aws

开发人员指南

Amazon Simple Queue Service



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Simple Queue Service: 开发人员指南

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆或者贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 Amazon SQS？ 1
使用 Amazon SQS 的优势 1
基本架构 1
分布式队列 1
消息生命周期 2
Amazon SQS、Amazon MQ 和 Amazon SNS 之间的区别 4
开始使用 6
设置 6
注册获取 AWS 账户 6
准备好使用示例代码吧 6
后续步骤 7
了解 Amazon SQS 控制台 7
队列类型 8
在 Amazon SQS 中实现请求-响应系统 10
创建标准队列 10
创建队列 11
使用标准队列发送消息 12
创建 FIFO 队列 13
创建队列 13
使用 FIFO 队列发送消息 15
常见任务 15
管理队列 17
编辑队列 17
接收和删除消息 17
确认队列为空 19
删除队列 20
清除队列 21
标准队列 22
亚马逊 SQS at-least-once 配送 22
队列和消息标识符 22
标准队列的标识符 23
FIFO 队列 25
FIFO 队列关键术语 26
FIFO 传送逻辑 27

发送消息 27
接收消息 28
多次重试 28
关于 FIFO 行为的其他注意事项 29
示例（帮助更好地理解） 29
仅处理一次 30
从标准队列移至 FIFO 队列 30
FIFO 队列和 Lambda 并发行为 31
FIFO 队列消息分组 32
Lambda 与 FIFO 队列的并发处理 32
使用案例示例 32
FIFO 队列的高吞吐量 33
使用案例 33
分区和数据分发 33
为 FIFO 队列启用高吞吐量 36
队列和消息标识符 36
FIFO 队列的标识符 23
FIFO 队列的其他标识符 38
限额 39
FIFO 队列配额 39
Amazon SQS 配额 39
标准队列配额 40
消息配额 42
策略配额 47
特征和功能 48
Dead-letter 队列 48
使用死信队列策略 48
了解死信队列的消息保留期 49
配置死信队列 49
配置死信队列重新驱动 50
CloudTrail 更新和权限要求 59
使用 Amazon 为死信队列创建警报 CloudWatch 62
Amazon SQS 的消息元数据 63
消息属性 63
消息系统属性 67
处理消息所需的资源 67

列表队列分页 68
成本分配标签 68
短轮询和长轮询 69
通过短轮询来使用消息 69
通过长轮询来使用消息 70
长轮询和短轮询之间的区别 71
可见性超时 71
可见性超时使用案例 71
设置和调整可见性超时 72
传输中消息和配额 72
了解标准队列和 FIFO 队列中的可见性超时 73
处理故障 73
更改和终止可见性超时 73
最佳实践 73
公平队列 74
如何使用公平队列 76
何时使用公平队列 76
公平队列 CloudWatch 指标 76
公平排队是如何运作的 77
延迟队列 79
临时队列 80
虚拟队列 80
Request-response 消息传送模式（虚拟队列） 81
示例场景：处理登录请求 82
清除队列 84
消息计时器 84
访问 EventBridge 管道 85
管理大型消息 86
使用适用于 Java 的扩展型客户端库 87
使用适用于 Python 的扩展型客户端库 93
配置 Amazon SQS 96
Amazon SQS 的 ABAC 96
什么是 ABAC？ 96
为什么应该在 Amazon SQS 中使用 ABAC？ 97
访问控制标签 97
创建 IAM 用户和 Amazon SQS 队列 98

测试基于属性的访问控制 100
配置队列参数 101
配置访问策略 103
为队列配置 SSE-SQS 104
为队列配置 SSE-KMS 105
为队列配置标签 106
为队列订阅主题 107
跨账户订阅 108
跨区域订阅 108
配置 Lambda 触发器 109
先决条件 109
使用自动发送通知 EventBridge 110
消息属性 111
最佳实践 113
错误处理和有问题的消息 113
处理 Amazon SQS 中的请求错误 113
捕获 Amazon SQS 中有问题的消息 113
在 Amazon SQS 中设置死信队列保留时间 114
消息重复数据删除和分组 114
避免 Amazon SQS 中出现不一致的消息处理 114
使用消息重复数据删除 ID 115
使用消息组 ID 117
使用接收请求尝试 ID 118
消息处理和定时 118
在 Amazon SQS 中及时处理消息 119
在 Amazon SQS 中设置长轮询 119
在 Amazon SQS 中使用合适的轮询模式 120
Java SDK 示例 121
使用服务器端加密 121
向现有队列添加 SSE 121
为队列禁用 SSE 122
使用 SSE 创建队列 123
检索 SSE 属性 123
配置标签 123
列出标签 124
添加或更新标签 124

移除标签 125
发送消息属性 125
定义属性 126
发送带有属性的消息 127
使用 API 129
使用 AWS JSON 协议发出查询 API 请求 130
构造端点 130
发出 POST 请求 131
解释 Amazon SQS JSON API 响应 132
亚马逊 SQS AWS JSON 协议 FAQs 133
使用查询协议发出 AWS 查询 API 请求 136
构造端点 136
提出 GET 请求 137
发出 POST 请求 131
解释 Amazon SQS XML API 响应 138
对请求进行身份验证 139
基本身份验证流程 HMAC-SHA 140
第 1 部分：来自用户的请求 141
第 2 部分：来自的回应 AWS 142
批处理操作 143
批处理消息操作 144
启用客户端缓冲和请求批处理功能，并将其与 Amazon SQS 结合使用 144
利用水平扩缩和操作批处理，借助 Amazon SQS 来提高吞吐量 153
使用 AWS 软件开发工具包 165
使用 JMS 167
先决条件 167
使用 Java Messaging Library 168
创建 JMS 连接 168
创建 Amazon SQS 队列 169
同步发送消息 170
同步接收消息 171
异步接收消息 173
使用客户端确认模式 174
使用无序确认模式 175
将 JMS 客户端与其他 Amazon SQS 客户端配合使用 176
实际可用的 Java 示例：将 JMS 与标准队列相结合使用 177

ExampleConfiguration.java ... 177
TextMessageSender.java ... 180
SyncMessageReceiver.java ... 182
AsyncMessageReceiver.java ... 183
SyncMessageReceiverClientAcknowledge.java ... 186
SyncMessageReceiverUnorderedAcknowledge.java ... 189
SpringExampleConfiguration.xml ... 193
SpringExample.java ... 194
ExampleCommon.java ... 196
支持的 JMS 1.1 实施 ... 198
支持的常用接口 ... 198
支持的消息类型 ... 198
支持的消息确认模式 ... 199
JMS 定义的标头和预留属性 ... 199
教程 ... 201
使用创建 Amazon SQS 队列 CloudFormation ... 201
从 VPC 发送消息 ... 203
步骤 1：创建 Amazon EC2 密钥对 ... 203
步骤 2：创建 AWS 资源 ... 204
步骤 3：确认您的 EC2 实例不可公开访问 ... 205
步骤 4：为 Amazon SQS 创建 Amazon VPC 端点 ... 206
步骤 5：向 Amazon SQS 队列发送消息 ... 207
代码示例 ... 208
基本功能 ... 209
Hello Amazon SQS ... 210
操作 ... 222
场景 ... 395
创建消息应用程序 ... 396
创建 Messenger 应用程序 ... 397
创建 Amazon Textract 浏览器应用程序 ... 398
创建并发布到 FIFO 主题 ... 399
检测视频中的人物和对象 ... 411
使用 S3 管理大型消息 ... 412
处理 S3 事件通知 ... 416
将消息发布到队列 ... 420
发送和接收批量消息 ... 556

将适用于.NET 的 AWS 消息处理框架与 Amazon SQS 配合使用 587
使用 Amazon SQS Java Messaging Library 处理 JMS 接口 588
处理队列标签 610
无服务器示例 613
通过 Amazon SQS 触发器调用 Lambda 函数 614
报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败 622
故障排除 633
访问被拒绝错误 633
Amazon SQS 队列策略和 IAM 策略 634
AWS Key Management Service (AWS KMS) 权限 634
VPC 端点策略 636
组织服务控制策略 636
API 错误 636
QueueDoesNotExist 错误 636
InvalidAttributeValue 错误 637
ReceiptHandle 错误 638
死信队列和死信队列重新驱动问题 638
死信队列问题 638
死信队列重新驱动问题 640
FIFO 节流问题 641
ReceiveMessage API 调用未返回的消息 642
空队列 642
已达到传输中消息数量限制 642
消息延迟 642
消息正在传输中 643
轮询方法 643
网络错误 643
ETIMEOUT error 643
UnknownHostException error 644
使用 X-Ray 排查队列问题 645
安全性 646
数据保护 646
数据加密 647
互联网络流量隐私 658
使用双栈端点实现连接 660
Identity and access management 660

受众 660
使用身份进行身份验证 660
使用策略管理访问 661
概述 663
Amazon Simple Queue Service 如何与 IAM 结合使用 669
AWS 托管策略 674
问题排查 675
使用 策略 677
日志记录和监控 720
记录 API 调用 722
监控队列 726
合规性验证 740
恢复能力 740
分布式队列 740
基础结构安全性 741
最佳实践 741
确保队列不可公开访问 742
实施最低权限访问 742
对需要 Amazon SQS 访问权限的应用程序和 AWS 服务使用 IAM 角色 743
实施服务器端加密 743
实施传输中数据加密 743
考虑使用 VPC 端点来访问 Amazon SQS 743
相关资源 744
文档历史记录 745
..... dccli

什么是 Amazon Simple Queue Service？

Amazon Simple Queue Service (Amazon SQS) 提供了一个安全、持久且可用的托管队列，以允许您集成和分离分布式软件系统与组件。Amazon SQS 提供常见构造，例如死信队列和成本分配标签。它提供了一个通用 Web 服务 API，您可以使用 AWS SDK 支持的任何编程语言访问该API。

使用 Amazon SQS 的优势

- 安全性 – 您可以控制谁能向 Amazon SQS 队列发送消息以及谁能从该队列接收消息。您可以选择通过使用默认 Amazon SQS 托管服务器端加密 (SSE) 或使用在 AWS Key Management Service (AWS KMS) 中管理的自定义 SSE 密钥来保护队列中的消息内容，从而传输敏感数据。
- 持久性 – 为确保您消息的安全，Amazon SQS 将消息存储在多个服务器上。标准队列支持至少一次消息传送，而 FIFO 队列支持仅处理一次和高吞吐量模式。
- 可用性 – Amazon SQS 使用冗余基础设施为生成和使用消息提供高度并发的消息访问和高可用性。
- 可扩展性 – Amazon SQS 可独立处理各个缓冲的请求，并可透明扩展以处理任何负载增加或峰值，无需任何预配置指令。
- 可靠性 – Amazon SQS 在处理期间锁定消息，以便多个创建者同时发送消息，多个使用者同时接收消息。
- 自定义 – 您的队列不必完全相同，例如，您可以设置队列的默认延迟。您可以使用 Amazon Simple Storage Service（Amazon S3）或 Amazon DynamoDB 存储大于 1 MiB 的消息内容，Amazon SQS 保留指向 Amazon S3 对象的指针；或者，您也可以将一条大消息拆分为几条小消息。

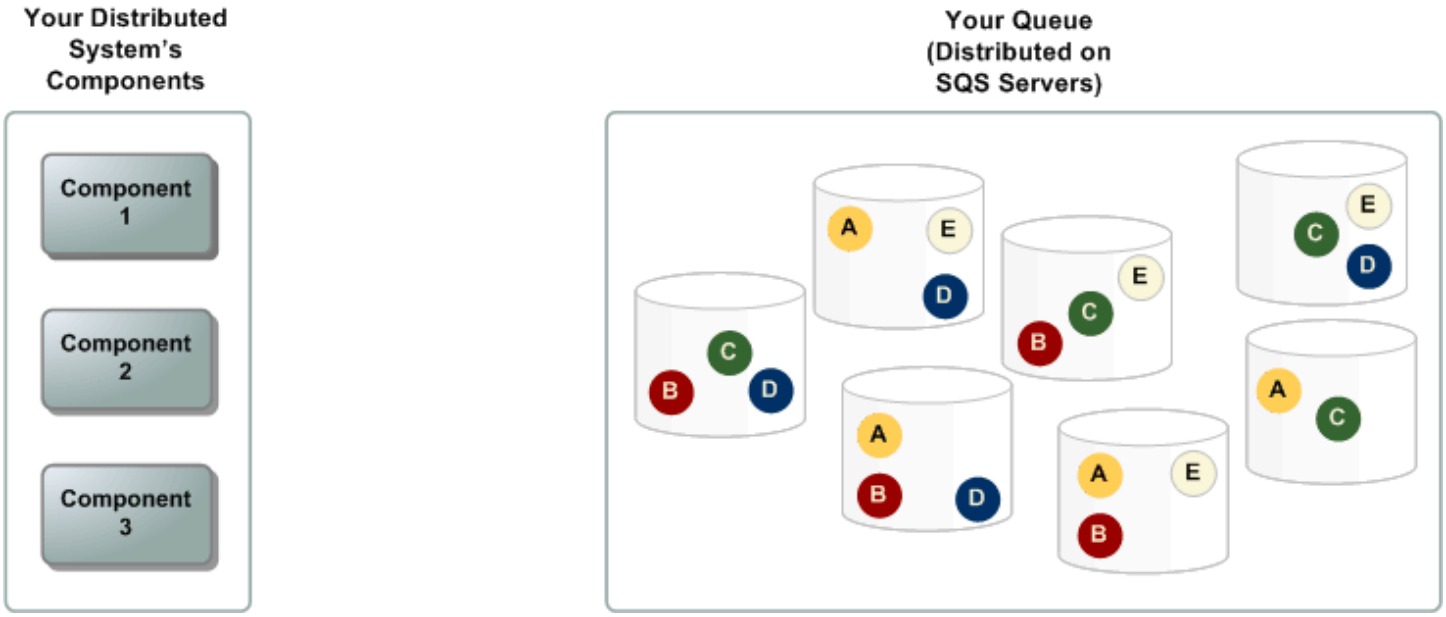
基本 Amazon SQS 架构

本部分简要介绍了分布式消息收发系统的组成部分，并说明了 Amazon SQS 消息的生命周期。

分布式队列

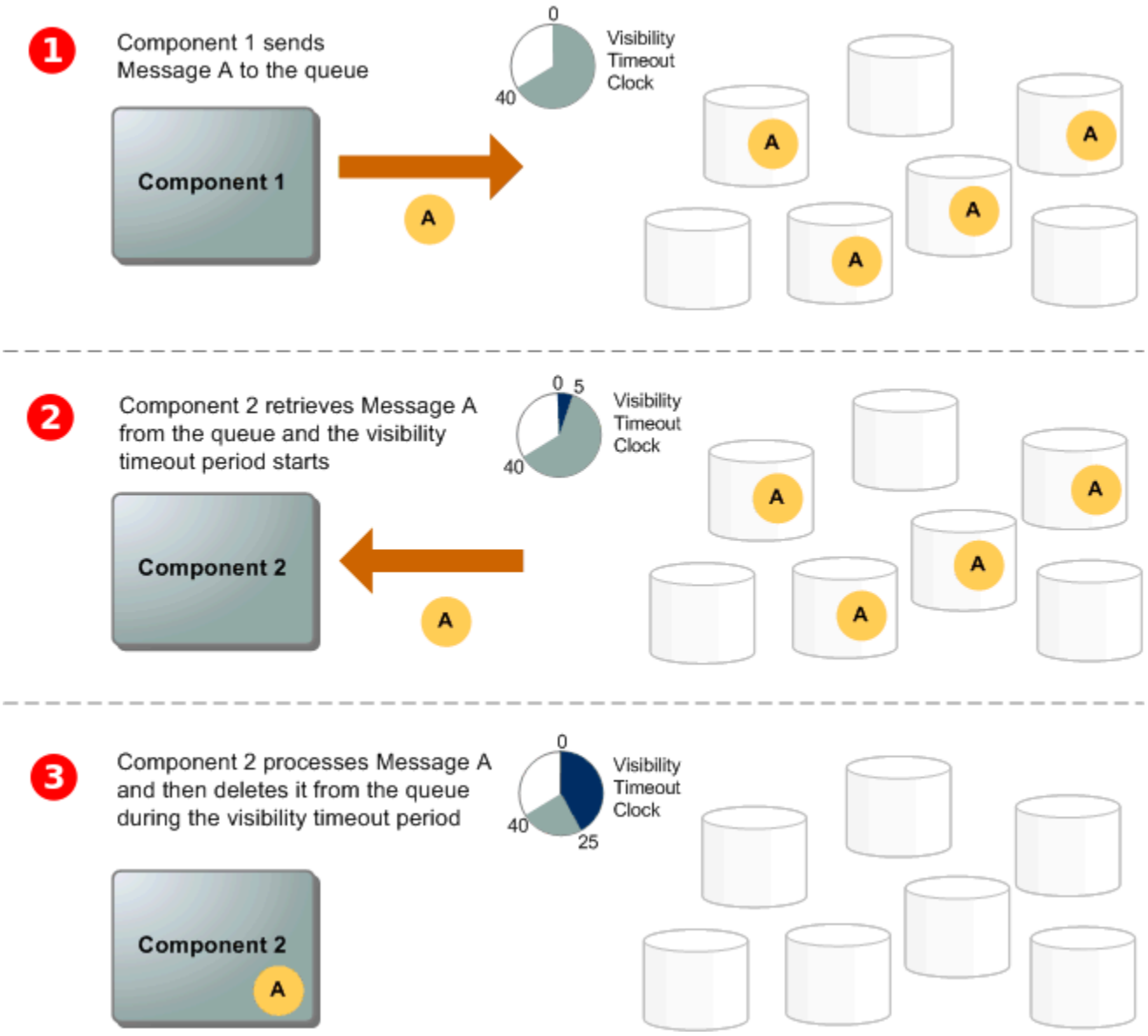
分布式消息收发系统有三个主要部分：分布式系统的组件、队列（分布在 Amazon SQS 服务器上）以及队列中的消息。

在以下场景中，您的系统有多个创建者（向队列发送消息的组件）和使用者（从队列接收消息的组件）。队列（保存从 A 到 E 的消息）在多个 Amazon SQS 服务器上冗余存储消息。



消息生命周期

以下场景介绍 Amazon SQS 消息在队列中从创建到删除的整个生命周期。



1

创建者（Component 1）将消息 A 发送到一个队列，该消息以冗余方式在 Amazon SQS 服务器间分布。

2

使用者（Component 2）准备好处理消息时，会从队列使用消息，然后返回消息 A。在处理消息 A 期间，它仍保留在队列中，并且在可见性超时期间不返回至后续接收请求。

3

使用者（Component 2）从队列中删除消息 A，以阻止该消息在可见性超时过期后被再次接收和处理。

> **Note**
>
> Amazon SQS 会自动删除在队列中已过了最大消息保存期的消息。默认的消息保存期为 4 天。不过，您可使用 `SetQueueAttributes` 操作将消息保存期设为介于 60 秒和 1209600 秒（14 天）之间的值。

Amazon SQS、Amazon MQ 和 Amazon SNS 之间的区别

Amazon SQS、Amazon SNS 和 Amazon MQ 提供高度可扩展且易于使用的托管式消息收发服务，每种服务在分布式系统中都有特定的作用。以下是对这些服务之间的区别的详细介绍：

Amazon SQS 作为队列服务，可以实现分布式软件系统和组件的分离和扩展。它通常通过单个订阅用户来处理消息，非常适合对顺序和防止消息丢失有严格要求的工作流程。如果需要更广泛的分发，可以将 Amazon SQS 与 Amazon SNS 集成，启用扇出消息传送模式，从而有效地将消息同时推送给多个订阅用户。

Amazon SNS 让发布者通过作为通信渠道的主题向多个订阅用户发送消息。订阅用户可以使用受支持的端点类型接收已发布的消息，例如，Amazon Data Firehose、Amazon SQS、Lambda、HTTP、电子邮件、移动推送通知和移动短信（SMS）。该服务非常适合需要即时通知的场景，例如实时用户参与或警报系统。为了防止订阅用户离线时消息丢失，将 Amazon SNS 与 Amazon SQS 队列消息集成可以确保消息传递的一致性。

Amazon MQ 最适合希望从传统消息代理迁移的企业，支持 AMQP 和 MQTT 等标准消息协议，以及 Apache ActiveMQ 和 RabbitMQ。它对需要稳定、可靠消息收发的遗留系统提供兼容支持，而无需进行大量重新配置。

下表概述了每种服务的资源类型：

资源类型	Amazon SNS	Amazon SQS	Amazon MQ
同步	否	否	是
异步	支持	是	是
队列	否	是	是
Publisher-subscriber 消息	是	否	是

资源类型	Amazon SNS	Amazon SQS	Amazon MQ
消息代理	否	否	是

我们建议对新的应用程序使用 Amazon SQS 和 Amazon SNS，这样可以实现几乎不受限制的可扩展性和简单 API。对于高容量应用程序来说，它们采用的即用即付定价模式通常可以提供更具成本效益的解决方案。我们建议 Amazon MQ 从依赖于 JMS 等 API 的兼容性或高级消息队列协议 (AMQP)、MQTT 和面向简单文本的消息协议 (STOMP) 等协议的兼容性的现有消息代理迁移应用程序。 OpenWire

Amazon SQS 入门

本主题将指导您使用 Amazon SQS 控制台创建和管理标准队列与 FIFO 队列。您将了解如何浏览控制台、查看队列属性以及区分队列类型。关键任务包括发送、接收和配置消息，调整可见性超时和消息保留等参数，以及通过策略管理队列访问权限。

主题

- 设置 Amazon SQS
- 了解 Amazon SQS 控制台
- Amazon SQS 队列类型
- 创建 Amazon SQS 标准队列并发送消息
- 创建 Amazon SQS FIFO 队列并发送消息
- Amazon SQS 入门的常见任务

设置 Amazon SQS

在首次使用 Amazon SQS 之前，您必须完成以下步骤：

注册获取 AWS 账户

要开始使用 AWS，你需要一个 AWS 账户。有关创建的信息 AWS 账户，请参阅《AWS 账户管理 参考指南》 AWS 账户中的入门指南。

准备好使用示例代码吧

本指南包括使用适用于 Java 的 AWS SDK 的示例。要运行示例代码，请按照适用于 Java 的 AWS SDK 2.0 使用入门中的设置说明进行操作。

您可以使用其他编程语言开发 AWS 应用程序，例如 Go、JavaScript、Python 和 Ruby。有关更多信息，请参阅构建工具 AWS。

> **Note**
>
> 使用 AWS Command Line Interface (AWS CLI) 或 Windows 之类的工具，你无需编写代码即可浏览 Amazon SQS。 PowerShell您可以在《AWS CLI 命令参考》的 Amazon SQS 部分中

找到 AWS CLI 示例。你可以在 AWS Tools for PowerShell Cmdlet 参考的 “亚马逊简单队列服务” 部分找到 Windows PowerShell 示例。

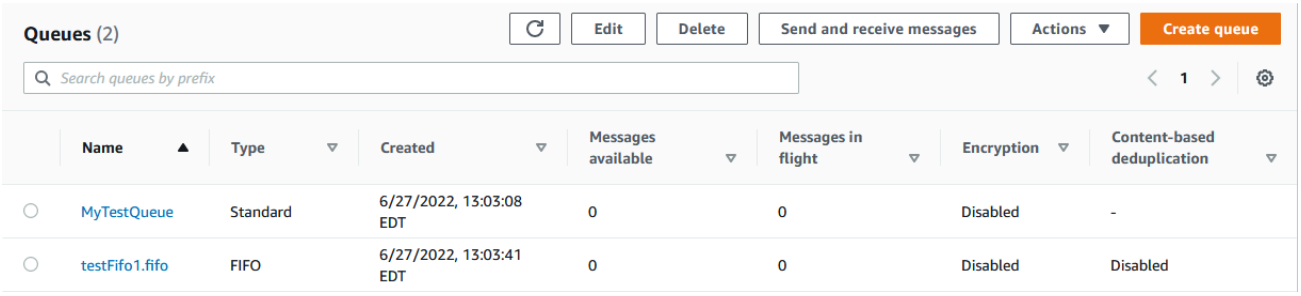
后续步骤

现在，您可以开始使用 AWS 管理控制台管理 Amazon SQS 队列和消息了。

了解 Amazon SQS 控制台

打开 Amazon SQS 控制台后，从导航窗格中选择队列。队列页面提供有关活动区域中所有队列的信息。

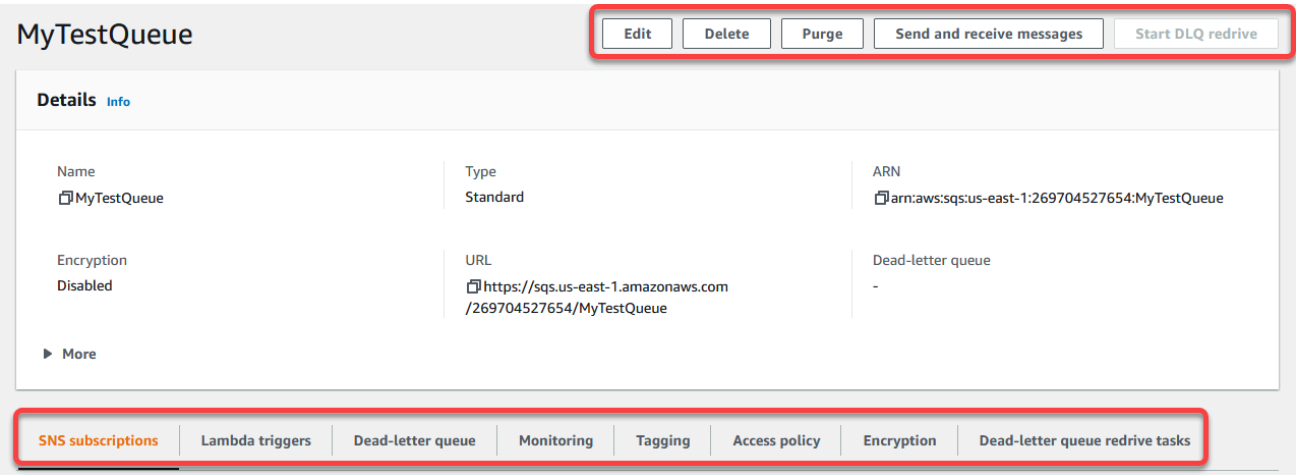
每个队列条目都提供有关队列的基本信息，包括队列类型和关键属性。标准队列经过优化，旨在实现最大吞吐量和最优消息排序，而先进先出（FIFO）队列则优先考虑消息排序和唯一性，适用于需要严格消息顺序的应用程序。



交互式元素和操作

在“队列”页面，您可以使用多个选项来管理队列：

1. 快速操作：您可以使用每个队列名称旁边的下拉菜单快速访问常见操作，例如发送、查看或删除消息、配置触发器以及删除队列本身。
2. 详细视图和配置：单击队列名称可以打开“队列详细信息”页面，您可以在其中更深入地了解队列设置和配置。在该页面，您可以调整消息保留期、可见性超时和最大消息大小等参数，以便根据应用程序的要求定制队列。



区域选择和资源标签

确保您处于正确的 AWS 区域，以便有效地访问和管理队列。此外，您可以考虑使用资源标签来整理和分类队列，从而在 AWS 共享环境中实现更好的资源管理、成本分配和访问控制。

借助 Amazon SQS 控制台中提供的特征和功能，您可以高效地管理消息基础设施，优化队列性能，并确保为应用程序提供可靠的消息传递。

Amazon SQS 队列类型

Amazon SQS 支持两种类型的队列：标准队列和 FIFO 队列。使用下表来确定哪种队列最适合您的需求。

标准队列	FIFO 队列
无限制吞吐量 – 标准队列支持每个操作（SendMessage、ReceiveMessage 或 DeleteMessage）每秒几乎无限次的 API 调用。这种高吞吐量使其非常适合需要快速处理大量消息的应用场景，例如实时数据流或大型应用程序。虽然标准队列会根据需求自动扩展，但您必须监控使用模式以确保实现最佳性能，特别是在工作负载较高的区域。 At-least-once 交付 — 保证至少传送一次，这意味着每条消息至少传送一次，但在某些情况下，由于重试或网络延迟，一条消息可能会多次传	高吞吐量 – 如果您使用批处理，则 FIFO 队列支持每个 API 方法（SendMessageBatch、ReceiveMessage 或 DeleteMessageBatch）每秒最多处理 3000 条消息。这种吞吐量依赖于每秒 300 次 API 调用，每次 API 调用都批量处理 10 条消息。通过启用高吞吐量模式，您可以将每秒事务数（TPS）扩展到 3 万个，但这会放宽消息组内的顺序要求。在不使用批处理的情况下，FIFO 队列支持每个 API 方法（SendMessage、ReceiveMessage 或 DeleteMes

标准队列	FIFO 队列
送。在设计应用程序时，您应该使用幂等性操作来处理可能出现的重复消息，从而确保多次处理同一条消息不会影响系统的状态。 Best-effort 排序 — 提供尽力排序，这意味着，尽管 Amazon SQS 尝试按照发送顺序传送消息，但不能保证这一点。在某些情况下，消息可能会无序到达，尤其是在高吞吐量或故障恢复的情况下。对于消息处理顺序至关重要的应用程序，您应该处理应用程序中的重新排序逻辑，或者使用 FIFO 队列来保证严格的排序。 耐久性和冗余 — 标准队列通过在多个 AWS 可用区存储每条消息的多个副本来确保高耐久性。这样可以确保即使基础设施出现故障，消息也不会丢失。 可见性超时：Amazon SQS 让您能够配置可见性超时以控制消息在被接收后隐藏多长时间，从而确保在消息得到完全处理或超时到期之前，其他用户无法处理该消息。	sage ）每秒最多 300 次 API 调用。如果您需要更高的吞吐量，可以通过 AWS Support Center 请求增加配额。要启用高吞吐量模式，请参阅为 Amazon SQS 中的 FIFO 队列启用高吞吐量。 Exactly-once 处理 — FIFO 队列将每条消息传送一次，并保持其可用状态，直到您对其进行处理和删除。通过使用 MessageDeduplicationId 或基于内容的重复数据删除等功能，即使由于网络问题或超时而重试，也可以防止消息重复。 First-in-first-out 交付 — FIFO 队列确保您按照每个消息组内发送消息的顺序接收消息。通过将消息分配到多个组中，您可以并行处理这些消息，同时保持每个组内的顺序。
	

标准队列	FIFO 队列
当吞吐量至关重要时，请使用标准队列在应用程序之间发送数据，例如： • 将实时用户请求从密集型后台工作中分离。允许用户快速上传媒体文件，同时您在后台处理诸如调整大小或编码等任务，从而确保快速响应而不会使系统过载。 • 将任务分配给多个 Worker 节点。将大量信用卡验证请求分配到多个 Worker 节点，并使用幂等性操作处理重复消息，以避免处理错误。 • 对消息进行批处理，以供未来使用。将多个条目排入队列，以便批量添加至数据库。由于消息顺序无法保证，因此您可以将系统设计为能够处理无序消息（如有需要）。	如果事件顺序非常重要，可以使用 FIFO 队列在应用程序之间发送数据，例如： • 确保按正确的顺序运行用户输入的命令。这是 FIFO 队列的关键应用场景，在这种场景下，命令的顺序至关重要。例如，如果用户在应用程序中执行一系列操作，FIFO 队列可以确保按输入的顺序执行操作。 • 通过按正确的顺序发送价格修改来显示正确的产品价格。FIFO 队列可以确保产品价格的多次更新按顺序到达并得到处理。如果不使用 FIFO，系统可能会在价格上涨后处理降价，从而导致显示不正确的数据。 • 防止学员在注册账户之前参加课程。借助 FIFO 队列，您可以确保注册流程按正确的顺序执行。系统首先处理账户注册，然后处理选课请求，从而防止系统提前处理选课请求。

在 Amazon SQS 中实现请求-响应系统

实施请求-响应和远程程序调用 (RPC) 系统时，请记住以下最佳实践：

- 在启动时创建回复队列：在启动时为每个创建者创建回复队列，而不是为每条消息创建回复队列。使用关联 ID 消息属性高效地将回复与请求进行对应。
- 避免在创建者之间共享回复队列：确保每个创建者都有自己的回复队列。共享回复队列可能会导致创建者收到原本应发送给另一个创建者的响应消息。

有关使用临时队列客户端实施请求-响应模式的更多信息，请参阅 Request-response 消息传送模式（虚拟队列）。

创建 Amazon SQS 标准队列并发送消息

您可以使用 Amazon SQS 控制台创建标准队列并发送消息。本主题还重点介绍了最佳实践，包括避免在队列名称中使用敏感信息以及使用托管服务器端加密。

使用 Amazon SQS 控制台创建标准队列

> **Important**
>
> 2022 年 8 月 17 日，默认服务器端加密 (SSE) 应用于所有 Amazon SQS 队列。
> 请勿在队列名称中添加个人身份信息 (PII) 或其他机密/敏感信息。许多 Amazon Web Services 都可以访问队列名称，包括账单和 CloudWatch 日志。队列名称不适合用于私有或敏感数据。

创建 Amazon SQS 标准队列

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 选择创建队列。
3. 对于类型，默认设置为标准队列类型。

   > **Note**
   >
   > 创建队列后无法更改队列类型。

4. 输入队列的名称。
5. （可选）控制台为队列配置参数设置默认值。在配置下，您可以为以下参数设置新值：

   a. 对于可见性超时，输入持续时间和单位。范围从 0 秒到 12 小时。默认 值为 30 秒。

   b. 对于消息保留期，输入持续时间和单位。范围从 1 分钟到 14 天。默认值为 4 天。

   c. 对于传送延迟，输入持续时间和单位。范围从 0 秒到 15 分钟。默认值为 0 秒。

   d. 对于最大消息大小，输入一个值。范围从 1 KiB 到 1024 KiB 不等。默认值为 1024 KiB。

   e. 在接收消息等待时间中，输入一个值。范围从 0 秒到 20 秒。默认值为 0 秒，这会设置短轮询。任何非零值都会设置长轮询。

6. （可选）定义访问策略。访问策略定义了可以访问队列的账户、用户和角色。访问策略还定义了用户可以访问的操作（例如 `SendMessage`、`ReceiveMessage` 或 `DeleteMessage`）。默认策略仅允许队列所有者发送和接收消息。

   要定义访问策略，请执行以下操作之一：

   - 选择基本，配置谁可以向队列发送消息以及谁可以从队列接收消息。控制台根据您的选择创建策略，并在只读 JSON 面板中显示生成的访问策略。

- 选择高级，直接修改 JSON 访问策略。这允许您指定每个主体（账户、用户或角色）可以执行的一组自定义操作。

7. 对于重新驱动允许策略，请选择启用。从以下选项中选择一个：全部允许、按队列或全部拒绝。选择按队列时，按 Amazon 资源名称 (ARN) 指定最多 10 个源队列的列表。
8. 默认情况下，Amazon SQS 提供托管服务器端加密。要选择加密密钥类型或禁用 Amazon SQS 托管服务器端加密，请展开加密。有关加密密钥类型的更多信息，请参阅使用 SQS 托管的加密密钥为队列配置服务器端加密和使用 Amazon SQS 控制台为队列配置服务器端加密。

   > **Note**
   >
   > 启用 SSE 后，对加密队列的匿名 `SendMessage` 和 `ReceiveMessage` 请求将被拒绝。Amazon SQS 安全最佳实践建议不要使用匿名请求。如果您想向 Amazon SQS 队列发送匿名请求，请确保禁用 SSE。

9. （可选）要将死信队列配置为接收无法投递的消息，请展开死信队列。
10. （可选）要向队列添加标签，请展开标签。
11. 选择创建队列。Amazon SQS 创建队列并显示队列的详细信息页面。

Amazon SQS 会在整个系统中传播有关新队列的信息。由于 Amazon SQS 是一种分布式系统，因此在控制台在队列页面上显示队列之前，您可能会遇到轻微的延迟。

使用标准队列发送消息

创建队列后，您可以向该队列发送消息。

1. 在左侧导航窗格中，选择队列。在队列列表中，选择刚刚创建的队列。
2. 从操作中，选择发送和接收消息。

   控制台会显示发送和接收消息页面。
3. 对于消息正文，输入消息文本。
4. 对于标准队列，您可以为传送延迟输入值并选择单位。例如，输入 `60` 并选择秒。有关更多信息，请参阅 Amazon SQS 消息计时器。
5. 选择发送消息。

   发送消息后，控制台会显示一条成功消息。选择查看详细信息，显示有关已发送消息的信息。

创建 Amazon SQS FIFO 队列并发送消息

您可以创建 Amazon SQS FIFO 队列并通过控制台发送消息。本主题介绍了如何设置队列参数，包括可见性超时、消息保留和重复数据删除，同时遵循安全最佳实践，例如避免在队列名称中使用敏感信息以及启用服务器端加密。它还包括定义访问策略、配置死信队列以及发送带有特定于 FIFO 的属性（例如消息组 ID 和重复数据删除 ID）的消息。

使用 Amazon SQS 控制台创建 FIFO 队列

您可以使用 Amazon SQS 控制台创建 FIFO 队列。该控制台为除队列名称之外的所有设置提供默认值。

> **Important**
>
> 2022 年 8 月 17 日，默认服务器端加密 (SSE) 应用于所有 Amazon SQS 队列。
> 请勿在队列名称中添加个人身份信息 (PII) 或其他机密/敏感信息。许多 Amazon Web Services 都可以访问队列名称，包括账单和 CloudWatch 日志。队列名称不适合用于私有或敏感数据。

创建 Amazon SQS FIFQ 队列

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 选择创建队列。
3. 对于类型，默认设置为标准队列类型。要创建 FIFO 队列，请选择 FIFO。

   > **Note**
   >
   > 创建队列后无法更改队列类型。

4. 输入队列的名称。

   FIFO 队列名称必须以 `.fifo` 后缀结尾。后缀计入 80 个字符的队列名称配额。要确定队列是否为 FIFO，您可以检查队列名称是否以该后缀结尾。

5. （可选）控制台为队列配置参数设置默认值。在配置下，您可以为以下参数设置新值：

   a. 对于可见性超时，输入持续时间和单位。范围从 0 秒到 12 小时。默认 值为 30 秒。
   b. 对于消息保留期，输入持续时间和单位。范围从 1 分钟到 14 天。默认值为 4 天。
   c. 对于传送延迟，输入持续时间和单位。范围从 0 秒到 15 分钟。默认值为 0 秒。

d. 对于最大消息大小，输入一个值。范围从 1 KiB 到 1024 KiB 不等。默认值为 1024 KiB。

e. 在接收消息等待时间中，输入一个值。范围从 0 秒到 20 秒。默认值为 0 秒，这会设置短轮询。任何非零值都会设置长轮询。

f. 对于 FIFO 队列，选择基于内容的重复数据删除，以启用基于内容的重复数据删除。默认情况下，该设置处于禁用状态。

g. （可选）要让 FIFO 队列启用更高的吞吐量以便在队列中发送和接收消息，请选择启用高吞吐量 FIFO。

选择此选项会将相关选项（重复数据删除范围和 FIFO 吞吐量限制）更改为为 FIFO 队列启用高吞吐量所需的设置。如果更改使用高吞吐量 FIFO 所需的任何设置，则队列正常吞吐量生效，重复数据删除按规定进行。有关更多信息，请参阅Amazon SQS 中 FIFO 队列的高吞吐量和Amazon SQS 消息配额。

6. （可选）定义访问策略。访问策略定义了可以访问队列的账户、用户和角色。访问策略还定义了用户可以访问的操作（例如 `SendMessage`、`ReceiveMessage` 或 `DeleteMessage`）。默认策略仅允许队列所有者发送和接收消息。

要定义访问策略，请执行以下操作之一：

- 选择基本，配置谁可以向队列发送消息以及谁可以从队列接收消息。控制台根据您的选择创建策略，并在只读 JSON 面板中显示生成的访问策略。
- 选择高级，直接修改 JSON 访问策略。这允许您指定每个主体（账户、用户或角色）可以执行的一组自定义操作。

7. 对于重新驱动允许策略，请选择启用。从以下选项中选择一个：全部允许、按队列或全部拒绝。选择按队列时，按 Amazon 资源名称 (ARN) 指定最多 10 个源队列的列表。

8. 默认情况下，Amazon SQS 提供托管服务器端加密。要选择加密密钥类型或禁用 Amazon SQS 托管服务器端加密，请展开加密。有关加密密钥类型的更多信息，请参阅使用 SQS 托管的加密密钥为队列配置服务器端加密和使用 Amazon SQS 控制台为队列配置服务器端加密。

> **Note**
>
> 启用 SSE 后，对加密队列的匿名 `SendMessage` 和 `ReceiveMessage` 请求将被拒绝。Amazon SQS 安全最佳实践建议不要使用匿名请求。如果您想向 Amazon SQS 队列发送匿名请求，请确保禁用 SSE。

9. （可选）要将死信队列配置为接收无法投递的消息，请展开死信队列。

10. （可选）要向队列添加标签，请展开标签。

11. 选择创建队列。Amazon SQS 创建队列并显示队列的详细信息页面。

Amazon SQS 会在整个系统中传播有关新队列的信息。由于 Amazon SQS 是一种分布式系统，因此在控制台在队列页面上显示队列之前，您可能会遇到轻微的延迟。

创建队列后，您可以向该队列发送消息，以及接收和删除消息。除队列类型外，您还可以编辑任何队列配置设置。

使用 FIFO 队列发送消息

创建队列后，您可以向该队列发送消息。

1. 在左侧导航窗格中，选择队列。在队列列表中，选择刚刚创建的队列。
2. 从操作中，选择发送和接收消息。

   控制台会显示发送和接收消息页面。
3. 对于消息正文，输入消息文本。
4. 对于 First-In-First-Out (FIFO) 队列，请输入消息组 ID。有关更多信息，请参阅 Amazon SQS 中的 FIFO 队列传递逻辑。
5. （可选）对于 FIFO 队列，您可以输入消息重复数据删除 ID。如果您为队列启用了基于内容的重复数据删除，则不需要消息重复数据删除 ID。有关更多信息，请参阅 Amazon SQS 中的 FIFO 队列传递逻辑。
6. FIFO 队列不支持单个消息的计时器。有关更多信息，请参阅 Amazon SQS 消息计时器。
7. 选择发送消息。

   发送消息后，控制台会显示一条成功消息。选择查看详细信息，显示有关已发送消息的信息。

Amazon SQS 入门的常见任务

现在，您已经创建了队列，并了解了如何发送、接收和删除消息，您可能想要尝试以下操作：

- 触发 Lambda 函数以自动处理传入的消息，从而无需持续轮询即可实现事件驱动的工作流。
- 配置队列，包括 SSE 和其他特征。
- 发送带有属性的消息。
- 从 VPC 发送消息。
- 了解 Amazon SQS 的功能和架构。

- 了解可帮助您充分利用 Amazon SQS 的指南和注意事项。
- 浏览软件开发工具包的 Amazon AWS SQS 示例，例如AWS SDK for Java 2.x 开发者指南。
- 了解亚马逊 SQS 命令。 AWS CLI
- 了解 Amazon SQS API 操作。
- 了解如何通过编程方式与 Amazon SQS 交互。参见使用 APIs和探索AWS 开发中心：
  - Java
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Windows 和 .NET

- 了解如何监控成本和资源。
- 了解如何保护您的数据。
- 了解有关 Amazon SQS 工作流的更多信息。

管理 Amazon SQS 队列

了解如何通过控制台管理 Amazon SQS 队列，包括编辑队列设置、接收和删除消息、确认队列是否为空，以及删除或清除队列。了解高效处理消息的最佳实践，例如使用长轮询、管理可见性超时以及通过监控控制面板或 AWS CLI验证指标。遵循实际步骤来有效维护队列和处理消息，同时最大限度地减少中断。

使用控制台编辑 Amazon SQS 队列

您可以使用 Amazon SQS 控制台编辑队列配置参数（队列类型除外），并根据需要修改或删除特征。

编辑 Amazon SQS 队列（控制台）

1. 打开 Amazon S3 控制台的队列页面。
2. 选择一个队列，然后选择编辑。
3. （可选）在配置下，更新队列的配置参数。
4. （可选）要更新访问策略，请在访问策略下修改 JSON 策略。
5. （可选）要更新死信队列重新驱动允许策略，请展开重新驱动允许策略。
6. （可选）要更新或删除加密，请展开加密。
7. （可选）要添加、更新或删除死信队列（允许您接收无法投递的消息），请展开死信队列。
8. （可选）要添加、更新或删除队列的标签，请展开标签。
9. 选择保存。

    - 控制台会显示队列的详细信息页面。

在 Amazon SQS 中接收和删除消息

向 Amazon SQS 队列发送消息后，您可以检索并删除这些消息，以处理应用程序工作流。此过程可确保消息处理的安全性和可靠性。本主题将指导您使用 Amazon SQS 控制台检索和删除消息，并说明优化此操作的关键设置。以下是消息接收和删除的重要概念：

1. 接收消息

    - 从 Amazon SQS 队列检索消息时，您无需指定特定消息。相反，您需要在单次请求中指定要检索的最大消息数量（最多 10 条）。

- 由于 Amazon SQS 的分布式特性，从消息较少的队列中检索时可能会返回空响应。为缓解此问题：
  - 使用长轮询，等待有消息可用或投票超时。此方法可减少不必要的轮询成本并提升效率。
  - 必要时重新发出请求。

2. 消息可见性和删除
   - 消息在检索后不会自动删除。此功能可确保您在应用程序故障或网络中断时可以重新处理消息。
   - 处理完成后，您必须明确发送删除请求才能永久移除消息。此操作将确认消息已成功处理。
   - 通过 Amazon SQS 控制台检索的消息仍然可见，可供重新检索。调整自动环境的可见性超时设置，以便在消息处理期间暂时隐藏来自其他使用者的消息。
3. 可见性超时
   - 此设置决定了消息在检索后保持隐藏状态的时长。请设置适当的超时时间，以确保消息仅被处理一次，并防止分布式处理过程中出现重复处理的情况。

使用控制台接收和删除消息的步骤

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 在队列页面上，选择要从中接收消息的队列，然后选择发送和接收消息。
4. 在发送和接收消息页面中，选择轮询消息。

   Amazon SQS 会显示一个进度条，用以指示轮询持续时间。检索到的消息将显示在消息部分，并显示：

   - 消息 ID
   - 发送日期
   - Size
   - 重启计数
5. 要删除消息，请选择要移除的消息，然后选择删除。

   在删除消息对话框中选择删除，确认删除操作。

有关高级操作（包括基于 API 的消息检索和删除）的更多详情，请参阅 Amazon SQS API 参考指南。

确认 Amazon SQS 队列为空

在大多数情况下，您可以使用长轮询来确定队列是否为空。在极少数情况下，即使队列中仍包含消息，您也可能会收到空响应，特别是在您在创建队列时为接收消息等待时间指定了较低的值时。本节将介绍如何确认队列为空。

确认队列为空（控制台）

1. 阻止所有创建者发送消息。
2. 通过以下网址打开 Amazon SQS 控制台：https://console.aws.amazon.com/sqs/。
3. 在导航窗格中，选择 Queues (队列)。
4. 在队列页面，选择队列。
5. 选择监控选项卡。
6. 在“监控”仪表板的右上角，选择“刷新”符号旁边的向下箭头。从下拉菜单中，选择自动刷新。将刷新间隔保留为 1 分钟。
7. 观察以下仪表板：

   - 延迟的消息的大致数量
   - 不可见消息的大致数量
   - 可见消息的大致数量

   当它们全部显示几分钟的 0 值时，队列为空。

确认队列为空（AWS CLI、AWS API）

1. 阻止所有创建者发送消息。
2. 重复运行以下命令之一：

   - AWS CLI: `get-queue-attributes`
   - AWSAPI –`GetQueueAttributes`
3. 观察以下属性的指标：

   - `ApproximateNumberOfMessagesDelayed`
   - `ApproximateNumberOfMessagesNotVisible`
   - `ApproximateNumberOfMessagesVisible`

当它们全部为几分钟的 0 时，队列为空。

如果您依赖于 Amazon CloudWatch 指标，请确保在考虑该队列为空之前，看到多个连续的零数据点。有关 CloudWatch 指标的更多信息，请参阅亚马逊 SQS 的可用 CloudWatch 指标。

删除 Amazon SQS 队列

如果您不再使用某个 Amazon SQS 队列，且近期无使用计划，请删除该队列。

> **Tip**
>
> 如果要在删除队列之前验证队列是否为空，请参阅确认 Amazon SQS 队列为空。

您可以删除队列，即使它不为空。要删除队列中的消息，但不删除队列本身，请清除队列。

删除队列（控制台）

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 在队列页面上，选择要删除的队列。
4. 选择删除。
5. 在删除队列对话框中，输入 **delete** 以确认删除。
6. 选择删除。

删除队列（AWS CLI 和 API）

根据您的需求选择适当的方法来删除队列：

- AWS CLI: `aws sqs delete-queue`
- AWS API: `DeleteQueue`

使用 Amazon SQS 控制台从队列中清除消息

要保留 Amazon SQS 队列但移除其所有消息，您可以清除该队列。这将删除所有消息，包括当前不可见（传输中）的消息。清除过程最多可能需要 60 秒，因此无论队列大小如何，都要等待整整 60 秒。

> **Important**
>
> 当清除队列时，您不能检索任何已从中删除的消息。

清除队列（控制台）

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 在队列页面上，选择要清除的队列。
4. 在操作中，选择清除。
5. 在清除队列对话框中，输入 **`purge`** 并选择清除，以确认清除。
   - 所有消息将从队列中清除。控制台会显示确认横幅。

Amazon SQS 标准队列

Amazon SQS 提供标准队列作为默认队列类型，支持几乎无限数量的每秒 API 调用，以便进行 SendMessage、ReceiveMessage 和 DeleteMessage 等操作。标准队列可确保 at-least-once消息传送，但由于高度分散的架构，可能会传送一个以上的消息副本，而且消息偶尔会出现乱序的情况。尽管如此，标准队列还是会尽最大努力保持消息的发送顺序。

当您使用发送消息时SendMessage，Amazon SQS 会在确认消息之前将其冗余存储在多个可用区 (AZs) 中。这种冗余可以确保即使某个计算机、网络或可用区出现故障，您依然可以访问消息。

您可以使用 Amazon SQS 控制台创建和配置队列。有关详细说明，请参阅使用 Amazon SQS 控制台创建标准队列。有关特定于 Java 的示例，请参阅 Amazon SQS Java SDK 示例。

标准队列的应用场景

标准消息队列适用于各种场景，前提是应用程序能够处理可能多次到达或无序到达的消息。例子包括：

- 将实时用户请求从密集型后台工作中分离：用户可以在系统在后台调整媒体大小或对媒体编码时上传媒体。
- 将任务分配给多个 Worker 节点：例如，处理大量信用卡验证请求。
- 批量处理消息以供进一步处理：安排在稍后的时间将多个条目添加到数据库中。

要了解与标准队列相关的配额，请参阅 Amazon SQS 标准队列配额。

有关使用标准队列的最佳实践，请参阅Amazon SQS 最佳实践。

亚马逊 SQS at-least-once 配送

Amazon SQS 会在多台服务器上存储消息的副本，以实现冗余和高可用性。在极少数情况下，当您接收或删除消息时，存储消息副本的某台服务器可能不可用。

如果出现这种情况，该服务器上的消息副本不会被删除，并且您在接收消息时可能会再次收到该消息副本。将应用程序设计为幂等 应用程序（多次处理同一消息时，它们不应受到不利影响）。

Amazon SQS 队列和消息标识符

本主题介绍了标准队列和 FIFO 队列的标识符。这些标识符可帮助您查找并操作特定队列和消息。

Amazon SQS 标准队列的标识符

有关以下标识符的更多信息，请参阅 Amazon Simple Queue Service API 参考。

队列名称和 URL

在创建新的队列时，您必须为 AWS 账户和区域指定唯一的队列名称。Amazon SQS 会为您创建的每个队列分配一个名为队列 URL 的标识符，其中包含队列名称和其他 Amazon SQS 组件。每当您要对队列执行操作时，都需要提供其队列 URL。

以下是名为 MyQueue 的队列的队列 URL，该队列由 AWS 账号为 123456789012 的用户所拥有。

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue
```

您可以通过列出队列并解析账号后的字符串，以编程方式检索队列的 URL。有关更多信息，请参阅 `ListQueues`。

消息 ID

每条消息都会收到一个系统分配的消息 ID，该 ID 由 Amazon SQS 在 `SendMessage` 响应中返回给您。此标识符用于识别消息。消息 ID 的最大长度为 100 个字符。

接收句柄

每当收到来自队列的消息时，您都会收到该消息的接收句柄。此句柄与接收消息的操作相关联，与消息本身无关。要删除消息或更改消息可见性，您必须提供接收句柄（而不是消息 ID）。因此，您必须始终先接收消息，然后才能删除它（您不能将消息放入队列中，然后重新调用它）。接收句柄的最大长度为 1024 个字符。

> **Important**
>
> 如果多次接收某条消息，则每次接收该消息时，您都会获得不同的接收句柄。在请求删除该消息时，您必须提供最近收到的接收句柄（否则，可能无法删除该消息）。

以下是一个跨三行显示的接收句柄示例。

```
MbZj6wDWli+JvwwJaBV+3dcjk2YW2vA3+STFFljTM8tJJg6HRG6PYSasuWXPJB+Cw
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq2OKpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE
```

```
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Amazon SQS FIFO 队列

FIFO（先进先出）队列除了具有标准队列的所有功能之外，还能在操作和事件的顺序很重要或不能容忍重复项的情况下增强应用程序之间的消息收发。

FIFO 队列最重要的特征是 FIFO（先进先出）传递和仅处理一次：

- 发送和接收消息的顺序严格保持一致；一条消息只会被传递一次，并且在使用者处理并删除该消息之前，其他使用者无法进行处理。
- 不会将重复项引入到队列中。

此外，FIFO 队列还支持消息组，此类组允许一个队列中存在多个有序的消息组。一个 FIFO 队列中的消息组数量没有配额。

您可以使用 FIFO 队列的情况示例如下：

1. 订单至关重要的电子商务订单管理系统
2. 与需要按顺序处理事件的第三方系统集成
3. 按输入顺序处理用户输入的内容
4. 通信和联网 - 按相同的顺序发送和接收数据与信息
5. 计算机系统 - 确保用户输入的命令按正确的顺序运行
6. 教育学院 - 防止学员在注册账户之前参加课程
7. 在线售票系统 - 票按先到先得的原则分发

> **Note**
>
> FIFO 队列还提供“仅处理一次”功能，但每秒事务数 (TPS) 有限。您可以将 Amazon SQS 高吞吐量模式与 FIFO 队列配合使用，以提高事务限额。有关使用高吞吐量模式的详细信息，请参阅 Amazon SQS 中 FIFO 队列的高吞吐量。有关吞吐量配额的信息，请参阅the section called “消息配额”。

Amazon SQS FIFO 队列在所有提供 Amazon SQS 的区域推出。

有关使用具有复杂排序功能的 FIFO 队列的更多信息，请参阅使用 Amazon SQS FIFO 队列解决复杂的排序难题。

有关如何使用 Amazon SQS 控制台创建和配置队列的信息，请参阅使用 Amazon SQS 控制台创建标准队列。有关 Java 的示例，请参阅 Amazon SQS Java SDK 示例。

有关使用 FIFO 队列的最佳实践，请参阅 Amazon SQS 最佳实践。

Amazon SQS FIFO 队列关键术语

以下关键术语有助于您更好地了解 FIFO 队列的功能。有关更多信息，请参阅 Amazon Simple Queue Service API 参考。

客户端

Amazon SQS 缓冲异步客户端目前不支持 FIFO 队列。

消息重复数据删除 ID

Amazon SQS FIFO 队列中使用的令牌，用于唯一地标识消息并防止重复。如果在 5 分钟的重复数据删除间隔内发送了多条具有相同重复数据删除 ID 的消息，则这些消息将被视为重复消息，并且系统只传递其中一条。如果您未指定重复数据删除 ID 并且启用了基于内容的重复数据删除，Amazon SQS 会通过对消息正文进行哈希处理来生成重复数据删除 ID。这一机制可以在指定时间范围内消除重复消息，从而确保仅传递一次。

> **Note**
>
> 即使消息已被接收并删除，Amazon SQS 仍会继续跟踪重复数据删除 ID。

消息组 ID

在 FIFO（先进先出）队列中，`MessageGroupId` 是一个将消息组织成不同组的属性。同一消息组内的消息始终按照严格的顺序逐条处理，从而确保同一组内绝不会有两条消息同时被处理。在标准队列中，使用 `MessageGroupId` 启用公平队列。如果需要严格排序，请使用 FIFO 队列。

接收请求尝试 ID

接收请求尝试 ID 是 Amazon SQS 中用于删除 `ReceiveMessage` 重复调用的唯一令牌。

序列号

Amazon SQS 为每条消息分配的大型非连续数字。

服务

如果您的应用程序使用多个 AWS 服务，或者混合使用外部服务，那么了解哪些服务功能不支持 FIFO 队列非常重要。 AWS

尽管允许您将 FIFO 队列设置为目标，但向 Amazon SQS 发送通知的某些 AWS 或外部服务可能与 FIFO 队列不兼容。

AWS 服务的以下功能目前与 FIFO 队列不兼容：

- Amazon S3 事件通知
- Auto Scaling 生命周期挂钩
- AWS IoT 规则操作
- AWS Lambda 死信队列

有关其他服务与 FIFO 队列的兼容性的信息，请参阅服务文档。

Amazon SQS 中的 FIFO 队列传递逻辑

以下概念阐明了 Amazon SQS FIFO 队列如何处理消息的发送和接收，尤其是在处理消息排序和消息组时。 IDs

发送消息

Amazon SQS FIFO 队列使用唯一的重复数据删除 IDs 和消息组来保留消息顺序。 IDs本主题重点介绍了消息组 IDs 对保持组内严格排序的重要性，并重点介绍了确保在多个生产者之间可靠且有序地交付消息的最佳实践。

1. 顺序保存
   - 当多条消息连续发送到具有唯一消息重复数据删除功能的 FIFO 队列时 IDs，Amazon SQS 会存储这些消息并确认其传输。随后，这些消息将按其确切的传输顺序进行接收和处理。
2. 消息组 ID
   - 在 FIFO 队列中，消息基于其消息组 ID 进行排序。如果多个创建者或线程发送具有相同消息组 ID 的消息，Amazon SQS 会确保按照消息的到达顺序进行存储和处理。
   - 最佳实践：为确保多个创建者之间的严格消息顺序，应为来自每个创建者的所有消息分配唯一的消息组 ID。
3. 按组排序
   - FIFO 队列逻辑应用于每个消息组 ID：

- 每个消息组 ID 表示一个不同的、有序的消息组。
- 在每一个消息组 ID 内，所有消息的发送和接收均严格遵循一定的顺序。
- 具有不同消息组的消息 IDs 可能会到达或处理顺序混乱。

- 要求：您必须将消息组 ID 与每条消息关联。如果消息发送时未指定消息组 ID，则操作将失败。
- 单个组场景：如果需要严格遵循一定的顺序处理所有消息，请为每条消息使用相同的消息组 ID。

接收消息

Amazon SQS FIFO 队列处理消息检索，包括批处理、FIFO 订单保证和请求特定消息组的限制。 IDs 本主题介绍 Amazon SQS 如何在保持严格的排序和可见性规则的 IDs 同时检索消息组内和跨消息组的消息。

1. 批量检索
   - 从包含多个消息组的 FIFO 队列接收消息时 IDs，Amazon SQS：
     - 尝试在单次调用中返回尽可能多的具有相同消息组 ID 的消息。
     - 允许其他使用者 IDs 同时处理来自不同消息组的消息。
   - 重要澄清
     - 您可以一次性收到来自同一消息组 ID 的多条消息（使用 `MaxNumberOfMessages` 参数，一次调用最多可接收 10 条消息）。
     - 但是，在后续请求中，您无法接收来自同一消息组 ID 的其他消息，除非：
       - 当前接收到的消息被删除，或
       - 它们再次变为可见（例如，在可见性超时结束之后）。
2. FIFO 顺序保证
   - 批量检索的消息在组中保留其 FIFO 顺序。
   - 如果同一个消息组 ID 的可用消息少于 10 条，则 Amazon SQS 可能会在同一批次 IDs 中包含来自其他消息组的消息，但每个组都保留 FIFO 顺序。
3. 使用者限制
   - 您无法显式请求接收具有特定消息组 ID 的消息。

多次重试

创建者和使用者可以安全地重试 Amazon SQS FIFO 队列中失败的操作，而不会打乱消息顺序或引入重复消息。本主题重点介绍重复数据删除 IDs 和可见性超时如何确保重试期间的消息完整性。

1. 制作人重试
   - 如果 SendMessage 操作失败，创建者可以使用相同的消息重复数据删除 ID 多次重试发送消息。
   - 只要创建者在重复数据删除间隔到期前收到至少一个确认，重试操作将：
     - 不会引入重复消息。
     - 不会打乱消息顺序。
2. 消费者重试
   - 如果 ReceiveMessage 操作失败，使用者可以根据需要使用相同的接收请求尝试 ID 重试多次。
   - 只要使用者在可见性超时结束之前收到至少一次确认，则重试：
     - 不会打乱消息顺序。

关于 FIFO 行为的其他注意事项

了解如何处理可见性超时、启用多个消息组的并行处理以及确保在单组 IDs场景中进行严格的顺序处理。

1. 处理可见性超时
   - 当消息被检索但未删除时，将保持不可见状态，指导可见性超时结束。
   - 在第一条消息被删除或再次可见之前，系统不会返回来自同一消息组 ID 的其他消息。
2. 并发和并行处理
   - FIFO 队列允许并行处理不同消息组 IDs中的消息。
   - 为了最大限度地提高并发性，请设计具有多个消息组的系统， IDs 以实现独立的工作流程。
3. 单个组场景
   - 要严格按顺序处理 FIFO 队列中的所有消息，请为队列中的所有消息使用单个消息组 ID。

示例（帮助更好地理解）

以下实际场景演示了 Amazon SQS 中 FIFO 队列行为。

1. 场景 1：单个组 ID
   - 创建者发送了 5 条具有相同消息组 ID（Group A）的消息。
   - 使用者按照 FIFO 顺序接收这些消息。在使用者删除这些消息或可见性超时结束之前，不会接收来自 Group A 的其他消息。
2. 场景 2：多个群组 IDs

- 创建者向 Group A 发送了 5 条消息，向 Group B 发送了 5 条消息
- Consumer 1 处理来自 Group A 的消息，而 Consumer 2 处理来自 Group B 的消息。这实现了并行处理，并在每个组内保持严格的顺序。

3. 场景 3：批量检索
   - 创建者向 Group A 发送了 7 条消息，向 Group B 发送了 3 条消息
   - 单个使用者最多可检索 10 条消息。如果队列允许，它可能会返回：
     - 来自 Group A 的 7 条消息和来自 Group B 的 3 条消息（如果单组可用的消息不足，则返回的数量更少）。

Amazon SQS 中的仅处理一次

不同于标准队列，FIFO 队列不会引入重复消息。FIFO 队列可帮助您避免向一个队列发送重复消息。如果您在 5 分钟的重复数据删除时间间隔内重试 SendMessage 操作，则 Amazon SQS 不会将任何重复消息引入队列。

要配置重复数据删除，必须执行以下操作之一：

- 启用基于内容的重复数据删除。这将指示 Amazon SQS 使用 SHA-256 哈希通过消息的正文（而不是消息的属性）生成消息重复数据删除 ID。有关更多信息，请参阅 Amazon Simple Queue Service API 参考中 CreateQueue、GetQueueAttributes 和 SetQueueAttributes 操作的相关文档。
- 为消息显式提供消息重复数据删除 ID（或查看序列号）。有关更多信息，请参阅 Amazon Simple Queue Service API 参考中 SendMessage、SendMessageBatch 和 ReceiveMessage 操作的相关文档。

在 Amazon SQS 中，从标准队列移至 FIFO 队列

如果您的现有应用程序使用标准队列，并且您想要利用 FIFO 队列的排序或“仅处理一次”功能，则需要正确地配置队列和应用程序。

重要注意事项

- 创建 FIFO 队列：您无法将现有标准队列转换为 FIFO 队列。您必须为应用程序创建新的 FIFO 队列，或者删除现有标准队列并重新将其创建为 FIFO 队列。
- 延迟参数：FIFO 队列不支持每消息延迟，仅支持每队列延迟。如果您的应用程序在每条消息上设置 DelaySeconds 参数，您必须将应用程序修改为在整个队列上设置 DelaySeconds。

- 消息组 ID：为每条发送的消息提供一个消息组 ID。该 ID 支持并行处理消息，同时保持各组内消息的顺序。为了更好地扩展 FIFO 队列，请为消息组 ID 使用更精细的业务维度。 IDs 您向其分发消息的消息组越多，可供使用的消息数量就越多。
- 高吞吐量模式：使用推荐的 FIFO 队列高吞吐量模式来提高吞吐量。有关消息配额的更多信息，请参阅Amazon SQS 消息配额。

移至 FIFO 队列的核对清单

在将消息发送到 FIFO 队列之前，请确认以下内容：

1. 配置延迟设置
   - 修改应用程序以取消每消息延迟。
   - 在整个队列上设置 `DelaySeconds` 参数。
2. 设置消息组 IDs
   - 通过基于业务维度指定消息组 ID，将消息整理到消息组中。
   - 使用更精细的业务维度来提高可扩展性。
3. 处理消息重复数据删除
   - 如果您的应用程序无法发送具有相同消息正文的消息，请针对每条消息提供唯一的消息重复数据删除 ID。
   - 如果您的应用程序发送具有独特的消息正文的消息，请启用基于内容的重复数据删除。
4. 配置使用者
   - 通常，使用者不需要更改代码。
   - 如果处理消息需要较长时间并且您设置了较高的可见性超时时间，请考虑向每个 `ReceiveMessage` 操作添加接收请求尝试 ID。这样做有助于在网络发生故障时重试接收尝试，并防止由于接收尝试失败而导致队列暂停。

通过执行这些步骤，您可以确保您的应用程序在使用 FIFO 队列时能够正常运行，并充分利用其排序和“仅处理一次”功能。有关更多详细信息，请参阅《Amazon Simple Queue Service API Reference》。

Amazon SQS FIFO 队列和 Lambda 并发行为

通过将 FIFO（先进先出）队列与 Lambda 结合使用，您可以确保每个消息组内消息的有序处理。Lambda 函数不会同时为同一个消息组运行多个实例，从而保证消息处理的顺序。但是，它可以纵

向扩展以并行处理多个消息组，从而确保高效处理队列的工作负载。以下几点描述了 Lambda 函数在处理来自消息组的 Amazon SQS FIFO 队列的消息时的行为： IDs

- 每个消息组对应单个实例：在任何时候，只有一个 Lambda 实例会处理来自特定消息组 ID 的消息。这样可以确保同一组内的消息按照顺序处理，从而保持 FIFO 顺序的完整性。
- 并行处理不同的组：Lambda 可以使用多个实例同时处理来自不同消息组 IDs 的消息。这意味着，当 Lambda 函数的一个实例处理来自一个消息组 ID 的消息时，其他实例可以同时处理来自其他消息组的消息 IDs，利用 Lambda 的并发功能并行处理多个组。

FIFO 队列消息分组

FIFO 队列可以确保严格按照消息的发送顺序对其进行处理。它们使用消息组 ID 对需要按顺序处理的消息进行分组。

同一消息组中的消息按顺序处理，并且系统每次仅处理每个消息组中的一条消息，以便保持该顺序。

Lambda 与 FIFO 队列的并发处理

创建队列后，您可以向该队列发送消息。

当您设置 Lambda 函数来处理来自 Amazon SQS FIFO 队列的消息时，Lambda 会遵守 FIFO 队列提供的顺序保证。以下几点描述了在使用消息组时处理来自 Amazon SQS FIFO 队列的消息时，Lambda 函数在并发性和扩展方面的行为。 IDs

- 消息组内的并发性：一次只有一个 Lambda 实例处理某个特定消息组 ID 的消息。这样可以确保同一组内的消息按照顺序处理。
- 扩展性和多个消息组：虽然 Lambda 可以纵向扩展以并发处理消息，但这种扩展是针对不同的消息组进行的。如果您有多个消息组，Lambda 可以并行处理多个消息组，每个消息组由单独的 Lambda 实例处理。

有关更多信息，请参阅《AWS Lambda Operator Guide》中的 Scaling and concurrency in Lambda。

使用案例示例

假设您的 FIFO 队列收到具有相同消息组 ID 的消息，并且您的 Lambda 函数具有较高的并发限制（最多 1000 条）。

如果系统正在处理来自消息组 ID“A”的一条消息，而来自消息组 ID“A”的另一条消息到达，则在第一条消息完全处理完毕之前，第二条消息不会触发新的 Lambda 实例。

但是，如果来自组 IDs “A” 和 “B” 的消息到达，则这两条消息可以由单独的 Lambda 实例同时处理。

Amazon SQS 中 FIFO 队列的高吞吐量

Amazon SQS 中的高吞吐量 FIFO 队列可以有效地管理高消息吞吐量，同时保持严格的消息顺序，从而确保处理大量消息的应用程序的可靠性和可扩展性。该解决方案非常适合要求高吞吐量和有序消息传递的场景。

在严格的消息排序并不重要、传入消息量相对较低或偶尔发生的情况下，不需要使用 Amazon SQS 高吞吐量 FIFO 队列。例如，如果您有一个处理到达频率低或不需要按顺序处理的消息的小型应用程序，那么使用高吞吐量 FIFO 队列所产生的复杂性和成本不会带来足够的好处。此外，如果您的应用程序不需要高吞吐量 FIFO 队列提供的增强吞吐能力，那么选择标准 Amazon SQS 队列可能更具成本效益，并且更易于管理。

为了提高高吞吐量 FIFO 队列中的请求容量，建议增加消息组的数量。有关高吞吐量消息配额的更多信息，请参阅《Amazon Web Services 一般参考》中的 Amazon SQS 服务限额。

有关每个队列的配额和数据分配策略的信息，请参阅Amazon SQS 消息配额和SQS FIFO 队列高吞吐量的分区和数据分布。

Amazon SQS FIFO 队列的高吞吐量使用案例

以下应用场景重点介绍了高吞吐量 FIFO 队列的不同应用，展示了它们在各行业和场景中的有效性：

1. Real-time 数据处理：处理实时数据流（例如事件处理或遥测数据摄取）的应用程序可以受益于高吞吐量 FIFO 队列，以处理持续涌入的消息，同时保留其顺序以进行准确分析。
2. E-commerce 订单处理：在维护客户交易顺序至关重要的电子商务平台中，高吞吐量 FIFO 队列可确保按顺序处理订单，即使在购物旺季也不会出现延迟。
3. 金融服务：金融机构在处理高频交易或交易数据时，依靠高吞吐量 FIFO 队列以最小的延迟处理市场数据和交易，同时遵守严格的消息顺序监管要求。
4. 媒体流式传输：流媒体平台和媒体分发服务利用高吞吐量 FIFO 队列来管理媒体文件和流媒体内容的分发，确保用户获得流畅的播放体验，同时保持正确的内容分发顺序。

SQS FIFO 队列高吞吐量的分区和数据分布

Amazon SQS 以分区形式存储 FIFO 队列数据。分区是为队列分配的存储空间，该队列可自动复制到一个 AWS 区域内的多个可用区。您无需管理分区。相反，Amazon SQS 会负责分区管理。

对于 FIFO 队列，在以下情况下，Amazon SQS 会修改队列中的分区数量：

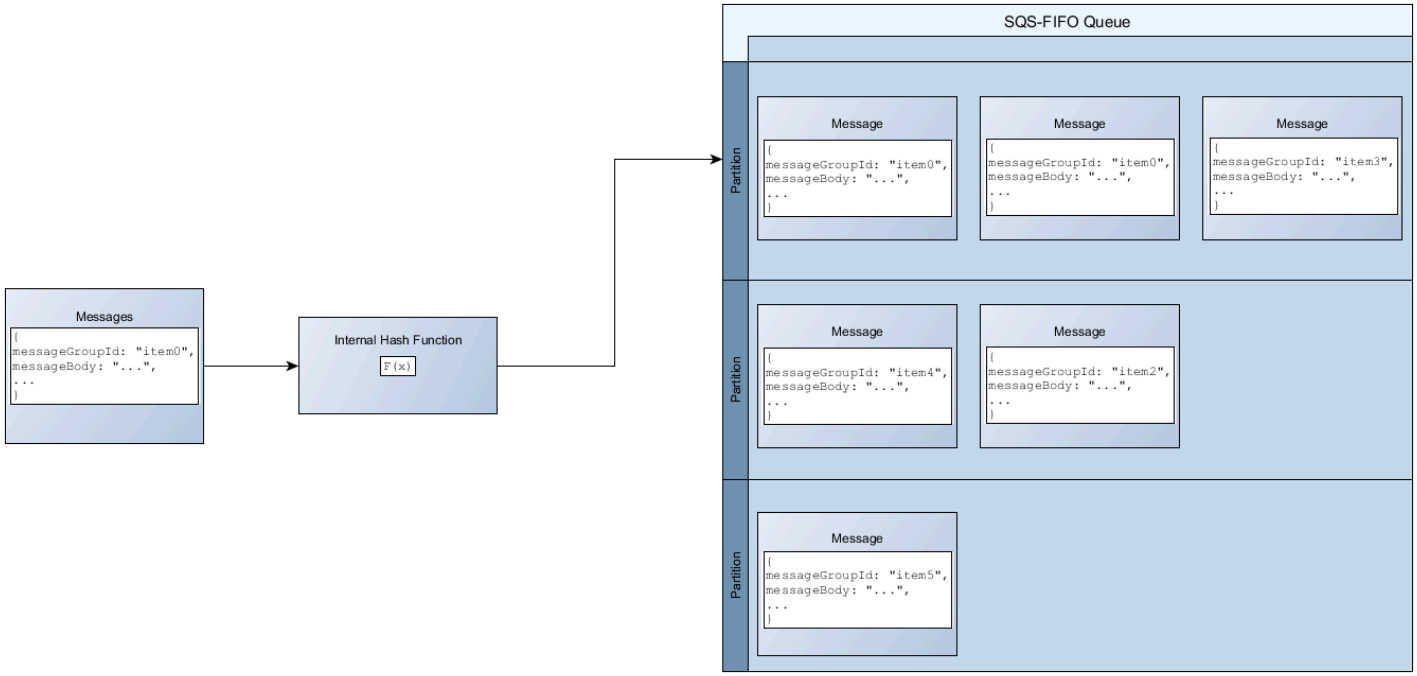
- 如果当前请求速率接近或超过现有分区所能支持的速率，则会分配其他分区，直到队列达到区域配额。有关配额的信息，请参阅Amazon SQS 消息配额。
- 如果当前分区的利用率较低，则分区的数量可能会减少。

分区管理在后台自动进行，对程序是透明的。您的队列和消息始终可用。

按消息组 ID 分发数据

为了将消息添加到 FIFO 队列，Amazon SQS 使用每条消息的消息组 ID 的值作为内部哈希函数的输入。散列函数的输出值决定了哪个分区会存储消息。

下图显示了跨越多个分区的队列。队列的消息组 ID 基于项编号。Amazon SQS 使用其哈希函数决定新项的存储位置；在本例中，它基于字符串 `item0` 的哈希值。请注意，这些项的存储顺序与它们添加到队列的顺序相同。每个项的位置由其消息组 ID 的哈希值决定。



> **Note**
>
> Amazon SQS 经过优化，无论分区数量多少，均可在 FIFO 队列的分区上均匀分配项目。AWS 建议您使用可以有大量不同值的消息组 ID。

优化分区利用率

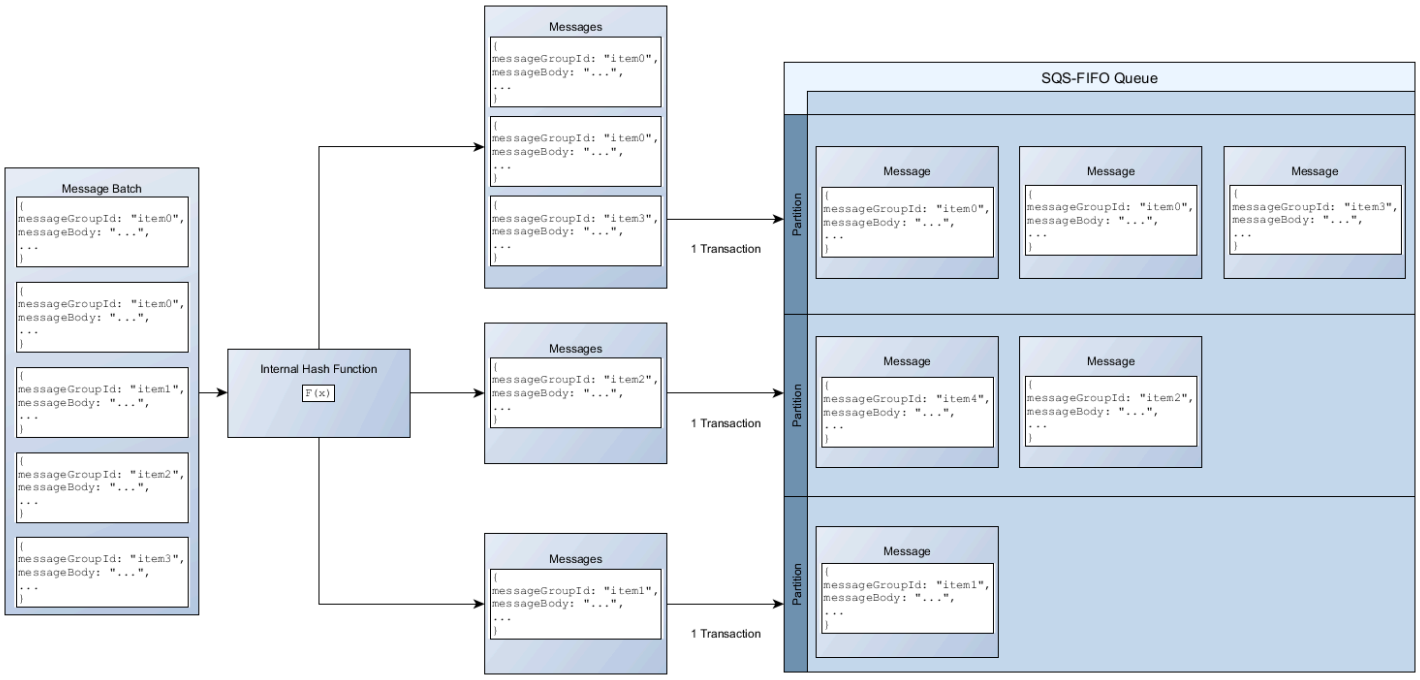
在支持区域，每个分区支持每秒最多 3000 条消息进行批处理，或者支持每秒最多 300 条消息用于发送、接收和删除操作。有关高吞吐量消息配额的更多信息，请参阅《Amazon Web Services 一般参考》中的 Amazon SQS 服务限额。

使用批处理 API 时，每条消息都将根据按消息组 ID 分发数据中所述的过程进行路由。路由到同一分区的消息在单个事务中进行分组和处理。

为了优化 SendMessageBatch API 的分区利用率， AWS 建议尽可能批处理具有相同消息组 ID 的消息。

为了优化DeleteMessageBatch和 ChangeMessageVisibilityBatch API 的分区利用率， AWS 建议使用MaxNumberOfMessages参数设置为 10 的ReceiveMessage请求，并对单个请求返回的收据句柄进行批处理。ReceiveMessage

在以下示例中，发送了一批具有不同消息组 ID 的消息。该批次分为三组，每组都计入分区的配额。



> **Note**
>
> Amazon SQS 仅保证将具有相同消息组 ID 的内部哈希函数的消息分组到批处理请求中。根据内部哈希函数的输出和分区数量，可能会对具有不同消息组 ID 的消息进行分组。由于哈希函数或分区数量可以随时更改，因此，在某一时刻分组的消息以后可能无法分组。

为 Amazon SQS 中的 FIFO 队列启用高吞吐量

您可以为任何新的或现有的 FIFO 队列启用高吞吐量。创建和编辑 FIFO 队列时，该特征包括三个新选项：

- 启用高吞吐量 FIFO - 增加当前 FIFO 队列消息的吞吐量。
- 重复数据删除范围 - 指定是在队列级别还是在消息组级别进行重复数据删除。
- FIFO 吞吐量限制 - 指定 FIFO 队列中消息的吞吐量配额是在队列级别还是在消息组级别设置的。

为 FIFO 队列启用高吞吐量（控制台）

1. 开始创建或编辑 FIFO 队列。
2. 为队列指定选项时，选择启用高吞吐量 FIFO。

   为 FIFO 队列启用高吞吐量会设置相关选项，如下所示：

   - 将重复数据删除范围设置为消息组，这是为 FIFO 队列使用高吞吐量的必需设置。
   - 将 FIFO 吞吐量限制设置为每个消息组 ID，这是为 FIFO 队列使用高吞吐量的必需设置。

   如果更改为 FIFO 队列使用高吞吐量所需的任何设置，则队列正常吞吐量生效，重复数据删除按规定进行。
3. 继续为队列指定所有选项。完成后，选择创建队列或保存。

创建或编辑 FIFO 队列后，您可以向该队列发送消息以及接收和删除消息，所有这些都以更高的 TPS 完成。有关高吞吐量配额，请参阅Amazon SQS 消息配额中的消息吞吐量。

Amazon SQS 中的 FIFO 队列和消息标识符

本节介绍 FIFO 队列的标识符。这些标识符可帮助您查找并操作特定队列和消息。

Amazon SQS 中的 FIFO 队列标识符

有关以下标识符的更多信息，请参阅 Amazon Simple Queue Service API 参考。

队列名称和 URL

在创建新的队列时，您必须为 AWS 账户和区域指定唯一的队列名称。Amazon SQS 会为您创建的每个队列分配一个名为队列 URL 的标识符，其中包含队列名称和其他 Amazon SQS 组件。每当您要对队列执行操作时，都需要提供其队列 URL。

FIFO 队列名称必须以 `.fifo` 后缀结尾。后缀计入 80 个字符的队列名称配额。要确定队列是否为 FIFO，您可以检查队列名称是否以该后缀结尾。

以下是名为 `MyQueue` 的 FIFO 队列的队列 URL，该队列由 AWS 账号为 `123456789012` 的用户所拥有。

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue.fifo
```

您可以通过列出队列并解析账号后的字符串，以编程方式检索队列的 URL。有关更多信息，请参阅 `ListQueues`。

消息 ID

每条消息都会收到一个系统分配的消息 ID，该 ID 由 Amazon SQS 在 `SendMessage` 响应中返回给您。此标识符用于识别消息。消息 ID 的最大长度为 100 个字符。

接收句柄

每当收到来自队列的消息时，您都会收到该消息的接收句柄。此句柄与接收消息的操作相关联，与消息本身无关。要删除消息或更改消息可见性，您必须提供接收句柄（而不是消息 ID）。因此，您必须始终先接收消息，然后才能删除它（您不能将消息放入队列中，然后重新调用它）。接收句柄的最大长度为 1024 个字符。

> **Important**
>
> 如果多次接收某条消息，则每次接收该消息时，您都会获得不同的接收句柄。在请求删除该消息时，您必须提供最近收到的接收句柄（否则，可能无法删除该消息）。

以下是一个跨三行显示的接收句柄示例。

```
MbZj6wDWli+JvwwJaBV+3dcjk2YW2vA3+STFFljTM8tJJg6HRG6PYSasuWXPJB+Cw
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq2OKpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE
```

```
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Amazon SQS FIFO 队列的其他标识符

有关以下标识符的更多信息，请参阅Amazon SQS 中的仅处理一次和 Amazon Simple Queue Service API 参考。

消息重复数据删除 ID

Amazon SQS FIFO 队列中使用的令牌，用于唯一地标识消息并防止重复。如果在 5 分钟的重复数据删除间隔内发送了多条具有相同重复数据删除 ID 的消息，则这些消息将被视为重复消息，并且系统只传递其中一条。如果您未指定重复数据删除 ID 并且启用了基于内容的重复数据删除，Amazon SQS 会通过对消息正文进行哈希处理来生成重复数据删除 ID。这一机制可以在指定时间范围内消除重复消息，从而确保仅传递一次。

消息组 ID

`MessageGroupId` 属性仅用于 Amazon SQS FIFO（先进先出）队列，可将消息组织成不同的组。同一消息组内的消息始终按照严格的顺序逐条处理，从而确保同一组内绝不会有两条消息同时被处理。标准队列不使用 `MessageGroupId`，也不提供顺序保证。如果需要严格排序，请使用 FIFO 队列。

序列号

Amazon SQS 为每条消息分配的大型非连续数字。

Amazon SQS 配额

本主题介绍了 Amazon SQS FIFO 队列和标准队列的配额与限制，详细说明了这些限制如何影响队列创建、配置及消息处理。了解消息保留期限、传输中消息上限、吞吐量阈值等约束条件，以及通过批处理、API 调用优化和长轮询等最大限度提高效率的策略。本主题还涵盖命名约定、标记规则以及请求配额提升以满足高需求工作负载的方法，从而确保实现有效的队列管理和最佳性能。

Amazon SQS FIFO 队列配额

Amazon SQS 配额

下表列出了与 FIFO 队列相关的配额。

配额	描述
延迟队列	队列的默认（最小）延迟为 0 秒。最大值为 15 分钟。
列出的队列	每个 `ListQueues` 请求 1000 个队列。
长轮询等待时间	最长长轮询等待时间为 20 秒。
消息组	一个 FIFO 队列中的消息组数量没有配额。
每个队列的消息数（积压）	Amazon SQS 队列可以存储的消息数量不受限制。
每个队列的消息数（传输中）	FIFO 队列最多支持 12 万条传输中消息（使用者已接收但尚未删除的消息）。如果达到此上限，Amazon SQS 不会返回错误，但处理可能会受到影响。您可以联系 AWS Support 申请提高此上限。
队列名称	FIFO 队列名称必须以 `.fifo` 后缀结尾。后缀计入 80 个字符的队列名称配额。要确定队列是否为 FIFO，您可以检查队列名称是否以该后缀结尾。
队列标签	建议不要向队列中添加超过 50 个标签。标记支持 UTF-8 格式的 Unicode 字符。
	标签 `Key` 是必需的，而标签 `Value` 是可选的。

配额	描述
	标签 Key 和标签 Value 区分大小写。
	标签 Key 和标签 Value 可包含 Unicode 字母数字字符（采用 UTF-8 格式）和空格。允许使用以下特殊字符：_ . : / = + - @
	标签 Key 或 Value 不得包含预留前缀 aws:（您不能删除带此前缀的标签键或值）。
	最大标签 Key 长度为 128 个 Unicode 字符（采用 UTF-8 格式）。标签 Key 不得为空或为 null。
	最大标签 Value 长度为 256 个 Unicode 字符（采用 UTF-8 格式）。标签 Value 可以为空或为 null。
	标记操作限制为每 AWS 账户 30 TPS。如果您的应用程序需要更高的吞吐量，请提交请求。

Amazon SQS 标准队列配额

下表列出了与标准队列相关的配额。

配额	描述
延迟队列	队列的默认（最小）延迟为 0 秒。最大值为 15 分钟。
列出的队列	每个 ListQueues 请求 1000 个队列。
长轮询等待时间	最长长轮询等待时间为 20 秒。
每个队列的消息数（积压）	Amazon SQS 队列可以存储的消息数量不受限制。
每个队列的消息数（传输中）	对于大多数标准队列（取决于队列流量和消息积压），最多可以有大约 120000 条传输中消息（使用者已从队列中接收，但尚未从队列中删除）。如果您在使用短轮询时达到此配额，则 Amazon SQS 会返回 OverLimit 错误消息。如果您使用长轮询，则 Amazon SQS 不返回任何错误

配额	描述
	消息。为避免达到此配额，您应该在处理消息后将其从队列中删除。您还可以增加用来处理消息的队列的数量。要申请提高配额，请提交支持请求。
队列名称	队列名称可以包含最多 80 个字符。接受以下字符：字母数字字符、连字符 (-) 和下划线 (_)。 **Note** 队列名称区分大小写（例如 `Test-queue` 和 `test-queue` 是不同的队列）。
队列标签	建议不要向队列中添加超过 50 个标签。标记支持 UTF-8 格式的 Unicode 字符。
	标签 `Key` 是必需的，而标签 `Value` 是可选的。
	标签 `Key` 和标签 `Value` 区分大小写。
	标签 `Key` 和标签 `Value` 可包含 Unicode 字母数字字符（采用 UTF-8 格式）和空格。允许使用以下特殊字符：`_ . : / = + - @`
	标签 `Key` 或 `Value` 不得包含预留前缀 `aws:`（您不能删除带此前缀的标签键或值）。
	最大标签 `Key` 长度为 128 个 Unicode 字符（采用 UTF-8 格式）。标签 `Key` 不得为空或为 null。
	最大标签 `Value` 长度为 256 个 Unicode 字符（采用 UTF-8 格式）。标签 `Value` 可以为空或为 null。
	标记操作限制为每 AWS 账户 30 TPS。如果您的应用程序需要更高的吞吐量，请提交请求。

Amazon SQS 消息配额

下表列出了与消息相关的配额。

配额	说明
批处理消息 ID	批处理消息 ID 最多可包含 80 个字符。接受以下字符：字母数字字符、连字符 (-) 和下划线 (_)。
消息属性	一条消息可以包含最多 10 个元数据属性。
消息批	一个消息批请求中最多可包含 10 条消息。有关更多信息，请参阅 Amazon SQS 批处理操作 部分中的 配置亚马逊 SQSBuffered AsyncClient。
消息内容	消息可以仅包含 XML、JSON 和非格式化的文本。允许以下 Unicode 字符：#x9 | #xA | #xD | #x20 至 #xD7FF | #xE000 至 #xFFFD | #x10000 至 #x10FFFF 此列表中未包含的任何字符将被拒绝。有关更多信息，请参阅字符的 W3C 规范。
消息组 ID	MessageGroupId 是 FIFO 队列所必需的。如果您在向 FIFO 队列发送消息时未提供 MessageGroupId ，操作将失败。在标准队列中，使用 MessageGroupId 启用公平队列。我们建议您在使用公平队列时在所有消息中添加 MessageGroupId 。 MessageGroupId 的长度为 128 个字符。有效值：字母数字字符和标点符号 (!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~) 。
消息保留	默认情况下，消息将保留 4 天。最小值为 60 秒（1 分钟）。最大值为 1209600 秒（14 天）。
消息吞吐量	标准队列 标准队列支持每个操作（SendMessage 、ReceiveMessage 或 DeleteMessage ）每秒几乎无限次的 API

配额	说明
	调用。这种高吞吐量使其非常适合需要快速处理大量消息的应用场景，例如实时数据流或大型应用程序。虽然标准队列会根据需求自动扩展，但您必须监控使用模式以确保实现最佳性能，特别是在工作负载较高的区域。

配额	说明
	FIFO 队列 • 对于每个 API 操作（SendMessage 、ReceiveMessage 和 DeleteMessage ），FIFO 队列中每个分区的事务处理速率限制为每秒 300 次。这一限制特别适用于非高吞吐量模式。通过切换到高吞吐量模式，您可以超过该默认限制。要启用高吞吐量模式，请参阅为 Amazon SQS 中的 FIFO 队列启用高吞吐量。 • 如果您使用批处理，非高吞吐量 FIFO 队列支持每个 API 操作（SendMessage 、ReceiveMessage 和 DeleteMessage ）每秒最多 3000 条消息。每秒 3000 个事务代表 300 次 API 调用，每次调用带有包含 10 条消息的一个批处理。 FIFO 队列的高吞吐量 Amazon SQS FIFO 限制基于 API 请求的数量，而不是消息数量限制。在高吞吐量模式下，API 请求的限制如下： 交易吞吐量限制（Non-batching API 调用） 这些限制定义了每个 API 操作（例如SendMessageReceiveMessage、或 DeleteMessage）可以独立执行的频率，从而确保在每秒允许的事务数 (TPS) 范围内实现高效的系统性能。 以下限制基于非批量 API 调用： • 美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）和欧洲地区（爱尔兰）：每秒最多 7 万个事务。 • 美国东部（俄亥俄州）和欧洲地区（法兰克福）：每秒最多 1.9 万个事务。 • 亚太地区（孟买）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）和欧洲（西班牙）：高达 9,000 TPS。

配额	说明
	• 欧洲地区（伦敦）和南美洲（圣保罗）：每秒最多 4500 个事务。 • 所有其他 AWS 区域：默认吞吐量为 2,400 TPS。 通过批处理最大限度地提高吞吐量 在一次 API 调用中处理多条消息，从而显著提高效率。您可以通过批处理在一次 API 请求中发送、接收或删除最多 10 条消息，而无需单独处理每条消息。这减少了 API 调用的总数，让您能够在不超出区域的每秒事务数（TPS）限制的情况下每秒处理更多消息，从而最大限度地提高吞吐量和系统性能。有关更多信息，请参阅 利用水平扩缩和操作批处理，借助 Amazon SQS 来提高吞吐量。 以下限制基于批量 API 调用： • 美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）和欧洲地区（爱尔兰）：每秒最多 70 万条消息，是非批量限制（每秒 7 万个事务）的 10 倍。 • 美国东部（俄亥俄州）和欧洲地区（法兰克福）：每秒最多 19 万条消息。 • 亚太地区（孟买）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）和欧洲（西班牙）：每秒最多 90,000 条消息。 • 欧洲地区（伦敦）和南美洲（圣保罗）：每秒最多 4.5 万条消息。 • 所有其他 AWS 区域：每秒最多 24,000 条消息。 利用批处理之外的其他方式优化吞吐量 虽然批处理可以大大提高吞吐量，但您还需要考虑使用其他策略来优化 FIFO 性能： • 将消息分配到多个消息组 ID 中：由于同一组中的消息是按顺序处理的，因此将工作负载分配到多个消息组可以

配额	说明
	实现更有效的并行处理和更高的整体吞吐量。有关更多信息，请参阅 SQS FIFO 队列高吞吐量的分区和数据分布。 • 高效使用 API 调用：最大限度地减少不必要的 API 调用，例如频繁更改可见性或重复删除消息，以便优化可用 TPS 的使用并提高效率。 • 使用长轮询接收：通过在接收请求中设置 `WaitTimeSeconds` 来使用长轮询，减少没有可用消息时的空响应，从而减少不必要的 API 调用并更好地利用 TPS 配额。 • 请求增加吞吐量：如果您的应用程序需要的吞吐量高于默认限制，请使用服务配额控制台请求增加吞吐量。对于高需求的工作负载或默认限制较低的区域，有必要请求增加吞吐量。要启用高吞吐量模式，请参阅为 Amazon SQS 中的 FIFO 队列启用高吞吐量。
消息定时器	消息的默认（最小）延迟为 0 秒。最大值为 15 分钟。
消息大小	最小消息大小为 1 字节（1 个字符）。最大为 1048576 字节（1 MiB）。 要发送大于 1 MiB 的消息，您可以使用适用于 Java 的 Amazon SQS 扩展型客户端库和适用于 Python 的 Amazon SQS 扩展型客户端库。此库允许您发送包含对 Amazon S3 中消息负载的引用的 Amazon SQS 消息。最大负载大小为 2 GB。 **Note** 该扩展型客户端库仅适用于同步客户端。
消息可见性超时	消息的默认可见性超时为 30 秒。最短时间为 0 秒。最长时间为 12 小时。

配额	说明
策略信息	最大配额为 8192 个字节、20 个语句、50 个主体或 10 个条件。有关更多信息，请参阅 Amazon SQS 策略配额。

Amazon SQS 策略配额

下表列出了与策略相关的配额。

名称	最大值
字节	8192
条件	10
主体	50
语句	20
每个语句的操作	7

Amazon SQS 特征和功能

本主题介绍了 Amazon SQS 中用于管理消息队列、优化性能、确保可靠的消息传递以及高效处理消息的常用功能。

在 Amazon SQS 中使用死信队列

Amazon SQS 支持死信队列（DLQ）。死信队列是源队列无法成功处理的消息的目标队列。死信队列对于调试应用程序非常有用，因为您可以隔离未使用的消息，以确定其处理失败的原因。死信队列必须 AWS 账户 与源队列位于同一区域中。消息进入死信队列后，您可以：

- 查看日志，以了解可能导致消息移动到死信队列的异常。
- 分析移动到死信队列的消息内容，以诊断应用程序问题。
- 确定是否为使用者提供了充足的时间来处理消息。
- 使用死信队列重新驱动将消息移出死信队列。

您必须先创建一个新队列，然后才能将其配置为死信队列。有关使用 Amazon SQS 控制台配置死信队列的信息，请参阅使用 Amazon SQS 控制台配置死信队列。要获得死信队列方面的帮助，例如如何为移动到死信队列的任何消息配置警报，请参阅使用 Amazon 为死信队列创建警报 CloudWatch。

> **Note**
>
> 如果不想中断消息或操作的准确顺序，请不要对 FIFO 队列使用死信队列。例如，请不要对视频编辑套件的编辑决策列表 (EDL) 中的指令使用死信队列，此情况下，更改编辑的顺序将更改后续编辑的上下文。

使用死信队列策略

使用重新驱动策略指定 `maxReceiveCount`。`maxReceiveCount` 是使用者可以从源队列接收消息的次数，超过该次数后，消息会被移动到死信队列。例如，如果将 `maxReceiveCount` 设置为较低的值（例如 1），那么只要接收消息失败一次，该消息就会被直接移动到死信队列。要确保您的系统能够抵御错误，请将 `maxReceiveCount` 设置得足够高，以允许足够的重试。

对于采用大`maxReceiveCount`于 3 的重新驱动策略的标准队列，如果一条消息被接收 3 次或更多次但未被删除，则 SQS 会将其移到队列的后面。然后，该`ApproximateAgeOfOldestMessage`指标会反映下一封未超过此阈值的消息的存在时间。

重新驱动允许策略指定哪些源队列可以访问死信队列。您可以选择允许所有源队列使用死信队列、允许特定的源队列使用死信队列或拒绝所有源队列使用死信队列。默认设置是允许所有源队列使用死信队列。如果您选择允许特定队列使用 byQueue 选项，则可以使用源队列 Amazon 资源名称（ARN）指定最多 10 个源队列。如果您指定 denyAll，则队列不能用作死信队列。

了解死信队列的消息保留期

对于标准队列，消息的到期时间始终基于其原始入队时间戳。将消息移至死信队列时，入队时间戳保持不变。ApproximateAgeOfOldestMessage 指标指示消息何时移入死信队列，而不是消息最初发送的时间。例如，假设一条消息在原始队列中停留了 1 天，然后才移至死信队列。如果死信队列的保留期为 4 天，则消息将在 3 天后从死信队列中删除，且 ApproximateAgeOfOldestMessage 为 3 天。因此，最佳实践是始终将死信队列的保留期设置为比原始队列的保留期长。

对于 FIFO 队列，当消息移到死信队列时，入队时间戳会重置。ApproximateAgeOfOldestMessage 指标指示消息何时移动到死信队列。在上面的同一个示例中，消息在 4 天后从死信队列中删除，且 ApproximateAgeOfOldestMessage 为 4 天。

使用 Amazon SQS 控制台配置死信队列

死信队列（DLQ）是一种队列，可接收来自其他队列（称为源队列）的未成功处理的消息。Amazon SQS 不会自动创建死信队列。必须先创建队列，然后才能将其用作死信队列。配置 DLQ 时，队列类型必须与源队列类型一致：FIFO 队列只能使用 FIFO DLQ，标准队列只能使用标准 DLQ。您可以在创建或编辑队列时配置死信队列。有关更多详细信息，请参阅在 Amazon SQS 中使用死信队列。

为现有队列配置死信队列（控制台）

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 选择源队列（该队列会将未成功处理的消息发送到死信队列），然后选择编辑。
4. 滚动到死信队列部分，然后切换为启用。
5. 在死信队列设置下，选择您要用作死信队列的现有队列的 Amazon 资源名称（ARN）。
6. 设置最大接收数量值，该值定义了消息在被发送到死信队列前可被接收的次数（有效范围：1 至 1000）。
7. 选择保存。

了解如何在 Amazon SQS 中配置死信队列重新驱动

使用死信队列重新驱动，将死信队列中未使用的消息移到其他目标进行处理。默认情况下，死信队列重新驱动会将消息从死信队列移动到源队列。但是，您还可以将任何其他队列配置为重新驱动目的地，前提是两个队列属于同一类型。例如，如果死信队列是 FIFO 队列，则重新驱动目的地队列也必须是 FIFO 队列。此外，您可以配置重新驱动速度以设置 Amazon SQS 移动消息的速率。

> **Note**
>
> 当消息从 FIFO 队列移动到 FIFO DLQ 时，原始消息的重复数据删除 ID 将替换为原始消息的 ID。这是为了确保 DLQ 重复数据删除不会阻止存储恰好共享重复数据删除 ID 的两条独立消息。

死信队列按接收顺序重新驱动消息，从最旧的消息开始。但是，目标队列会根据接收消息的顺序摄取重新驱动的消息以及来自其他创建者的新消息。例如，如果创建者在向源 FIFO 队列发送消息的同时从死信队列接收重新驱动的消息，则重新驱动的消息将与创建者的新消息交织在一起。

> **Note**
>
> 重新驱动任务会重置保留期。所有重新驱动的消息都被视为新消息，并且系统会为其分配新的 `messageID` 和 `enqueueTime`。

使用 Amazon SQS API 为现有标准队列配置死信队列重新驱动

您可以使用 `StartMessageMoveTask`、`ListMessageMoveTasks` 和 `CancelMessageMoveTask` API 操作配置死信队列重新驱动：

API 操作	说明
`StartMessageMoveTask`	启动异步任务，将消息从指定的源队列移动到指定的目标队列。
`ListMessageMoveTasks`	获取特定源队列下的最新消息移动任务（最多 10 个）。
`CancelMessageMoveTask`	取消指定的消息移动任务。只有在当前状态为“正在运行”时，才能取消消息移动。

使用 Amazon SQS 控制台为现有标准队列配置死信队列重新驱动

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 选择已配置为死信队列的队列名称。
4. 选择开始 DLQ 重新驱动。
5. 在重新驱动配置下，对于消息目标，执行以下任一操作：
   - 要将消息重新驱动到其源队列，请选择重新驱动到源队列。
   - 要将消息重新驱动到其他队列，请选择驱动到自定义目标。然后，输入现有目标队列的 Amazon 资源名称 (ARN)。
6. 在速度控制设置下，选择以下一个选项：
   - 系统优化 - 以每秒最大消息数重新驱动死信队列消息。
   - 自定义最大速度 - 使用自定义每秒最大消息速率重新驱动死信队列消息。允许的最大速率为每秒 500 条消息。
     - 建议从“自定义最大速度”的较小值开始，并验证源队列不会被消息淹没。在这里，逐渐增加“自定义最大速度”值，继续监控源队列的状态。
7. 配置完死信队列重新驱动后，选择重新驱动消息。

> **Important**
>
> Amazon SQS 不支持在将消息从死信队列中重新驱动时筛选和修改消息。
> 死信队列重新驱动任务最多可以运行 36 小时。Amazon SQS 支持每个账户最多 100 个活跃的重新驱动任务。

8. 如果要取消消息重新驱动任务，请在队列的详细信息页面上，选择取消 DLQ 重新驱动。取消正在进行的消息重新驱动时，任何已经成功移至其移动目标队列的消息都将保留在目标队列中。

为死信队列重新驱动配置队列权限

您可以通过向策略添加权限来授予用户访问特定死信队列操作的权限。死信队列重新驱动所需的最低权限如下：

最小权限	必需的 API 方法
启动消息重新驱动	• 添加死信队列的 `sqs:StartMessageMoveTask` 、`sqs:ReceiveMessage` 、`sqs:DeleteMessage` 和 `sqs:GetQueueAttributes` 。如果死信队列或原始源队列经过加密（也称为 SSE 队列），则还需要使用已用于加密消息的任何 KMS 密钥的 `kms:Decrypt` 。 • 添加目标队列的 `sqs:SendMessage` 。如果目标队列已加密，则还需要 `kms:GenerateDataKey` 和 `kms:Decrypt` 。
取消进行中的消息重新驱动	• 添加死信队列的 `sqs:CancelMessageMoveTask` 、`sqs:ReceiveMessage` 、`sqs:DeleteMessage` 和 `sqs:GetQueueAttributes` 。如果死信队列已加密（也称为 SSE 队列），则还需要 `kms:Decrypt` 。
显示消息移动状态	• 添加死信队列的 `sqs:ListMessageMoveTasks` 和 `sqs:GetQueueAttributes` 。

为加密队列对（带有死信队列的源队列）配置权限

使用以下步骤配置死信队列（DLQ）重新驱动的最低权限：

1. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择策略。
3. 创建新策略并添加以下权限。将该策略附加到将执行重新驱动操作的 IAM 用户或角色。
   - DLQ（源队列）的权限：
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:GetQueueAttributes`

    - sqs:ListDeadLetterSourceQueues
    - 指定 DLQ（源队列）的资源 ARN（例如，“arn: aws: sqs:::”）。*<DLQ_region> <DLQ_accountId> <DLQ_name>*
- 目标队列的权限：
    - sqs:SendMessage
    - 指定目标队列Resource ARN的（例如，“arn: aws: sqs:”）。*<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*
- KMS 密钥的权限：
    - kms:Decrypt（需要，用于解密 DLQ 中的消息。）
    - kms:GenerateDataKey（需要，用于加密目标队列中的消息。）
        - Resource ARNs:
            - 用于加密 DLQ（源队列）中消息的 KMS 密钥的 ARN（例如，“arn: aws: kms::: key/”）。*<region> <accountId> <SourceQueueKeyId>*
            - 用于加密目标队列中消息的 KMS 密钥的 ARN（例如，“arn: aws: kms:: kms:: key/”）。*<region> <accountId> <DestinationQueueKeyId>*

访问策略应类似下文：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:StartMessageMoveTask",
                "sqs:CancelMessageMoveTask",
                "sqs:ListMessageMoveTasks",
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:ListDeadLetterSourceQueues"
            ],
            "Resource": "arn:aws:sqs:us-west-1:123456789012:<DLQ_name>",
            "Condition": {
```

```
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "source"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource": "arn:aws:sqs:us-
west-1:123456789012:<DestQueue_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "destination"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": [
                "arn:aws:kms:us-west-1:123456789012:key/<SourceQueueKeyId>",
                "arn:aws:kms:us-west-1:123456789012:key/<DestQueueKeyId>"
            ]
        }
    ]
}
```

使用非加密队列对（带有死信队列的源队列）配置权限

遵循以下步骤配置处理未加密的标准死信队列（DLQ）所需的最低权限。所需的最低权限包括：从死信队列中接收、删除和获取属性，以及向源队列发送属性。

1. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择策略。
3. 创建新策略并添加以下权限。将该策略附加到将执行重新驱动操作的 IAM 用户或角色。
   - DLQ（源队列）的权限：
     - `sqs:StartMessageMoveTask`

  - sqs:CancelMessageMoveTask
  - sqs:ListMessageMoveTasks
  - sqs:ReceiveMessage
  - sqs:DeleteMessage
  - sqs:ListDeadLetterSourceQueues
  - 指定 DLQ（源队列）的资源 ARN（例如，“arn: aws: sqs:::”）。*<DLQ_region> <DLQ_accountId> <DLQ_name>*
- 目标队列的权限：
  - sqs:SendMessage
  - 指定目标队列Resource ARN的（例如，“arn: aws: sqs:”）。*<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*

访问策略应类似下文：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:StartMessageMoveTask",
                "sqs:CancelMessageMoveTask",
                "sqs:ListMessageMoveTasks",
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:ListDeadLetterSourceQueues"
            ],
            "Resource": "arn:aws:sqs:us-west-1:111122223333:<DLQ_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "source"
                }
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource": "arn:aws:sqs:us-
west-1:111122223333:<DestQueue_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "destination"
                }
            }
        }
    ]
}
```

使用死信队列重新驱动和 VPC 端点访问控制

当 VPCs 使用`aws:sourceVpc`条件将队列访问权限限制为特定队列时，需要将 AWS 服务设为例外才能启用死信队列 (DLQ) 重新驱动功能。这是因为在移动消息时，Amazon SQS 服务在您的 VPC 之外运行。

要允许 DLQ 重新驱动操作，请将 `aws:CalledViaLast` 条件添加到您的队列策略中。这将允许 Amazon SQS 代表您进行 API 调用，同时保持 VPC 限制以便直接访问。

要同时允许 VPC 受限访问和 DLQ 重新驱动，请执行以下操作：

1. 在队列策略中使用 `aws:CalledViaLast` 条件。
2. 将策略应用于源队列和 DLQ
3. 保持 VPC 限制，以便从其他来源直接访问

以下是实现了这些要求的策略示例：

JSON

```
{
  "Version":"2012-10-17",
  "Id": "SQSRedriveWithVpcRestriction",
  "Statement": [
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DestQueue",
      "Effect": "Deny",
```

```
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:DestQueue",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "StringNotEqualsIfExists": {
          "aws:CalledViaLast": "sqs.amazonaws.com"
        }
      }
    },
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DLQ",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:Dlq",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "StringNotEqualsIfExists": {
          "aws:CalledViaLast": "sqs.amazonaws.com"
        }
      }
    }
  ]
}
```

- 请将占位符值替换为实际的值
- 此策略使用带条件的“拒绝”语句，这比使用“允许”语句更安全
- StringNotEqualsIfExists 运算符负责处理请求上下文中可能不存在条件键的情况。

或者，您也可以使用 aws:ViaAWSService 条件键允许基于服务的访问，同时保持 VPC 限制。此条件键表示请求是否来自 AWS 服务。以下是使用 aws:ViaAWSService 方法代替 aws:CalledViaLast 的策略示例：

JSON

```
{
  "Version":"2012-10-17",
  "Id": "SQSRedriveWithVpcRestriction",
  "Statement": [
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DestQueue",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:DestQueue",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    },
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DLQ",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:Dlq",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

带aws:ViaAWSService条件的 BoolIfExists 操作员可确保允许来自服务的请求，同时保持 VPC 对直接访问的限制。这可能更易于理解和维护，因为它直接检查请求是否由 AWS 服务发出，而不是检查上次呼叫的是哪个服务。

有关 IAM 和资源策略中使用的条件键的更多信息，请参阅“IAM JSON 策略元素：条件”。

CloudTrail Amazon SQS 死信队列重新驱动的更新和权限要求

2023 年 6 月 8 日，亚马逊 SQS 推出了适用于 SDK AWS 和 (CLI) 的死信队列 (DLQ) 重启。 AWS Command Line Interface 此功能是对已支持的主机 DLQ 重新驱动功能的补充。 AWS 如果您之前曾使用 AWS 控制台重新驱动死信队列消息，则可能会受到以下更改的影响：

CloudTrail 事件重命名

2023 年 10 月 15 日，亚马逊 SQS 控制台上死信队列重启 CloudTrail 的事件名称将发生变化。如果您已为这些 CloudTrail 事件设置了警报，则必须立即对其进行更新。以下是 DLQ redrive 的新 CloudTrail 事件名称：

上一个事件名称	新事件名称
CreateMoveTask	StartMessageMoveTask
CancelMoveTask	CancelMessageMoveTask

更新权限

在 SDK 和 CLI 版本中，Amazon SQS 还更新了 DLQ 重新驱动的队列权限，以遵守安全最佳实践。使用以下队列权限类型从您的 DLQs队列中重新驱动消息。

1. 基于操作的权限（更新 DLQ API 操作）
2. 托管 Amazon SQS 策略权限
3. 使用 sqs:* 通配符的权限策略

> **Important**
>
> 要使用适用于 SDK 或 CLI 的 DLQ 重新驱动，您需要具有与上述选项之一相匹配的 DLQ 重新驱动权限策略。

如果您的 DLQ 重新驱动队列权限与上述选项之一不匹配，则您必须在 2023 年 8 月 31 日之前更新您的权限。从现在起至 2023 年 8 月 31 日，您的账户只能在您之前使用过 DLQ 重新驱动的区域使用您在 AWS 控制台上配置的权限重新驱动消息。例如，假设您在 us-east-1 和 eu-west-1 中都有“账户 A”。在 2023 年 6 月 8 日之前，“账户 A” 用于在 us-east-1 中在 AWS 主机上重放消息，但在 eu-west-1 中却没有。在 2023 年 6 月 8 日至 2023 年 8 月 31 日之间，如果 “账户 A” 的策略权限与上述任一选项不匹配，则该权限只能用于在 us-east-1 中在 AWS 主机上重新发送消息，而不能在 eu-west-1 中重新驱动消息。

> **Important**
>
> 如果在 2023 年 8 月 31 日之后您的 DLQ 重新驱动权限与其中一个选项不匹配，则您的账户将无法再使用 AWS 控制台重新驱动 DLQ 消息。
> 但是，如果您在 2023 年 8 月期间在 AWS 主机上使用了 DLQ 重制版功能，则根据其中一个选项，您可以将新权限延长至 2023 年 10 月 15 日。
> 有关更多信息，请参阅 the section called “标识受影响的策略”。

以下是每个 DLQ 重新驱动选项的队列权限示例。使用服务器端加密 (SSE) 队列时，需要相应的 AWS KMS 密钥权限。

基于操作

JSON

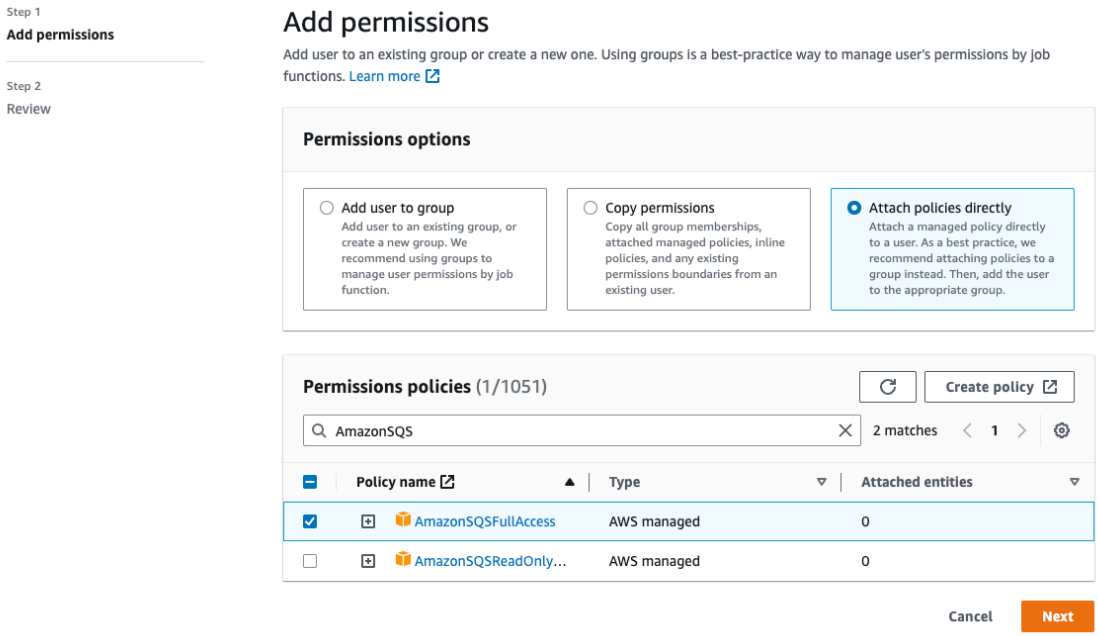
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:StartMessageMoveTask",
                "sqs:ListMessageMoveTasks",
                "sqs:CancelMessageMoveTask"
            ],
            "Resource": "arn:aws:sqs:us-west-1:123456789012:<DLQ_name>"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource": "arn:aws:sqs:us-west-1:123456789012:<DestQueue_name>"
        }
    ]
}
```

托管策略

以下托管策略包含所需的更新权限：

- Amazon A SQSFull cces s — 包括以下死信队列重新驱动任务：启动、取消和列出。
- Amazon SQSRead OnlyAccess — 提供只读访问权限，并包括列表死信队列重新驱动任务。



使用 sqs* 通配符的权限策略

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": "sqs:*",
      "Resource": "*"
    }
  ]
}
```

标识受影响的策略

如果您使用的是客户托管策略 (CMPs)，则可以使用 AWS CloudTrail 和 IAM 来识别受队列权限更新影响的策略。

> **Note**
>
> 如果您使用的是 `AmazonSQSFullAccess` 和 `AmazonSQSReadOnlyAccess`，则无需采取进一步操作。

1. 登录 AWS CloudTrail 控制台。
2. 在事件历史记录页面的查找属性下，使用下拉菜单选择事件名称。然后搜索 `CreateMoveTask`。
3. 选择事件，打开详细信息页面。在事件记录部分，从 `userIdentity` ARN 中检索 `UserName` 或 `RoleName`。
4. 登录到 IAM 控制台。
   - 对于用户，选择“用户”。通过上一步中标识的 `UserName` 选择用户。
   - 对于角色，选择“角色”。通过上一步中标识的 `RoleName` 搜索用户。
5. 在详细信息页面的权限部分，查看 `Action` 中前缀为 `sqs:` 的所有策略，或查看 `Resource` 中定义了 Amazon SQS 队列的策略。

使用 Amazon 为死信队列创建警报 CloudWatch

设置 CloudWatch 警报，使用该指标监控死信队列中的消息。`ApproximateNumberOfMessagesVisible`有关详细说明，请参阅为 Amazon SQS 指标创建 CloudWatch 警报。当警报触发，表明消息已移到死信队列时，您可以轮询该队列以查看和检索消息。

Amazon SQS 的消息元数据

使用消息属性将自定义元数据添加到应用程序的 Amazon SQS 消息。使用消息系统属性存储元数据以便与其他元数据集成 AWS 服务，例如 AWS X-Ray。

Amazon SQS 消息属性

Amazon SQS 支持您使用消息属性，在消息中添加结构化元数据（如时间戳、地理空间数据、签名和标识符）。每条消息最多可以包含 10 个属性。消息属性是可选的，并独立于消息正文（不过会随之一起发送）。使用者可以使用消息属性以特定方式处理消息，而无需先处理消息正文。有关使用 Amazon SQS 控制台发送带有属性的消息的信息，请参阅使用 Amazon SQS 发送带有属性的消息。

> **Note**
>
> 不要将消息属性与消息系统属性混淆：虽然您可以使用消息属性将自定义元数据附加到应用程序的 Amazon SQS 消息中，但您可以使用消息系统属性存储其他 AWS 服务的元数据，例如。AWS X-Ray

主题

- 消息属性组件
- 消息属性数据类型
- 计算消息属性的 MD5 消息摘要

消息属性组件

> **Important**
>
> 消息属性的所有组件都包括在 1 MiB 的消息大小限制中。
> `Name`、`Type`、`Value` 和消息正文不得为空或 null。

每个消息属性包含以下组件：

- 名称 – 消息属性名称可以包含以下字符：A-Z、a-z、0-9、下划线 (_)、连字符 (-) 和句点 (.)。以下限制适用：

- 最长可为 256 个字符
- 不能以 AWS. 或 Amazon.（或任意大小写变化形式）开头
- 区分大小写
- 必须在消息的所有属性名中唯一
- 不能以句点开头或结尾
- 序列中不能有句点

- 类型 – 消息属性数据类型。支持的类型包括 String、Number 和 Binary。您也可以添加有关任意数据类型的自定义信息。数据类型与消息正文具有相同的限制（有关更多信息，请参阅 Amazon Simple Queue Service API 参考中的 SendMessage）。此外，以下限制将适用：
  - 最长可为 256 个字符
  - 区分大小写
- 值 – 消息属性值。对于 String 数据类型，属性值具有与消息正文相同的限制。

消息属性数据类型

消息属性数据类型指示 Amazon SQS 如何处理对应的消息属性值。例如，如果类型为 Number，Amazon SQS 会验证数字值。

Amazon SQS 支持使用可选自定义数据类型标签（格式为 *.custom-data-type*）的逻辑数据类型 String、Number 和 Binary。

- String – String 属性可以存储使用任意有效 XML 字符的 Unicode 文本。
- Number – Number 属性可以存储正数或负数数值。数字最多可精确到 38 位，并且介于 10^-128 和 10^+126 之间。

> **Note**
>
> Amazon SQS 会删除开头和结尾的零。

- Binary – Binary 属性可以存储任何二进制数据，例如压缩数据、加密数据或图像。
- Custom – 要创建自定义数据类型，请将 custom-type 标签附加到任意数据类型。例如：
  - Number.byte、Number.short、Number.int 和 Number.float 可帮助区分各种数字类型。
  - Binary.gif 和 Binary.png 可帮助区分文件类型。

> **Note**
>
> Amazon SQS 不会解释、验证或使用附加数据。
> custom-type 标签与消息正文具有相同的限制。

计算消息属性的 MD5 消息摘要

如果您使用 适用于 Java 的 AWS SDK，则可以跳过此部分。适用于 Java 的 SDK `MessageMD5ChecksumHandler` 类支持 Amazon SQS 消息属性的 MD5 消息摘要。

如果您使用查询 API 或不支持 Amazon SQ AWS S 消息属性的 MD5 消息摘要的软件开发工具包，则必须使用以下指南来执行 MD5 消息摘要计算。

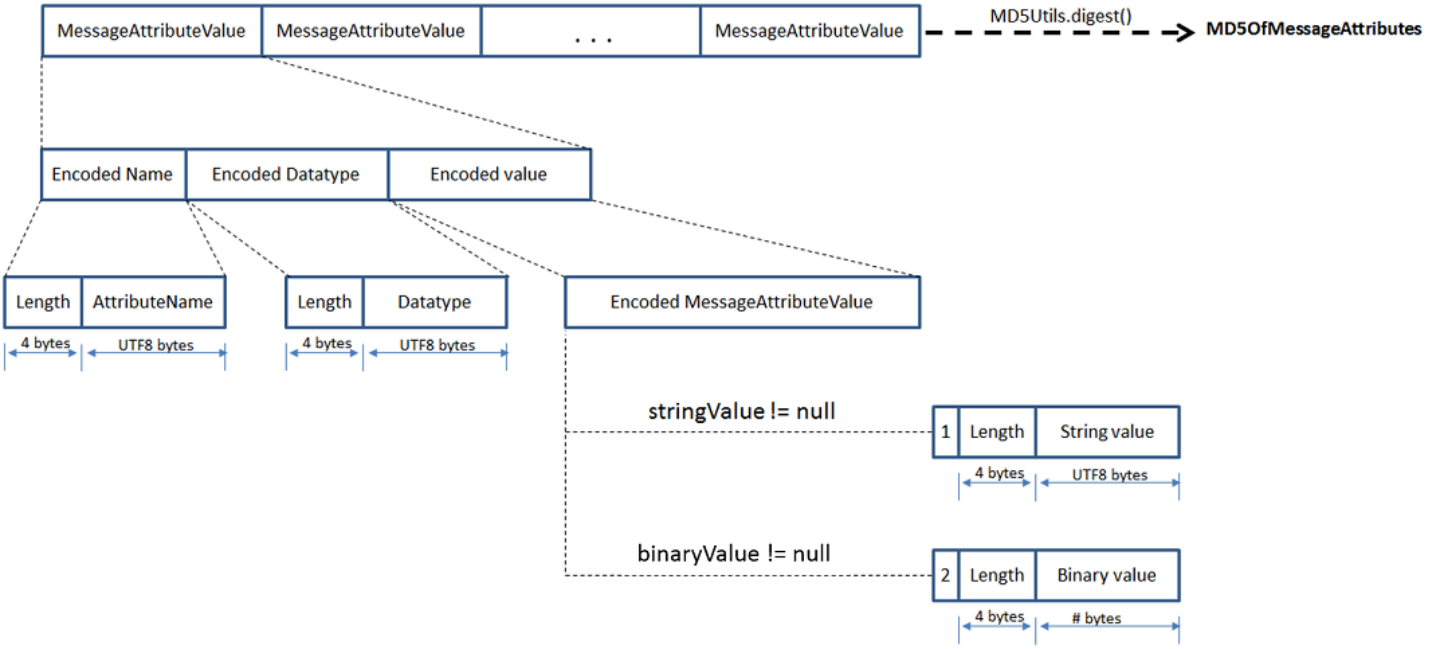
> **Note**
>
> 请始终在 MD5 消息摘要计算中包含自定义数据类型后缀。

概述

以下概述了 MD5 消息摘要计算算法：

1. 按名称的升序对所有消息属性进行排序。
2. 将每个属性的各个部分（`Name`、`Type` 和 `Value`）进行编码并存入缓冲区。
3. 计算整个缓冲区的消息摘要。

下图演示单个消息属性的 MD5 消息摘要的编码：



对单个 Amazon SQS 消息属性编码

1. 对名称进行编码：名称的长度（4 字节）和 UTF-8 字节。
2. 对数据类型进行编码：数据类型的长度（4 UTF-8 字节）和字节。
3. 对值（1 个字节）的传输类型（String 或 Binary）编码。

> **Note**
>
> 逻辑数据类型 String 和 Number 使用 String 传输类型。
> 逻辑数据类型 Binary 使用 Binary 传输类型。

   a. 对于 String 传输类型，编码为 1。
   b. 对于 Binary 传输类型，编码为 2。
4. 对属性值编码。
   a. 对于String传输类型，对属性值进行编码：值的长度（4 字节）和 UTF-8 字节。
   b. 对于 Binary 传输类型，对属性值编码：值的长度（4 字节）和原始字节。

Amazon SQS 消息系统属性

您可以使用消息属性将自定义元数据附加到应用程序的 Amazon SQS 消息，可以使用消息系统属性来存储其他 AWS 服务（例如 AWS X-Ray）的元数据。有关更多信息，请参阅 Amazon Simple Queue Service API 参考中 SendMessage 和 SendMessageBatch API 操作的 MessageSystemAttribute 请求参数、ReceiveMessage API 操作的 AWSTraceHeader 属性及 MessageSystemAttributeValue 数据类型。

消息系统属性的结构与消息属性完全一样，除了以下例外：

- 当前唯一受支持的消息系统属性是 AWSTraceHeader。它的类型必须是String，其值必须是格式正确的 AWS X-Ray 跟踪标头字符串。
- 消息系统属性的大小不会计入消息的总大小。

处理 Amazon SQS 消息所需的资源

Amazon SQS 会估算队列中的延迟消息、可见消息和不可见消息的大致数量，帮助您评估处理消息所需的资源。有关可见性的更多信息，请参阅“Amazon SQS 可见性超时”。

> **Note**
>
> 由于 Amazon SQS 采用分布式架构，某些指标的结果为近似值。在大多数情况下，该计数应接近队列中的实际消息数。

下表列出了用于 GetQueueAttributes 操作的属性名称：

Task	属性名称
获取可从队列检索的大致消息数。	ApproximateNumberOfMessages Visible
获取队列中延迟且无法立即读取的大致消息数。如果队列被配置为延迟队列，或者使用了延迟参数来发送消息，则会出现这种情况。	ApproximateNumberOfMessages Delayed

Task	属性名称
获取“处于飞行状态”的大致消息数。如果消息已发送到客户端，但尚未删除或尚未到达其可见性窗口末尾，则消息被视为处于飞行状态。	`ApproximateNumberOfMessagesNotVisible`

Amazon SQS 列表队列分页

`listQueues` 和 `listDeadLetterQueues` API 方法支持可选的分页控件。默认情况下，这些 API 方法在响应消息中最多返回 1000 个队列。您可以将 `MaxResults` 参数设置为在每个响应中返回更少的结果。

在 `MaxResults` 或 `listQueues` 请求中设置参数 `listDeadLetterQueues`，以指定要在响应中返回的最大结果数。如果您未设置 `MaxResults`，则响应最多包含 1000 个结果，并且响应中的 `NextToken` 值为 null。

如果您设置了 `MaxResults`，则在有更多结果需要显示时，响应将包含 `NextToken` 的值。在对 `NextToken` 的下一个请求中将 `listQueues` 作为参数，以接收下一页结果。如果没有更多结果需要显示，则响应中的 `NextToken` 值为 null。

Amazon SQS 成本分配标签

要组织和标识 Amazon SQS 队列以进行成本分配，您可以添加元数据标签来标识队列的用途、所有者或环境。这在您拥有许多队列时尤其有用。要使用 Amazon SQS 控制台配置标签，请参阅the section called “为队列配置标签”

您可以使用成本分配标签来整理 AWS 账单，以反映您自己的成本结构。为此，请注册以获取包含标签键和值的 AWS 账户 账单。有关更多信息，请参阅《AWS Billing 用户指南》中的设置月度成本分配报告。

每个标签均包含您定义的一个键-值对。例如，如果您按如下标签队列，则可轻松标识这些队列：生产和测试队列

队列	键	值
`MyQueueA`	`QueueType`	`Production`

队列	键	值
MyQueueB	QueueType	Testing

> **Note**
>
> 使用队列标签时，请记住以下准则：
>
> - 建议不要向队列中添加超过 50 个标签。标记支持 UTF-8 格式的 Unicode 字符。
> - 标签没有任何语义含义。Amazon SQS 将标签按字符串解释。
> - 标签区分大小写。
> - 如果新标签的键与现有标签相同，新标签会覆盖现有标签。
> - 标记操作限制为每次 30 TPS。 AWS 账户如果您的应用程序需要更高的吞吐量，请提交请求。
>
> 有关标签限制的完整列表，请参阅Amazon SQS 标准队列配额。

Amazon SQS 短轮询和长轮询

Amazon SQS 提供短轮询和长轮询选项，用于接收来自队列的消息。选择使用哪种轮询方式时，您需要考虑应用程序对响应能力和成本效益的要求：

- 短轮询（默认）：`ReceiveMessage` 请求查询服务器子集（基于加权随机分布）以查找可用消息并立即发送响应，即使未找到任何消息也是如此。
- 长轮询：`ReceiveMessage` 查询所有服务器的消息，在至少有一条消息可用时发送响应，不超过指定的最大值。只有在轮询等待时间到期时，系统才会发送空响应。这一选项可以减少空响应的数量，并有可能降低成本。

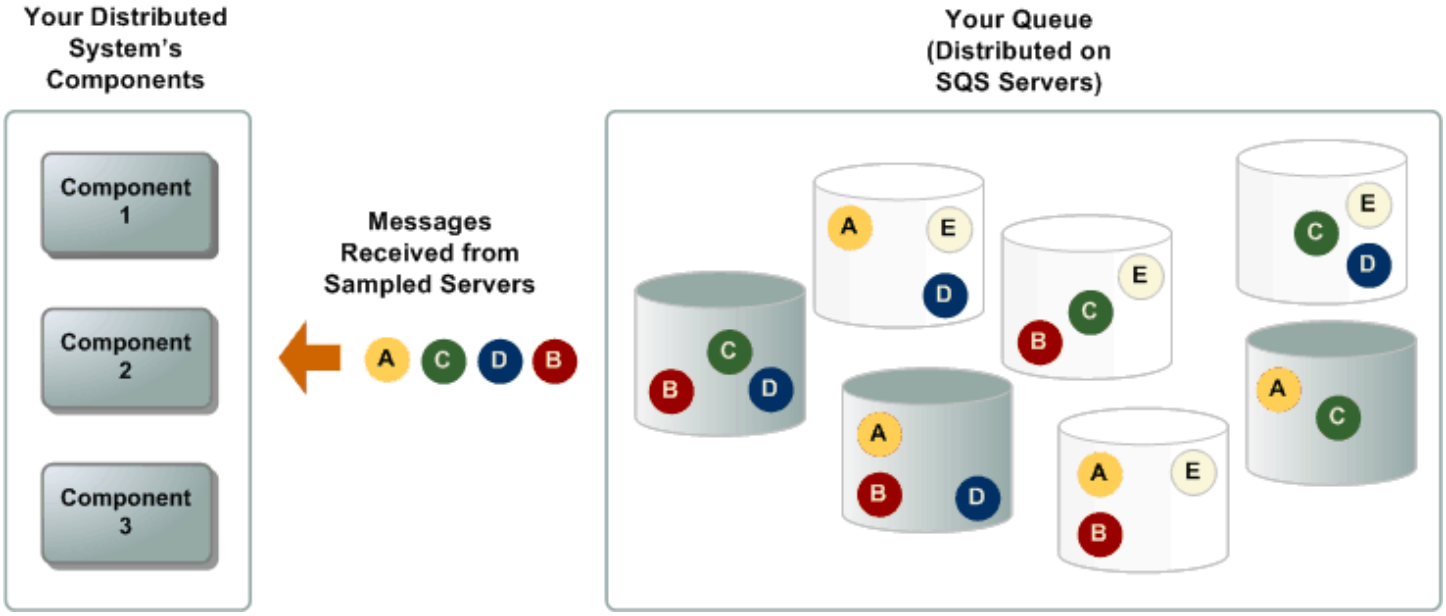
以下部分解释短轮询和长轮询的详细信息。

通过短轮询来使用消息

当通过短轮询从队列（FIFO 队列或标准队列）中使用消息时，Amazon SQS 会对服务器的一个子集进行采样（基于加权随机分布），并且仅返回来自这些服务器的消息。因此，特定 `ReceiveMessage` 请

求可能不会返回您的所有消息。但是，如果您的队列中的消息少于 1000 条，一个后续请求将返回您的消息。如果继续从您的队列中使用消息，则 Amazon SQS 会对其所有服务器进行采样，然后您将收到您的所有消息。

下图显示了系统组件之一发出接收请求后从标准队列返回的消息的短轮询行为。Amazon SQS 对其若干服务器（显示为灰色）进行采样并从这些服务器返回消息 A、C、D 和 B。系统不会为此请求返回消息 E，但将为后续请求返回该消息。



通过长轮询来使用消息

当 `ReceiveMessage` API 操作的等待时间大于 0 时，长轮询生效。最长长轮询等待时间为 20 秒。长轮询通过减少空响应（当`ReceiveMessage`请求没有消息可用时）和虚假空响应（消息可用但未包含在响应中时）的数量来帮助降低使用 Amazon SQS 的成本。有关使用 Amazon SQS 控制台为新队列或现有队列启用长轮询的信息，请参阅使用 Amazon SQS 控制台配置队列参数。有关最佳实践，请参阅在 Amazon SQS 中设置长轮询。

长轮询具有以下优势：

- 在发送响应之前，允许 Amazon SQS 等到队列中的消息可用为止，从而消除空响应。除非连接超时，否则对 `ReceiveMessage` 请求的响应将至少包含一条可用的消息，并且最多包含 `ReceiveMessage` 操作中指定的最大数量的消息。在极少数情况下，即使队列中仍有消息，您也可能会收到空响应，尤其是在您为 `ReceiveMessageWaitTimeSeconds` 参数指定的值较低的情况下。
- 通过查询所有（而不是其子集）Amazon SQS 服务器来减少错误空响应。
- 在消息可用时立即返回消息。

有关如何确认队列为空的信息，请参阅确认 Amazon SQS 队列为空。

长轮询和短轮询之间的区别

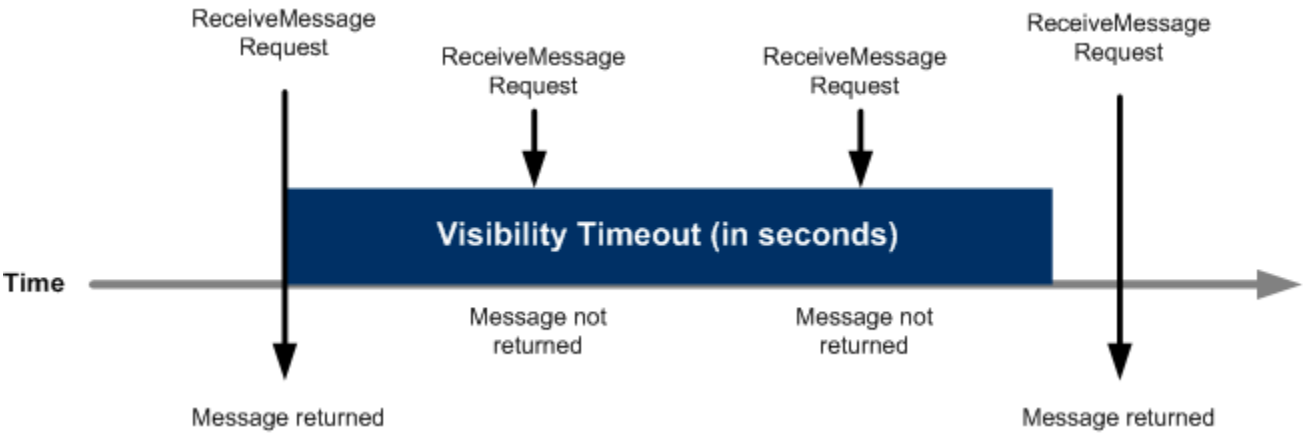
如果以下列两种方式之一将 ReceiveMessage 请求的 WaitTimeSeconds 参数设置为 0，则会发生短轮询：

- ReceiveMessage 调用将 WaitTimeSeconds 设置为 0。
- ReceiveMessage 调用不会设置 WaitTimeSeconds，但队列属性 ReceiveMessageWaitTimeSeconds 将设置为 0。

Amazon SQS 可见性超时

当您从 Amazon SQS 队列接收到一条消息时，该消息仍然保留在队列中，但会暂时对其他使用者不可见。这种不可见性由可见性超时控制，用于确保在您处理该消息期间，其他使用者无法对其进行处理。Amazon SQS 提供了两种在处理后删除消息的选项：

- 手动删除 – 使用 DeleteMessage 操作明确删除消息。
- 自动删除 – 在某些 AWS SDK 中受支持，消息在成功处理后自动删除，从而简化工作流程。



可见性超时使用案例

管理长时间运行的任务 – 使用可见性超时来处理需要延长处理时间的任务。为需要延长处理时间的消息设置适当的可见性超时。这可确保在消息处理期间，其他使用者不会接收到该消息，从而防止重复工作并保持系统效率。

实现重试机制 – 通过编程方式延长在初始超时内未完成的任务的可见性超时。如果任务在初始可见性超时内未完成，您可以通过编程方式延长超时。这使您的系统能够重新尝试处理消息，同时又不会让其他使用者看到该消息，从而提高容错能力和可靠性。与Dead-Letter 队列 (DLQ) 结合使用可管理持续故障。

协调分布式系统 – 使用 SQS 可见性超时来协调分布式系统中的任务。设置可见性超时，使其与不同组件的预期处理时间保持一致。这有助于保持一致性，并防止在复杂的分布式架构中出现竞争条件。

优化资源利用率 – 调整 SQS 可见性超时以优化应用程序中的资源利用率。通过设置适当的超时，您可以确保消息得到高效处理，避免不必要地占用资源。这可以提高整体系统性能和成本效益。

设置和调整可见性超时

可见性超时从消息送达给您时开始计算。在此期间，您需要处理并删除消息。如果您未在超时到期之前删除消息，该消息将在队列中再次变得可见，并且可以被其他使用者检索。队列的默认可见性超时为 30 秒，但您可以将其调整为您的应用程序处理和删除消息所需的时间。您也可以为每条消息设置特定的可见性超时，而无需更改队列的整体设置。根据需要，使用 `ChangeMessageVisibility` 操作通过编程方式延长或缩短超时。

传输中消息和配额

在 Amazon SQS 中，传输中消息是指已被使用者接收但尚未删除的消息。对于标准队列，传输中消息的数量上限约为 12 万条，具体取决于队列流量和消息积压情况。如果您达到此上限，Amazon SQS 会返回 `OverLimit` 错误，提示在删除某些传输中消息之前，无法接收更多的消息。对于 FIFO 队列，上限取决于活动消息组。

- 当使用短轮询时 – 如果使用短轮询时达到此上限，Amazon SQS 会返回 `OverLimit` 错误，提示在删除某些传输中消息之前，无法接收更多的消息。
- 当使用长轮询时 – 如果您使用的是长轮询，当达到传输中消息数量上限时，Amazon SQS 不会返回错误。不过，在传输中消息的数量降到限制以下之前，它不会返回任何新消息。

要有效管理传输中消息，请执行以下操作：

1. 及时删除 – 处理消息后将其删除（手动或自动），以便减少传输中消息的数量。
2. 监控 CloudWatch — 设置高飞行次数警报，防止达到极限。
3. 分配负载 – 如果您处理的消息量很大，请使用额外的队列或使用者来平衡负载并避免出现瓶颈。
4. 申请增加配额 – 如果需要提高上限，请向 AWS Support 提交申请。

了解标准队列和 FIFO 队列中的可见性超时

在标准队列和 FIFO (First-In-First-Out) 队列中，可见性超时有助于防止多个消费者同时处理相同的消息。但是，由于 Amazon SQS 采用“至少传递一次”模式，因此无法绝对保证在可见性超时期间消息不会被多次传递。

- 标准队列 – 在标准队列中，可见性超时有助于防止多个使用者同时处理同一消息。但是，由于 Amazon SQS 采用“至少传递一次”模式，因此无法保证在可见性超时期间消息不会被多次传递。
- FIFO 队列 – 对于 FIFO 队列，具有相同消息组 ID 的消息将按严格的顺序处理。当具有消息组 ID 的消息处于传输中状态时，该组中的后续消息在传输中消息被删除或可见性超时到期后才可用。但是，这并不会无限期“锁定”该组 – 每条消息都按顺序处理，只有当每条消息被删除或再次变得可见时，该组中的下一条消息才可供使用者使用。这种方法可确保在组内进行有序处理，同时避免不必要地阻止该组传递消息。

处理故障

如果您未在可见性超时到期之前处理和删除消息（由于应用程序错误、崩溃或连接问题），则该消息将在队列中再次变得可见。届时，该消息可被同一或不同的使用者检索，进行下一次处理尝试。这可确保即使初始处理失败，消息也不会丢失。但是，如果将可见性超时设置得过高，则会延迟未处理的消息重新出现在队列中的时间，这可能导致重试过程变慢。请务必根据预期处理时间设置适当的可见性超时，以便及时处理消息。

更改和终止可见性超时

您可以使用 `ChangeMessageVisibility` 操作更改或终止可见性超时：

- 更改超时 – 使用 `ChangeMessageVisibility` 动态调整可见性超时。这使您能够根据处理需求延长或缩短超时持续时间。
- 终止超时 – 如果您决定不处理接收到的消息，则可以通过 `ChangeMessageVisibility` 操作将 `VisibilityTimeout` 设置为 0 秒，从而终止可见性超时。这样做可以立即使消息可供其他使用者处理。

最佳实践

使用以下最佳实践来管理 Amazon SQS 中的可见性超时，包括设置、调整和延长超时，以及使用 Dead-Letter 队列 (DLQ) 处理未处理的消息。

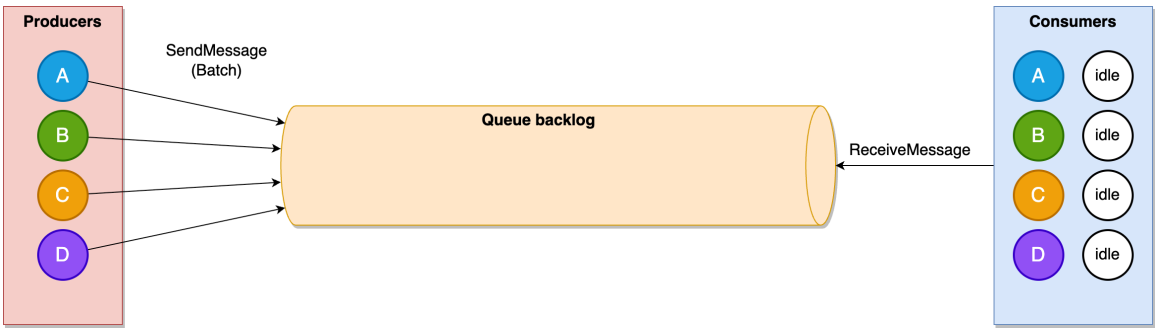
- 设置和调整超时。首先，将可见性超时设置为您的应用程序处理和删除消息通常所需的最长时间。如果您不确定确切的处理时间，可以从较短的超时（例如 2 分钟）开始，然后根据需要延长。实施心跳机制来定期延长可见性超时，确保消息在处理完成之前保持不可见状态。这样做可以最大限度地减少重新处理未处理的消息时的延迟，并防止其过早变得可见。
- 延长超时时间并处理 12 小时限制。如果您的处理时间会有所变化或可能超过最初设置的超时，请在处理消息时使用 `ChangeMessageVisibility` 操作延长可见性超时。请注意，可见性超时的最大限制为首次收到消息后的 12 小时。延长超时时间不会重置该 12 小时限制。如果您的处理需要的时间超过此限制，请考虑使用任务 AWS Step Functions 或将任务分成较小的步骤。
- 处理未处理的消息。要管理多次处理尝试失败的邮件，请配置 Dead-Letter 队列 (DLQ)。这样做可以确保单独捕获在多次重试后仍无法处理的消息，以供进一步分析或处理，从而防止它们在主队列中反复出现。

Amazon SQS 公平队列

对于包含来自多个逻辑实体（例如客户、客户端应用程序或消息类型）的消息的多租户队列，Amazon SQS 公平队列可自动减轻嘈杂邻居的影响。在这些共享队列环境中，一个关键的性能指标是停留时间，它衡量消息从到达到处理在队列中花费的总时间。当一个租户发布的消息数量超出系统处理能力，导致队列中出现积压情况时，公平队列可以最大限度地减轻对其他租户停留时间的影响。

稳定状态

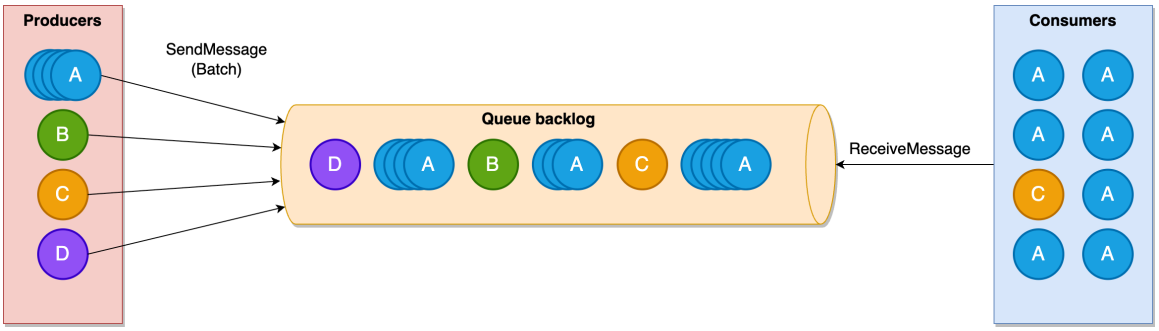
下图展示了一个多租户队列，其中包含来自四个不同租户（标记为 A、B、C 和 D）的消息。队列在稳定状态下运行，并且没有消息积压，因为使用者在消息出现在队列中后立即收到消息。所有租户的停留时间都很短。在这种稳定状态下，并非所有的使用者容量都得到充分利用。



嘈杂邻居影响

当多租户队列中的一个租户出现积压情况时，就会产生嘈杂邻居影响，导致其他所有租户的消息停留时间增加。租户可能会因发送的消息量超过其他租户而成为嘈杂邻居，或者当使用者处理来自该特定租户的消息耗时更长时，租户也可能会成为嘈杂邻居。

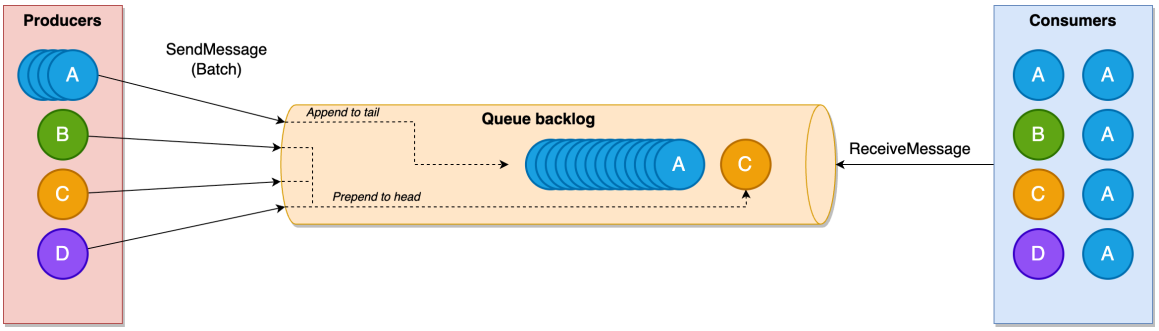
此图展示了来自租户 A 的流量增加是如何导致队列出现积压的。使用者忙于处理仅来自租户 A 的消息，而来自其他租户的消息则积压在队列中，导致所有租户的停留时间延长。



通过公平队列进行缓解

Amazon SQS 通过监控处理期间（“传输中”状态）租户之间的消息分发情况来检测嘈杂邻居。当某个租户的传输中消息数量远超其他租户时，Amazon SQS 会将该租户识别为嘈杂邻居，并优先处理其他租户的消息。这种方法减轻了对其他租户的停留时间影响。

此图展示了 Amazon SQS 公平队列是如何解决嘈杂邻居问题的。当某个租户（租户 A）变得嘈杂时，Amazon SQS 会优先返回其他租户（B、C 和 D）的消息。这种优先级排序有助于为安静租户（B、C 和 D）保持较低的停留时间，而租户 A 消息的停留时间则会延长，直到队列中积压的消息处理完毕，且不会影响其他租户。



> **Note**
>
> Amazon SQS 不限制每个租户的使用速率。当使用者具备处理能力且队列中没有其他消息需要返回时，它允许使用者接收来自嘈杂邻居租户的消息。与 Amazon SQS 标准队列一样，公平队列允许几乎无限的吞吐量，而且您在队列中可以拥有的租户数量没有限制。

有关 Amazon SQS 如何检测嘈杂的邻居并管理消息传送顺序的详细信息，请参阅 Amazon SQS 公平队列的工作原理。

如何使用公平队列

要启用公平队列，消息制作者应通过在传出消息MessageGroupId上设置 a 来添加租户标识符：

```
// Send message with tenant identifier
SendMessageRequest request = new SendMessageRequest()
    .withQueueUrl(queueUrl)
    .withMessageBody(messageBody)
    .withMessageGroupId("tenant-123"); // Tenant identifier
sqs.sendMessage(request);
```

公平性功能将自动应用于所有 Amazon SQS 标准队列中，用于处理带有该 MessageGroupId 属性的消息。它不需要对使用者代码进行任何更改，对 API 延迟没有影响，并且没有任何吞吐量限制。

> **Note**
>
> MessageGroupId在具有公平队列的标准队列上，其行为与MessageGroupId在 FIFO 队列上的行为不同。在标准队列上，MessageGroupId仅用作公平队列的租户标识符，不强制执行消息排序。有关 FIFO 队列的详细信息，请参阅MessageGroupId在 Amazon SQS FIFO 队列中使用消息组 ID。

何时使用公平队列

当以下所有条件都适用于您的队列时，可以考虑使用公平队列：

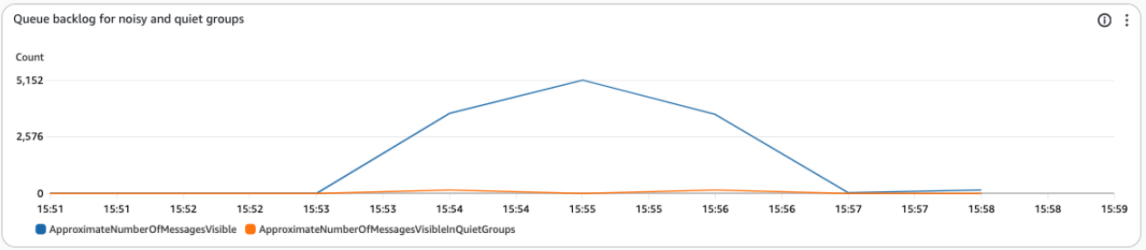
- 队列是多租户的。队列中包含属于多个逻辑实体（例如客户、客户端应用程序或请求类型）的消息，您可以用标识每个实体MessageGroupId。
- 队列吞吐量很高。在低吞吐量下，一个租户的突发很少会造成影响其他租户的待办事项。在高吞吐量下，过度扩展消费群以吸收每一次爆发是不切实际的，即使采用自动缩放功能，新消费者上线之前的延迟也可能导致积压。
- 停留时间是应用程序服务质量的一部分。公平的排队可以保护安静的租户免受邻居吵闹而导致的停留时间延长。如果您的应用程序对停留时间不敏感，则可能不需要公平队列提供的噪音邻居保护。

公平队列 CloudWatch 指标

Amazon SQS 提供了其他 CloudWatch 指标来帮助您监控邻居噪音影响的缓解情况。例如，您可以将 Approximate..InQuietGroups 指标与标准队列级别的指标进行比较。在特定租户的流量激增期

间，队列级别的一般指标可能会显示积压的消息增加或消息停留时间延长。但是，如果单独查看安静组，您可以发现大多数非嘈杂消息组或租户都没有受到影响。

您可以在下面找到一个示例，其中标准队列待办事项指标 (ApproximateNumberOfMessagesVisible) 因租户噪音而增加，而非嘈杂租户的积压待办事项 (ApproximateNumberOfMessagesVisibleInQuietGroups) 仍然很低。



有关亚马逊 SQS CloudWatch 指标及其描述的完整列表，请参阅亚马逊 SQ S CloudWatch 指标。

亚马逊 SQS 公平队列是如何运作的

Amazon SQS 公平队列可防止单个租户减慢其他租户的消息处理速度，从而自动减轻多租户队列中邻居噪音的影响。当公平队列检测到一个租户在队列的使用者容量中占用了不成比例的份额时，他们会优先考虑来自安静租户的消息传送。

租户识别

公平队列在每条消息上按`MessageGroupId`房产识别租户。共享相同内容的消息`MessageGroupId`属于同一个租户。公平队列可以包含带有和不带的混合消息`MessageGroupId`。不带的每条消息`MessageGroupId`都被视为属于其自己的不同租户。

我们建议您使用映射到系统中真实实体的标识符（例如客户 ID、客户端应用程序 ID 或请求类型）对发送到公平队列的每条消息都设置一个。`MessageGroupId`

检测的工作原理

Amazon SQS 使用两种衡量标准来检测邻居吵闹的声音：

- 并发份额 — 租户的飞行中消息占队列中所有飞行中消息的一小部分。这可以检测出同时处理多条消息的租户，无论是由于发送量大、处理速度慢还是两者兼而有之。
- 处理时间份额 — 租户最近在消费者处理总时间中所占的份额。它可以检测到消息数量很少但处理速度较慢的租户，因此租户会占用大量的消费者时间，而不会同时传输很多消息。

当以下任一条件为真时，租户将被标记为吵闹的邻居：

- 并发份额：租户占队列中传输中消息的10%以上，并且至少有30条自己的消息正在传输中。
- 处理时间份额：租户最近在消费者处理总时间中所占的份额超过 10%。

由于 Amazon SQS 是一个分布式系统，因此这些阈值是近似值，检测可能无法在所描述的确切值下激活。

留言传送顺序如何变化

一旦租户被检测为吵闹的邻居，Amazon SQS 就会优先发送来自安静租户的消息。每当有静默租户消息可用时，轮询消息的使用者实例就会收到来自安静租户的消息。这种优先级划分一直持续到吵闹的邻居的并发份额和处理时间份额下降到与安静的租户相当的水平。

嘈杂的邻居消息不会被丢弃或限制。他们的停留时间会增加，而安静的租户消息会被优先排序，待处理的消息清理完毕后就会恢复正常。当没有静默租户消息在等待时，使用者实例会像往常一样收到嘈杂的邻居消息。

如果同时检测到多个租户是嘈杂的邻居，则消费者实例会首先从嘈杂的租户那里收到消息，其中飞行中消息最少。这有助于在嘈杂的租户之间平衡处理时间。

当吵闹的邻居变得安静时

当满足以下任一条件时，租户不再被视为吵闹的邻居：

- 租户的待办事项已全部用完。
- 在连续 5 分钟内，租户没有发送任何消息。

之后，租户的消息将不再被取消优先级，其处理方式与来自任何其他安静租户的消息相同。

消费者容量注意事项

为了使并发共享衡量标准有效地发挥作用，您的消费者需要同时处理足够的消息，这样一个租户在飞行中消息中所占份额才能脱颖而出。在调整消费者车队规模时，请考虑这一点。

通过事件源映射将 Lambda 用作使用者时，传输中消息的数量取决于 Lambda 函数并发性和批处理大小。在调整消费者规模时，请同时评估这些设置。

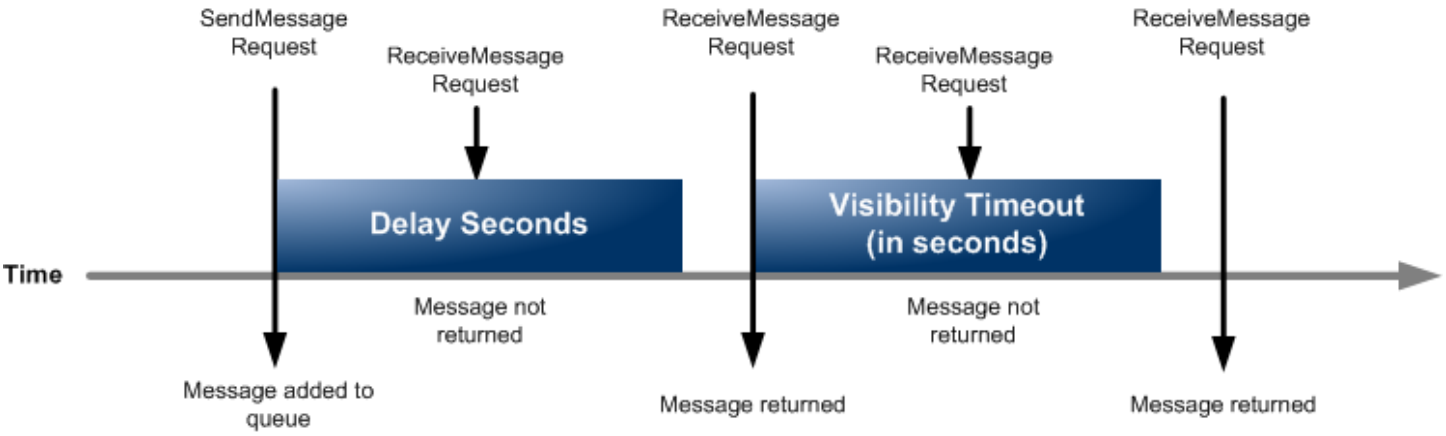
当传输中的消息数量太低而无法触发并发共享阈值时，处理时间共享度量仍然可以检测到有噪音的邻居。但是，当消费者同时处理足够多的消息以供评估这两种衡量标准时，公平队列效果最好。

Amazon SQS 延迟队列

延迟队列允许您将向使用者发送新消息推迟几秒钟，例如，当您的使用者应用程序需要更多时间来处理消息时。如果您创建延迟队列，则发送到该队列的任何消息在延迟期间对使用者都不可见。队列的默认（最小）延迟为 0 秒。最大值为 15 分钟。有关使用控制台配置延迟队列的信息，请参阅使用 Amazon SQS 控制台配置队列参数。

> **Note**
>
> 对于标准队列，每队列延迟设置是不可追溯的 — 更改此设置不会影响队列中已有的消息的延迟。
>
> 对于 FIFO 队列，每队列延迟设置可追溯，更改此设置会影响队列中已有的消息的延迟。

延迟队列类似于可见性超时，因为这两种特征都使得使用者在特定的时间段内无法获得消息。二者之间的区别在于：对于延迟队列，消息在首次添加到队列时 是隐藏的；而对于可见性超时，消息只有在从队列使用后 才是隐藏的。下图说明了延迟队列和可见性超时之间的关系。



延长调度选项

虽然 Amazon SQS 延迟队列和消息计时器支持在将来最多 15 分钟内调度消息传递，但您可能需要更灵活的调度功能。在这种情况下，可以考虑使用EventBridge 调度器，它使您能够不受时间限制地安排数十亿次一次性或重复性API操作。 EventBridge Scheduler 是高级消息调度用例的推荐解决方案。

要对单条消息而非整个队列设置延迟（以秒为单位），请使用消息计时器以允许 Amazon SQS 使用消息计时器的 `DelaySeconds` 值，而不是使用延迟队列的 `DelaySeconds` 值。EventBridge 调度器还支持调度单条消息。

Amazon SQS 临时队列

使用 request-response 等常见消息模式时，临时队列可帮助您节省开发时间和部署成本。您可以使用临时队列客户端创建高吞吐量、经济实惠、由应用程序管理的临时队列。

客户端会自动将多个临时队列（按需为特定流程创建的应用程序管理队列）映射到单个 Amazon SQS 队列。当指向每个临时队列的流量较低时，这就使得您的应用程序发出较少的 API 调用，实现更高的吞吐量。当临时队列不再使用时，客户端自动清除临时队列，即使部分使用客户端的进程没有完全关闭也是如此。

以下是临时队列的优势：

- 它们充当特定线程或进程的轻型通信渠道。
- 创建和删除临时队列不会产生额外的成本。
- 它们 API-compatible 使用静态（普通）亚马逊 SQS 队列。这意味着发送和接收消息的现有代码可以针对虚拟队列发送和接收消息。

虚拟队列

虚拟队列是临时队列客户端创建的本地数据结构。使用虚拟队列让您可以将多个低流量目标合并成单个 Amazon SQS 队列。有关最佳实践，请参阅与避免在虚拟队列中重用相同的消息组 ID 相关的内容。

> **Note**
>
> - 创建虚拟队列操作仅为接收消息的使用者创建临时数据结构。虚拟队列不对 Amazon SQS 发出 API 调用，因此不会产生任何成本。
> - TPS 配额适用于单个主机队列中的所有虚拟队列。有关更多信息，请参阅 Amazon SQS 消息配额。

`AmazonSQSVirtualQueuesClient` 包装程序类增加了对与虚拟队列相关属性的支持。要创建虚拟队列，您必须使用 `HostQueueURL` 属性调用 `CreateQueue` API 操作。此属性指定托管虚拟队列的现有队列。

虚拟队列 URL 采用以下格式。

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue#MyVirtualQueueName
```

创建者在虚拟队列 URL 上调用 SendMessage 或 SendMessageBatch API 操作时，临时队列客户端执行以下操作：

1. 提取虚拟队列名称。
2. 将虚拟队列名称作为额外的消息属性进行附加。
3. 发送消息到主机队列。

当创建者发送消息时，后台线程轮询主机队列，并根据对应的消息属性将收到的消息发送到虚拟队列。

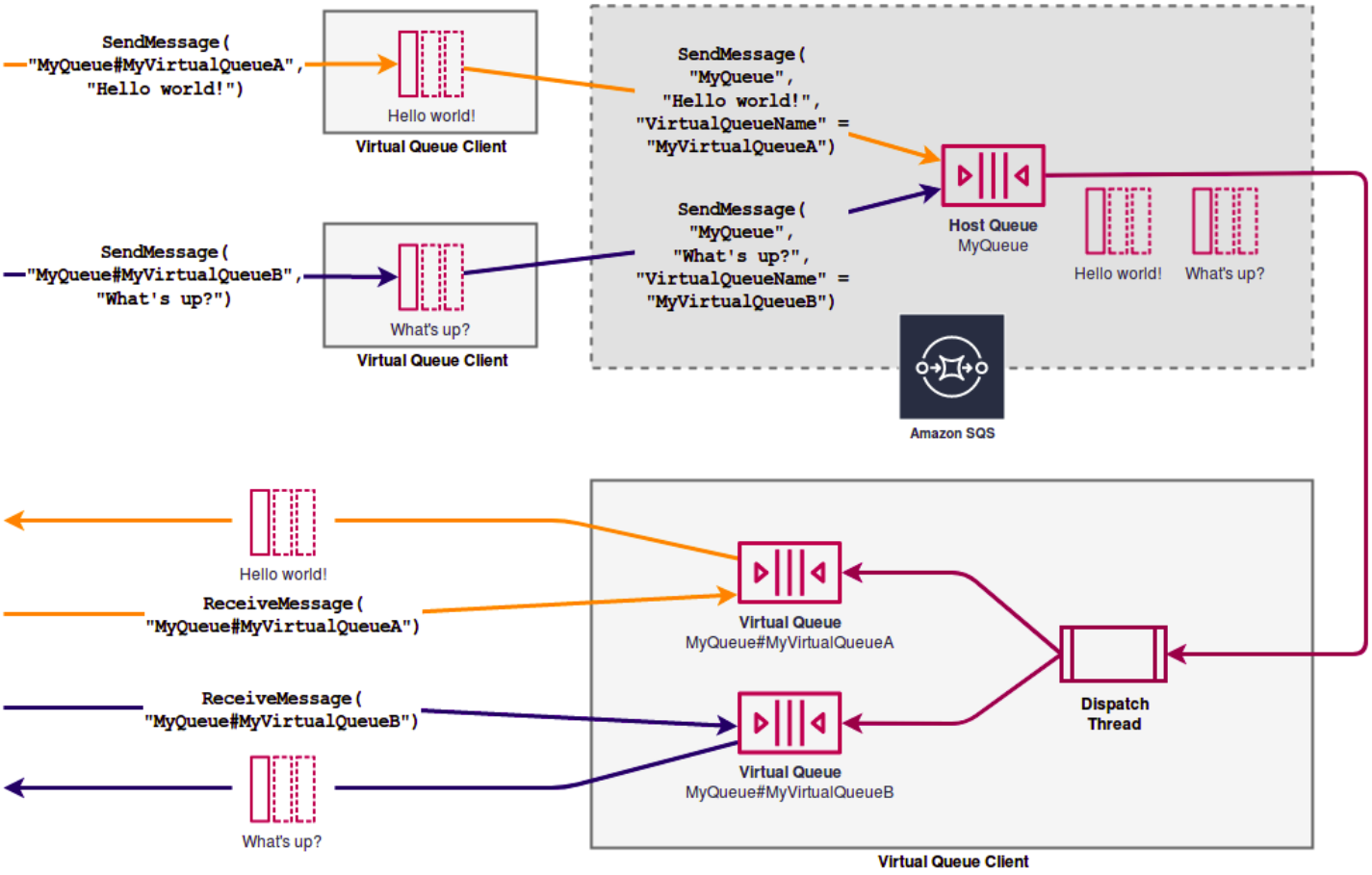
当使用者在虚拟队列 URL 上调用 ReceiveMessage API 操作时，临时队列客户端在本地阻止调用，直至后台线程将消息发送到虚拟队列中。（此过程类似于 Buffered Asynchronous Client 中的消息预取操作：单个 API 操作可以将消息提供到最多 10 个虚拟队列。）删除虚拟队列操作可以在不调用 Amazon SQS 本身的情况下删除任何客户端资源。

AmazonSQSTemporaryQueuesClient 类自动将它创建的所有队列转入临时队列中。它还会自动使用相同的队列属性，按需创建主机队列。这些队列的名称共用共同的可配置前缀（默认情况下为 __RequesterClientQueues__），该前缀将队列标识为临时队列。这使得客户端充当了简易替代，可优化创建和删除队列的现有代码。客户端还包括 AmazonSQSRequester 和 AmazonSQSResponder 接口，以允许队列之间的双向通信。

Request-response 消息传送模式（虚拟队列）

临时队列的最常见使用案例为请求-响应消息收发模式，此时请求者创建临时队列来接收各个响应消息。为了避免为每个响应消息创建 Amazon SQS 队列，临时队列客户端允许您创建和删除多个临时队列，而无需进行任何 Amazon SQS API 调用。有关更多信息，请参阅与实现请求-响应系统相关的内容。

下图显示了使用此模式时的常见配置。



示例场景：处理登录请求

以下示例场景展示了如何使用 AmazonSQSRequester 和 AmazonSQSResponder 接口来处理用户的登录请求。

在客户端

```
public class LoginClient {

    // Specify the Amazon SQS queue to which to send requests.
    private final String requestQueueUrl;

    // Use the AmazonSQSRequester interface to create
    // a temporary queue for each response.
    private final AmazonSQSRequester sqsRequester =
            AmazonSQSRequesterClientBuilder.defaultClient();

    LoginClient(String requestQueueUrl) {
        this.requestQueueUrl = requestQueueUrl;
```

```
    }

    // Send a login request.
    public String login(String body) throws TimeoutException {
        SendMessageRequest request = new SendMessageRequest()
                .withMessageBody(body)
                .withQueueUrl(requestQueueUrl);

        // If no response is received, in 20 seconds,
        // trigger the TimeoutException.
        Message reply = sqsRequester.sendMessageAndGetResponse(request,
                20, TimeUnit.SECONDS);

        return reply.getBody();
    }
}
```

发送登录请求将执行以下操作：

1. 创建一个临时队列。
2. 将临时队列的 URL 作为属性附加到消息。
3. 发送消息。
4. 从临时队列接收响应。
5. 删除临时队列。
6. 返回响应。

在服务器端

以下示例假设在构造时创建了一个线程，它为每个消息轮询该队列并调用 handleLoginRequest() 方法。此外，假设使用了 doLogin() 方法。

```
public class LoginServer {

    // Specify the Amazon SQS queue to poll for login requests.
    private final String requestQueueUrl;

    // Use the AmazonSQSResponder interface to take care
    // of sending responses to the correct response destination.
    private final AmazonSQSResponder sqsResponder =
            AmazonSQSResponderClientBuilder.defaultClient();
```

```
    LoginServer(String requestQueueUrl) {
        this.requestQueueUrl = requestQueueUrl;
    }

    // Process login requests from the client.
    public void handleLoginRequest(Message message) {

        // Process the login and return a serialized result.
        String response = doLogin(message.getBody());

        // Extract the URL of the temporary queue from the message attribute
        // and send the response to the temporary queue.
        sqsResponder.sendResponseMessage(MessageContent.fromMessage(message),
                new MessageContent(response));
    }
}
```

清除队列

为确保 Amazon SQS 回收虚拟队列使用的任何内存中资源，应用程序不再需要临时队列客户端时，它应调用 `shutdown()` 方法。您还可以使用 `AmazonSQSRequester` 接口的 `shutdown()` 方法。

临时队列客户端还提供了一种方法来消除孤立的主机队列。对于在一段时间内（默认情况下为 5 分钟）接收 API 调用的每个队列，客户端使用 `TagQueue` API 操作来标记仍在使用的队列。

> **Note**
>
> 在队列上执行的任意 API 操作将标记为非空闲，包括不返回任何消息的 `ReceiveMessage` 操作。

后台线程使用 `ListQueues` 和 `ListTags` API 操作，以检查具有已配置前缀的所有队列，并删除至少五分钟内没有被标记的所有队列。这样，如果一个客户端未完全关闭，则其他活动客户端在之后会进行清除。为了减少重复工作，具有相同前缀的所有客户端会通过共享的内部工作队列通信，该队列以前缀命名。

Amazon SQS 消息计时器

借助消息计时器，您可以为添加到队列中的消息指定初始不可见时段。例如，如果您发送的消息带有一个 45 秒的计时器，则该消息在前 45 秒内将对使用者保持隐藏状态。消息的默认（最小）延迟为 0

秒。最大值为 15 分钟。有关使用控制台发送带有计时器的消息的信息，请参阅使用标准队列发送消息。

> **Note**
>
> FIFO 队列不支持单个消息的计时器。

要对整个队列（而不是单条消息）设置延迟时段，请使用延迟队列。单个消息的消息计时器设置将覆盖 Amazon SQS 延迟队列上的任何 `DelaySeconds` 值。

延长调度选项

虽然 Amazon SQS 延迟队列和消息计时器支持在将来最多 15 分钟内调度消息传递，但您可能需要更灵活的调度功能。在这种情况下，请考虑使用 EventBridge 调度器，它使您能够调度数十亿次一次性或重复性 API 操作，且无时间限制。EventBridge 调度器是适合高级消息调度使用案例的推荐解决方案。

通过亚马逊 SQS EventBridge 控制台访问 Amazon Pipes

Amazon Pip EventBridge es 将源与目标连接起来。Pipes 旨在在支持的源和目标之间 point-to-point进行集成，并支持高级转换和扩展。 EventBridge 管道提供了一种高度可扩展的方式，可以将您的亚马逊 SQS 队列与 Step Functions、Amazon SQS 和 API Gateway 等 AWS 服务以及 Salesforce 等第三方软件即服务 (SaaS) 应用程序连接起来。

要设置管道，请选择来源，添加可选筛选，定义可选扩充，然后为事件数据选择目标。

在 Amazon SQS 队列的详细信息页面上，您可以查看使用该队列作为来源的管道。在这里，您还可以：

- 启动 EventBridge 控制台以查看管道详细信息。
- 启动 EventBridge 控制台以创建以队列为源的新管道。

有关将 Amazon SQS 队列配置为管道源的更多信息，请参阅亚马逊用户指南中的亚马逊 SQS 队列作为来源。 EventBridge 有关一般 EventBridge 管道的更多信息，请参见EventBridge 管道。

访问给定 Amazon SQS 队列的 EventBridge 管道

1. 打开 Amazon S3 控制台的队列页面。

2. 选择队列。
3. 在队列详细信息页面上，选择EventBridge 管道选项卡。

   “EventBridge 管道” 选项卡包含当前配置为使用所选队列作为源的所有管道的列表，包括：

   - 管道名称
   - 当前状态
   - 管道目标
   - 上次修改管道的时间
4. 查看更多管道详细信息或创建新管道（如需要）：

   - 访问有关管道的更多详细信息：

     选择管道名称。

     这将启动 EventBridge控制台的 Pipe 详细信息页面。
   - 创建新管道：

     选择将 Amazon SQS 队列连接到管道。

     这将启动 EventBridge控制台的 “创建管道” 页面，将 Amazon SQS 队列指定为管道源。有关更多信息，请参阅《Amazon EventBridge 用户指南》中的创建 EventBridge管道。

     > **Important**
     >
     > Amazon SQS 队列中的消息先由单个管道读取，然后在处理完毕后从队列中删除，无论该消息是否与您可以为该管道配置的筛选条件匹配。在将多个管道配置为使用相同队列作为来源时，请谨慎操作。

使用扩展型客户端库和 Amazon Simple Storage Service 管理大型 Amazon SQS 消息

使用适用于 Java 的 Amazon SQS 扩展型客户端库和适用于 Python 的 Amazon SQS 扩展型客户端库发送大型消息，尤其是有效载荷在 256 KB 和 2 GB 之间的消息。这些库将消息有效载荷存储到 Amazon S3 存储桶中，并将对已存储对象的引用发送到 Amazon SQS 队列。

> **Note**
>
> Amazon SQS 扩展型客户端库与标准队列和 FIFO 队列都兼容。

使用 Java 和 Amazon S3 管理大型 Amazon SQS 消息

使用适用于 Java 的 Amazon SQS 扩展型客户端库和 Amazon S3 来管理大型 Amazon SQS 消息，尤其是有效载荷在 256 KB 到 2 GB 之间的消息。该库将消息有效载荷存储到 Amazon S3 存储桶中，并将包含对已存储对象的引用的消息发送到 Amazon SQS 队列。

借助适用于 Java 的 Amazon SQS 扩展型客户端库，您可以执行以下操作：

- 指定消息是始终存储在 Amazon S3 中还是仅在其大小超过 256 KB 时存储在 Amazon S3 中。
- 发送引用存储在 S3 存储桶中的单个消息对象的消息
- 从 Amazon S3 存储桶检索消息对象
- 从 Amazon S3 存储桶中删除消息对象

先决条件

以下示例使用的是 AWS Java 开发工具包。要安装和设置开发工具包，请参阅 适用于 Java 的 AWS SDK 开发人员指南中的设置 AWS SDK for Java。

运行示例代码之前，请配置您的 AWS 凭证。有关更多信息，请参阅《适用于 Java 的 AWS SDK 开发人员指南》中的设置 AWS 凭据和开发区域。

适用于 Java 的 SDK 和适用于 Java 的 Amazon SQS 扩展型客户端库需要使用 J2SE Development Kit 8.0 或更高版本。

> **Note**
>
> 您可以通过适用于 Java 的 Amazon SQS 扩展型客户端库，仅利用 Amazon S3 与适用于 Java 的 AWS SDK 来管理 Amazon SQS 消息。您无法使用 AWS CLI、Amazon SQS 控制台、Amazon SQS HTTP API 或任何其他 AWS SDK，执行此操作。

AWS SDK for Java 2.x 示例：使用 Amazon S3 管理大型 Amazon SQS 消息

以下适用于 Java 2.x 的 SDK 示例使用适用于 Java 的扩展型客户端库来处理大型消息。在构造函数中，以下代码：

- 创建了具有随机名称的 Amazon S3 存储桶
- 创建了以 `MyQueue` 开头的 SQS 队列
- 在 `AmazonSQSExtendedClient` 的实例中封装了标准 Java SDK Amazon S3 客户端

在 `sendAnReceiveMessage` 方法中，该示例会发送一条存储在 Amazon S3 存储桶中的随机消息，因为该消息大小超过 256 KB（标准最大消息大小）。最后，该方法会检索消息并将其相关信息显示到控制台。

您可以在 https://github.com/awsdocs/aws-doc-sdk-examples/blob/94d1b24df12deda0f4fd91433b8231fed6d18b85/javav2/example_code/sqs/src/main/java/com/example/sqs/SqsExtendedClientExample.java#L1 中查看完整示例。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

            import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.BucketLifecycleConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
```

```
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.ExpirationStatus;
import software.amazon.awssdk.services.s3.model.LifecycleExpiration;
import software.amazon.awssdk.services.s3.model.LifecycleRule;
import software.amazon.awssdk.services.s3.model.LifecycleRuleFilter;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsRequest;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.PutBucketLifecycleConfigurationRequest;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;

import java.util.Arrays;
import java.util.List;
import java.util.UUID;


/**
 * Examples of using Amazon SQS Extended Client Library for Java 2.x
 *
 */
public class SqsExtendedClientExamples {
    // Create an Amazon S3 bucket with a random name.
    private final static String amzn-s3-demo-bucket = UUID.randomUUID() + "-"
            + DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new DateTime());

    public static void main(String[] args) {

        /*
         * Create a new instance of the builder with all defaults (credentials
         * and region) set automatically. For more information, see
         * Creating Service Clients in the AWS SDK for Java Developer Guide.
         */
        final S3Client s3 = S3Client.create();

        /*
```

```
         * Set the Amazon S3 bucket name, and then set a lifecycle rule on the
         * bucket to permanently delete objects 14 days after each object's
         * creation date.
         */
        final LifecycleRule lifeCycleRule = LifecycleRule.builder()
                .expiration(LifecycleExpiration.builder().days(14).build())
                .filter(LifecycleRuleFilter.builder().prefix("").build())
                .status(ExpirationStatus.ENABLED)
                .build();
        final BucketLifecycleConfiguration lifecycleConfig =
 BucketLifecycleConfiguration.builder()
                .rules(lifeCycleRule)
                .build();

        // Create the bucket and configure it
        s3.createBucket(CreateBucketRequest.builder().bucket(amzn-s3-demo-
bucket).build());

 s3.putBucketLifecycleConfiguration(PutBucketLifecycleConfigurationRequest.builder()
                .bucket(amzn-s3-demo-bucket)
                .lifecycleConfiguration(lifecycleConfig)
                .build());
        System.out.println("Bucket created and configured.");

        // Set the Amazon SQS extended client configuration with large payload support
 enabled
        final ExtendedClientConfiguration extendedClientConfig = new
 ExtendedClientConfiguration().withPayloadSupportEnabled(s3, amzn-s3-demo-bucket);

        final SqsClient sqsExtended = new
 AmazonSQSExtendedClient(SqsClient.builder().build(), extendedClientConfig);

        // Create a long string of characters for the message object
        int stringLength = 300000;
        char[] chars = new char[stringLength];
        Arrays.fill(chars, 'x');
        final String myLongString = new String(chars);

        // Create a message queue for this example
        final String queueName = "MyQueue-" + UUID.randomUUID();
        final CreateQueueResponse createQueueResponse =
 sqsExtended.createQueue(CreateQueueRequest.builder().queueName(queueName).build());
        final String myQueueUrl = createQueueResponse.queueUrl();
        System.out.println("Queue created.");
```

```
        // Send the message
        final SendMessageRequest sendMessageRequest = SendMessageRequest.builder()
                .queueUrl(myQueueUrl)
                .messageBody(myLongString)
                .build();
        sqsExtended.sendMessage(sendMessageRequest);
        System.out.println("Sent the message.");

        // Receive the message
        final ReceiveMessageResponse receiveMessageResponse =
 sqsExtended.receiveMessage(ReceiveMessageRequest.builder().queueUrl(myQueueUrl).build());
        List<Message> messages = receiveMessageResponse.messages();

        // Print information about the message
        for (Message message : messages) {
            System.out.println("\nMessage received.");
            System.out.println("  ID: " + message.messageId());
            System.out.println("  Receipt handle: " + message.receiptHandle());
            System.out.println("  Message body (first 5 characters): " +
 message.body().substring(0, 5));
        }

        // Delete the message, the queue, and the bucket
        final String messageReceiptHandle = messages.get(0).receiptHandle();

 sqsExtended.deleteMessage(DeleteMessageRequest.builder().queueUrl(myQueueUrl).receiptHandle(me
        System.out.println("Deleted the message.");


 sqsExtended.deleteQueue(DeleteQueueRequest.builder().queueUrl(myQueueUrl).build());
        System.out.println("Deleted the queue.");

        deleteBucketAndAllContents(s3);
        System.out.println("Deleted the bucket.");

    }

    private static void deleteBucketAndAllContents(S3Client client) {
        ListObjectsV2Response listObjectsResponse =
 client.listObjectsV2(ListObjectsV2Request.builder().bucket(amzn-s3-demo-
bucket).build());

        listObjectsResponse.contents().forEach(object -> {
```

```
            client.deleteObject(DeleteObjectRequest.builder().bucket(amzn-s3-demo-
bucket).key(object.key()).build());
        });

        ListObjectVersionsResponse listVersionsResponse =
 client.listObjectVersions(ListObjectVersionsRequest.builder().bucket(amzn-s3-demo-
bucket).build());

        listVersionsResponse.versions().forEach(version -> {
            client.deleteObject(DeleteObjectRequest.builder().bucket(amzn-s3-demo-
bucket).key(version.key()).versionId(version.versionId()).build());
        });

        client.deleteBucket(DeleteBucketRequest.builder().bucket(amzn-s3-demo-
bucket).build());
    }
}
```

您可以使用 Apache Maven 配置和构建适用于 Java 项目的 Amazon SQS 扩展型客户端或构建 SDK 本身。指定您的应用程序中使用的 SDK 的各个模块。

```
<properties>
    <aws-java-sdk.version>2.20.153</aws-java-sdk.version>
</properties>

<dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>sqs</artifactId>
      <version>${aws-java-sdk.version}</version>
    </dependency>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>s3</artifactId>
      <version>${aws-java-sdk.version}</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>amazon-sqs-java-extended-client-lib</artifactId>
      <version>2.0.4</version>
    </dependency>
```

```
    <dependency>
      <groupId>joda-time</groupId>
      <artifactId>joda-time</artifactId>
      <version>2.12.6</version>
    </dependency>
</dependencies>
```

使用 Python 和 Amazon S3 管理大型 Amazon SQS 消息

使用适用于 Python 的 Amazon SQS 扩展型客户端库和 Amazon S3 来管理大型 Amazon SQS 消息，尤其是有效载荷在 256 KB 和 2 GB 之间的消息。该库将消息有效载荷存储到 Amazon S3 存储桶中，并将包含对已存储对象的引用的消息发送到 Amazon SQS 队列。

借助适用于 Python 的 Amazon SQS 扩展型客户端库，您可以执行以下操作：

- 指定有效载荷是始终存储在 Amazon S3 中，还是仅当有效载荷大小超过 256 KB 时才存储在 Amazon S3 中
- 发送引用存储在 Amazon S3 存储桶中的单个消息对象的消息
- 从 Amazon S3 存储桶检索相应的有效载荷对象
- 从 Amazon S3 存储桶删除相应的有效载荷对象

先决条件

以下是使用适用于 Python 的 Amazon SQS 扩展型客户端库的先决条件：

- 具有必要凭证的 AWS 账户。要创建 AWS 帐户，请导航到AWS 主页，然后选择创建 AWS 帐户。按照说明进行操作。有关凭证的信息，请参阅 Credentials。
- AWS 软件开发工具包：本页上的示例使用 AWS Python SDK Boto3。要安装和设置 SDK，请参阅《适用于 Python 的AWS SDK 开发人员指南》中的适用于 Python 的AWS SDK 文档
- Python 3.x（或更高版本）和 `pip`。
- 适用于 Python 的 Amazon SQS 扩展型客户端库（可以从 PyPI 中获得）

> **Note**
>
> 只有使用 AWS 适用于 Python 的软件开发工具包，您才能使用亚马逊 SQS Python 扩展客户端库通过亚马逊 S3 管理亚马逊 SQS 消息。你无法 AWS 使用 CLI、亚马逊 SQS 控制台、亚马逊 SQS HTTP API 或任何其他方法来做到这一点。 AWS SDKs

配置消息存储

Amazon SQS 扩展型客户端使用以下消息属性来配置 Amazon S3 消息存储选项：

- `large_payload_support`：用于存储大型消息的 Amazon S3 存储桶名称。
- `always_through_s3`：如果设置为 `True`，则所有消息都会存储在 Amazon S3 中。如果设置为 `False`，则小于 256KB 的消息不会被序列化并存储到 S3 存储桶中。默认值为 `False`。
- `use_legacy_attribute`：如果设置为 `True`，则所有已发布的消息都使用旧版保留消息属性（`SQSLargePayloadSize`），而不是当前的保留消息属性（`ExtendedPayloadSize`）。

使用适用于 Python 的扩展型客户端库管理大型 Amazon SQS 消息

以下示例创建了具有随机名称的 Amazon S3 存储桶。然后，它会创建一个名为 `MyQueue` 的 Amazon SQS 队列并向该队列发送一条存储在 S3 存储桶中且大小超过 256KB 的随机消息。最后，代码会检索该消息，返回该消息的相关信息，并删除该消息、队列和存储桶。

```
import boto3
import sqs_extended_client

#Set the Amazon SQS extended client configuration with large payload.
sqs_extended_client = boto3.client("sqs", region_name="us-east-1")
sqs_extended_client.large_payload_support = "amzn-s3-demo-bucket"
sqs_extended_client.use_legacy_attribute = False


# Create an SQS message queue for this example. Then, extract the queue URL.
queue = sqs_extended_client.create_queue(
    QueueName = "MyQueue"
)
queue_url = sqs_extended_client.get_queue_url(
    QueueName = "MyQueue"
```

```
)['QueueUrl']


# Create the S3 bucket and allow message objects to be stored in the bucket.
sqs_extended_client.s3_client.create_bucket(Bucket=sqs_extended_client.large_payload_support)

# Sending a large message
small_message = "s"
large_message = small_message * 300000 # Shall cross the limit of 256 KB

send_message_response = sqs_extended_client.send_message(
    QueueUrl=queue_url,
    MessageBody=large_message
)
assert send_message_response['ResponseMetadata']['HTTPStatusCode'] == 200

# Receiving the large message
receive_message_response = sqs_extended_client.receive_message(
    QueueUrl=queue_url,
    MessageAttributeNames=['All']
)
assert receive_message_response['Messages'][0]['Body'] == large_message
receipt_handle = receive_message_response['Messages'][0]['ReceiptHandle']

# Deleting the large message
# Set to True for deleting the payload from S3
sqs_extended_client.delete_payload_from_s3 = True
delete_message_response = sqs_extended_client.delete_message(
    QueueUrl=queue_url,
    ReceiptHandle=receipt_handle
)

assert delete_message_response['ResponseMetadata']['HTTPStatusCode'] == 200

# Deleting the queue
delete_queue_response = sqs_extended_client.delete_queue(
    QueueUrl=queue_url
)

assert delete_queue_response['ResponseMetadata']['HTTPStatusCode'] == 200
```

使用 Amazon SQS 控制台配置 Amazon SQS 队列

使用 Amazon SQS 控制台来配置和管理 Amazon SQS 队列及其特征。您也可以：

- 启用服务器端加密以增强安全性。
- 关联一个死信队列来处理未处理的消息。
- 设置触发器来调用 Lambda 函数，以实现事件驱动型处理。

适用于 Amazon SQS 的基于属性的访问控制

什么是 ABAC？

基于属性的访问控制 (ABAC) 是一种授权过程，它根据附加到用户和资源的标签来定义权限。 AWS ABAC 根据属性和值提供精细而灵活的访问控制，降低与重新配置的基于角色的策略相关的安全风险，并集中审计和访问策略管理。有关 ABAC 的更多详细信息，请参阅《IAM 用户指南》中的什么是 AWS ABAC。

Amazon SQS 支持 ABAC，允许您根据与 Amazon SQS 队列关联的标签和别名来控制对 Amazon SQS 队列的访问权限。Amazon SQS 中启用 ABAC 的标签和别名条件键授权 IAM 主体使用 Amazon SQS 队列，而无需编辑策略或管理授权。要进一步了解 ABAC 条件键，请参阅《服务授权参考》中的 Condition keys for Amazon SQS。

借助 ABAC，您可以使用标签为 Amazon SQS 队列配置 IAM 访问权限和策略，这有助于您扩展权限管理。您可以使用添加到每个业务角色的标签在 IAM 中创建单个权限策略，而不必在每次添加新资源时都更新策略。您还可以向 IAM 主体附加标签以创建 ABAC 策略。您可以将 ABAC 策略设计为在进行调用的 IAM 用户角色上的标签与 Amazon SQS 队列标签匹配时允许 Amazon SQS 操作。要了解有关添加标签的更多信息 AWS，请参阅AWS 标记策略和。Amazon SQS 成本分配标签

> **Note**
>
> ABAC for Amazon SQS 目前已在 AWS 所有提供亚马逊 SQS 的商业区域推出，但以下情况除外：
>
> - 亚太地区（海得拉巴）
> - 亚太地区（墨尔本）
> - 欧洲（西班牙）

- 欧洲（苏黎世）

为什么应该在 Amazon SQS 中使用 ABAC？

以下是在 Amazon SQS 中使用 ABAC 的一些好处：

- ABAC for Amazon SQS 需要更少的权限策略。您无需为不同工作职能创建不同策略。您可以使用适用于多个队列的资源和请求标签，这样可以减少操作开销。
- 使用 ABAC 快速扩大团队规模。当资源在创建过程中被适当地标记时，将根据标签自动授予新资源的权限。
- 使用 IAM 主体的权限来限制资源访问。您可以为 IAM 主体创建标签，并使用它们来限制对与 IAM 主体标签匹配的特定操作的访问权限。这可以帮助您自动执行授予请求权限的过程。
- 跟踪谁在访问您的资源。您可以通过查看 AWS CloudTrail中的用户属性来确定会话的身份。

主题

- Amazon SQS 中用于访问控制的标记
- 创建 IAM 用户和 Amazon SQS 队列
- 在 Amazon SQS 中测试基于属性的访问控制

Amazon SQS 中用于访问控制的标记

以下是在 Amazon SQS 中使用标签进行访问控制的示例。该 IAM 策略限制 IAM 用户只能针对包括特定资源标签（键：environment，值：production）的所有队列执行所有 Amazon SQS 操作。有关更多信息，请参阅使用标签进行基于属性的访问控制和 Organizations。 AWS

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessForProd",
      "Effect": "Allow",
      "Action": "sqs:*",
```

```
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "prod"
        }
      }
    }
  ]
}
```

创建 IAM 用户和 Amazon SQS 队列

以下示例说明如何使用 AWS 管理控制台 和创建 ABAC 策略来控制对 Amazon SQS 的访问。CloudFormation

使用 AWS 管理控制台

创建 IAM 用户

1. 登录 AWS 管理控制台 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
2. 从左侧导航窗格中选择用户。
3. 选择添加用户，然后在用户名文本框中输入名称。
4. 选择访问密钥 - 编程访问框，然后选择下一步: 权限。
5. 选择下一步: 标签。
6. 将标签键添加为 `environment`，将标签值添加为 `beta`。
7. 选择下一步: 审核，然后选择创建用户。
8. 复制访问密钥 ID 和秘密访问密钥并将其存储在安全位置。

添加 IAM 用户权限

1. 选择您创建的 IAM 用户。
2. 选择添加内联策略。
3. 在 JSON 选项卡上，粘贴以下策略。
4. 选择查看策略。
5. 选择创建策略。

使用 AWS CloudFormation

使用以下示例 CloudFormation 模板创建附有内联策略和 Amazon SQS 队列的 IAM 用户：

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "CloudFormation template to create IAM user with custom inline policy"
Resources:
    IAMPolicy:
        Type: "AWS::IAM::Policy"
        Properties:
            PolicyDocument: |
                {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "AllowAccessForSameResTag",
                            "Effect": "Allow",
                            "Action": [
                                "sqs:SendMessage",
                                "sqs:ReceiveMessage",
                                "sqs:DeleteMessage"
                            ],
                            "Resource": "*",
                            "Condition": {
                                "StringEquals": {
                                    "aws:ResourceTag/environment": "${aws:PrincipalTag/
environment}"
                                }
                            }
                        },
                        {
                            "Sid": "AllowAccessForSameReqTag",
                            "Effect": "Allow",
                            "Action": [
                                "sqs:CreateQueue",
                                "sqs:DeleteQueue",
                                "sqs:SetQueueAttributes",
                                "sqs:tagqueue"
                            ],
                            "Resource": "*",
                            "Condition": {
                                "StringEquals": {
                                    "aws:RequestTag/environment": "${aws:PrincipalTag/
environment}"
```

```
                            }
                        }
                    },
                    {
                        "Sid": "DenyAccessForProd",
                        "Effect": "Deny",
                        "Action": "sqs:*",
                        "Resource": "*",
                        "Condition": {
                            "StringEquals": {
                                "aws:ResourceTag/stage": "prod"
                            }
                        }
                    }
                ]
            }

        Users:
          - "testUser"
        PolicyName: tagQueuePolicy

IAMUser:
    Type: "AWS::IAM::User"
    Properties:
        Path: "/"
        UserName: "testUser"
        Tags:
          -
            Key: "environment"
            Value: "beta"
```

在 Amazon SQS 中测试基于属性的访问控制

以下示例展示了如何在 Amazon SQS 中测试基于属性的访问控制。

创建一个队列，将标签键设置为环境，将标签值设置为生产

运行此 AWS CLI 命令来测试创建队列，标签键设置为 environment，标签值设置为 prod。如果您没有 AWS CLI，则可以为您的计算机下载并配置它。

```
aws sqs create-queue --queue-name prodQueue –region us-east-1 –tags "environment=prod"
```

您会收到来自 Amazon SQS 端点的 `AccessDenied` 错误：

```
An error occurred (AccessDenied) when calling the CreateQueue operation: Access to the
 resource <queueUrl> is denied.
```

这是因为 IAM 用户的标签值与 CreateQueue API 调用中传递的标签不一致。请记住，我们向 IAM 用户应用了一个标签，其键设置为 environment，值设置为 beta。

创建一个队列，将标签键设置为环境，将标签值设置为测试

运行此 CLI 命令来测试创建队列，将标签键设置为 environment，将标签值设置为 beta。

```
aws sqs create-queue --queue-name betaQueue –region us-east-1 –tags "environment=beta"
```

您会收到一条消息，确认队列已成功创建，如下所示。

```
{
"QueueUrl": "<queueUrl>“
}
```

向队列发送消息

运行此 CLI 命令以测试向队列发送消息。

```
aws sqs send-message --queue-url <queueUrl> --message-body testMessage
```

响应会显示消息已成功传送到 Amazon SQS 队列。IAM 用户权限允许您向带有 beta 标签的队列发送消息。响应包括 MD5OfMessageBody 和包含消息的 MessageId。

```
{
"MD5OfMessageBody": "<MD5OfMessageBody>",
"MessageId": "<MessageId>"
}
```

使用 Amazon SQS 控制台配置队列参数

创建或编辑队列时，可配置以下参数：

- 可见性超时 - 从队列（由一个使用者）收到的消息对其他消息使用者不可见的时间长度。相关详情，请参阅可见性超时。

> **Note**
>
> 使用控制台配置可见性超时可配置队列中所有消息的超时值。要配置单条或多条消息的超时时间，必须使用其中一条 AWS SDKs。

- 消息保留期 - Amazon SQS 保留留在队列中的消息的时间长度。默认情况下，队列会将消息保留四天。您可以配置队列以将消息保留最多 14 天。相关详情，请参阅消息保留期。
- 传送延迟 - Amazon SQS 在传送已添加到队列的消息之前将会延迟的时间。相关详情，请参阅传送延迟。
- 最大消息大小 - 此队列的最大消息大小。相关详情，请参阅最大消息大小。
- 接收消息等待时间 - Amazon SQS 在队列收到接收请求后等待消息变为可用的最长时间。有关更多信息，请参阅 Amazon SQS 短轮询和长轮询。
- 启用基于内容的重复数据删除 — Amazon SQS 可以 IDs 根据消息正文自动创建重复数据删除。有关更多信息，请参阅 Amazon SQS FIFO 队列。
- 启用高吞吐量 FIFO - 用于为队列中的消息启用高吞吐量。选择此选项会将相关选项（重复数据删除范围和 FIFO 吞吐量限制）更改为为 FIFO 队列启用高吞吐量所需的设置。有关更多信息，请参阅Amazon SQS 中 FIFO 队列的高吞吐量和Amazon SQS 消息配额。
- 重新驱动允许策略：定义哪些源队列可以将此队列用作死信队列。有关更多信息，请参阅 在 Amazon SQS 中使用死信队列。

为现有队列配置队列参数（控制台）

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。选择队列并选择编辑。
3. 滚动到配置部分。
4. 对于可见性超时，输入持续时间和单位。范围为 0 秒至 12 小时。默认值为 30 秒。
5. 对于消息保留期，输入持续时间和单位。范围为 1 分钟至 14 天。默认值为 4 天。
6. 对于标准队列，为接收消息等待时间输入一个值。范围为 0-20 秒。默认值为 0 秒，这会设置短轮询。任何非零值都会设置长轮询。
7. 对于传送延迟，输入持续时间和单位。范围为 0 秒到 15 分钟。默认值为 0 秒。
8. 对于最大消息大小，输入一个值。范围从 1 KiB 到 1024 KiB 不等。默认值为 1024 KiB。

9. 对于 FIFO 队列，选择启用基于内容的重复数据删除以启用基于内容的重复数据删除。默认情况下，该设置处于禁用状态。
10. （可选）要让 FIFO 队列启用更高的吞吐量以便在队列中发送和接收消息，请选择启用高吞吐量 FIFO。

    选择此选项会将相关选项（重复数据删除范围和 FIFO 吞吐量限制）更改为为 FIFO 队列启用高吞吐量所需的设置。如果更改使用高吞吐量 FIFO 所需的任何设置，则队列正常吞吐量生效，重复数据删除按规定进行。有关更多信息，请参阅Amazon SQS 中 FIFO 队列的高吞吐量和Amazon SQS 消息配额。
11. 对于重新驱动允许策略，请选择启用。从以下选项中选择：全部允许（默认）、按队列或全部拒绝。选择按队列时，按 Amazon 资源名称 (ARN) 指定最多 10 个源队列的列表。
12. 配置完队列参数后，选择保存。

在 Amazon SQS 中配置访问策略

编辑队列时，您可以配置其访问策略，以控制哪些用户可以与之交互。

- 访问策略定义了哪些账户、用户和角色具有访问该队列的权限。
- 它指定了允许的操作，例如 `SendMessage`、`ReceiveMessage` 或 `DeleteMessage`。
- 默认情况下，只有队列所有者有权发送和接收消息。

为现有队列配置访问策略（控制台）

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 选择队列并选择编辑。
4. 滚动到访问策略部分。
5. 在输入框中编辑访问策略语句。有关访问策略语句的更多信息，请参阅 Amazon SQS 中的身份和访问管理。
6. 配置完访问策略后，选择保存。

使用 SQS 托管的加密密钥为队列配置服务器端加密

除了默认 Amazon SQS 托管服务器端加密 (SSE) 选项外，Amazon SQS 托管 SSE (SSE-SQS) 还允许您创建自定义托管服务器端加密，以使用 SQS 托管加密密钥来保护通过消息队列发送的敏感数据。使用 SSE-SQS，您无需创建和管理加密密钥，也无需修改代码即可加密数据。SSE-SQS 可让您安全地传输数据，并帮助您满足严格的加密合规性和监管要求，而无需额外费用。

SSE-SQS 使用 256 位高级加密标准 (AES-256) 加密保护静态数据。一旦 Amazon SQS 收到消息，SSE 就会对消息进行加密。Amazon SQS 以加密形式存储消息，只有在将消息发送给授权使用者时才会将其解密。

> **Note**
>
> - 默认 SSE 选项仅在不指定加密属性的情况下创建队列时才有效。
> - Amazon SQS 允许您关闭所有队列加密。因此，关闭 KMS-SSE 不会自动启用 SQS-SSE。如果您希望在关闭 KMS-SSE 后启用 SQS-SSE，则必须在请求中添加属性更改。

为队列配置 SSE-SQS 加密（控制台）

> **Note**
>
> 默认情况下，使用 HTTP（非 TLS）端点创建的任何新队列都不会启用 SSE-SQS 加密。使用 HTTPS 或 Signature Version 4 端点创建 Amazon SQS 队列是一种安全最佳实践。

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 选择队列，然后选择编辑。
4. 展开加密。
5. 对于服务器端加密，选择启用（默认）。

> **Note**
>
> 启用 SSE 后，对加密队列的匿名 `SendMessage` 和 `ReceiveMessage` 请求将被拒绝。Amazon SQS 安全最佳实践建议不要使用匿名请求。如果您想向 Amazon SQS 队列发送匿名请求，请确保禁用 SSE。

6. 选择 Amazon SQS 密钥 (SSE-SQS)。使用此选项不会产生额外费用。
7. 选择保存。

使用 Amazon SQS 控制台为队列配置服务器端加密

为了保护队列消息中的数据，Amazon SQS 默认为所有新创建的队列启用服务器端加密 (SSE)。Amazon SQS 与 Amazon Web Services Key Management Service (Amazon Web Services KMS) 集成，用于管理服务器端加密 (SSE) 的 KMS 密钥。有关使用 SSE 的信息，请参阅Amazon SQS 中的静态加密。

您分配给队列的 KMS 密钥必须具有密钥政策，其中包括所有有权使用该队列的主体的权限。有关信息，请参阅密钥管理。

如果您不是 KMS 密钥的拥有者，或者您登录的账户没有 `kms:ListAliases` 和 `kms:DescribeKey` 权限，则您无法在 Amazon SQS 控制台上查看有关 KMS 密钥的信息。要求 KMS 密钥的拥有者授予您这些权限。有关更多信息，请参阅密钥管理。

创建或编辑队列时，您可以配置 SSE-KMS。

为现有队列配置 SSE-KMS（控制台）

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 选择队列，然后选择编辑。
4. 展开加密。
5. 对于服务器端加密，选择启用（默认）。

> **Note**
>
> 启用 SSE 后，对加密队列的匿名 SendMessage 和 ReceiveMessage 请求将被拒绝。Amazon SQS 安全最佳实践建议不要使用匿名请求。如果您想向 Amazon SQS 队列发送匿名请求，请确保禁用 SSE。

6. 选择 AWS Key Management Service 密钥 (SSE-KMS)。

   控制台会显示 KMS 密钥的描述、账户和 KMS 密钥 ARN。

7. 为队列指定 KMS 密钥 ID。有关更多信息，请参阅 关键术语。

   a. 选择选择 KMS 密钥别名选项。

   b. 默认密钥是 Amazon SQS 的 Amazon Web Services 托管 KMS 密钥。要使用此密钥，请从 KMS 密钥列表中选择它。

   c. 要使用您 Amazon Web Services 账户中的自定义 KMS 密钥，请从 KMS 密钥列表中选择该密钥。有关创建自定义 KMS 密钥的说明，请参阅《Amazon Web Services Key Management Service 开发人员指南》中的创建密钥。

   d. 要使用不在列表中的自定义 KMS 密钥或来自其他 Amazon Web Services 账户的自定义 KMS 密钥，请选择输入 KMS 密钥别名，并输入 KMS 密钥 Amazon 资源名称 (ARN)。

8. （可选）对于数据密钥重用周期，指定一个介于 1 分钟到 24 小时之间的值。默认值为 5 分钟。有关更多信息，请参阅 了解数据密钥重用周期。

9. 配置完 SSE-KMS 后，选择保存。

使用 Amazon SQS 控制台为队列配置成本分配标签

您可以向 Amazon SQS 队列添加成本分配标签来帮助组织和标识这些队列。有关更多信息，请参阅 Amazon SQS 成本分配标签。

- “详情”页面上的“标记”选项卡会显示队列的标签。
- 在创建或编辑队列时，您可以添加或修改标签。

为现有队列配置标签（控制台）

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。

3. 选择队列并选择编辑。
4. 滚动到标签部分。
5. 添加、修改或删除队列标签：

   a. 要添加标签，请选择添加新标签，输入键和值，然后选择添加新标签。

   b. 要更新标签，请更改其键和值。

   c. 要删除标签，请选择键值对旁边的删除。
6. 配置完标签后，选择保存。

使用 Amazon SQS 控制台为队列订阅 Amazon SNS 主题

您可以为一个或多个 Amazon SQS 队列订阅 Amazon SNS 主题。当您将消息发布到主题时，Amazon SNS 将向每个已订阅队列发送此消息。Amazon SQS 负责管理订阅并处理所需的权限。有关 Amazon SNS 的更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的什么是 Amazon SNS？

当您为 Amazon SQS 队列订阅 Amazon SNS 主题时，Amazon SNS 会使用 HTTPS 将消息转发给 Amazon SQS。有关将 Amazon SNS 用于加密的 Amazon SQS 队列的信息，请参阅配置 KMS 权限 AWS 务。

> **Important**
>
> Amazon SQS 支持每个访问策略最多 20 个语句。订阅 Amazon SNS 主题会添加这样一个语句。超过此数量将导致主题订阅交付失败。

为队列订阅 Amazon SNS 主题（控制台）

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 从队列列表中，选择 queue（排队）以订阅 Amazon SNS 主题。
4. 从 Actions（操作）中，选择 Subscribe to Amazon SNS topic（订阅 Amazon SNS 主题）。
5. 从指定可用于此队列的 Amazon SNS 主题菜单中，为队列选择 Amazon SNS 主题。

   如果 SNS 主题未列出，请选择输入 Amazon SNS 主题 ARN，然后输入主题的 Amazon 资源名称（ARN）。

6. 选择保存。
7. 要验证订阅，请向主题发布消息，并在队列中查看消息。有关更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的 Amazon SNS 消息发布。

跨账户订阅

如果您的 Amazon SQS 队列和 Amazon SNS 主题 AWS 账户不同，则需要额外的权限。

主题所有者（账户 A）

修改亚马逊 SNS 主题的访问策略以允许亚马逊 SQS 队列 AWS 账户 进行订阅。策略语句示例：

```
{
  "Effect": "Allow",
  "Principal": { "AWS": "arn:aws:iam::111122223333:root" },
  "Action": "sns:Subscribe",
  "Resource": "arn:aws:sns:us-east-1:123456789012:MyTopic"
}
```

此策略允许账户 111122223333 订阅 MyTopic。

队列所有者（账户 B）

修改 Amazon SQS 队列的访问策略以允许 Amazon SNS 主题发送消息。策略语句示例：

```
{
  "Effect": "Allow",
  "Principal": { "Service": "sns.amazonaws.com" },
  "Action": "sqs:SendMessage",
  "Resource": "arn:aws:sqs:us-east-1:111122223333:MyQueue",
  "Condition": {
    "ArnEquals": { "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012:MyTopic" }
  }
}
```

此策略允许 MyTopic 向 MyQueue 发送消息。

跨区域订阅

要订阅其他主题中的 Amazon SNS 主题 AWS 区域，请确保：

- Amazon SNS 主题的访问策略允许跨区域订阅。
- Amazon SQS 队列的访问策略允许 Amazon SNS 主题跨区域发送消息。

有关更多信息，请参阅亚马逊简单通知服务开发者指南中的向不同区域的 Amazon SQS 队列 AWS Lambda 或函数发送 Amazon SNS 消息。

配置 Amazon SQS 队列以触发函数 AWS Lambda

您可以使用 Lambda 函数处理来自 Amazon SQS 队列的消息。Lambda 会轮询队列并同步调用您的函数，将一批消息作为事件传递。

配置可见性超时

将队列可见性超时设置为函数超时的至少六倍。这可以确保在函数处理前一批任务期间被节流时，Lambda 有足够的时间进行重试。

使用死信队列（DLQ）

指定一个死信队列，用于捕获 Lambda 函数无法处理的消息。

处理多个队列和函数

Lambda 函数可通过为每个队列创建独立的事件源来处理多个队列。您还可以将多个 Lambda 函数关联到同一队列。

加密队列的权限

如果您将加密队列与 Lambda 函数相关联，但 Lambda 不轮询消息，请将 `kms:Decrypt` 权限添加到您的 Lambda 执行角色中。

限制

队列和 Lambda 函数必须位于同一 AWS 区域。

使用默认密钥（Amazon SQS 的AWS 托管 KMS 密钥）的加密队列无法在其他队列中调用 Lambda 函数。 AWS 账户

有关实施详细信息，请参阅《AWS Lambda 开发人员指南》中的将 AWS Lambda 与 Amazon SQS 结合使用。

先决条件

要配置 Lambda 函数触发器，您必须满足以下要求：

- 如果您使用用户，则您的 Amazon SQS 角色必须包括以下权限：
  - `lambda:CreateEventSourceMapping`
  - `lambda:ListEventSourceMappings`
  - `lambda:ListFunctions`
- Lambda 执行角色必须包括以下权限：
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `sqs:ReceiveMessage`
- 如果您将加密队列与 Lambda 函数相关联，请将 `kms:Decrypt` 权限添加到 Lambda 执行角色中。

有关更多信息，请参阅 管理 Amazon SQS 中的访问权限概述。

配置队列以触发 Lambda 函数（控制台）

1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 在队列页面上，选择要配置的队列。
4. 在队列页面上，选择 Lambda 触发器选项卡。
5. 在 Lambda 触发器页面上，选择 Lambda 触发器。

   如果列表中没有您需要的 Lambda 触发器，请选择配置 Lambda 函数触发器。输入 Lambda 函数的 Amazon 资源名称 (ARN)，或选择现有资源。然后选择保存。
6. 选择保存。控制台会保存配置，并显示队列的详细信息页面。

   在详细信息页面上，Lambda 触发器选项卡显示 Lambda 函数及其状态。Lambda 函数大约需要 1 分钟时间与队列关联。
7. 要验证配置的结果，请向您的队列发送消息，然后在 Lambda 控制台中查看触发的 Lambda 函数。

使用亚马逊自动将 AWS 服务发送到亚马逊 SQS 的通知 EventBridge

Amazon EventBridge 允许您近乎实时地自动处理 AWS 服务 和响应事件，例如应用程序问题或资源更改。

- 您可以创建规则来筛选特定事件，并在规则与事件匹配时定义自动化操作。
- EventBridge 支持多个目标，包括接收 JSON 格式事件的 Amazon SQS 标准队列和 FIFO 队列。

有关更多信息，请参阅《亚马逊 EventBridge 用户指南》中的亚马逊 EventBridge 目标。

使用 Amazon SQS 发送带有属性的消息

对于标准队列和 FIFO 队列，您可以在消息中添加结构化元数据，包括时间戳、地理空间数据、签名和标识符。有关更多信息，请参阅 Amazon SQS 消息属性。

使用 Amazon SQS 控制台向队列发送带有属性的消息

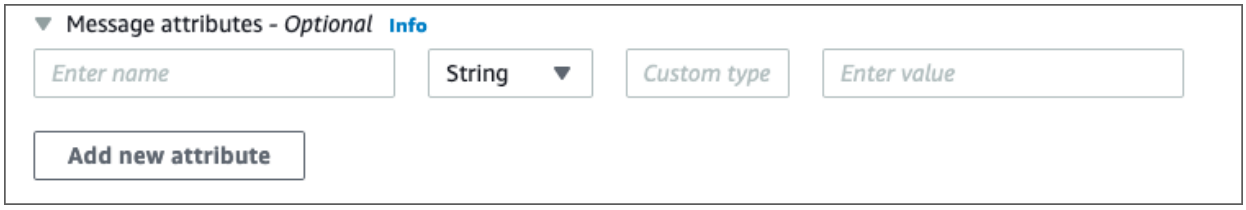
1. 打开 Amazon SQS 控制台，网址为。https://console.aws.amazon.com/sqs/
2. 在导航窗格中，选择队列。
3. 在队列页面，选择队列。
4. 选择发送和接收消息。
5. 输入消息属性参数。

   a. 在名称文本框中，输入最多 256 个字符的唯一名称。

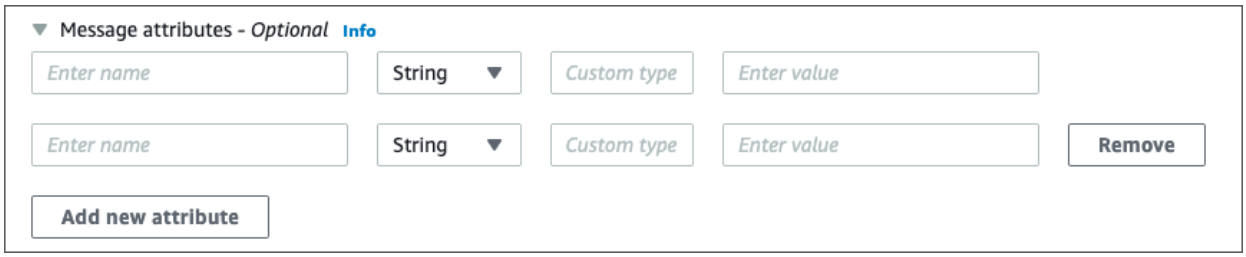
   b. 对于属性类型，选择字符串、数字或二进制。

   c. （可选）输入自定义数据类型。例如，您可以添加 **`byte`**、**`int`** 或 **`float`** 作为数字的自定义数据类型。

   d. 在值文本框中，输入消息属性值。



6. 要添加其他消息属性，请选择添加新属性。



7. 您可在发送消息之前随时修改属性值。
8. 要删除属性，请选择删除。要删除第一个属性，请关闭消息属性。
9. 完成将属性添加到消息之后，选择发送消息。发送消息后，控制台会显示一条成功消息。要查看有关已发送消息的消息属性的信息，请选择查看详细信息。选择完成，以关闭消息详细信息对话框。

Amazon SQS 最佳实践

Amazon SQS 可以管理和处理消息队列，使应用程序的不同部分能够可靠且大规模地交换消息。本主题提供了关键的操作最佳实践，包括使用长轮询来减少空响应、实施死信队列来处理错误，以及优化队列权限以确保安全。

主题

- Amazon SQS 错误处理和有问题的消息
- Amazon SQS 消息重复数据删除和分组
- Amazon SQS 消息处理和定时

Amazon SQS 错误处理和有问题的消息

本主题提供有关管理和缓解 Amazon SQS 中的错误的详细说明，包括处理请求错误、捕获有问题的消息以及配置死信队列保留时间以确保消息可靠性的技术。

主题

- 处理 Amazon SQS 中的请求错误
- 捕获 Amazon SQS 中有问题的消息
- 在 Amazon SQS 中设置死信队列保留时间

处理 Amazon SQS 中的请求错误

要处理请求错误，请使用下列策略之一：

- 如果您使用 AWS SDK，则已经有自动重试和退避逻辑可供您使用。有关更多信息，请参阅《Amazon Web Services 一般参考》中的 AWS中的错误重试和指数回退。
- 如果您不使用 AWS 软件开发工具包功能进行重试和退避，则在未收到来自 Amazon SQS 的消息、超时或错误消息后，请允许暂停（例如 200 毫秒），然后重试ReceiveMessage操作。对于将产生相同结果的 `ReceiveMessage` 的后续使用，应允许更长的暂停时间（例如 400 毫秒）。

捕获 Amazon SQS 中有问题的消息

要捕获所有无法处理的消息并收集准确的 CloudWatch 指标，请配置死信队列。

- 在源队列无法将消息处理指定次数后，重新驱动策略会将消息重定向到死信队列。
- 使用死信队列将减少消息数并减小向您公开毒丸消息（可接收但无法处理的消息）的几率。
- 在队列中加入毒丸消息可能会给出错误的毒丸消息的年龄，从而扭曲ApproximateAgeOfOldestMessage CloudWatch 指标。配置死信队列有助于避免在使用此指标时发出错误警报。

在 Amazon SQS 中设置死信队列保留时间

对于标准队列，消息的到期时间始终基于其原始入队时间戳。将消息移至死信队列时，入队时间戳保持不变。ApproximateAgeOfOldestMessage 指标指示消息何时移入死信队列，而不是消息最初发送的时间。例如，假设一条消息在原始队列中停留了 1 天，然后才移至死信队列。如果死信队列的保留期为 4 天，则消息将在 3 天后从死信队列中删除，且 ApproximateAgeOfOldestMessage 为 3 天。因此，最佳实践是始终将死信队列的保留期设置为比原始队列的保留期长。

对于 FIFO 队列，当消息移到死信队列时，入队时间戳会重置。ApproximateAgeOfOldestMessage 指标指示消息何时移动到死信队列。在上面的同一个示例中，消息在 4 天后从死信队列中删除，且 ApproximateAgeOfOldestMessage 为 4 天。

Amazon SQS 消息重复数据删除和分组

本主题提供了可确保在 Amazon SQS 中实现一致消息处理的最佳实践，并说明了如何使用：

- MessageDeduplicationId 防止 FIFO 队列中出现重复消息。
- MessageGroupId 管理不同消息组内的消息排序。

主题

- 避免 Amazon SQS 中出现不一致的消息处理
- 在 Amazon SQS 中使用消息重复数据删除 ID
- 结合使用消息组 ID 和 Amazon SQS FIFO 队列
- 使用 Amazon SQS 接收请求尝试 ID

避免 Amazon SQS 中出现不一致的消息处理

由于 Amazon SQS 是一个分布式系统，因此，即使在从 ReceiveMessage API 方法调用成功返回时 Amazon SQS 将消息标记为已发送，使用者也可能不会收到消息。在这种情况下，Amazon SQS 将消

息记录为至少已发送一次，即使使用者从未收到过也是如此。由于在这些情况下不会再尝试发送消息，因此我们不建议将死信队列的最大接收数量设置为 1。

在 Amazon SQS 中使用消息重复数据删除 ID

`MessageDeduplicationId` 是一种令牌，仅用于防止在 Amazon SQS FIFO 队列中重复传递消息。它可以确保在 5 分钟的重复数据删除时段内，仅处理和传送具有相同重复数据删除 ID 的消息的一个实例。

如果 Amazon SQS 已接受具有特定重复数据删除 ID 的消息，则任何具有相同 ID 的后续消息都将被确认，但不会传送给使用者。

> **Note**
>
> 即使消息已被接收并删除，Amazon SQS 仍会继续跟踪重复数据删除 ID。

主题

- 在 Amazon SQS 中提供消息重复数据删除 ID
- 在 Amazon SQS 为单个创建者/使用者系统启用重复数据删除
- Amazon SQS 中的中断恢复场景
- 在 Amazon SQS 中配置可见性超时

在 Amazon SQS 中提供消息重复数据删除 ID

创建者应在以下场景中指定消息重复数据删除 ID：

- 发送相同消息正文但必须将其视为唯一项时。
- 发送内容相同但消息属性不同的消息时，确保对每条消息进行独立处理。
- 发送内容不同（例如消息正文包含重试计数器）但要求 Amazon SQS 将其识别为重复项的消息时。

在 Amazon SQS 为单个创建者/使用者系统启用重复数据删除

如果您有单一的创建者和单一的使用者，且消息因包含特定于应用程序的消息 ID 而具有唯一性，请遵循以下最佳实践：

- 为队列启用基于内容的重复数据删除（每条消息都具有唯一的正文）。创建者可忽略消息重复数据删除 ID。

- 如果您为 Amazon SQS FIFO 队列启用了基于内容的重复数据删除，并且发送了具有重复数据删除 ID 的消息，则 SendMessage 重复数据删除 ID 将覆盖生成的基于内容的重复数据删除 ID。
- 尽管使用者无需为每个请求提供接收请求尝试 ID，但最好提供，因为这样可以更快地执行失败-重试序列。
- 请求不会干扰消息在 FIFO 队列中的顺序，因此可重试发送或接收请求。

Amazon SQS 中的中断恢复场景

FIFO 队列中的重复数据删除过程具有时效性。在设计应用程序时，请确保创建者和使用者均可在客户端或网络中断时恢复运行，而不会引入重复数据或出现处理失败。

创建者注意事项

- Amazon SQS 强制执行 5 分钟的重复数据删除时段。
- 如果创建者在 5 分钟后重试 SendMessage 请求，Amazon SQS 会将其视为新消息，这可能导致消息重复。

使用者注意事项

- 如果使用者未能在可见性超时结束前处理消息，其他使用者可能会接收并处理该消息，从而导致重复处理。
- 根据应用程序的处理时间调整可见性超时。
- 使用 ChangeMessageVisibility API 在消息仍在处理时延长超时。
- 如果消息反复处理失败，请将其路由至死信队列（DLQ），而不是允许无限期地对其进行重新处理。

- 创建者必须了解队列的重复数据删除间隔。Amazon SQS 的重复数据删除间隔为 5 分钟。在重复数据删除时间间隔过期后重试 SendMessage 请求可能会将重复的消息引入队列中。例如，车辆中的移动设备将发送其顺序很重要的消息。如果车辆在接收确认前一段时间失去手机网络连接，则在重新获得手机网络连接之前重试请求可能产生重复项。
- 使用者必须具有可见性超时，以便将在可见性超时过期之前无法处理消息的风险降至最低。您可通过调用 ChangeMessageVisibility 操作延长处理消息时的可见性超时。但是，如果可见性超时过期，其他使用者可立即开始处理消息，从而导致多次处理消息。要避免这种情况，请配置死信队列。

在 Amazon SQS 中配置可见性超时

为确保可靠的消息处理，请将可见性超时设置为比 AWS SDK 读取超时长。这一设置适用于使用 ReceiveMessage API 进行短轮询和长轮询的情况。较长的可见性超时可防止消息在原始请求完成之前提供给其他使用者，从而降低重复处理的风险。

结合使用消息组 ID 和 Amazon SQS FIFO 队列

在 FIFO（先进先出）队列中，MessageGroupId 是一个将消息组织成不同组的属性。同一消息组内的消息始终按照严格的顺序逐条处理，从而确保同一组内绝不会有两条消息同时被处理。在标准队列中，使用 MessageGroupId 启用公平队列。如果需要严格排序，请使用 FIFO 队列。

主题

- 在 Amazon SQS 中交错多个有序消息组
- 在 Amazon SQS 的多创建者/使用者系统中避免处理重复消息
- 避免在 Amazon SQS 中出现具有相同消息组 ID 的大量消息积压
- 避免在 Amazon SQS 中的虚拟队列中重复使用相同的消息组 ID

在 Amazon SQS 中交错多个有序消息组

要在交错一个 FIFO 队列中的多个有序消息组，请为每个组分配一个 MessageGroupId（例如，不同用户的会话数据）。这可以让多个使用者能够同时从队列中读取消息，同时确保同一组内的消息按顺序处理。

当具有特定 MessageGroupId 的消息正在进行处理且不可见时，任何其他使用者均无法处理来自同一组的消息，直至可见性超时结束或消息被删除。

在 Amazon SQS 的多创建者/使用者系统中避免处理重复消息

在一个高吞吐量、低延迟且消息顺序并非优先事项的系统中，创建者可为每条消息分配唯一的 MessageGroupId。这可以确保即使在多创建者/多使用者设置中，Amazon SQS FIFO 队列也能消除重复项。虽然这种方法可以防止重复的消息，但无法保证消息的顺序，因为每条消息都被视为自己的独立组。

在任何具有多个创建者和使用者的系统中，都存在重复传递消息的风险。如果使用者未能在可见性超时结束前处理消息，Amazon SQS 会使该消息再次可用，从而导致其他使用者可能会接收该消息。为了缓解这个问题，请确保根据处理时间正确设置消息确认和可见性超时。

避免在 Amazon SQS 中出现具有相同消息组 ID 的大量消息积压

FIFO 队列最多支持 12 万条传输中消息（使用者已接收但尚未删除的消息）。如果达到此上限，Amazon SQS 不会返回错误，但处理可能会受到影响。您可以联系 AWS Support 申请提高此上限。

FIFO 队列会浏览前 12 万条消息，以确定可用的消息组。如果单个消息组中积压了大量消息，则在积压消息处理完毕之前，其他组后续发送的消息将持续被阻止。

> **Note**
>
> 当使用者反复多次处理失败时，就可能会导致消息积压。这可能是由于消息内容问题或使用者端故障所致。为防止消息处理延迟，请配置死信队列，用于在多次处理失败后转移未处理的消息。这样可以确保同一消息组中的其他消息得到处理，从而避免系统出现瓶颈。

避免在 Amazon SQS 中的虚拟队列中重复使用相同的消息组 ID

将虚拟队列与共享主机队列一起使用时，请避免在不同的虚拟队列中重复使用相同的 `MessageGroupId`。如果多个虚拟队列共享同一主机队列且包含具有相同 `MessageGroupId` 的消息，这些消息可能会相互阻塞，以至阻碍高效处理。为确保顺利处理消息，请为不同虚拟队列中的消息分配唯一的 `MessageGroupId` 值。

使用 Amazon SQS 接收请求尝试 ID

接收请求尝试 ID 是 Amazon SQS 中用于删除 `ReceiveMessage` 重复调用的唯一令牌。当您的应用程序与 Amazon SQS 之间出现网络中断或连接问题时，最佳实践是：

- 在调用 `ReceiveMessage` 时提供接收请求尝试 ID。
- 如果操作失败，请使用相同的接收请求尝试 ID 重试。

Amazon SQS 消息处理和定时

本主题提供了有关优化 Amazon SQS 中消息处理速度和效率的全面指导，包括及时处理消息、选择最佳轮询模式以及配置长轮询以提高性能的策略。

主题

- 在 Amazon SQS 中及时处理消息

- 在 Amazon SQS 中设置长轮询
- 在 Amazon SQS 中使用合适的轮询模式

在 Amazon SQS 中及时处理消息

可见性超时设置取决于您的应用程序需要多长时间来处理和删除消息。例如，如果您的应用程序处理一条消息需要花费 10 秒，并且您将可见性超时设置为 15 分钟，则在上一次处理尝试失败的情况下，您必须等待一个相对较长的时间才能再次尝试处理消息。或者，如果您的应用程序处理一条消息需要花费 10 秒，但您将可见性超时仅设置为 2 秒，则当原始使用者仍在处理消息时，另一个使用者会收到重复消息。

要确保有足够的时间处理消息，请使用下列策略之一：

- 如果您知道（或者可以合理地估计）处理消息所需的时间，则将消息的可见性超时 延长至处理消息所需的最长时间并删除消息。有关更多信息，请参阅配置可见性超时。
- 如果您不知道处理消息需要多长时间，请为您的使用者流程创建检测信号：指定初始可见性超时（例如 2 分钟），然后（只要您的使用者仍在处理该消息），就可以每分钟将可见性超时延长 2 分钟。

> **Important**
>
> 最大可见性超时为从 Amazon SQS 收到 `ReceiveMessage` 请求时起 12 小时。延长可见性超时不会重置 12 小时的最大值。
> 此外，您可能无法将单个消息的超时时间设置为 `ReceiveMessage` 请求启动计时器后的整整 12 小时（即 43200 秒）。例如，如果您收到一条消息，并立即通过发送 `ChangeMessageVisibility` 调用，将 `VisibilityTimeout` 设置为 43200 秒，以设置 12 小时的最大值，则该消息很可能会失败。但是，除非通过 `ReceiveMessage` 请求消息和更新可见性超时之间存在明显的延迟，否则使用 43195 秒这个值会起到作用。如果您的使用者需要超过 12 小时，请考虑使用 Step Functions。

在 Amazon SQS 中设置长轮询

当 `ReceiveMessage` API 操作的等待时间大于 0 时，长轮询生效。最长长轮询等待时间为 20 秒。长轮询通过减少空响应（当`ReceiveMessage`请求没有消息可用时）和虚假空响应（当消息可用但未包含在响应中时）的数量来帮助降低使用 Amazon SQS 的成本。有关更多信息，请参阅 Amazon SQS 短轮询和长轮询。

要确保最佳消息处理，请使用以下策略：

- 在大多数情况下，您可以将 ReceiveMessage 等待时间设置为 20 秒。如果 20 秒对您的应用程序来说太长，则可设置较短的 ReceiveMessage 等待时间（最少 1 秒）。如果您不使用 AWS 软件开发工具包来访问 Amazon SQS，或者将 AWS 软件开发工具包配置为缩短等待时间，则可能需要修改您的 Amazon SQS 客户端，以允许更长的请求或使用更短的等待时间进行长时间轮询。
- 如果您对多个队列实施长轮询，则对每个队列使用一个线程，而不是对所有队列使用单个线程。如果对每个队列使用一个线程，您的应用程序将能够在各队列中的消息可用时处理这些消息；如果使用单个线程来轮询多个队列，则可能导致应用程序在等待（最长 20 秒）没有任何可用消息的队列时无法处理其他队列中的可用消息。

> **Important**
>
> 为避免 HTTP 错误，请确保 ReceiveMessage 请求的 HTTP 响应超时比 WaitTimeSeconds 参数更长。有关更多信息，请参阅 ReceiveMessage。

在 Amazon SQS 中使用合适的轮询模式

- 长轮询使您能够在消息可用后立即从 Amazon SQS 队列中使用消息。
  - 要降低使用 Amazon SQS 的成本并减少空队列的空接收次数（对不返回任何消息的 ReceiveMessage 操作的响应次数），请启用长轮询。有关更多新信息，请参阅 Amazon SQS 长轮询。
  - 要提高轮询具有多次接收的多个线程的效率，请减少线程数。
  - 在大多数情况下，长轮询优于短轮询。
- 短轮询会立即返回响应，即使轮询的 Amazon SQS 队列为空。
  - 要满足应立即响应 ReceiveMessage 请求的应用程序的要求，请使用短轮询。
  - 短轮询的计费与长轮询相同。

Amazon SQS Java SDK 示例

适用于 Java 的 AWS SDK 支持您构建与 Amazon SQS 及其他 AWS 服务交互的 Java 应用程序。

- 要安装和设置 SDK，请参阅《AWS SDK for Java 2.x 开发人员指南》中的开始使用。
- 有关基本队列操作（例如创建队列或发送消息），请参阅《AWS SDK for Java 2.x 开发人员指南》中的使用 Amazon SQS 消息队列。
- 该指南还包括其他 Amazon SQS 功能的示例，例如：
  - 在 Amazon SQS 队列中使用服务器端加密
  - 为 Amazon SQS 队列配置标签
  - 将消息属性发送到 Amazon SQS 队列

在 Amazon SQS 队列中使用服务器端加密

使用 适用于 Java 的 AWS SDK 将服务器端加密 (SSE) 添加到 Amazon SQS 队列。每个队列都使用 AWS Key Management Service (AWS KMS) KMS 密钥生成数据加密密钥。此示例设置 AWS 托管 KMS 密钥用于 Amazon SQS。

有关使用 SSE 和 KMS 密钥角色的更多信息，请参阅Amazon SQS 中的静态加密。

向现有队列添加 SSE

要为现有队列启用服务器端加密，请使用 `SetQueueAttributes` 方法设置 `KmsMasterKeyId` 属性。

以下代码示例将设置 AWS KMS key 为 Amazon SQS 的 AWS 托管 KMS 密钥。该示例还会将 AWS KMS key 重用周期设置为 140 秒。

在运行示例代码之前，请确保已设置 AWS 凭据。有关更多信息，请参阅《AWS SDK for Java 2.x 开发人员指南》中的设置开发 AWS 凭证和区域。

```
    public static void addEncryption(String queueName, String kmsMasterKeyAlias) {
        SqsClient sqsClient = SqsClient.create();

        GetQueueUrlRequest urlRequest = GetQueueUrlRequest.builder()
                .queueName(queueName)
                .build();
```

```
        GetQueueUrlResponse getQueueUrlResponse;
        try {
            getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest);
        } catch (QueueDoesNotExistException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        }
        String queueUrl = getQueueUrlResponse.queueUrl();


        Map<QueueAttributeName, String> attributes = Map.of(
                QueueAttributeName.KMS_MASTER_KEY_ID, kmsMasterKeyAlias,
                QueueAttributeName.KMS_DATA_KEY_REUSE_PERIOD_SECONDS, "140" // Set the
data key reuse period to 140 seconds.
        );                                                                  // This is
how long SQS can reuse the data key before requesting a new one from KMS.

        SetQueueAttributesRequest attRequest = SetQueueAttributesRequest.builder()
                .queueUrl(queueUrl)
                .attributes(attributes)
                .build();
        try {
            sqsClient.setQueueAttributes(attRequest);
            LOGGER.info("The attributes have been applied to {}", queueName);
        } catch (InvalidAttributeNameException | InvalidAttributeValueException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        } finally {
            sqsClient.close();
        }
    }
```

为队列禁用 SSE

要为现有队列禁用服务器端加密，请使用 SetQueueAttributes 方法将 KmsMasterKeyId 属性设置为空字符串。

> **Important**
>
> null 对于 KmsMasterKeyId 是无效值。

使用 SSE 创建队列

要在创建队列时启用 SSE，请将 KmsMasterKeyId 属性添加到 CreateQueue API 方法中。

以下示例将在启用了 SSE 的情况下创建新队列。队列将 AWS 托管 KMS 密钥用于 Amazon SQS。该示例还会将 AWS KMS key 重用周期设置为 160 秒。

在运行示例代码之前，请确保已设置 AWS 凭据。有关更多信息，请参阅《AWS SDK for Java 2.x 开发人员指南》中的设置开发 AWS 凭证和区域。

```
// Create an SqsClient for the specified Region.
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Create a hashmap for the attributes. Add the key alias and reuse period to the
 hashmap.
HashMap<QueueAttributeName, String> attributes = new HashMap<QueueAttributeName,
 String>();
final String kmsMasterKeyAlias = "alias/aws/sqs";  // the alias of the AWS managed KMS
 key for Amazon SQS.
attributes.put(QueueAttributeName.KMS_MASTER_KEY_ID, kmsMasterKeyAlias);
attributes.put(QueueAttributeName.KMS_DATA_KEY_REUSE_PERIOD_SECONDS, "140");

// Add the attributes to the CreateQueueRequest.
CreateQueueRequest createQueueRequest =
                CreateQueueRequest.builder()
                        .queueName(queueName)
                        .attributes(attributes)
                        .build();
sqsClient.createQueue(createQueueRequest);
```

检索 SSE 属性

有关检索队列属性的信息，请参阅 Amazon Simple Queue Service API 参考中的示例。

要检索特定队列的 KMS 密钥 ID 或数据密钥重用周期，请运行 GetQueueAttributes 方法并检索 KmsMasterKeyId 和 KmsDataKeyReusePeriodSeconds 值。

为 Amazon SQS 队列配置标签

使用 cost-allocation 标签来帮助组织和标识 Amazon SQS 队列。以下示例演示如何使用 适用于 Java 的 AWS SDK来配置标签。有关更多信息，请参阅 Amazon SQS 成本分配标签。

在运行示例代码之前，请确保已设置 AWS 凭据。有关更多信息，请参阅《AWS SDK for Java 2.x 开发人员指南》中的设置开发 AWS 凭证和区域。

列出标签

要列出队列的标签，请使用 ListQueueTags 方法。

```
// Create an SqsClient for the specified region.
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Get the queue URL.
String queueName = "MyStandardQ1";
GetQueueUrlResponse getQueueUrlResponse =

 sqsClient.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
String queueUrl = getQueueUrlResponse.queueUrl();

// Create the ListQueueTagsRequest.
final ListQueueTagsRequest listQueueTagsRequest =

 ListQueueTagsRequest.builder().queueUrl(queueUrl).build();

// Retrieve the list of queue tags and print them.
final ListQueueTagsResponse listQueueTagsResponse =
                                    sqsClient.listQueueTags(listQueueTagsRequest);
System.out.println(String.format("ListQueueTags: \tTags for queue %s are %s.\n",
                queueName, listQueueTagsResponse.tags() ));
```

添加或更新标签

要为队列添加或更新标签值，请使用 TagQueue 方法。

```
 // Create an SqsClient for the specified Region.
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Get the queue URL.
String queueName = "MyStandardQ1";
GetQueueUrlResponse getQueueUrlResponse =

 sqsClient.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
String queueUrl = getQueueUrlResponse.queueUrl();
```

```
// Build a hashmap of the tags.
final HashMap<String, String> addedTags = new HashMap<>();
        addedTags.put("Team", "Development");
        addedTags.put("Priority", "Beta");
        addedTags.put("Accounting ID", "456def");

//Create the TagQueueRequest and add them to the queue.
final TagQueueRequest tagQueueRequest = TagQueueRequest.builder()
        .queueUrl(queueUrl)
        .tags(addedTags)
        .build();
sqsClient.tagQueue(tagQueueRequest);
```

移除标签

要从队列中删除一个或多个标签，请使用 `UntagQueue` 方法。以下示例将删除 `Accounting ID` 标签。

```
// Create the UntagQueueRequest.
final UntagQueueRequest untagQueueRequest = UntagQueueRequest.builder()
        .queueUrl(queueUrl)
        .tagKeys("Accounting ID")
        .build();

// Remove the tag from this queue.
sqsClient.untagQueue(untagQueueRequest);
```

将消息属性发送到 Amazon SQS 队列

您可以使用消息属性，在消息中包括结构化元数据（如时间戳、地理空间数据、签名和标识符）。有关更多信息，请参阅 Amazon SQS 消息属性。

在运行示例代码之前，请确保已设置 AWS 凭据。有关更多信息，请参阅《AWS SDK for Java 2.x 开发人员指南》中的设置开发 AWS 凭证和区域。

定义属性

要为消息定义属性，请添加使用 `MessageAttributeValue` 数据类型的以下代码。有关更多信息，请参阅消息属性组件和消息属性数据类型。

会 适用于 Java 的 AWS SDK 自动计算消息正文和消息属性的校验和，并将它们与 Amazon SQS 返回的数据进行比较。有关更多信息，请参阅 AWS SDK for Java 2.x 开发人员指南；有关其他编程语言，请参阅计算消息属性的 MD5 消息摘要。

String

此示例定义 `String` 属性，其名称为 `Name`，值为 `Jane`。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("Name", new MessageAttributeValue()
.withDataType("String")
.withStringValue("Jane"));
```

Number

此示例定义 `Number` 属性，其名称为 `AccurateWeight`，值为 `230.00000000000000001`。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccurateWeight", new MessageAttributeValue()
.withDataType("Number")
.withStringValue("230.00000000000000001"));
```

Binary

此示例定义 `Binary` 属性，其名称为 `ByteArray`，值为未初始化的 10 字节数组。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("ByteArray", new MessageAttributeValue()
.withDataType("Binary")
.withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

String (custom)

此示例定义自定义属性 `String.EmployeeId`，其名称为 `EmployeeId`，值为 `ABC123456`。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
```

```
messageAttributes.put("EmployeeId", new MessageAttributeValue()
.withDataType("String.EmployeeId")
.withStringValue("ABC123456"));
```

Number (custom)

此示例定义自定义属性 Number.AccountId，其名称为 AccountId，值为 000123456。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccountId", new MessageAttributeValue()
.withDataType("Number.AccountId")
.withStringValue("000123456"));
```

> **Note**
>
> 由于基本数据类型为 Number，ReceiveMessage 方法会返回 123456。

Binary (custom)

此示例定义自定义属性 Binary.JPEG，其名称为 ApplicationIcon，值为未初始化的 10 字节数组。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("ApplicationIcon", new MessageAttributeValue()
.withDataType("Binary.JPEG")
.withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

发送带有属性的消息

此示例在发送消息之前将属性添加到 SendMessageRequest 中。

```
// Send a message with an attribute.
final SendMessageRequest sendMessageRequest = new SendMessageRequest();
sendMessageRequest.withMessageBody("This is my message text.");
sendMessageRequest.withQueueUrl(myQueueUrl);
sendMessageRequest.withMessageAttributes(messageAttributes);
sqs.sendMessage(sendMessageRequest);
```

Important

如果您向 First-In-First-Out (FIFO) 队列发送消息，请确保在提供消息组 ID 后执行该`sendMessage`方法。
如果您使用 `SendMessageBatch` 方法而非 `SendMessage`，则必须指定批处理中每条消息的消息属性。

在 Amazon SQS 中使用 API

本主题提供有关构建 Amazon SQS 端点、通过 GET 和 POST 方法发出查询 API 请求以及使用批处理 API 操作的信息。有关 Amazon SQS 操作（包括参数、错误、示例和数据类型）的详细信息，请参阅 Amazon Simple Queue Service API 参考。

要使用各种编程语言访问 Amazon SQS，您还可以使用包含以下自动功能的 AWS SDK：

- 对服务请求进行加密签名
- 重试请求
- 处理错误响应

有关更多信息，请参阅 the section called “使用 AWS 软件开发工具包”。

有关命令行工具信息，请参阅 AWS CLI 命令参考和AWS Tools for PowerShell Cmdlet 参考中的 Amazon SQS 部分。

采用 JSON 协议的亚马逊 SQS API AWS

亚马逊 SQS 使用 AWS JSON 协议作为指定软件开发工具包版本上所有亚马逊 SQS API 的AWS 传输机制。 AWS JSON 协议提供了更高的吞吐量、更低的延迟和更快的应用程序到应用程序的通信。AWS 与 AWS 查询协议相比，JSON 协议在请求和响应方面的 serialization/deserialization 效率更高。如果您仍然倾向于在 SQS API 中使用 AWS 查询协议，亚马逊 SQS APIs 中使用的 AWS JSON 协议支持哪些语言？请参阅，了解支持 Amazon AWS S AWS QS 查询协议的软件开发工具包版本。

亚马逊 SQS 使用 AWS JSON 协议在 S AWS DK 客户端（例如 Java、Python、Golang 等 JavaScript）和亚马逊 SQS 服务器之间进行通信。Amazon SQS API 操作的 HTTP 请求接受 JSON 格式的输入。将执行 Amazon SQS 操作，并将执行响应以 JSON 格式发送回 SDK 客户端。与 AWS 查询相比， AWS JSON 更简单、更快速、更高效地在客户端和服务器之间传输数据。

- AWS JSON 协议充当 Amazon SQS 客户端和服务器之间的中介。
- 服务器不理解创建 Amazon SQS 操作所用的编程语言，但它能理解 AWS JSON 协议。
- AWS JSON 协议在 Amazon SQS 客户端和服务器之间使用序列化（将对象转换为 JSON 格式）和反序列化（将 JSON 格式转换为对象）。

有关使用 Amazon SQS 的 AWS JSON 协议的更多信息，请参阅。亚马逊 SQS AWS JSON 协议 FAQs

AWS JSON 协议适用于指定的 AWS SDK 版本。要查看不同语言变体的 SDK 版本和发布日期，请参阅《AWS SDK 和工具参考指南》中的 AWS SDK 和工具版本支持列表。

在 Amazon SQS 中使用 AWS JSON 协议发出查询 API 请求

本主题介绍了如何构造 Amazon SQS 端点、发出 POST 请求以及解释响应。

> **Note**
>
> AWS 大多数语言变体都支持 JSON 协议。有关受支持语言变体的完整列表，请参阅亚马逊 SQS APIs 中使用的 AWS JSON 协议支持哪些语言？。

构造端点

为了使用 Amazon SQS 队列，您必须构造一个端点。有关 Amazon SQS 端点的信息，请参阅 Amazon Web Services 一般参考中的以下页面：

- 区域端点
- Amazon Simple Queue Service 端点和配额

每个 Amazon SQS 端点都是独立的。例如，如果有两个名为 MyQueue 的队列，其中一个队列具有终端节点 `sqs.us-east-2.amazonaws.com`，另一个队列具有终端节点 `sqs.eu-west-2.amazonaws.com`，则这两个队列不会相互共享任何数据。

以下是一个提出创建队列请求的端点的示例。

```
POST / HTTP/1.1
Host: sqs.us-west-2.amazonaws.com
X-Amz-Target: AmazonSQS.CreateQueue
X-Amz-Date: <Date>
Content-Type: application/x-amz-json-1.0
Authorization: <AuthParams>
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
{
    "QueueName":"MyQueue",
    "Attributes": {
        "VisibilityTimeout": "40"
```

```
    },
    "tags": {
        "QueueType": "Production"
    }
}
```

> **Note**
>
> 队列名称和队 URLs 列区分大小写。
> *AUTHPARAMS* 的结构取决于 API 请求的签名。有关更多信息，请参阅《亚马逊 Web Services 一般参考》中的 “签署 AWS API 请求”。

发出 POST 请求

Amazon SQS POST 请求在 HTTP 请求的正文中以表单的形式发送查询参数。

以下是将 `X-Amz-Target` 设置为 `AmazonSQS.<operationName>` 的 HTTP 标头以及将 `Content-Type` 设置为 `application/x-amz-json-1.0` 的 HTTP 标头的示例。

```
POST / HTTP/1.1
Host: sqs.<region>.<domain>
X-Amz-Target: AmazonSQS.SendMessage
X-Amz-Date: <Date>
Content-Type: application/x-amz-json-1.0
Authorization: <AuthParams>
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
{
    "QueueUrl": "https://sqs.<region>.<domain>/<awsAccountId>/<queueName>/",
    "MessageBody": "This is a test message"
}
```

此 HTTP POST 请求将消息发送到 Amazon SQS 队列。

> **Note**
>
> HTTP 标头 `X-Amz-Target` 和 `Content-Type` 均为必需项。
> 根据客户端的 HTTP 版本，您的 HTTP 客户端可能会向 HTTP 请求添加其他项目。

解释 Amazon SQS JSON API 响应

当您向 Amazon SQS 发送请求时，它会返回包含结果的 JSON 响应。响应结构取决于您使用的 API 操作。

要了解这些响应的详细信息，请参阅：

- 《Amazon Simple Queue Service API 参考》的 API 操作中的特定 API 操作
- 这些区域有：亚马逊 SQS AWS JSON 协议 FAQs

成功的 JSON 响应结构

如果请求成功，则主响应元素为 `x-amzn-RequestId`，其中包含请求的通用唯一标识符 (UUID) 以及其他附加的响应字段。例如，以下 `CreateQueue` 响应包含 `QueueUrl` 字段，后者又包含所创建队列的 URL。

```
HTTP/1.1 200 OK
x-amzn-RequestId: <requestId>
Content-Length: <PayloadSizeBytes>
Date: <Date>
Content-Type: application/x-amz-json-1.0
{
    "QueueUrl":"https://sqs.us-east-1.amazonaws.com/111122223333/MyQueue"
}
```

JSON 错误响应结构

如果请求失败，则 Amazon SQS 将返回主响应，包括 HTTP 标头和正文。

在 HTTP 标头中，`x-amzn-RequestId` 包含请求的 UUID。`x-amzn-query-error` 包含两条信息：错误类型，以及错误是创建者错误还是使用者错误。

在响应正文中，`"__type"` 表示其他错误详细信息，`Message` 以可读的格式指明错误情况。

以下是 JSON 格式的错误响应示例：

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: 66916324-67ca-54bb-a410-3f567a7a0571
x-amzn-query-error: AWS.SimpleQueueService.NonExistentQueue;Sender
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

```
Content-Type: application/x-amz-json-1.0
{
    "__type": "com.amazonaws.sqs#QueueDoesNotExist",
    "message": "The specified queue does not exist."
}
```

亚马逊 SQS AWS JSON 协议 FAQs

本主题涵盖了有关在 Amazon SQS 中使用 AWS JSON 协议的常见问题。

什么是 AWS JSON 协议，它与现有的 Amazon SQS API 请求和响应有何不同？

JSON 是在异构系统之间进行通信时最广为使用和接受的连接方法之一。亚马逊 SQS 使用 JSON 作为媒介在 S AWS DK 客户端（例如 Java、Python、Golang JavaScript）和亚马逊 SQS 服务器之间进行通信。Amazon SQS API 操作的 HTTP 请求接受 JSON 形式的输入。系统会执行 Amazon SQS 操作，然后将执行的响应以 JSON 的形式反过来共享给 SDK 客户端。与 AWS 查询相比，JSON 在客户端和服务器之间的数据传输方面效率更高。

- 亚马逊 SQS AWS JSON 协议充当亚马逊 SQS 客户端和服务器之间的中介。
- 服务器不理解创建 Amazon SQS 操作所用的编程语言，但它能理解 AWS JSON 协议。
- 亚马逊 SQS AWS JSON 协议在亚马逊 SQS 客户端和服务器之间使用序列化（将对象转换为 JSON 格式）和反序列化（将 JSON 格式转换为对象）。

如何开始使用适用于亚马逊 SQS 的 AWS JSON 协议？

要开始使用最新版本的 AWS SDK，以便更快地向 Amazon SQS 发送消息，请将您的 AWS 软件开发工具包升级到指定版本或任何后续版本。要详细了解 SDK 客户端，请参阅下表中的“指南”一列。

以下是适用于亚马逊 SQS APIs 的 JS AWS ON 协议跨语言变体的软件开发工具包版本列表：

语言	SDK 客户端存储库	所需的 SDK 客户端版本	指南
C++	啊/ aws-sdk-cpp	1.11.98	AWS 适用于 C++ 的 SDK
Golang 1.x	啊/ aws-sdk-go	v1.47.7	

语言	SDK 客户端存储库	所需的 SDK 客户端版本	指南
			AWS 适用于 Go 的 SDK
Golang 2.x	aws/ 2 aws-sdk-go-v	v1.28.0	AWS 适用于 Go V2 的 SDK
Java 1.x	啊/ aws-sdk-java	1.12.585	AWS 适用于 Java 的 SDK
Java 2.x	aws/ 2 aws-sdk-java-v	2.21.19	AWS 适用于 Java 的 SDK
JavaScript v2.x	啊/ aws-sdk-js	JavaScript on AWS	
JavaScript v3.x	aws/ 3 aws-sdk-js-v	v3.447.0	JavaScript on AWS
.NET	啊/ aws-sdk-net	3.7.681.0	AWS 适用于 .NET 的 SDK
PHP	啊/ aws-sdk-php	3.285.2	AWS 适用于 PHP 的 SDK
Python-boto3	boto/boto3	1.28.82	AWS 适用于 Python (Boto3) 的 SDK
Python-botocore	boto/botocore	1.31.82	AWS 适用于 Python (Boto3) 的 SDK
awscli	AWS CLI	1.29.82	AWS命令行界面

语言	SDK 客户端存储库	所需的 SDK 客户端版本	指南
Ruby	啊/ aws-sdk-ruby	1.67.0	AWS 适用于 Ruby 的 SDK

为我的 Amazon SQS 工作负载启用 JSON 协议有什么风险？

如果您使用软件开发工具包的自定义实现或自定义客户端和 AWS AWS 软件开发工具包的组合来与生成基于 AWS 查询（又名基于 XML）的响应的 Amazon SQS 进行交互，则可能与 JSON 协议不兼容。AWS 如果您遇到任何问题，请联系 Supp AWS ort。

如果我已经使用最新的 AWS SDK 版本，但我的开源解决方案不支持 JSON，该怎么办？

您必须将 SDK 版本更改为当前所用版本之前的版本。有关如何开始使用适用于亚马逊 SQS 的 AWS JSON 协议？更多信息，请参阅。 AWS 中列出的软件开发工具包版本如何开始使用适用于亚马逊 SQS 的 AWS JSON 协议？使用适用于 Amazon SQS APIs 的 JSON 有线协议。如果您将 AWS 软件开发工具包更改为先前版本，则您的 Amazon SQS APIs 将使用该查询。 AWS

亚马逊 SQS APIs 中使用的 AWS JSON 协议支持哪些语言？

Amazon SQS 支持所有通用语言变体 (GA)。 AWS SDKs 目前，我们不支持 Kotlin、Rust 或 Swift。要详细了解其他语言变体，请参阅用于在 AWS上进行构建的工具。

亚马逊 SQS 中使用的 AWS JSON 协议支持哪些区域 APIs

亚马逊 SQS 在所有提供亚马逊 SQS 的AWS 地区都支持 AWS JSON 协议。

使用 JS AWS ON 协议升级到适用于 Amazon SQS 的指定 AWS 软件开发工具包版本时，我可以期待哪些延迟改善？

AWS 与查询协议相比，JSON 协议在请求和响应的序列化和反序列化方面效率更高。 AWS 根据对 5 KB 消息负载的 AWS 性能测试，适用于 Amazon SQS 的 JSON 协议可将 end-to-end消息处理延迟减少多达 23%，并减少应用程序客户端 CPU 和内存使用量。

AWS 查询协议会被弃用吗？

AWS 将继续支持查询协议。只要你的 AWS SDK 版本设置了除[如何开始使用 Amazon SQS 的 JS AWS ON 协议中列出的版本之外的任何先前版本，你就可以继续使用 AWS 查询协议](#)。

在哪里可以找到有关 AWS JSON 协议的更多信息？

您可以在 Smithy 文档的 [AWS JSON 1.0 协议](#)中找到有关 JSON 协议的更多信息。有关使用 AWS JSON 协议的 Amazon SQS API 请求的更多信息，请参阅[在 Amazon SQS 中使用 AWS JSON 协议发出查询 API 请求](#)。

在 Amazon SQS 中使用 AWS 查询协议发出查询 API 请求

本主题介绍了如何构造 Amazon SQS 端点、发出 GET 和 POST 请求以及解释响应。

构造端点

为了使用 Amazon SQS 队列，您必须构造一个端点。有关 Amazon SQS 端点的信息，请参阅 Amazon Web Services 一般参考中的以下页面：

- [区域端点](#)
- [Amazon Simple Queue Service 端点和配额](#)

每个 Amazon SQS 端点都是独立的。例如，如果有两个名为 MyQueue 的队列，其中一个队列具有终端节点 `sqs.us-east-2.amazonaws.com`，另一个队列具有终端节点 `sqs.eu-west-2.amazonaws.com`，则这两个队列不会相互共享任何数据。

以下是一个提出创建队列请求的端点的示例。

```
https://sqs.eu-west-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> 队列名称和队 URLs 列区分大小写。

AUTHPARAMS 的结构取决于 API 请求的签名。有关更多信息，请参阅亚马逊 Web Services 一般参考中的签署 AWS API 请求。

提出 GET 请求

Amazon SQS GET 请求的结构是一个 URL，其中包含以下部分：

- 端点 – 作为请求操作对象的资源（队列名称和 URL），例如：`https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue`
- 操作 – 要对端点执行的操作。端点与操作之间用问号 (?) 分隔，例如：`?Action=SendMessage&MessageBody=Your%20Message%20Text`
- 参数 - 任何请求参数。各参数以和号 (&) 分隔，例如：`&Version=2012-11-05&AUTHPARAMS`

以下是一个 GET 请求的示例，该请求向 Amazon SQS 队列发送一条消息。

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue
?Action=SendMessage&MessageBody=Your%20message%20text
&Version=2012-11-05
&AUTHPARAMS
```

Note

队列名称和队 URLs 列区分大小写。
由于 GET 请求是 URLs，因此必须对所有参数值进行 URL 编码。由于不允许使用空格 URLs，因此每个空格都以 URL 编码为。`%20`示例的其余部分没有进行 URL 编码，以方便您阅读。

发出 POST 请求

Amazon SQS POST 请求在 HTTP 请求的正文中以表单的形式发送查询参数。

下面是一个将 `Content-Type` 设置为 `application/x-www-form-urlencoded` 的 HTTP 标头的示例。

```
POST /123456789012/MyQueue HTTP/1.1
Host: sqs.us-east-2.amazonaws.com
```

```
Content-Type: application/x-www-form-urlencoded
```

该标头后跟一个 form-urlencoded GET 请求，该请求向 Amazon SQS 队列发送一条消息。各参数以和号 (&) 分隔。

```
Action=SendMessage
&MessageBody=Your+Message+Text
&Expires=2020-10-15T12%3A00%3A00Z
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> 仅 Content-Type HTTP 标头是必需的。*AUTHPARAMS* 对于 GET 请求是相同的。根据客户端的 HTTP 版本，您的 HTTP 客户端可能会向 HTTP 请求添加其他项目。

解释 Amazon SQS XML API 响应

当您向 Amazon SQS 发送请求时，它会返回包含请求结果的 XML 响应。要了解这些响应的结构和详情，请参阅《Amazon Simple Queue Service API 参考》中的特定 API 操作。

成功的 XML 响应结构

如果请求成功，则主要响应元素将以操作命名并附加上 Response（例如，*ActionName*Response)。

此元素包含以下子元素：

- ***ActionName*Result** – 包含一个特定于操作的元素。例如，CreateQueueResult 元素包含 QueueUrl 元素，后者又包含所创建队列的 URL。
- **ResponseMetadata** – 包含 RequestId，后者又包含请求的通用唯一标识符 (UUID)。

以下是 XML 格式的成功响应的示例：

```
<CreateQueueResponse
   xmlns=https://sqs.us-east-2.amazonaws.com/doc/2012-11-05/
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:type=CreateQueueResponse>
   <CreateQueueResult>
```

```
      <QueueUrl>https://sqs.us-east-2.amazonaws.com/770098461991/queue2</QueueUrl>
   </CreateQueueResult>
   <ResponseMetadata>
      <RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
   </ResponseMetadata>
</CreateQueueResponse>
```

XML 错误响应结构

如果请求不成功，则 Amazon SQS 将始终返回主要响应元素 `ErrorResponse`。此元素包含一个 `Error` 元素和一个 `RequestId` 元素。

`Error` 元素包含以下子元素：

- **`Type`** – 指定错误是创建者错误还是使用者错误。
- **`Code`** – 指定错误类型。
- **`Message`** – 以可读格式指定错误情况。
- **`Detail`** – （可选）指定有关错误的其他详细信息。

`RequestId` 元素包含请求的 UUID。

下面是 XML 格式的错误响应的示例：

```
<ErrorResponse>
   <Error>
      <Type>Sender</Type>
      <Code>InvalidParameterValue</Code>
      <Message>
         Value (quename_nonalpha) for parameter QueueName is invalid.
         Must be an alphanumeric String of 1 to 80 in length.
      </Message>
   </Error>
   <RequestId>42d59b56-7407-4c4a-be0f-4c88daeea257</RequestId>
</ErrorResponse>
```

对 Amazon SQS 请求进行身份验证

身份验证是用于识别和验证发送请求的当事方的过程。在身份验证的第一个阶段， AWS 将验证创建者的身份以及创建者是否已注册使用 AWS（有关更多信息，请参阅设置 Amazon SQS）。接下来，AWS 遵守以下程序：

1. 创建者（发件人）获取必要的凭证。
2. 创建者向使用者（接收方）发送请求和凭证。
3. 使用者使用证书来验证创建者是否发送了该请求。
4. 将出现以下情况之一：
   - 如果身份验证成功，使用者将处理该请求。
   - 如果身份验证失败，使用者将拒绝请求并返回错误。

基本身份验证流程 HMAC-SHA

使用查询 API 访问 Amazon SQS 时，必须提供以下项来对请求进行身份验证：

- 用于识别您的AWS 访问密钥 ID AWS 账户， AWS 用于查找您的私有访问密钥。
- HMAC-SHA 请求签名使用您的私有访问密钥（只有您知道的共享密钥，以及 AWS—有关更多信息，请参阅 RFC2104）计算得出。AWS 开发工具包可处理签名过程；但是，如果您通过 HTTP 或 HTTPS 提交查询请求，则必须在每个查询请求中包含一个签名。

1. 派生签名版本 4 签名密钥。有关更多信息，请参阅使用 Java 派生签名密钥。

   > **Note**
   >
   > Amazon SQS 支持签名版本 4，与之前的版本相比，该版本提供了更高的SHA256-based 安全性和性能。创建使用 Amazon SQS 的新应用程序时，应使用 Signature Version 4。

2. Base64-encode 请求签名。下面的示例 Java 代码将执行此操作：

```
package amazon.webservices.common;

// Define common routines for encoding data in AWS requests.
public class Encoding {

    /* Perform base64 encoding of input bytes.
     * rawData is the array of bytes to be encoded.
     * return is the base64-encoded string representation of rawData.
     */
    public static String EncodeBase64(byte[] rawData) {
        return Base64.encodeBytes(rawData);
    }
```

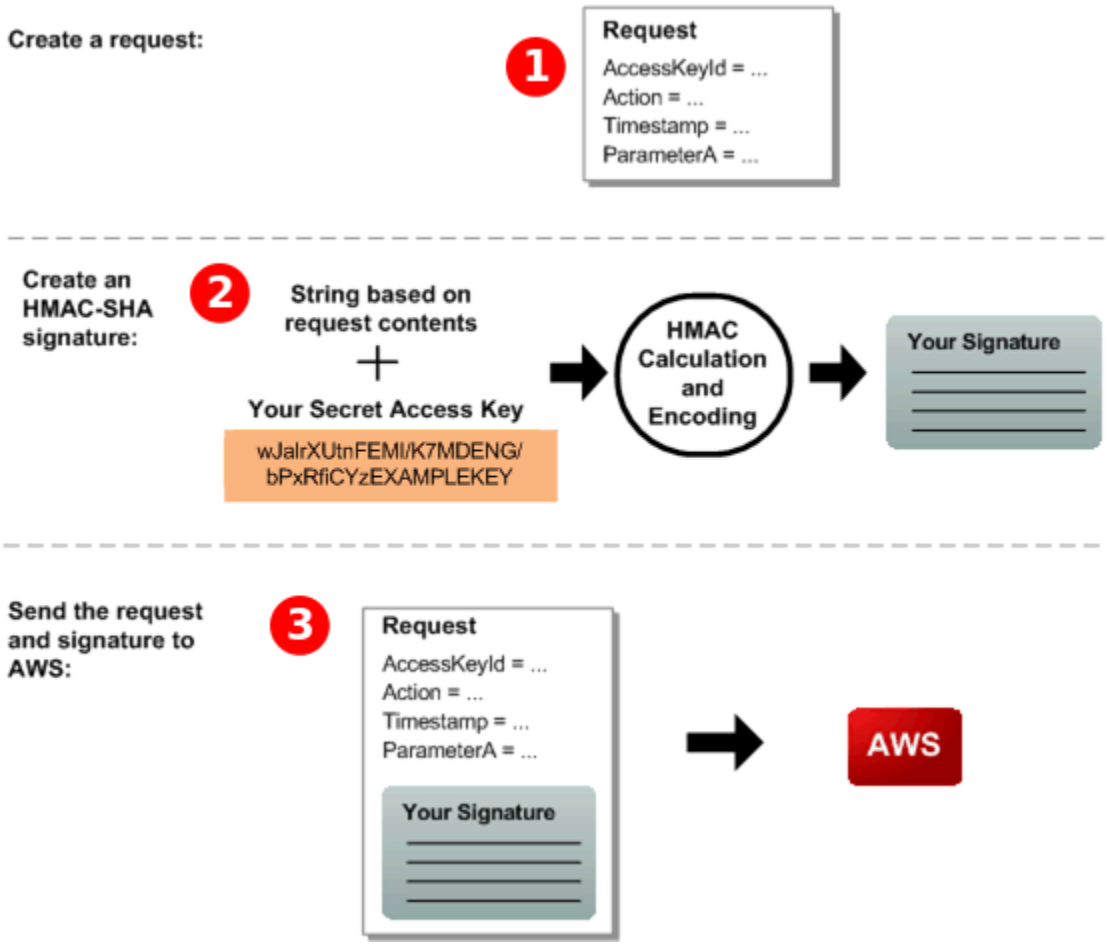
```
}
```

- 请求的 时间戳（或到期时间）。在请求中使用的时间戳必须是 dateTime 对象，并包含完整的日期以及小时、分钟和秒。例如 2007-01-31T23:59:59Z。尽管没有强制要求，但还是建议您使用协调世界时（格林威治标准时间）时区提供该对象。

> **Note**
>
> 确保您的服务器时间设置正确。如果您指定时间戳（而不是过期时间），则请求将在指定时间后 15 分钟自动过期（AWS 不处理时间戳比 AWS 服务器当前时间早 15 分钟以上的请求）。
>
> 如果使用 .NET，则不得发送过于具体的时间戳（因为对如何降低额外时间精度的解释不同）。在这种情况下，应手动构造精度不超过 1 毫秒的 dateTime 对象。

第 1 部分：来自用户的请求

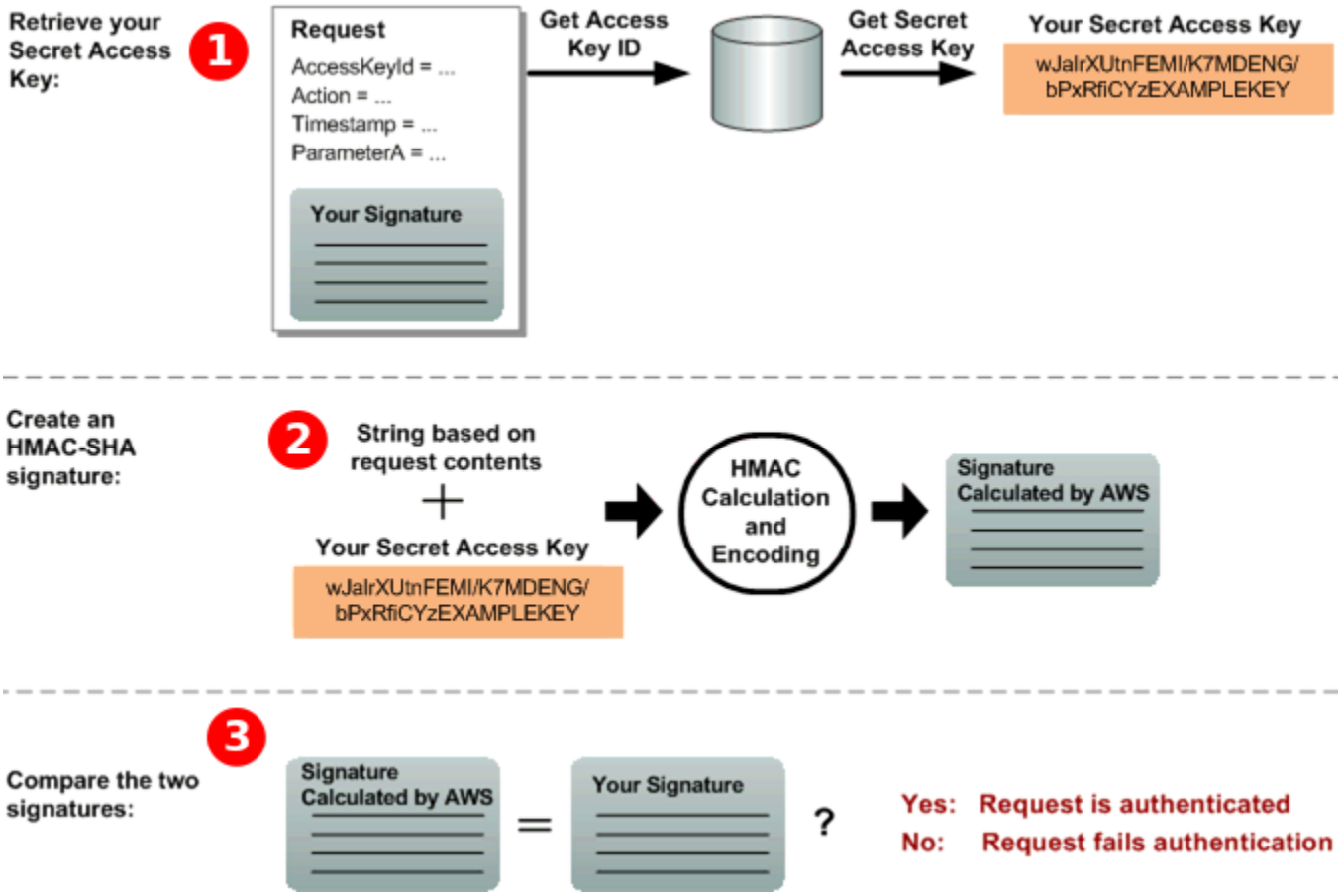
以下是使用请求签名对 AWS 请求进行身份验证时必须遵循的HMAC-SHA 流程。



1. 构造对的请求 AWS。
2. 使用您的私有访问密钥计算密钥哈希消息身份验证码 (HMAC-SHA) 签名。
3. 在请求中包括签名和您的访问密钥 ID，然后将请求发送至 AWS。

第 2 部分：来自的回应 AWS

AWS 作为响应，开始以下过程。



1. AWS 使用访问密钥 ID 来查找您的私有访问密钥。
2. AWS 使用与计算请求中发送的签名相同的算法，根据请求数据和私有访问密钥生成签名。
3. 将出现以下情况之一：
   - 如果 AWS 生成的签名与您在请求中发送的签名相匹配，则 AWS 认为该请求是真实的。
   - 如果比较失败，则请求将被丢弃，并 AWS 返回错误。

Amazon SQS 批处理操作

Amazon SQS 支持批处理操作，帮助您降低成本，并通过单次操作处理最多 10 条消息。这些批处理操作包括：

- `SendMessageBatch`
- `DeleteMessageBatch`
- `ChangeMessageVisibilityBatch`

使用批处理操作，您可以在一次 API 调用中执行多个操作，这有助于优化性能并降低成本。您可以使用查询 API 或任何支持 Amazon SQS 批处理操作的 AWS 软件开发工具包来利用批处理功能。

重要详细信息

- 消息大小限制：在一次 `SendMessageBatch` 调用中发送的所有消息的总大小不能超过 1048576 字节（1 MiB）。
- 权限：您无法显式设置 `SendMessageBatch`、`DeleteMessageBatch` 或 `ChangeMessageVisibilityBatch` 的权限。相反，设置 `SendMessage`、`DeleteMessage` 或 `ChangeMessageVisibility` 操作的权限会同时设置其对应的批处理版本的权限。
- 控制台支持：Amazon SQS 控制台不支持批处理操作。您必须使用查询 API 或 S AWS DK 来执行批量操作。

批处理消息操作

为了进一步优化成本和效率，请考虑以下批处理消息操作的最佳实践：

- 批处理 API 操作：使用 Amazon SQS 批处理 API 操作来发送、接收和删除消息，并通过单次操作更改多条消息的可见性超时。这样做可以减少 API 调用的次数和相关成本。
- 客户端缓冲和长轮询数：将长轮询与 适用于 Java 的 AWS SDK中的缓冲异步客户端一起使用，从而将客户端缓冲与请求批处理功能相结合。这种方法有助于最大限度地减少请求数量并优化对大量消息的处理。

> **Note**
>
> Amazon SQS 缓冲异步客户端目前不支持 FIFO 队列。

启用客户端缓冲和请求批处理功能，并将其与 Amazon SQS 结合使用

适用于 Java 的 AWS SDK 包括可访问 Amazon SQS 的 `AmazonSQSBufferedAsyncClient`。该客户端支持使用客户端缓冲进行简单的请求批处理。客户端进行的调用首先被缓冲，然后作为批处理请求发送到 Amazon SQS。

客户端缓冲最多允许缓冲 10 个请求并将这些请求作为一个批处理请求发送，从而减少使用 Amazon SQS 的成本并减少发送的请求数。`AmazonSQSBufferedAsyncClient` 会缓冲同步和异步调用。批

量请求和对长轮询的支持还有助于提高吞吐量。有关更多信息，请参阅 利用水平扩缩和操作批处理，借助 Amazon SQS 来提高吞吐量。

由于 AmazonSQSBufferedAsyncClient 实施与 AmazonSQSAsyncClient 相同的接口，因此从 AmazonSQSAsyncClient 迁移到 AmazonSQSBufferedAsyncClient 通常只需要对现有代码进行少量的更改。

> **Note**
>
> Amazon SQS 缓冲异步客户端目前不支持 FIFO 队列。

使用亚马逊 SQSBuffered AsyncClient

在开始之前，请完成 设置 Amazon SQS 中的步骤。

AWS 适用于 Java 的 SDK 1.x

对于 AWS 适用于 Java 的 SDK 1.x，你可以AmazonSQSBufferedAsyncClient根据以下示例创建一个新的：

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

在创建新的 AmazonSQSBufferedAsyncClient 之后，您可以使用它将多个请求发送到 Amazon SQS（就像使用 AmazonSQSAsyncClient 所做的那样），例如：

```
final CreateQueueRequest createRequest = new
 CreateQueueRequest().withQueueName("MyQueue");

final CreateQueueResult res = bufferedSqs.createQueue(createRequest);

final SendMessageRequest request = new SendMessageRequest();
final String body = "Your message text" + System.currentTimeMillis();
request.setMessageBody( body );
request.setQueueUrl(res.getQueueUrl());

final Future<SendMessageResult> sendResult = bufferedSqs.sendMessageAsync(request);
```

```
final ReceiveMessageRequest receiveRq = new ReceiveMessageRequest()
    .withMaxNumberOfMessages(1)
    .withQueueUrl(queueUrl);
final ReceiveMessageResult rx = bufferedSqs.receiveMessage(receiveRq);
```

配置亚马逊 SQSBuffered AsyncClient

AmazonSQSBufferedAsyncClient 预配置了适用于大多数使用案例的设置。您可以进一步配置 AmazonSQSBufferedAsyncClient，例如：

1. 使用必需的配置参数来创建 QueueBufferConfig 类的实例。
2. 将该实例提供给 AmazonSQSBufferedAsyncClient 构造函数。

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

final QueueBufferConfig config = new QueueBufferConfig()
    .withMaxInflightReceiveBatches(5)
    .withMaxDoneReceiveBatches(15);

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync, config);
```

QueueBufferConfig 配置参数

参数	默认 值	说明
longPoll	true	如果 longPoll 设置为 true，AmazonSQS BufferedAsyncClient 会在使用消息时尝试使用长轮询。
longPollWaitTimeoutSeconds	20 秒	在返回空接收结果前，ReceiveMessage 调用在服务器上阻塞以等待消息显示在队列中的最长时间（以秒为单位）。

参数	默认 值	说明
		Note 如果禁用长轮询，则此设置不起作用。
`maxBatchOpenMs`	200 毫秒	传出调用等待其他要一起对同类型的消息进行批处理的调用的最长时间（以毫秒为单位）。 设置的时间越长，则执行等量工作所需的批处理次数就越少（但是，批处理中的首次调用必须等待更长的时间）。 如果将此参数设置为 `0`，则提交的请求不会等待其他请求，从而有效地禁用批处理。
`maxBatchSize`	每批 10 个请求	在一个请求中一起进行批处理的消息的最大数量。该设置越大，则执行等量请求所需的批处理就越少。 **Note** Amazon SQS 允许的最大值为每批 10 个请求。

参数	默认 值	说明
`maxBatchSizeBytes`	1 MiB	客户端尝试向 Amazon SQS 发送的消息批处理的最大大小（以字节为单位）。 **Note** Amazon SQS 允许的最大值为 1 MiB。
`maxDoneReceiveBatches`	10 个批处理	`AmazonSQSBufferedAsyncClient` 在客户端预取和存储的接收批处理的最大数量。 设置的值越高，则可满足越多的接收请求而不必调用 Amazon SQS（但是，预取的消息越多，则消息在缓冲区中停留的时间就越长，从而导致它们的可见性超时过期）。 **Note** `0` 表示所有消息预取操作将被禁用，消息只能按需使用。

参数	默认 值	说明
maxInflightOutboundBatches	5 个批处理	可以同时处理的最大活跃出站批处理数量。 设置的值越高，发送出站批处理的速度就越快（受限于其他配额，例如 CPU 或带宽），并且 AmazonSQSBufferedAsyncClient 使用的线程就越多。
maxInflightReceiveBatches	10 个批处理	可以同时处理的最大活跃接收批处理数量。 设置的值越高，可接收的消息就越多（受限于其他配额，例如 CPU 或带宽），并且 AmazonSQSBufferedAsyncClient 使用的线程就越多。 **Note** 0 表示所有消息预取操作将被禁用，消息只能按需使用。

参数	默认 值	说明
visibilityTimeoutSeconds	–1	如果此参数设置为正值（非零值），则此处设置的可见性超时将覆盖在使用的消息所在的队列上设置的可见性超时。 **Note** -1 表示为队列选择默认设置。 不能将可见性超时设置为 0。

AWS 适用于 Java 的 SDK 2.x

对于 AWS 适用于 Java 的 SDK 2.x，你可以SqsAsyncBatchManager根据以下示例创建一个新的：

```
// Create the basic Sqs Async Client
SqsAsyncClient sqs = SqsAsyncClient.builder()
    .region(Region.US_EAST_1)
    .build();

// Create the batch manager
SqsAsyncBatchManager sqsAsyncBatchManager = sqs.batchManager();
```

在创建新的 SqsAsyncBatchManager 之后，您可以使用它将多个请求发送到 Amazon SQS（就像使用 SqsAsyncClient 所做的那样），例如：

```
final String queueName = "MyAsyncBufferedQueue" + UUID.randomUUID();
final CreateQueueRequest request =
 CreateQueueRequest.builder().queueName(queueName).build();
final String queueUrl = sqs.createQueue(request).join().queueUrl();
System.out.println("Queue created: " + queueUrl);
```

```
// Send messages
CompletableFuture<SendMessageResponse> sendMessageFuture;
for (int i = 0; i < 10; i++) {
    final int index = i;
    sendMessageFuture = sqsAsyncBatchManager.sendMessage(
            r -> r.messageBody("Message " + index).queueUrl(queueUrl));
    SendMessageResponse response= sendMessageFuture.join();
    System.out.println("Message " + response.messageId() + " sent!");
}

// Receive messages with customized configurations
CompletableFuture<ReceiveMessageResponse> receiveResponseFuture =
 customizedBatchManager.receiveMessage(
        r -> r.queueUrl(queueUrl)
                .waitTimeSeconds(10)
                .visibilityTimeout(20)
                .maxNumberOfMessages(10)
);
System.out.println("You have received " +
 receiveResponseFuture.join().messages().size() + " messages in total.");

// Delete messages
DeleteQueueRequest deleteQueueRequest =
 DeleteQueueRequest.builder().queueUrl(queueUrl).build();
int code = sqs.deleteQueue(deleteQueueRequest).join().sdkHttpResponse().statusCode();
System.out.println("Queue is deleted, with statusCode " + code);
```

正在配置 SqsAsyncBatchManager

SqsAsyncBatchManager 预配置了适用于大多数使用案例的设置。您可以进一步配置 SqsAsyncBatchManager，例如：

通过 SqsAsyncBatchManager.Builder 创建自定义配置：

```
SqsAsyncBatchManager customizedBatchManager = SqsAsyncBatchManager.builder()
    .client(sqs)
    .scheduledExecutor(Executors.newScheduledThreadPool(5))
    .overrideConfiguration(b -> b
        .maxBatchSize(10)
        .sendRequestFrequency(Duration.ofMillis(200))
        .receiveMessageMinWaitDuration(Duration.ofSeconds(10))
        .receiveMessageVisibilityTimeout(Duration.ofSeconds(20))
        .receiveMessageAttributeNames(Collections.singletonList("*"))
```

```
.receiveMessageSystemAttributeNames(Collections.singletonList(MessageSystemAttributeName.ALL))
   .build();
```

`BatchOverrideConfiguration` 参数

参数	默认 值	说明
`maxBatchSize`	每批 10 个请求	在一个请求中一起进行批处理的消息的最大数量。该设置越大，则执行等量请求所需的批处理就越少。 **Note** Amazon SQS 允许的最大值为每批次 10 个请求。
`sendRequestFrequency`	200 毫秒	传出调用等待其他要一起对同类型的消息进行批处理的调用的最长时间（以毫秒为单位）。 设置的时间越长，则执行等量工作所需的批处理次数就越少（但是，批处理中的首次调用必须等待更长的时间）。 如果将此参数设置为 `0`，则提交的请求不会等待其他请求，从而有效地禁用批处理。
`receiveMessageVisibilityTimeout`	–1	如果此参数设置为正值（非零值），则此处设置的可见性超时将覆盖在使用的消息所在的队列上设置的可见性超时。

参数	默认 值	说明
		Note 1 表示为队列选择默认设置。不能将可见性超时设置为 0。
receiveMessageMinWaitDuration	50 毫秒	receiveMessage 调用等待可用消息被获取的最短时间（以毫秒为单位）。设置的值越高，则执行等量请求所需的批量操作就越少。

利用水平扩缩和操作批处理，借助 Amazon SQS 来提高吞吐量

Amazon SQS 支持高吞吐量消息传递。有关吞吐量限制的详细信息，请参阅 Amazon SQS 消息配额。

要最大限度提高吞吐量：

- 通过增加创建者和使用者的实例数量，实现水平扩展。
- 使用操作批处理在单次请求中发送或接收多条消息，从而减少 API 调用开销。

横向扩展

由于您通过 HTTP 请求-响应协议来访问 Amazon SQS，因此，请求延迟（启动请求和接收响应之间的时间间隔）会限制您可以通过单一连接利用单一线程达到的吞吐量。例如，如果从基于 Amazon EC2 的客户端到同一区域内的 Amazon SQS 的延迟时间平均为 20 毫秒，则通过单一连接利用单一线程达到的最大吞吐量平均为 50 TPS。

水平扩展 涉及到增加消息创建者（发出 SendMessage 请求）和使用者（发出 ReceiveMessage 和 DeleteMessage 请求）的数量，以提高整个队列的吞吐量。可以通过三种方式进行水平扩展：

- 增加每个客户端的线程数量
- 添加更多客户端

- 增加每个客户端的线程数量并添加更多客户端

在添加更多客户端后，基本上可以实现队列吞吐量的线性增长。例如，如果将客户端数量翻倍，吞吐量也会翻倍。

操作批处理

批处理 可在与服务的每次往返操作中执行更多的工作（例如，当您通过单个 SendMessageBatch 请求发送多条消息时）。Amazon SQS 批处理操作包括 SendMessageBatch、DeleteMessageBatch 和 ChangeMessageVisibilityBatch。要在不更改创建者或使用者的情况下利用批处理，您可以使用 Amazon SQS 缓冲异步客户端。

> **Note**
>
> 由于 ReceiveMessage 一次可以处理 10 条消息，因此没有 ReceiveMessageBatch 操作。

批处理会在一个批处理请求中的多条消息之间分配批处理操作的延迟时间，而不是接受单一消息（例如，SendMessage 请求）的整个延迟时间。由于每次往返操作都会执行更多工作，因此，批处理请求可以更高效地使用线程和连接，从而提高吞吐量。

可以将批处理与水平扩展结合使用来提供吞吐量，所需的线程、连接和请求的数量比单独的消息请求所需的数量更少。您可以使用 Amazon SQS 批处理操作一次性发送、接收或删除多达 10 条消息。由于 Amazon SQS 按请求收费，因此，批处理可以大幅降低您的成本。

批处理会为您的应用程序带来一些复杂性（例如，您的应用程序必须先积累消息然后才能发送，或者有时候必须花费较长的时间等待响应）。但是，批处理在以下情况下仍然会很有效：

- 您的应用程序在短时间内生成很多消息，因此，延迟时间从来不会很长。
- 消息使用者从队列中自行获取消息，这与需要发送消息来响应其无法控制的事件的典型消息创建者不同。

> **Important**
>
> 即使批处理中的个别消息失败了，批处理请求也可能会成功。发出批处理请求后，始终检查各条消息是否失败，并在必要时重试操作。

单一操作和批处理请求的有效 Java 示例

先决条件

将 aws-java-sdk-sqs.jar、aws-java-sdk-ec2.jar 和 commons-logging.jar 程序包添加到 Java 生成类路径中。以下示例说明了 Maven 项目的 pom.xml 文件中的这些依赖关系。

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version>LATEST</version>
    </dependency>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-ec2</artifactId>
        <version>LATEST</version>
    </dependency>
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>LATEST</version>
    </dependency>
</dependencies>
```

SimpleProducerConsumer.java

以下 Java 代码示例将实施一个简单的创建者-使用者模式。主线程会生成大量创建者和使用者线程，这些线程会在指定时间内处理 1KB 消息。此示例包括发出单一操作请求的创建者和使用者，以及发出批处理请求的创建者和使用者。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
```

```
 * permissions and limitations under the License.
 *
 */

import com.amazonaws.AmazonClientException;
import com.amazonaws.ClientConfiguration;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.*;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Start a specified number of producer and consumer threads, and produce-consume
 * for the least of the specified duration and 1 hour. Some messages can be left
 * in the queue because producers and consumers might not be in exact balance.
 */
public class SimpleProducerConsumer {

    // The maximum runtime of the program.
    private final static int MAX_RUNTIME_MINUTES = 60;
    private final static Log log = LogFactory.getLog(SimpleProducerConsumer.class);

    public static void main(String[] args) throws InterruptedException {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the number of producers: ");
        final int producerCount = input.nextInt();

        System.out.print("Enter the number of consumers: ");
        final int consumerCount = input.nextInt();
```

```
        System.out.print("Enter the number of messages per batch: ");
        final int batchSize = input.nextInt();

        System.out.print("Enter the message size in bytes: ");
        final int messageSizeByte = input.nextInt();

        System.out.print("Enter the run time in minutes: ");
        final int runTimeMinutes = input.nextInt();

        /*
         * Create a new instance of the builder with all defaults (credentials
         * and region) set automatically. For more information, see Creating
         * Service Clients in the AWS SDK for Java Developer Guide.
         */
        final ClientConfiguration clientConfiguration = new ClientConfiguration()
                .withMaxConnections(producerCount + consumerCount);

        final AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()
                .withClientConfiguration(clientConfiguration)
                .build();

        final String queueUrl = sqsClient
                .getQueueUrl(new GetQueueUrlRequest(queueName)).getQueueUrl();

        // The flag used to stop producer, consumer, and monitor threads.
        final AtomicBoolean stop = new AtomicBoolean(false);

        // Start the producers.
        final AtomicInteger producedCount = new AtomicInteger();
        final Thread[] producers = new Thread[producerCount];
        for (int i = 0; i < producerCount; i++) {
            if (batchSize == 1) {
                producers[i] = new Producer(sqsClient, queueUrl, messageSizeByte,
                        producedCount, stop);
            } else {
                producers[i] = new BatchProducer(sqsClient, queueUrl, batchSize,
                        messageSizeByte, producedCount,
                        stop);
            }
            producers[i].start();
        }

        // Start the consumers.
```

```
        final AtomicInteger consumedCount = new AtomicInteger();
        final Thread[] consumers = new Thread[consumerCount];
        for (int i = 0; i < consumerCount; i++) {
            if (batchSize == 1) {
                consumers[i] = new Consumer(sqsClient, queueUrl, consumedCount,
                        stop);
            } else {
                consumers[i] = new BatchConsumer(sqsClient, queueUrl, batchSize,
                        consumedCount, stop);
            }
            consumers[i].start();
        }

        // Start the monitor thread.
        final Thread monitor = new Monitor(producedCount, consumedCount, stop);
        monitor.start();

        // Wait for the specified amount of time then stop.
        Thread.sleep(TimeUnit.MINUTES.toMillis(Math.min(runTimeMinutes,
                MAX_RUNTIME_MINUTES)));
        stop.set(true);

        // Join all threads.
        for (int i = 0; i < producerCount; i++) {
            producers[i].join();
        }

        for (int i = 0; i < consumerCount; i++) {
            consumers[i].join();
        }

        monitor.interrupt();
        monitor.join();
    }

    private static String makeRandomString(int sizeByte) {
        final byte[] bs = new byte[(int) Math.ceil(sizeByte * 5 / 8)];
        new Random().nextBytes(bs);
        bs[0] = (byte) ((bs[0] | 64) & 127);
        return new BigInteger(bs).toString(32);
    }

    /**
     * The producer thread uses {@code SendMessage}
```

```
     * to send messages until it is stopped.
     */
    private static class Producer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        Producer(AmazonSQS sqsQueueBuffer, String queueUrl, int messageSizeByte,
                 AtomicInteger producedCount, AtomicBoolean stop) {
            this.sqsClient = sqsQueueBuffer;
            this.queueUrl = queueUrl;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /*
         * The producedCount object tracks the number of messages produced by
         * all producer threads. If there is an error, the program exits the
         * run() method.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    sqsClient.sendMessage(new SendMessageRequest(queueUrl,
                            theMessage));
                    producedCount.incrementAndGet();
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("Producer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * The producer thread uses {@code SendMessageBatch}
     * to send messages until it is stopped.
```

```
     */
    private static class BatchProducer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final int batchSize;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        BatchProducer(AmazonSQS sqsQueueBuffer, String queueUrl, int batchSize,
                      int messageSizeByte, AtomicInteger producedCount,
                      AtomicBoolean stop) {
            this.sqsClient = sqsQueueBuffer;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        public void run() {
            try {
                while (!stop.get()) {
                    final SendMessageBatchRequest batchRequest =
                            new SendMessageBatchRequest().withQueueUrl(queueUrl);

                    final List<SendMessageBatchRequestEntry> entries =
                            new ArrayList<SendMessageBatchRequestEntry>();
                    for (int i = 0; i < batchSize; i++)
                        entries.add(new SendMessageBatchRequestEntry()
                                .withId(Integer.toString(i))
                                .withMessageBody(theMessage));
                    batchRequest.setEntries(entries);

                    final SendMessageBatchResult batchResult =
                            sqsClient.sendMessageBatch(batchRequest);
                    producedCount.addAndGet(batchResult.getSuccessful().size());

                    /*
                     * Because SendMessageBatch can return successfully, but
                     * individual batch items fail, retry the failed batch items.
                     */
                    if (!batchResult.getFailed().isEmpty()) {
                        log.warn("Producer: retrying sending "
```

```
                            + batchResult.getFailed().size() + " messages");
                    for (int i = 0, n = batchResult.getFailed().size();
                         i < n; i++) {
                        sqsClient.sendMessage(new
                                SendMessageRequest(queueUrl, theMessage));
                        producedCount.incrementAndGet();
                    }
                }
            }
        } catch (AmazonClientException e) {
            /*
             * By default, AmazonSQSClient retries calls 3 times before
             * failing. If this unlikely condition occurs, stop.
             */
            log.error("BatchProducer: " + e.getMessage());
            System.exit(1);
        }
    }
}

/**
 * The consumer thread uses {@code ReceiveMessage} and {@code DeleteMessage}
 * to consume messages until it is stopped.
 */
private static class Consumer extends Thread {
    final AmazonSQS sqsClient;
    final String queueUrl;
    final AtomicInteger consumedCount;
    final AtomicBoolean stop;

    Consumer(AmazonSQS sqsClient, String queueUrl, AtomicInteger consumedCount,
             AtomicBoolean stop) {
        this.sqsClient = sqsClient;
        this.queueUrl = queueUrl;
        this.consumedCount = consumedCount;
        this.stop = stop;
    }

    /*
     * Each consumer thread receives and deletes messages until the main
     * thread stops the consumer thread. The consumedCount object tracks the
     * number of messages that are consumed by all consumer threads, and the
     * count is logged periodically.
     */
```

```
        public void run() {
            try {
                while (!stop.get()) {
                    try {
                        final ReceiveMessageResult result = sqsClient
                                .receiveMessage(new
                                        ReceiveMessageRequest(queueUrl));

                        if (!result.getMessages().isEmpty()) {
                            final Message m = result.getMessages().get(0);
                            sqsClient.deleteMessage(new
                                    DeleteMessageRequest(queueUrl,
                                    m.getReceiptHandle()));
                            consumedCount.incrementAndGet();
                        }
                    } catch (AmazonClientException e) {
                        log.error(e.getMessage());
                    }
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("Consumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * The consumer thread uses {@code ReceiveMessage} and {@code
     * DeleteMessageBatch} to consume messages until it is stopped.
     */
    private static class BatchConsumer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final int batchSize;
        final AtomicInteger consumedCount;
        final AtomicBoolean stop;

        BatchConsumer(AmazonSQS sqsClient, String queueUrl, int batchSize,
                      AtomicInteger consumedCount, AtomicBoolean stop) {
            this.sqsClient = sqsClient;
```

```
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        public void run() {
            try {
                while (!stop.get()) {
                    final ReceiveMessageResult result = sqsClient
                            .receiveMessage(new ReceiveMessageRequest(queueUrl)
                                    .withMaxNumberOfMessages(batchSize));

                    if (!result.getMessages().isEmpty()) {
                        final List<Message> messages = result.getMessages();
                        final DeleteMessageBatchRequest batchRequest =
                                new DeleteMessageBatchRequest()
                                        .withQueueUrl(queueUrl);

                        final List<DeleteMessageBatchRequestEntry> entries =
                                new ArrayList<DeleteMessageBatchRequestEntry>();
                        for (int i = 0, n = messages.size(); i < n; i++)
                            entries.add(new DeleteMessageBatchRequestEntry()
                                    .withId(Integer.toString(i))
                                    .withReceiptHandle(messages.get(i)
                                            .getReceiptHandle()));
                        batchRequest.setEntries(entries);

                        final DeleteMessageBatchResult batchResult = sqsClient
                                .deleteMessageBatch(batchRequest);
                        consumedCount.addAndGet(batchResult.getSuccessful().size());

                        /*
                         * Because DeleteMessageBatch can return successfully,
                         * but individual batch items fail, retry the failed
                         * batch items.
                         */
                        if (!batchResult.getFailed().isEmpty()) {
                            final int n = batchResult.getFailed().size();
                            log.warn("Producer: retrying deleting " + n
                                    + " messages");
                            for (BatchResultErrorEntry e : batchResult
                                    .getFailed()) {
```

```
                                sqsClient.deleteMessage(
                                        new DeleteMessageRequest(queueUrl,
                                                messages.get(Integer
                                                        .parseInt(e.getId()))
                                                        .getReceiptHandle()));

                                consumedCount.incrementAndGet();
                            }
                        }
                    }
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("BatchConsumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * This thread prints every second the number of messages produced and
     * consumed so far.
     */
    private static class Monitor extends Thread {
        private final AtomicInteger producedCount;
        private final AtomicInteger consumedCount;
        private final AtomicBoolean stop;

        Monitor(AtomicInteger producedCount, AtomicInteger consumedCount,
                AtomicBoolean stop) {
            this.producedCount = producedCount;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        public void run() {
            try {
                while (!stop.get()) {
                    Thread.sleep(1000);
                    log.info("produced messages = " + producedCount.get()
                            + ", consumed messages = " + consumedCount.get());
```

```
                }
            } catch (InterruptedException e) {
                // Allow the thread to exit.
            }
        }
    }
}
```

监控示例运行中的数量指标

Amazon SQS 会针对发送、接收和删除的消息自动生成容量指标。可以通过队列的监控选项卡或者通过 CloudWatch 控制台访问上述指标和其他指标。

> **Note**
>
> 队列启动后，这些指标最多可能需要花费 15 分钟才可用。

将 Amazon SQS 与 AWS SDK

AWS 软件开发套件 (SDK) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
适用于 C++ 的 AWS SDK	适用于 C++ 的 AWS SDK 代码示例
AWS CLI	AWS CLI 代码示例
适用于 Go 的 AWS SDK	适用于 Go 的 AWS SDK 代码示例
适用于 Java 的 AWS SDK	适用于 Java 的 AWS SDK 代码示例
适用于 JavaScript 的 AWS SDK	适用于 JavaScript 的 AWS SDK 代码示例
适用于 Kotlin 的 AWS SDK	适用于 Kotlin 的 AWS SDK 代码示例
适用于 .NET 的 AWS SDK	适用于 .NET 的 AWS SDK 代码示例
适用于 PHP 的 AWS SDK	适用于 PHP 的 AWS SDK 代码示例

SDK 文档	代码示例
AWS Tools for PowerShell	AWS Tools for PowerShell 代码示例
AWS 适用于 Python 的 SDK v4	AWS 适用于 Python 的 SDK v4 代码示例
适用于 Python (Boto3) 的 AWS SDK	适用于 Python (Boto3) 的 AWS SDK 代码示例
适用于 Ruby 的 AWS SDK	适用于 Ruby 的 AWS SDK 代码示例
适用于 Rust 的 AWS SDK	适用于 Rust 的 AWS SDK 代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
适用于 Swift 的 AWS SDK	适用于 Swift 的 AWS SDK 代码示例

示例可用性

找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

将 JMS 与 Amazon SQS 结合使用

Amazon SQS Java Messaging Library 是适用于 Amazon SQS 的 Java Message Service (JMS) 接口，允许您在已经使用 JMS 的应用程序中利用 Amazon SQS。利用此接口，对代码稍作更改即可将 Amazon SQS 用作 JMS 提供程序。结合使用 适用于 Java 的 AWS SDK 与 Amazon SQS Java Messaging Library，您可以创建 JMS 连接和会话以及与 Amazon SQS 队列之间收发消息的创建者和使用者。

此库支持依据 JMS 1.1 规范的队列消息收发（JMS 点对点模式）；该库支持将文本、字节或对象消息同步发送到 Amazon SQS 队列。还支持同步或异步接收对象。

有关支持 JMS 1.1 规范的 Amazon SQS Java Messaging Library 特征的信息，请参阅Amazon SQS 支持的 JMS 1.1 实现和 Amazon SQS 常见问题。

使用 JMS 和 Amazon SQS 的先决条件

在开始之前，您必须满足以下先决条件：

- 适用于 Java 的 SDK

  可通过两种不同的方式将适用于 Java 的 SDK 包含在项目中：

  - 下载并安装适用于 Java 的 SDK。
  - 使用 Maven 获取 Amazon SQS Java Messaging Library。

  > **Note**
  >
  > 适用于 Java 的 SDK 作为一个依赖项包含在内。
  > 适用于 Java 的 SDK 和适用于 Java 的 Amazon SQS 扩展型客户端库需要使用 J2SE Development Kit 8.0 或更高版本。

  有关下载适用于 Java 的 SDK 的信息，请参阅适用于 Java 的 SDK。

- Amazon SQS Java Messaging Library

  如果未使用 Maven，则必须将 `amazon-sqs-java-messaging-lib.jar` 程序包添加到 Java 类路径中。有关下载该库的信息，请参阅 Amazon SQS Java Messaging Library。

> **Note**
>
> Amazon SQS Java Messaging Library 包括对 Maven 和 Spring Framework 的支持。有关使用 Maven、Spring Framework 和 Amazon SQS Java Messaging Library 的代码示例，请参阅实际可用的 Java 示例：将 JMS 与 Amazon SQS 标准队列相结合使用。
>
> ```
> <dependency>
>   <groupId>com.amazonaws</groupId>
>   <artifactId>amazon-sqs-java-messaging-lib</artifactId>
>   <version>1.0.4</version>
>   <type>jar</type>
> </dependency>
> ```

- Amazon SQS 队列

  使用适用于 Amazon SQS 的 AWS 管理控制台、`CreateQueue` API 或 Amazon SQS Java Messaging Library 中包括的封装 Amazon SQS 客户端，创建队列。

  - 有关使用 AWS 管理控制台或 `CreateQueue` API 创建用于 Amazon SQS 的队列的信息，请参阅创建队列。
  - 有关使用 Amazon SQS Java Messaging Library 的信息，请参阅使用 Amazon SQS Java Messaging Library。

使用 Amazon SQS Java Messaging Library

要开始结合使用 Java Message Service (JMS) 与 Amazon SQS，请使用本节中的代码示例。以下部分介绍如何创建 JMS 连接和会话以及如何发送和接收消息。

Amazon SQS Java Messaging Library 中包含了封装的 Amazon SQS 客户端对象，该对象会检查是否存在 Amazon SQS 队列。如果队列不存在，客户端将创建它。

创建 JMS 连接

在开始之前，请先查看使用 JMS 和 Amazon SQS 的先决条件中的先决条件。

1. 创建连接工厂并对该工厂调用 `createConnection` 方法。

   ```
   // Create a new connection factory with all defaults (credentials and region) set
    automatically
   ```

```
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        AmazonSQSClientBuilder.defaultClient()
        );

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();
```

SQSConnection 类对 javax.jms.Connection 进行了扩展。除了 JMS 标准连接方法，SQSConnection 还提供其他一些方法，例如 getAmazonSQSClient 和 getWrappedAmazonSQSClient。这两种方法均可让您执行未包含在 JMS 规范中的管理操作，例如创建新队列。不过，getWrappedAmazonSQSClient 方法还提供当前连接使用的 Amazon SQS 客户端的封装版本。该包装程序将客户端中的每个异常转变为 JMSException，从而让预期有 JMSException 发生的现有代码更方便地使用该异常。

2. 您可以使用从 getAmazonSQSClient 和 getWrappedAmazonSQSClient 返回的客户端对象来执行未包含在 JMS 规范中的管理操作（例如，可以创建 Amazon SQS 队列）。

   如果您现有的代码需要 JMS 异常，则应使用 getWrappedAmazonSQSClient：

   - 如果您使用 getWrappedAmazonSQSClient，则返回的客户端对象会将所有异常转变成 JMS 异常。
   - 如果您使用 getAmazonSQSClient，则这些异常将全部为 Amazon SQS 异常。

创建 Amazon SQS 队列

包装的客户端对象将检查是否存在 Amazon SQS 队列。

如果队列不存在，客户端将创建它。如果队列存在，则该功能不会返回任何值。有关更多信息，请参阅 TextMessageSender.java 示例中的“根据需要创建队列”一节。

创建标准队列

```
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an SQS queue named MyQueue, if it doesn't already exist
if (!client.queueExists("MyQueue")) {
    client.createQueue("MyQueue");
}
```

创建 FIFO 队列

```
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an Amazon SQS FIFO queue named MyQueue.fifo, if it doesn't already exist
if (!client.queueExists("MyQueue.fifo")) {
    Map<String, String> attributes = new HashMap<String, String>();
    attributes.put("FifoQueue", "true");
    attributes.put("ContentBasedDeduplication", "true");
    client.createQueue(new
 CreateQueueRequest().withQueueName("MyQueue.fifo").withAttributes(attributes));
}
```

> **Note**
>
> FIFO 队列名称必须以 `.fifo` 后缀结尾。
> 有关 `ContentBasedDeduplication` 属性的更多信息，请参阅Amazon SQS 中的仅处理一次。

同步发送消息

1. 当连接和基础 Amazon SQS 队列准备就绪时，将创建一个具有 `AUTO_ACKNOWLEDGE` 模式的非事务性 JMS 会话。

   ```
   // Create the nontransacted session with AUTO_ACKNOWLEDGE mode
   Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
   ```

2. 为了向队列发送文本消息，将创建一个 JMS 队列标识和消息创建者。

   ```
   // Create a queue identity and specify the queue name to the session
   Queue queue = session.createQueue("MyQueue");

   // Create a producer for the 'MyQueue'
   MessageProducer producer = session.createProducer(queue);
   ```

3. 创建文本消息并将它发送到队列。

   - 要向标准队列发送消息，您无需设置任何其他参数。

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

- 要向 FIFO 队列发送消息，必须设置消息组 ID。您也可以设置消息重复数据删除 ID。有关更多信息，请参阅Amazon SQS FIFO 队列关键术语。

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Set the message group ID
message.setStringProperty("JMSXGroupID", "Default");

// You can also set a custom message deduplication ID
// message.setStringProperty("JMS_SQS_DeduplicationId", "hello");
// Here, it's not needed because content-based deduplication is enabled for the
 queue

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
System.out.println("JMS Message Sequence Number " +
 message.getStringProperty("JMS_SQS_SequenceNumber"));
```

同步接收消息

1. 要接收消息，可为同一队列创建一个使用者，然后调用 `start` 方法。

   您可以随时对连接调用 `start` 方法。不过，使用者不会开始接收消息，直至调用此方法。

```
// Create a consumer for the 'MyQueue'
MessageConsumer consumer = session.createConsumer(queue);
// Start receiving incoming messages
connection.start();
```

2. 在超时设为 1 秒钟的情况下对该使用者调用 `receive` 方法，然后输出收到的消息内容。

- 收到来自标准队列的消息后，可以访问消息的内容。

```
// Receive a message from 'MyQueue' and wait up to 1 second
Message receivedMessage = consumer.receive(1000);

// Cast the received message as TextMessage and display the text
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
}
```

- 在收到来自 FIFO 队列的消息后，您可以访问消息的内容和其他特定于 FIFO 的消息属性，例如消息组 ID、消息重复数据删除 ID 和序列号。有关更多信息，请参阅 Amazon SQS FIFO 队列关键术语。

```
// Receive a message from 'MyQueue' and wait up to 1 second
Message receivedMessage = consumer.receive(1000);

// Cast the received message as TextMessage and display the text
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
    System.out.println("Group id: " +
 receivedMessage.getStringProperty("JMSXGroupID"));
    System.out.println("Message deduplication id: " +
 receivedMessage.getStringProperty("JMS_SQS_DeduplicationId"));
    System.out.println("Message sequence number: " +
 receivedMessage.getStringProperty("JMS_SQS_SequenceNumber"));
}
```

3. 关闭连接和会话。

```
// Close the connection (and the session).
connection.close();
```

输出看上去类似于以下内容：

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

> **Note**
>
> 您可以使用 Spring Framework 初始化这些对象。
> 有关其他信息，请参阅 SpringExampleConfiguration.xml、SpringExample.java，以及ExampleConfiguration.java部分中的 ExampleCommon.java 和 实际可用的 Java 示例：将 JMS 与 Amazon SQS 标准队列相结合使用 中的其他帮助程序类。

有关发送和接收对象的完整示例，请参阅 TextMessageSender.java 和 SyncMessageReceiver.java。

异步接收消息

在 使用 Amazon SQS Java Messaging Library中的示例中，消息发送到 MyQueue 并被同步接收。

以下示例介绍如何通过监听器异步接收消息。

1. 实施 MessageListener 接口。

```
class MyListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        try {
            // Cast the received message as TextMessage and print the text to
 screen.
            System.out.println("Received: " + ((TextMessage) message).getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

   在收到消息时，调用 onMessage 接口的 MessageListener 方法。在此监听器实现中，输出保存在消息中的文本。

2. 对 receive 实现实例设置使用者消息监听器，而不是对使用者显式调用 MyListener 方法。主线程会等待一秒钟。

```
// Create a consumer for the 'MyQueue'.
MessageConsumer consumer = session.createConsumer(queue);
```

```
// Instantiate and set the message listener for the consumer.
consumer.setMessageListener(new MyListener());

// Start receiving incoming messages.
connection.start();

// Wait for 1 second. The listener onMessage() method is invoked when a message is
 received.
Thread.sleep(1000);
```

其余步骤与 使用 Amazon SQS Java Messaging Library示例中的步骤相同。有关异步使用者的完整示例，请参阅AsyncMessageReceiver.java中的 实际可用的 Java 示例：将 JMS 与 Amazon SQS 标准队列相结合使用。

此示例的输出将与以下内容类似：

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

使用客户端确认模式

使用 Amazon SQS Java Messaging Library中的示例使用 AUTO_ACKNOWLEDGE 模式，该模式会自动确认收到的每条消息（因此会从基础 Amazon SQS 队列中删除消息）。

1. 要在消息处理完毕后显式确认消息，则必须创建具有 CLIENT_ACKNOWLEDGE 模式的会话。

```
// Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

2. 收到消息时，显示消息，然后显式确认消息。

```
// Cast the received message as TextMessage and print the text to screen. Also
 acknowledge the message.
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
    receivedMessage.acknowledge();
    System.out.println("Acknowledged: " + message.getJMSMessageID());
}
```

> **Note**
>
> 在此模式下，当确认某条消息时，也会隐式确认在该消息之前收到的所有消息。例如，如果收到 10 条消息，则仅确认第 10 条消息（按接收消息的顺序），然后还会确认先前的所有 9 条消息。

其余步骤与 使用 Amazon SQS Java Messaging Library示例中的步骤相同。有关具有客户端确认模式的同步使用者的完整示例，请参阅SyncMessageReceiverClientAcknowledge.java中的 实际可用的 Java 示例：将 JMS 与 Amazon SQS 标准队列相结合使用。

此示例的输出将与以下内容类似：

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5
Received: Hello World!
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

使用无序确认模式

在使用 CLIENT_ACKNOWLEDGE 模式时，将自动确认在显式确认的消息之前收到的所有消息。有关更多信息，请参阅 使用客户端确认模式。

Amazon SQS Java Messaging Library 提供另一种确认模式。在使用 UNORDERED_ACKNOWLEDGE 模式时，客户端必须单独显式确认收到的所有消息，不管消息的接收顺序如何。为此，请使用 UNORDERED_ACKNOWLEDGE 模式创建会话。

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

其余步骤与 使用客户端确认模式示例中的步骤相同。有关具有 UNORDERED_ACKNOWLEDGE 模式的同步使用者的完整示例，请参阅 SyncMessageReceiverUnorderedAcknowledge.java。

此示例中的输出将与以下内容类似：

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7
Received: Hello World!
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

将 Java Message Service 客户端与其他 Amazon SQS 客户端配合使用

使用带软件开发工具包的亚马逊 SQS Java 消息服务 (JMS) 客户端将亚马逊 AWS SQS 消息大小限制为 256 KB。不过，您可以使用任何 Amazon SQS 客户端创建 JMS 提供程序。例如，可以使用 JMS 客户端与适用于 Java 的 Amazon SQS 扩展型客户端库来发送包含对 Amazon S3 中的消息负载（最大为 2 GB）的引用的 Amazon SQS 消息。有关更多信息，请参阅 使用 Java 和 Amazon S3 管理大型 Amazon SQS 消息。

以下 Java 代码示例将为扩展型客户端库创建 JMS 提供程序。

在测试此示例之前，请参阅使用 JMS 和 Amazon SQS 的先决条件中的先决条件。

```
AmazonS3 s3 = new AmazonS3Client(credentials);
Region s3Region = Region.getRegion(Regions.US_WEST_2);
s3.setRegion(s3Region);

// Set the Amazon S3 bucket name, and set a lifecycle rule on the bucket to
// permanently delete objects a certain number of days after each object's creation
 date.
// Next, create the bucket, and enable message objects to be stored in the bucket.
BucketLifecycleConfiguration.Rule expirationRule = new
 BucketLifecycleConfiguration.Rule();
expirationRule.withExpirationInDays(14).withStatus("Enabled");
BucketLifecycleConfiguration lifecycleConfig = new
 BucketLifecycleConfiguration().withRules(expirationRule);

s3.createBucket(s3BucketName);
s3.setBucketLifecycleConfiguration(s3BucketName, lifecycleConfig);
System.out.println("Bucket created and configured.");

// Set the SQS extended client configuration with large payload support enabled.
ExtendedClientConfiguration extendedClientConfig = new ExtendedClientConfiguration()
    .withLargePayloadSupportEnabled(s3, s3BucketName);

AmazonSQS sqsExtended = new AmazonSQSExtendedClient(new AmazonSQSClient(credentials),
 extendedClientConfig);
Region sqsRegion = Region.getRegion(Regions.US_WEST_2);
sqsExtended.setRegion(sqsRegion);
```

以下 Java 代码示例将创建连接工厂：

```
// Create the connection factory using the environment variable credential provider.
// Pass the configured Amazon SQS Extended Client to the JMS connection factory.
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        sqsExtended
        );

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();
```

实际可用的 Java 示例：将 JMS 与 Amazon SQS 标准队列相结合使用

以下代码示例说明如何结合使用 Java Message Service (JMS) 与 Amazon SQS 标准队列。有关使用 FIFO 队列的更多信息，请参阅创建 FIFO 队列、同步发送消息和同步接收消息。（标准队列和 FIFO 队列的同步接收消息是相同的。但是，FIFO 队列中的消息包含更多属性。）

在测试以下示例之前，请参阅使用 JMS 和 Amazon SQS 的先决条件中的先决条件。

ExampleConfiguration.java

以下 Java SDK（版本 1.x）代码示例设置将用于其他 Java 示例的默认队列名称、区域和凭证。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class ExampleConfiguration {
    public static final String DEFAULT_QUEUE_NAME = "SQSJMSClientExampleQueue";
```

```
    public static final Region DEFAULT_REGION = Region.getRegion(Regions.US_EAST_2);

    private static String getParameter( String args[], int i ) {
        if( i + 1 >= args.length ) {
            throw new IllegalArgumentException( "Missing parameter for " + args[i] );
        }
        return args[i+1];
    }

    /**
     * Parse the command line and return the resulting config. If the config parsing
fails
     * print the error and the usage message and then call System.exit
     *
     * @param app the app to use when printing the usage string
     * @param args the command line arguments
     * @return the parsed config
     */
    public static ExampleConfiguration parseConfig(String app, String args[]) {
        try {
            return new ExampleConfiguration(args);
        } catch (IllegalArgumentException e) {
            System.err.println( "ERROR: " + e.getMessage() );
            System.err.println();
            System.err.println( "Usage: " + app + " [--queue <queue>] [--region
<region>] [--credentials <credentials>] ");
            System.err.println( "  or" );
            System.err.println( "       " + app + " <spring.xml>" );
            System.exit(-1);
            return null;
        }
    }

    private ExampleConfiguration(String args[]) {
        for( int i = 0; i < args.length; ++i ) {
            String arg = args[i];
            if( arg.equals( "--queue" ) ) {
                setQueueName(getParameter(args, i));
                i++;
            } else if( arg.equals( "--region" ) ) {
                String regionName = getParameter(args, i);
                try {
                    setRegion(Region.getRegion(Regions.fromName(regionName)));
```

```
                } catch( IllegalArgumentException e ) {
                    throw new IllegalArgumentException( "Unrecognized region " +
regionName );
                }
                i++;
            } else if( arg.equals( "--credentials" ) ) {
                String credsFile = getParameter(args, i);
                try {
                    setCredentialsProvider( new
PropertiesFileCredentialsProvider(credsFile) );
                } catch (AmazonClientException e) {
                    throw new IllegalArgumentException("Error reading credentials from
" + credsFile, e );
                }
                i++;
            } else {
                throw new IllegalArgumentException("Unrecognized option " + arg);
            }
        }
    }

    private String queueName = DEFAULT_QUEUE_NAME;
    private Region region = DEFAULT_REGION;
    private AWSCredentialsProvider credentialsProvider = new
DefaultAWSCredentialsProviderChain();

    public String getQueueName() {
        return queueName;
    }

    public void setQueueName(String queueName) {
        this.queueName = queueName;
    }

    public Region getRegion() {
        return region;
    }

    public void setRegion(Region region) {
        this.region = region;
    }

    public AWSCredentialsProvider getCredentialsProvider() {
        return credentialsProvider;
```

```
    }

    public void setCredentialsProvider(AWSCredentialsProvider credentialsProvider) {
        // Make sure they're usable first
        credentialsProvider.getCredentials();
        this.credentialsProvider = credentialsProvider;
    }
}
```

TextMessageSender.java

以下 Java 代码示例创建文本消息创建者。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class TextMessageSender {
    public static void main(String args[]) throws JMSException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("TextMessageSender", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );
```

```
        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer =
session.createProducer( session.createQueue( config.getQueueName() ) );

        sendMessages(session, producer);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }

    private static void sendMessages( Session session, MessageProducer producer ) {
        BufferedReader inputReader = new BufferedReader(
            new InputStreamReader( System.in, Charset.defaultCharset() ) );

        try {
            String input;
            while( true ) {
                System.out.print( "Enter message to send (leave empty to exit): " );
                input = inputReader.readLine();
                if( input == null || input.equals("" ) ) break;

                TextMessage message = session.createTextMessage(input);
                producer.send(message);
                System.out.println( "Send message " + message.getJMSMessageID() );
            }
        } catch (EOFException e) {
            // Just return on EOF
        } catch (IOException e) {
            System.err.println( "Failed reading input: " + e.getMessage() );
        } catch (JMSException e) {
            System.err.println( "Failed sending message: " + e.getMessage() );
            e.printStackTrace();
        }
    }
```

```
}
```

SyncMessageReceiver.java

以下 Java 代码示例创建同步消息使用者。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class SyncMessageReceiver {
public static void main(String args[]) throws JMSException {
    ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiver", args);

    ExampleCommon.setupLogging();

    // Create the connection factory based on the config
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                    .withRegion(config.getRegion().getName())
                    .withCredentials(config.getCredentialsProvider())
            );

    // Create the connection
    SQSConnection connection = connectionFactory.createConnection();

    // Create the queue if needed
    ExampleCommon.ensureQueueExists(connection, config.getQueueName());
```

```
    // Create the session
    Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
    MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

    connection.start();

    receiveMessages(session, consumer);

    // Close the connection. This closes the session automatically
    connection.close();
    System.out.println( "Connection closed" );
}

private static void receiveMessages( Session session, MessageConsumer consumer ) {
    try {
        while( true ) {
            System.out.println( "Waiting for messages");
            // Wait 1 minute for a message
            Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
            if( message == null ) {
                System.out.println( "Shutting down after 1 minute of silence" );
                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message " + message.getJMSMessageID() );
        }
    } catch (JMSException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
}
```

AsyncMessageReceiver.java

以下 Java 代码示例创建异步消息使用者。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
```

```
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class AsyncMessageReceiver {
    public static void main(String args[]) throws JMSException, InterruptedException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("AsyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

        // No messages are processed until this is called
        connection.start();

        ReceiverCallback callback = new ReceiverCallback();
        consumer.setMessageListener( callback );
```

```
        callback.waitForOneMinuteOfSilence();
        System.out.println( "Returning after one minute of silence" );

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }


    private static class ReceiverCallback implements MessageListener {
        // Used to listen for message silence
        private volatile long timeOfLastMessage = System.nanoTime();

        public void waitForOneMinuteOfSilence() throws InterruptedException {
            for(;;) {
                long timeSinceLastMessage = System.nanoTime() - timeOfLastMessage;
                long remainingTillOneMinuteOfSilence =
                    TimeUnit.MINUTES.toNanos(1) - timeSinceLastMessage;
                if( remainingTillOneMinuteOfSilence < 0 ) {
                    break;
                }
                TimeUnit.NANOSECONDS.sleep(remainingTillOneMinuteOfSilence);
            }
        }


        @Override
        public void onMessage(Message message) {
            try {
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message " +
 message.getJMSMessageID() );
                timeOfLastMessage = System.nanoTime();
            } catch (JMSException e) {
                System.err.println( "Error processing message: " + e.getMessage() );
                e.printStackTrace();
            }
        }
    }
}
```

SyncMessageReceiverClientAcknowledge.java

以下 Java 代码示例创建具有客户端确认模式的同步使用者。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

/**
 * An example class to demonstrate the behavior of CLIENT_ACKNOWLEDGE mode for received
 messages. This example
 * complements the example given in {@link SyncMessageReceiverUnorderedAcknowledge} for
 UNORDERED_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that no message is returned for this
 attempt since in CLIENT_ACKNOWLEDGE mode,
 * as expected, all the messages prior to the acknowledged messages are also
 acknowledged.
 *
 * This ISN'T the behavior for UNORDERED_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverUnorderedAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverClientAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
```

```
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
ExampleConfiguration.parseConfig("SyncMessageReceiverClientAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with client acknowledge mode
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

        // Open the connection
        connection.start();

        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
```

```
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
receiving no messages since
        // we have acknowledged the second message. Although we didn't explicitly
acknowledge the first message,
        // in the CLIENT_ACKNOWLEDGE mode, all the messages received prior to the
explicitly acknowledged message
        // are also acknowledged. Therefore, we have implicitly acknowledged the first
message.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
(TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
"Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
```

```
     * @param acknowledge If true and a message is received, the received message is
 acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
 throws JMSException {
        // Receive a message
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
}
```

SyncMessageReceiverUnorderedAcknowledge.java

以下 Java 代码示例创建具有无序确认模式的同步使用者。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */
```

```
/**
 * An example class to demonstrate the behavior of UNORDERED_ACKNOWLEDGE mode for
 received messages. This example
 * complements the example given in {@link SyncMessageReceiverClientAcknowledge} for
 CLIENT_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that the first message received in the
 prior attempt is returned again
 * for the second attempt. In UNORDERED_ACKNOWLEDGE mode, all the messages must be
 explicitly acknowledged no matter what
 * the order they're received.
 *
 * This ISN'T the behavior for CLIENT_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverClientAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverUnorderedAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiverUnorderedAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();
```

```
        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with unordered acknowledge mode
        Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

        // Open the connection
        connection.start();

        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
receiving the first message since
        // we have acknowledged only the second message. In the UNORDERED_ACKNOWLEDGE
mode, all the messages must
        // be explicitly acknowledged.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }
```

```
    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
(TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
"Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSException {
        // Receive a message
        Message message =
consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
```

```
}
```

SpringExampleConfiguration.xml

以下 XML 代码示例是 SpringExample.java 的 Bean 配置文件。

```
<!--
    Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.

    Licensed under the Apache License, Version 2.0 (the "License").
    You may not use this file except in compliance with the License.
    A copy of the License is located at

    https://aws.amazon.com/apache2.0

    or in the "license" file accompanying this file. This file is distributed
    on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
    express or implied. See the License for the specific language governing
    permissions and limitations under the License.
-->

<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/
schema/util/spring-util-3.0.xsd
    ">

    <bean id="CredentialsProviderBean"
 class="com.amazonaws.auth.DefaultAWSCredentialsProviderChain"/>

    <bean id="ClientBuilder" class="com.amazonaws.services.sqs.AmazonSQSClientBuilder"
 factory-method="standard">
        <property name="region" value="us-east-2"/>
        <property name="credentials" ref="CredentialsProviderBean"/>
    </bean>
```

```
    <bean id="ProviderConfiguration"
 class="com.amazon.sqs.javamessaging.ProviderConfiguration">
        <property name="numberOfMessagesToPrefetch" value="5"/>
    </bean>

    <bean id="ConnectionFactory"
 class="com.amazon.sqs.javamessaging.SQSConnectionFactory">
        <constructor-arg ref="ProviderConfiguration" />
        <constructor-arg ref="ClientBuilder" />
    </bean>

    <bean id="Connection" class="javax.jms.Connection"
        factory-bean="ConnectionFactory"
        factory-method="createConnection"
        init-method="start"
        destroy-method="close" />

    <bean id="QueueName" class="java.lang.String">
        <constructor-arg value="SQSJMSClientExampleQueue"/>
    </bean>
</beans>
```

SpringExample.java

以下 Java 代码示例使用 Bean 配置文件来初始化您的对象。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class SpringExample {
```

```
    public static void main(String args[]) throws JMSException {
        if( args.length != 1 || !args[0].endsWith(".xml")) {
            System.err.println( "Usage: " + SpringExample.class.getName() + " <spring
config.xml>" );
            System.exit(1);
        }

        File springFile = new File( args[0] );
        if( !springFile.exists() || !springFile.canRead() ) {
            System.err.println( "File " + args[0] + " doesn't exist or isn't
readable.");
            System.exit(2);
        }

        ExampleCommon.setupLogging();

        FileSystemXmlApplicationContext context =
            new FileSystemXmlApplicationContext( "file://" +
springFile.getAbsolutePath() );

        Connection connection;
        try {
            connection = context.getBean(Connection.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the JMS connection to use: " +
e.getMessage() );
            System.exit(3);
            return;
        }

        String queueName;
        try {
            queueName = context.getBean("QueueName", String.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the name of the queue to use: " +
e.getMessage() );
            System.exit(3);
            return;
        }

        if( connection instanceof SQSConnection ) {
            ExampleCommon.ensureQueueExists( (SQSConnection) connection, queueName );
        }
```

```
        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
 session.createConsumer( session.createQueue( queueName) );

        receiveMessages(session, consumer);

        // The context can be setup to close the connection for us
        context.close();
        System.out.println( "Context closed" );
    }

    private static void receiveMessages( Session session, MessageConsumer consumer ) {
        try {
            while( true ) {
                System.out.println( "Waiting for messages");
                // Wait 1 minute for a message
                Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
                if( message == null ) {
                    System.out.println( "Shutting down after 1 minute of silence" );
                    break;
                }
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message" );
            }
        } catch (JMSException e) {
            System.err.println( "Error receiving from SQS: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

ExampleCommon.java

以下 Java 代码示例首先查看 Amazon SQS 队列是否存在，如果不存在，则会创建一个。它还包含示例日历记录代码。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
```

```
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class ExampleCommon {
    /**
     * A utility function to check the queue exists and create it if needed. For most
     * use cases this is usually done by an administrator before the application is
 run.
     */
    public static void ensureQueueExists(SQSConnection connection, String queueName)
 throws JMSException {
        AmazonSQSMessagingClientWrapper client =
 connection.getWrappedAmazonSQSClient();

        /**
         * In most cases, you can do this with just a createQueue call, but
 GetQueueUrl
         * (called by queueExists) is a faster operation for the common case where the
 queue
         * already exists. Also many users and roles have permission to call
 GetQueueUrl
         * but don't have permission to call CreateQueue.
         */
        if( !client.queueExists(queueName) ) {
            client.createQueue( queueName );
        }
    }

    public static void setupLogging() {
        // Setup logging
        BasicConfigurator.configure();
        Logger.getRootLogger().setLevel(Level.WARN);
    }

    public static void handleMessage(Message message) throws JMSException {
        System.out.println( "Got message " + message.getJMSMessageID() );
```

```
        System.out.println( "Content: ");
        if( message instanceof TextMessage ) {
            TextMessage txtMessage = ( TextMessage ) message;
            System.out.println( "\t" + txtMessage.getText() );
        } else if( message instanceof BytesMessage ){
            BytesMessage byteMessage = ( BytesMessage ) message;
            // Assume the length fits in an int - SQS only supports sizes up to 256k so
 that
            // should be true
            byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
            byteMessage.readBytes(bytes);
            System.out.println( "\t" +  Base64.encodeAsString( bytes ) );
        } else if( message instanceof ObjectMessage ) {
            ObjectMessage objMessage = (ObjectMessage) message;
            System.out.println( "\t" + objMessage.getObject() );
        }
    }
}
```

Amazon SQS 支持的 JMS 1.1 实现

Amazon SQS Java Messaging Library 支持以下 JMS 1.1 实施。有关 Amazon SQS Java Messaging Library 支持的特征和功能的更多信息，请参阅 Amazon SQS 常见问题。

支持的常用接口

- `Connection`
- `ConnectionFactory`
- `Destination`
- `Session`
- `MessageConsumer`
- `MessageProducer`

支持的消息类型

- `ByteMessage`
- `ObjectMessage`
- `TextMessage`

支持的消息确认模式

- AUTO_ACKNOWLEDGE
- CLIENT_ACKNOWLEDGE
- DUPS_OK_ACKNOWLEDGE
- UNORDERED_ACKNOWLEDGE

> **Note**
>
> UNORDERED_ACKNOWLEDGE 模式不属于 JMS 1.1 规范。此模式有助于 Amazon SQS 允许 JMS 客户端显式确认消息。

JMS 定义的标头和预留属性

发送消息

在发送消息时，您可以为每条消息设置以下标头和属性：

- JMSXGroupID（对于 FIFO 队列是必需的，对于标准队列是不允许的）
- JMS_SQS_DeduplicationId（对于 FIFO 队列是可选的，对于标准队列是不允许的）

在发送消息后，Amazon SQS 将为每条消息设置以下标头和属性：

- JMSMessageID
- JMS_SQS_SequenceNumber（仅适用于 FIFO 队列）

接收消息

在接收消息时，Amazon SQS 将为每条消息设置以下标头和属性：

- JMSDestination
- JMSMessageID
- JMSRedelivered
- JMSXDeliveryCount

- JMSXGroupID（仅适用于 FIFO 队列）
- JMS_SQS_DeduplicationId（仅适用于 FIFO 队列）
- JMS_SQS_SequenceNumber（仅适用于 FIFO 队列）

Amazon SQS 教程

本主题提供了教程，帮助您探索 Amazon SQS 的各项特征和功能。

教程

- 使用创建 Amazon SQS 队列 CloudFormation
- 教程：从 Amazon Virtual Private Cloud 将消息发送到 Amazon SQS 队列

使用创建 Amazon SQS 队列 CloudFormation

使用 CloudFormation 控制台和 JSON 或 YAML 模板来创建亚马逊 SQS 队列。有关更多详情，请参阅《AWS CloudFormation 用户指南》中的使用 CloudFormation 模板和 AWS::SQS::Queue 资源。

要将 CloudFormation 创建亚马逊 SQS 队列。

1. 将以下 JSON 代码复制到名为 MyQueue.json 的文件中。要创建标准队列，请省略 FifoQueue 和 ContentBasedDeduplication 属性。有关基于内容的重复数据删除的更多信息，请参阅Amazon SQS 中的仅处理一次。

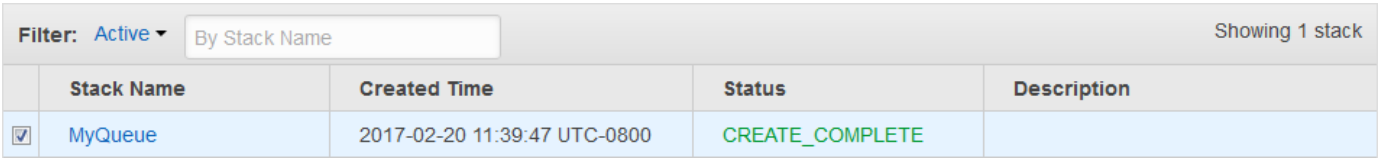
   > **Note**
   >
   > FIFO 队列名称必须以 .fifo 后缀结尾。

```
{
   "AWSTemplateFormatVersion": "2010-09-09",
   "Resources": {
      "MyQueue": {
         "Properties": {
            "QueueName": "MyQueue.fifo",
            "FifoQueue": true,
            "ContentBasedDeduplication": true
             },
         "Type": "AWS::SQS::Queue"
         }
      },
   "Outputs": {
      "QueueName": {
```

```
         "Description": "The name of the queue",
         "Value": {
            "Fn::GetAtt": [
               "MyQueue",
               "QueueName"
            ]
         }
      },
      "QueueURL": {
         "Description": "The URL of the queue",
         "Value": {
            "Ref": "MyQueue"
         }
      },
      "QueueARN": {
         "Description": "The ARN of the queue",
         "Value": {
            "Fn::GetAtt": [
               "MyQueue",
               "Arn"
            ]
         }
      }
   }
}
```

2. 登录 CloudFormation 控制台，然后选择创建堆栈。
3. 在指定模板面板上，选择上传模板文件，选择您的 MyQueue.json 文件，然后选择下一步。
4. 在指定详细信息页面上，为堆栈名称键入 MyQueue，然后选择下一步。
5. 在选项页面上，选择下一步。
6. 在 Review 页面上，选择 Create 。

   CloudFormation 开始创建MyQueue堆栈并显示 CREATE_IN_PROGRESS 状态。在此过程完成后， CloudFormation 将显示 CREATE_COMPLETE 状态。



7. （可选）要显示队列的名称、URL 和 ARN，请选择堆栈的名称，然后在下一页上展开 Outputs 部分。

教程：从 Amazon Virtual Private Cloud 将消息发送到 Amazon SQS 队列

本教程将向您演示如何通过安全的私有网络将消息发送到 Amazon SQS 队列。该网络包括：

- 一个包含 Amazon EC2 实例的 VPC。
- 一个接口 VPC 端点，它允许 Amazon EC2 实例在未使用公共互联网的情况下连接到 Amazon SQS。

即使在完全私有网络中，您也可以连接到 Amazon EC2 实例并将消息发送到 Amazon SQS 队列。有关更多信息，请参阅 Amazon SQS 的 Amazon Virtual Private Cloud 端点。

Important

- 您只能将 Amazon Virtual Private Cloud 与 HTTPS Amazon SQS 端点一起使用。
- 当您将 Amazon SQS 配置为从 Amazon VPC 发送消息时，必须启用私有 DNS，并为双栈端点指定格式为 `sqs.us-east-2.amazonaws.com` 或 `sqs.us-east-2.api.aws` 的端点。
- Amazon SQS 还通过 PrivateLink 使用终端节点服务支持 FIPS 终端节点。`com.amazonaws.region.sqs-fips`您可以连接到格式为 `sqs-fips.region.amazonaws.com` 的 FIPS 端点。
- 在 Amazon Virtual Private Cloud 中使用双栈终端节点时，将使用 IPv4 和 IPv6发送请求。
- 私有 DNS 不支持传统端点，例如 `queue.amazonaws.com` 或 `us-east-2.queue.amazonaws.com`。

步骤 1：创建 Amazon EC2 密钥对

利用密钥对，您可以连接到 Amazon EC2 实例。它包含一个用于加密您的登录信息的公有密钥和一个用于解密该信息的私有密钥。

1. 登录 Amazon EC2 控制台。
2. 在导航菜单上的网络和安全性下，选择密钥对。
3. 选择创建密钥对。

4. 在创建密钥对对话框中，对于密钥对名称，输入 SQS-VPCE-Tutorial-Key-Pair 并选择创建。
5. 您的浏览器会自动下载私有密钥文件 SQS-VPCE-Tutorial-Key-Pair.pem。

> **Important**
>
> 将此文件保存在安全位置。EC2 不会再次为同一密钥对生成 .pem 文件。

6. 要允许 SSH 客户端连接到您的 EC2 实例，请设置私有密钥文件的权限，以便只有您的用户有权读取该文件，例如：

```
chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
```

步骤 2：创建 AWS 资源

要设置必要的基础设施，您必须使用 CloudFormation 模板，该模板是创建由 Amazon EC2 实例和 Amazon SQS 队列等 AWS 资源组成的堆栈的蓝图。

本教程的堆栈包括以下资源：

- VPC 和关联的网络资源，包括子网、安全组、Internet 网关和路由表。
- 在 VPC 子网中启动的 Amazon EC2 实例
- 一个 Amazon SQS 队列

1. SQS-VPCE-Tutorial-CloudFormation.yaml从下载名为的 CloudFormation 模板GitHub。
2. 登录 CloudFormation 控制台。
3. 选择创建堆栈。
4. 在选择模板页面上，依次选择将模板上传到 Amazon S3、SQS-VPCE-SQS-Tutorial-CloudFormation.yaml 文件和下一步。
5. 在指定详细信息页面中，执行以下操作：
   a. 对于堆栈名称，输入 SQS-VPCE-Tutorial-Stack。
   b. 对于 KeyName，请选择 SQS-vpce-Tutorial-Key-p air。
   c. 选择下一步。

6. 在选项页面上，选择下一步。
7. 在 “查看” 页面的 “能力” 部分，选择我确认这 AWS CloudFormation 可能会创建带有自定义名称的 IAM 资源。 ，然后选择 “创建”。

CloudFormation 开始创建堆栈并显示 CREATE_IN_ PROGRESS 状态。在此过程完成后，CloudFormation 将显示 CREATE_COMPLETE 状态。

步骤 3：确认您的 EC2 实例不可公开访问

您的 CloudFormation 模板将在您的 VPC 中启动一个名`SQS-VPCE-Tutorial-EC2-Instance`为 EC2 实例。此 EC2 实例不允许出站流量，并且无法将消息发送到 Amazon SQS。要验证这一点，您必须连接到该实例，尝试连接到公有端点，然后尝试将消息发送到 Amazon SQS。

1. 登录 Amazon EC2 控制台。
2. 在导航菜单上的实例下，选择实例。
3. 选择 SQS-VPCE-。Tutorial-EC2Instance
4. 复制公有 DNS 下的主机名，例如 ec2-203-0-113-0.us-west-2.compute.amazonaws.com。
5. 从包含您之前创建的密钥对的目录中，使用以下命令连接到实例，例如：

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-
east-2.compute.amazonaws.com
```

6. 尝试连接到任何公有端点，例如：

```
ping amazon.com
```

   正如预期的那样，连接尝试失败。

7. 登录 Amazon SQS 控制台。
8. 从队列列表中，选择由您的 CloudFormation 模板创建的队列，例如 vpce-sqs-Tutorial-Stack-- 1 IJK。CFQueue ABCDEFGH2
9. 在详细信息表中，复制 URL，例如https://sqs.us-east-2.amazonaws.com/123456789012/。
10. 从您的 EC2 实例，尝试使用以下命令向该队列发布消息，例如：

```
aws sqs send-message --region us-east-2 --endpoint-url https://sqs.us-
east-2.amazonaws.com/ --queue-url https://sqs.us-east-2.amazonaws.com/123456789012/
 --message-body "Hello from Amazon SQS."
```

正如预期的那样，发送尝试失败。

> **Important**
>
> 稍后，当您为 Amazon SQS 创建 VPC 端点时，您的发送尝试将成功。

步骤 4：为 Amazon SQS 创建 Amazon VPC 端点

要将您的 VPC 连接到 Amazon SQS，您必须定义一个接口 VPC 端点。添加端点后，可以使用 VPC 中 EC2 实例的 Amazon SQS API。这使您无需通过公共 Internet 即可向 AWS 网络中的队列发送消息。

> **Note**
>
> EC2 实例仍然无法访问互联网上的其他 AWS 服务和终端节点。

1. 登录 Amazon VPC 控制台。
2. 在导航菜单上，选择端点。
3. 选择创建端点。
4. 在创建端点页面上，对于服务名称，选择 Amazon SQS 的服务名称。

   > **Note**
   >
   > 服务名称因当前 AWS 区域而异。例如，如果您在美国东部（俄亥俄州），则服务名称为 com.amazonaws。*us-east-2*.sqs。

5. 对于 VPC，选择 SQS-VPCE-Tutorial-VPC。
6. 对于子网，选择其子网 ID 包含 SQS-VPCE-Tutorial-Subnet 的子网。
7. 对于安全组，选择选择安全组，然后选择其组名称包含 SQS VPCE Tutorial Security Group 的安全组。
8. 选择创建端点。

   创建接口 VPC 端点并显示其 ID，例如 vpce-0ab1cdef2ghi3j456k。
9. 选择关闭。

Amazon VPC 控制台会打开端点页面。

Amazon VPC 开始创建端点，并显示待处理状态。在此过程完成后，Amazon VPC 将显示可用状态。

步骤 5：向 Amazon SQS 队列发送消息

现在您的 VPC 包含 Amazon SQS 的端点，您可以连接到您的 EC2 实例并将消息发送到您的队列。

1. 重新连接到您的 EC2 实例，例如：

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-
east-2.compute.amazonaws.com
```

2. 重新尝试使用以下命令向该队列发布消息，例如：

```
aws sqs send-message --region us-east-2 --endpoint-url https://sqs.us-
east-2.amazonaws.com/ --queue-url https://sqs.us-east-2.amazonaws.com/123456789012/
 --message-body "Hello from Amazon SQS."
```

发送尝试成功并显示消息正 MD5 文摘要和消息 ID，例如：

```
{
 "MD5OfMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
 "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
}
```

有关从您的 CloudFormation 模板创建的队列中接收和删除消息的信息（例如，vpce-sqs-Tutorial-Stack-- 1 IJK），请参阅。CFQueue ABCDEFGH2 在 Amazon SQS 中接收和删除消息

有关删除资源的信息，请参阅以下内容：

- 《Amazon VPC 用户指南》中的删除 VPC 端点
- 删除 Amazon SQS 队列
- 《Amazon EC2 用户指南》中的终止实例
- 《Amazon VPC 用户指南》中的删除 VPC
- 《AWS CloudFormation 用户指南》中的 “在 CloudFormation 控制台上删除堆栈”
- 《Amazon EC2 用户指南》中的删除密钥对

使用 Amazon SQS 的代码示例 AWS SDKs

以下代码示例展示了如何将 Amazon SQS 与 AWS 软件开发套件 (SDK) 配合使用。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 AWS 服务服务结合来完成特定任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

代码示例

- 使用 Amazon SQS 的基本示例 AWS SDKs
  - 开始使用 Amazon SQS
  - 使用 Amazon SQS 执行的操作 AWS SDKs
    - 将 AddPermission 与 CLI 配合使用
    - ChangeMessageVisibility与 AWS SDK 或 CLI 配合使用
    - 将 ChangeMessageVisibilityBatch 与 CLI 配合使用
    - CreateQueue与 AWS SDK 或 CLI 配合使用
    - DeleteMessage与 AWS SDK 或 CLI 配合使用
    - DeleteMessageBatch与 AWS SDK 或 CLI 配合使用
    - DeleteQueue与 AWS SDK 或 CLI 配合使用
    - GetQueueAttributes与 AWS SDK 或 CLI 配合使用
    - GetQueueUrl与 AWS SDK 或 CLI 配合使用
    - 将 ListDeadLetterSourceQueues 与 CLI 配合使用
    - ListQueues与 AWS SDK 或 CLI 配合使用
    - 将 PurgeQueue 与 CLI 配合使用
    - ReceiveMessage与 AWS SDK 或 CLI 配合使用
    - 将 RemovePermission 与 CLI 配合使用
    - SendMessage与 AWS SDK 或 CLI 配合使用
    - SendMessageBatch与 AWS SDK 或 CLI 配合使用
    - SetQueueAttributes与 AWS SDK 或 CLI 配合使用

- 使用 Amazon SQS 的场景 AWS SDKs
  - 使用 Amazon SQS 创建用于发送和检索消息的 Web 应用程序
  - 使用 Step Functions 创建 Messenger 应用程序
  - 创建 Amazon Textract 浏览器应用程序
  - 使用软件开发工具包创建并发布到 FIFO Amazon SNS 主题 AWS
  - 使用 Amazon Rekognition 使用软件开发工具包检测视频中的人物和物体 AWS
  - 使用带软件开发工具包的 Amazon S3 管理大型亚马逊 SQS 消息 AWS
  - 使用软件开发工具包接收和处理 Amazon AWS S3 事件通知
  - 使用软件开发工具包将 Amazon SNS 消息发布到亚马逊 SQS 队列 AWS
  - 使用软件开发工具包通过 Amazon SQS 发送和接收批量消息 AWS
  - 使用适用于.NET 的 AWS 消息处理框架发布和接收 Amazon SQS 消息
  - 使用 Amazon SQS Java Messaging Library 处理 Amazon SQS 的 Java Message Service（JMS）接口
  - 使用软件开发工具包处理队列标签和 Amazon SQS AWS
- Amazon SQS 无服务器示例
  - 通过 Amazon SQS 触发器调用 Lambda 函数
  - 报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败

使用 Amazon SQS 的基本示例 AWS SDKs

以下代码示例展示了如何将 Amazon 简单队列服务的基础知识与一起使用 AWS SDKs。

示例

- 开始使用 Amazon SQS
- 使用 Amazon SQS 执行的操作 AWS SDKs
  - 将 AddPermission 与 CLI 配合使用
  - ChangeMessageVisibility与 AWS SDK 或 CLI 配合使用
  - 将 ChangeMessageVisibilityBatch 与 CLI 配合使用
  - CreateQueue与 AWS SDK 或 CLI 配合使用
  - DeleteMessage与 AWS SDK 或 CLI 配合使用
  - DeleteMessageBatch与 AWS SDK 或 CLI 配合使用
  - DeleteQueue与 AWS SDK 或 CLI 配合使用

- GetQueueAttributes与 AWS SDK 或 CLI 配合使用
- GetQueueUrl与 AWS SDK 或 CLI 配合使用
- 将 ListDeadLetterSourceQueues 与 CLI 配合使用
- ListQueues与 AWS SDK 或 CLI 配合使用
- 将 PurgeQueue 与 CLI 配合使用
- ReceiveMessage与 AWS SDK 或 CLI 配合使用
- 将 RemovePermission 与 CLI 配合使用
- SendMessage与 AWS SDK 或 CLI 配合使用
- SendMessageBatch与 AWS SDK 或 CLI 配合使用
- SetQueueAttributes与 AWS SDK 或 CLI 配合使用

开始使用 Amazon SQS

以下代码示例展示了如何开始使用 Amazon SQS。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSActions;

public static class HelloSQS
{
    static async Task Main(string[] args)
    {
        var sqsClient = new AmazonSQSClient();
```

```
        Console.WriteLine($"Hello Amazon SQS! Following are some of your
 queues:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five queues.
        var response = await sqsClient.ListQueuesAsync(
            new ListQueuesRequest()
            {
                MaxResults = 5
            });

        foreach (var queue in response.QueueUrls)
        {
            Console.WriteLine($"\tQueue Url: {queue}");
            Console.WriteLine();
        }
    }
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListQueues中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CMakeLists.txt CMake 文件的代码。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS sqs)

# Set this project's name.
```

```
project("hello_sqs")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if(WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif()

add_executable(${PROJECT_NAME}
        hello_sqs.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_sqs.cpp 源文件的代码。

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/ListQueuesRequest.h>
```

```
#include <iostream>

/*
 *  A "Hello SQS" starter application that initializes an Amazon Simple Queue
 Service
 *  (Amazon SQS) client and lists the SQS queues in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_sqs'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::SQS::SQSClient sqsClient(clientConfig);

        Aws::Vector<Aws::String> allQueueUrls;
        Aws::String nextToken; // Next token is used to handle a paginated
 response.
        do {
            Aws::SQS::Model::ListQueuesRequest request;

            Aws::SQS::Model::ListQueuesOutcome outcome =
 sqsClient.ListQueues(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::String> &pageOfQueueUrls =
 outcome.GetResult().GetQueueUrls();
                if (!pageOfQueueUrls.empty()) {
                    allQueueUrls.insert(allQueueUrls.cend(),
 pageOfQueueUrls.cbegin(),
                                        pageOfQueueUrls.cend());
                }
            }
            else {
```

```
                std::cerr << "Error with SQS::ListQueues. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
            nextToken = outcome.GetResult().GetNextToken();
        } while (!nextToken.empty());


        std::cout << "Hello Amazon SQS! You have " << allQueueUrls.size() << "
 queue"
                  << (allQueueUrls.size() == 1 ? "" : "s") << " in your account."
                  << std::endl;

        if (!allQueueUrls.empty()) {
            std::cout << "Here are your queue URLs." << std::endl;
            for (const Aws::String &queueUrl: allQueueUrls) {
                std::cout << "  * " << queueUrl << std::endl;
            }
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考ListQueues中的。

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
package main
```

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get queues. Here's why: %v\n", err)
   break
  } else {
   queueUrls = append(queueUrls, output.QueueUrls...)
  }
 }
 if len(queueUrls) == 0 {
  fmt.Println("You don't have any queues!")
 } else {
  for _, queueUrl := range queueUrls {
   fmt.Printf("\t%v\n", queueUrl)
  }
 }
}
```

- 有关 API 的详细信息，请参阅 适用于 Go 的 AWS SDK API 参考ListQueues中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.SqsException;
import software.amazon.awssdk.services.sqs.paginators.ListQueuesIterable;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloSQS {
    public static void main(String[] args) {
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listQueues(sqsClient);
        sqsClient.close();
    }

    public static void listQueues(SqsClient sqsClient) {
        try {
            ListQueuesIterable listQueues = sqsClient.listQueuesPaginator();
            listQueues.stream()
```

```
                    .flatMap(r -> r.queueUrls().stream())
                    .forEach(content -> System.out.println(" Queue URL: " +
 content.toLowerCase()));

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListQueues中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

初始化 Amazon SQS 客户端并列出队列。

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
```

```
      queues.push(...page.QueueUrls);
    }
  }

  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your
 account.`,
  );
  console.log(queues.map((t) => `  * ${t}`).join("\n"));
};
```

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考ListQueues中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
package com.kotlin.sqs

import aws.sdk.kotlin.services.sqs.SqsClient
import aws.sdk.kotlin.services.sqs.paginators.listQueuesPaginated
import kotlinx.coroutines.flow.transform

suspend fun main() {
    listTopicsPag()
}

suspend fun listTopicsPag() {
    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient
            .listQueuesPaginated { }
            .transform { it.queueUrls?.forEach { queue -> emit(queue) } }
            .collect { queue ->
```

```
                println("The Queue URL is $queue")
            }
    }
}
```

- 有关 API 的详细信息，请参阅适用ListQueues于 K otlin 的AWS SDK API 参考。

Swift

适用于 Swift 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

`Package.swift` 文件。

```
import PackageDescription

let package = Package(
    name: "sqs-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
```

```
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "sqs-basics",
            dependencies: [
                .product(name: "AWSSQS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

Swift 源代码 `entry.swift`。

```
import ArgumentParser
import AWSClientRuntime
import AWSSQS
import Foundation

struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use (default: us-east-1)")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "sqs-basics",
        abstract: """
        This example shows how to list all of your available Amazon SQS queues.
        """,
        discussion: """
        """
    )

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        var queues: [String] = []
        let outputPages = sqsClient.listQueuesPaginated(
            input: ListQueuesInput()
```

```
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let urls = output.queueUrls else {
                print("No queues found.")
                return
            }

            // Iterate over the queue URLs listed on this page, adding them
            // to the `queues` array.

            for queueUrl in urls {
                queues.append(queueUrl)
            }
        }

        print("You have \(queues.count) queues:")
        for queue in queues {
            print("   \(queue)")
        }
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- 如需了解 API 的详细信息，请参阅适用ListQueues于 S wift 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 Amazon SQS 执行的操作 AWS SDKs

以下代码示例演示了如何使用执行各个 Amazon SQS 操作。 AWS SDKs每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

这些代码节选调用了 Amazon SQS API，是必须在上下文中运行的大型程序的代码节选。您可以在使用 Amazon SQS 的场景 AWS SDKs中结合上下文查看操作。

以下示例仅包括最常用的操作。有关完整列表，请参阅《Amazon Simple Queue Service API Reference》。

示例

- 将 AddPermission 与 CLI 配合使用
- ChangeMessageVisibility与 AWS SDK 或 CLI 配合使用
- 将 ChangeMessageVisibilityBatch 与 CLI 配合使用
- CreateQueue与 AWS SDK 或 CLI 配合使用
- DeleteMessage与 AWS SDK 或 CLI 配合使用
- DeleteMessageBatch与 AWS SDK 或 CLI 配合使用
- DeleteQueue与 AWS SDK 或 CLI 配合使用
- GetQueueAttributes与 AWS SDK 或 CLI 配合使用
- GetQueueUrl与 AWS SDK 或 CLI 配合使用
- 将 ListDeadLetterSourceQueues 与 CLI 配合使用
- ListQueues与 AWS SDK 或 CLI 配合使用
- 将 PurgeQueue 与 CLI 配合使用
- ReceiveMessage与 AWS SDK 或 CLI 配合使用
- 将 RemovePermission 与 CLI 配合使用
- SendMessage与 AWS SDK 或 CLI 配合使用
- SendMessageBatch与 AWS SDK 或 CLI 配合使用
- SetQueueAttributes与 AWS SDK 或 CLI 配合使用

将 **AddPermission** 与 CLI 配合使用

以下代码示例演示如何使用 AddPermission。

CLI

AWS CLI

向队列添加权限

此示例允许指定 AWS 账户向指定队列发送消息。

命令:

```
aws sqs add-permission --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue --aws-
account-ids 12345EXAMPLE --actions SendMessage
```

输出：

```
None.
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考AddPermission中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例允许指定队列 AWS 账户 发送来自指定队列的消息。

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) AddPermission中的。

适用于 PowerShell V5 的工具

示例 1：此示例允许指定队列 AWS 账户 发送来自指定队列的消息。

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) AddPermission中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ChangeMessageVisibility`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ChangeMessageVisibility`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Changes the visibility timeout of a message in an Amazon Simple Queue Service
//! (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param messageReceiptHandle: A message receipt handle.
  \param visibilityTimeoutSeconds: Visibility timeout in seconds.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::changeMessageVisibility(
        const Aws::String &queue_url,
        const Aws::String &messageReceiptHandle,
        int visibilityTimeoutSeconds,
```

```
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::ChangeMessageVisibilityRequest request;
    request.SetQueueUrl(queue_url);
    request.SetReceiptHandle(messageReceiptHandle);
    request.SetVisibilityTimeout(visibilityTimeoutSeconds);

    auto outcome = sqsClient.ChangeMessageVisibility(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully changed visibility of message " <<
                  messageReceiptHandle << " from queue " << queue_url <<
 std::endl;
    }
    else {
        std::cout << "Error changing visibility of message from queue "
                  << queue_url << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考[ChangeMessageVisibility](#)中的。

CLI

AWS CLI

更改消息的超时可见性

此示例将指定消息的超时可见性更改为 10 小时（10 小时 * 60 分钟 * 60 秒）。

命令:

```
aws sqs change-message-visibility --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBTpyI...t6HyQg== --
visibility-timeout 36000
```

输出：

```
None.
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ChangeMessageVisibility中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收 Amazon SQS 消息并更改其超时可见性。

```
import {
  ReceiveMessageCommand,
  ChangeMessageVisibilityCommand,
  SQSClient,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 1,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 1,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  const response = await client.send(
    new ChangeMessageVisibilityCommand({
```

```
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考[ChangeMessageVisibility](#)中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#)中查找完整示例，了解如何进行设置和运行。

接收 Amazon SQS 消息并更改其超时可见性。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
```

```
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-managing-visibility-timeout.html#sqs-examples-managing-visibility-timeout-setting。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考ChangeMessageVisibility中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例将指定队列中具有指定接收句柄的消息的可见性超时更改为 10 小时（10 小时 * 60 分钟 * 60 秒 = 36000 秒）。

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) ChangeMessageVisibility中的。

适用于 PowerShell V5 的工具

示例 1：此示例将指定队列中具有指定接收句柄的消息的可见性超时更改为 10 小时（10 小时 * 60 分钟 * 60 秒 = 36000 秒）。

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) ChangeMessageVisibility中的。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
require 'aws-sdk-sqs' # v2: require 'aws-sdk'
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
sqs = Aws::SQS::Client.new(region: 'us-west-2')

begin
  queue_name = 'my-queue'
  queue_url = sqs.get_queue_url(queue_name: queue_name).queue_url

  # Receive up to 10 messages
  receive_message_result_before = sqs.receive_message({
                                                        queue_url: queue_url,
                                                        max_number_of_messages:
 10
                                                      })

  puts "Before attempting to change message visibility timeout: received
 #{receive_message_result_before.messages.count} message(s)."
```

```
  receive_message_result_before.messages.each do |message|
    sqs.change_message_visibility({
                                    queue_url: queue_url,
                                    receipt_handle: message.receipt_handle,
                                    visibility_timeout: 30 # This message will
 not be visible for 30 seconds after first receipt.
                                  })
  end

  # Try to retrieve the original messages after setting their visibility timeout.
  receive_message_result_after = sqs.receive_message({
                                                       queue_url: queue_url,
                                                       max_number_of_messages: 10
                                                     })

  puts "\nAfter attempting to change message visibility timeout: received
 #{receive_message_result_after.messages.count} message(s)."
rescue Aws::SQS::Errors::NonExistentQueue
  puts "Cannot receive messages for a queue named '#{queue_name}', as it does not
 exist."
end
```

- 有关 API 的详细信息，请参阅 适用于 Ruby 的 AWS SDK API 参考ChangeMessageVisibility中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **ChangeMessageVisibilityBatch** 与 CLI 配合使用

以下代码示例演示如何使用 ChangeMessageVisibilityBatch。

CLI

AWS CLI

批量更改多条消息的可见性超时

此示例 2 条指定消息的超时可见性更改为 10 小时（ 10 小时 * 60 分钟 * 60 秒）。

命令:

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-
visibility-batch.json
```

输入文件 (change-message-visibility-batch.json)：

```
[
  {
    "Id": "FirstMessage",
        "ReceiptHandle": "AQEBhz2q...Jf3kaw==",
        "VisibilityTimeout": 36000
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBkTUH...HifSnw==",
        "VisibilityTimeout": 36000
  }
]
```

输出：

```
{
  "Successful": [
    {
      "Id": "SecondMessage"
    },
    {
      "Id": "FirstMessage"
    }
  ]
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ChangeMessageVisibilityBatch中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例更改了指定队列中具有指定接收句柄的 2 条消息的可见性超时。第一条消息的超时可见性更改为 10 小时（10 小时 * 60 分钟 * 60 秒 = 36000 秒）。第二条消息的超时可见性更改为 5 小时（5 小时 * 60 分钟 * 60 秒 = 18000 秒）。

```
$changeVisibilityRequest1 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest1.Id = "Request1"
$changeVisibilityRequest1.ReceiptHandle = "AQEBd329...v6gl8Q=="
$changeVisibilityRequest1.VisibilityTimeout = 36000

$changeVisibilityRequest2 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest2.Id = "Request2"
$changeVisibilityRequest2.ReceiptHandle = "AQEBgGDh...J/Iqww=="
$changeVisibilityRequest2.VisibilityTimeout = 18000

Edit-SQSMessageVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
 $changeVisibilityRequest2
```

输出：

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) ChangeMessageVisibilityBatch中的。

适用于 PowerShell V5 的工具

示例 1：此示例更改了指定队列中具有指定接收句柄的 2 条消息的可见性超时。第一条消息的超时可见性更改为 10 小时（10 小时 * 60 分钟 * 60 秒 = 36000 秒）。第二条消息的超时可见性更改为 5 小时（5 小时 * 60 分钟 * 60 秒 = 18000 秒）。

```
$changeVisibilityRequest1 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest1.Id = "Request1"
$changeVisibilityRequest1.ReceiptHandle = "AQEBd329...v6gl8Q=="
$changeVisibilityRequest1.VisibilityTimeout = 36000

$changeVisibilityRequest2 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest2.Id = "Request2"
$changeVisibilityRequest2.ReceiptHandle = "AQEBgGDh...J/Iqww=="
```

```
$changeVisibilityRequest2.VisibilityTimeout = 18000

Edit-SQSMessageVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
 $changeVisibilityRequest2
```

输出：

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) ChangeMessageVisibilityBatch中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`CreateQueue`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateQueue`。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 将消息发布到队列
- 发送和接收批量消息
- 使用 Amazon SQS Java Messaging Library 处理 JMS 接口

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用特定的名称创建队列。

```
    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }

            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
```

```
        return createResponse.QueueUrl;
    }
```

创建 Amazon SQS 队列并向其发送消息。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SQS;
    using Amazon.SQS.Model;

    public class CreateSendExample
    {
        // Specify your AWS Region (an example Region is shown).
        private static readonly string QueueName = "Example_Queue";
        private static readonly RegionEndpoint ServiceRegion =
RegionEndpoint.USWest2;
        private static IAmazonSQS client;

        public static async Task Main()
        {
            client = new AmazonSQSClient(ServiceRegion);
            var createQueueResponse = await CreateQueue(client, QueueName);

            string queueUrl = createQueueResponse.QueueUrl;

            Dictionary<string, MessageAttributeValue> messageAttributes = new
Dictionary<string, MessageAttributeValue>
            {
                { "Title",   new MessageAttributeValue { DataType = "String",
StringValue = "The Whistler" } },
                { "Author",  new MessageAttributeValue { DataType = "String",
StringValue = "John Grisham" } },
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
bestseller for week of 12/11/2016.";
```

```
            var sendMsgResponse = await SendMessage(client, queueUrl,
 messageBody, messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS
 client, string queueName)
        {
            var request = new CreateQueueRequest
            {
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

            return response;
        }

        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
```

```
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
            Console.WriteLine($"Sent a message with id : {response.MessageId}");

            return response;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考CreateQueue中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Create an Amazon Simple Queue Service (Amazon SQS) queue.
```

```
/*!
  \param queueName: An Amazon SQS queue name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::createQueue(const Aws::String &queueName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::CreateQueueRequest request;
    request.SetQueueName(queueName);

    const Aws::SQS::Model::CreateQueueOutcome outcome =
 sqsClient.CreateQueue(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created queue " << queueName << " with a queue
 URL "
                  << outcome.GetResult().GetQueueUrl() << "." << std::endl;
    }
    else {
        std::cerr << "Error creating queue " << queueName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考CreateQueue中的。

CLI

AWS CLI

创建队列

此示例使用指定名称创建队列，将消息保留期设置为 3 天（3 天 * 24 小时 * 60 分钟 * 60 秒），并将队列的死信队列设置为指定队列，其最大接收数量为 1,000 条消息。

命令:

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

输入文件（create-queue.json）：

```
{
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "MessageRetentionPeriod": "259200"
}
```

输出：

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateQueue中的。

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)
```

```
// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can
 specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}
```

- 有关 API 的详细信息，请参阅 适用于 Go 的 AWS SDK API 参考CreateQueue中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.ChangeMessageVisibilityRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlResponse;
import software.amazon.awssdk.services.sqs.model.ListQueuesRequest;
import software.amazon.awssdk.services.sqs.model.ListQueuesResponse;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SQSExample {
    public static void main(String[] args) {
        String queueName = "queue" + System.currentTimeMillis();
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();
```

```
        // Perform various tasks on the Amazon SQS queue.
        String queueUrl = createQueue(sqsClient, queueName);
        listQueues(sqsClient);
        listQueuesFilter(sqsClient, queueUrl);
        List<Message> messages = receiveMessages(sqsClient, queueUrl);
        sendBatchMessages(sqsClient, queueUrl);
        changeMessages(sqsClient, queueUrl, messages);
        deleteMessages(sqsClient, queueUrl, messages);
        sqsClient.close();
    }

    public static String createQueue(SqsClient sqsClient, String queueName) {
        try {
            System.out.println("\nCreate Queue");

            CreateQueueRequest createQueueRequest = CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();

            sqsClient.createQueue(createQueueRequest);

            System.out.println("\nGet queue url");

            GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
            return getQueueUrlResponse.queueUrl();

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static void listQueues(SqsClient sqsClient) {

        System.out.println("\nList Queues");
        String prefix = "que";

        try {
            ListQueuesRequest listQueuesRequest =
ListQueuesRequest.builder().queueNamePrefix(prefix).build();
```

```
            ListQueuesResponse listQueuesResponse =
sqsClient.listQueues(listQueuesRequest);
            for (String url : listQueuesResponse.queueUrls()) {
                System.out.println(url);
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void listQueuesFilter(SqsClient sqsClient, String queueUrl) {
        // List queues with filters
        String namePrefix = "queue";
        ListQueuesRequest filterListRequest = ListQueuesRequest.builder()
                .queueNamePrefix(namePrefix)
                .build();

        ListQueuesResponse listQueuesFilteredResponse =
sqsClient.listQueues(filterListRequest);
        System.out.println("Queue URLs with prefix: " + namePrefix);
        for (String url : listQueuesFilteredResponse.queueUrls()) {
            System.out.println(url);
        }

        System.out.println("\nSend message");
        try {
            sqsClient.sendMessage(SendMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageBody("Hello world!")
                    .delaySeconds(10)
                    .build());

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void sendBatchMessages(SqsClient sqsClient, String queueUrl) {

        System.out.println("\nSend multiple messages");
        try {
```

```
            SendMessageBatchRequest sendMessageBatchRequest =
SendMessageBatchRequest.builder()
                    .queueUrl(queueUrl)

.entries(SendMessageBatchRequestEntry.builder().id("id1").messageBody("Hello
from msg 1").build(),

SendMessageBatchRequestEntry.builder().id("id2").messageBody("msg
2").delaySeconds(10)
                                    .build())
                    .build();
            sqsClient.sendMessageBatch(sendMessageBatchRequest);

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static List<Message> receiveMessages(SqsClient sqsClient, String
queueUrl) {

        System.out.println("\nReceive messages");
        try {
            ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .maxNumberOfMessages(5)
                    .build();
            return sqsClient.receiveMessage(receiveMessageRequest).messages();

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static void changeMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {

        System.out.println("\nChange Message Visibility");
        try {
```

```
            for (Message message : messages) {
                ChangeMessageVisibilityRequest req =
 ChangeMessageVisibilityRequest.builder()
                        .queueUrl(queueUrl)
                        .receiptHandle(message.receiptHandle())
                        .visibilityTimeout(100)
                        .build();
                sqsClient.changeMessageVisibility(req);
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteMessages(SqsClient sqsClient, String queueUrl,
 List<Message> messages) {
        System.out.println("\nDelete Messages");

        try {
            for (Message message : messages) {
                DeleteMessageRequest deleteMessageRequest =
 DeleteMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .receiptHandle(message.receiptHandle())
                        .build();
                sqsClient.deleteMessage(deleteMessageRequest);
            }
        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CreateQueue中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 Amazon SQS 标准队列。

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

使用长轮询创建 Amazon SQS 队列。

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const response = await client.send(
    new CreateQueueCommand({
```

```
      QueueName: queueName,
      Attributes: {
        // When the wait time for the ReceiveMessage API action is greater than
 0,
        // long polling is in effect. The maximum long polling wait time is 20
        // seconds. Long polling helps reduce the cost of using Amazon SQS by,
        // eliminating the number of empty responses and false empty responses.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        ReceiveMessageWaitTimeSeconds: "20",
      },
    }),
  );
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-create-queue。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考CreateQueue中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 Amazon SQS 标准队列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
```

```
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

创建等待消息到达的 Amazon SQS 队列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-create-queue。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考CreateQueue中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun createQueue(queueNameVal: String): String {
    println("Create Queue")
    val createQueueRequest =
        CreateQueueRequest {
            queueName = queueNameVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.createQueue(createQueueRequest)
        println("Get queue url")

        val getQueueUrlRequest =
            GetQueueUrlRequest {
                queueName = queueNameVal
            }

        val getQueueUrlResponse = sqsClient.getQueueUrl(getQueueUrlRequest)
        return getQueueUrlResponse.queueUrl.toString()
    }
}
```

- 有关 API 的详细信息，请参阅适用CreateQueue于 K otlin 的AWS SDK API 参考。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例使用指定名称创建队列。

```
New-SQSQueue -QueueName MyQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) CreateQueue中的。

适用于 PowerShell V5 的工具

示例 1：此示例使用指定名称创建队列。

```
New-SQSQueue -QueueName MyQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) CreateQueue中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
def create_queue(name, attributes=None):
```

```
    """
    Creates an Amazon SQS queue.

    :param name: The name of the queue. This is part of the URL assigned to the
 queue.
    :param attributes: The attributes of the queue, such as maximum message size
 or
                       whether it's a FIFO queue.
    :return: A Queue object that contains metadata about the queue and that can
 be used
             to perform queue operations like sending and receiving messages.
    """
    if not attributes:
        attributes = {}

    try:
        queue = sqs.create_queue(QueueName=name, Attributes=attributes)
        logger.info("Created queue '%s' with URL=%s", name, queue.url)
    except ClientError as error:
        logger.exception("Couldn't create queue named '%s'.", name)
        raise error
    else:
        return queue
```

```
class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.
```

```
        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)


    def create_queue(self, queue_name: str, is_fifo: bool = False) -> str:
        """
        Create an SQS queue.

        :param queue_name: The name of the queue to create.
        :param is_fifo: Whether to create a FIFO queue.
        :return: The URL of the created queue.
        :raises ClientError: If the queue creation fails.
        """
        try:
            # Add .fifo suffix for FIFO queues
            if is_fifo and not queue_name.endswith('.fifo'):
                queue_name += '.fifo'

            attributes = {}
            if is_fifo:
                attributes['FifoQueue'] = 'true'

            response = self.sqs_client.create_queue(
                QueueName=queue_name,
                Attributes=attributes
            )

            queue_url = response['QueueUrl']
            logger.info(f"Created queue: {queue_name} with URL: {queue_url}")
            return queue_url

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error creating queue {queue_name}: {error_code} -
{e}")
            raise
```

- 有关 API 的详细信息，请参阅适用CreateQueue于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# This code example demonstrates how to create a queue in Amazon Simple Queue
 Service (Amazon SQS).

require 'aws-sdk-sqs'

# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_name [String] The name of the queue.
# @return [Boolean] true if the queue was created; otherwise, false.
# @example
#   exit 1 unless queue_created?(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'my-queue'
#   )
def queue_created?(sqs_client, queue_name)
  sqs_client.create_queue(queue_name: queue_name)
  true
rescue StandardError => e
  puts "Error creating queue: #{e.message}"
  false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Creating the queue named '#{queue_name}'..."

  if queue_created?(sqs_client, queue_name)
    puts 'Queue created.'
```

```
  else
    puts 'Queue not created.'
  end
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- 有关 API 的详细信息，请参阅 适用于 Ruby 的 AWS SDK API 参考CreateQueue中的。

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 Amazon SQS 标准队列。

```
    TRY.
        oo_result = lo_sqs->createqueue( iv_queuename = iv_queue_name ).        "
oo_result is returned for testing purposes. "
        MESSAGE 'SQS queue created.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedeldrecently.
        MESSAGE 'After deleting a queue, wait 60 seconds before creating another
queue with the same name.' TYPE 'E'.
      CATCH /aws1/cx_sqsqueuenameexists.
        MESSAGE 'A queue with this name already exists.' TYPE 'E'.
    ENDTRY.
```

创建等待接收消息的 Amazon SQS 队列。

```
    TRY.
        DATA lt_attributes TYPE /aws1/cl_sqsqueueattrmap_w=>tt_queueattributemap.
        DATA ls_attribute TYPE /aws1/
cl_sqsqueueattrmap_w=>ts_queueattributemap_maprow.
```

```
       ls_attribute-key = 'ReceiveMessageWaitTimeSeconds'.                " Time
in seconds for long polling, such as how long the call waits for a message to
arrive in the queue before returning. "
       ls_attribute-value = NEW /aws1/cl_sqsqueueattrmap_w( iv_value =
iv_wait_time ).
       INSERT ls_attribute INTO TABLE lt_attributes.
       oo_result = lo_sqs->createqueue(                   " oo_result is returned
for testing purposes. "
               iv_queuename = iv_queue_name
               it_attributes = lt_attributes ).
       MESSAGE 'SQS queue created.' TYPE 'I'.
     CATCH /aws1/cx_sqsqueuedeldrecently.
       MESSAGE 'After deleting a queue, wait 60 seconds before creating another
queue with the same name.' TYPE 'E'.
     CATCH /aws1/cx_sqsqueuenameexists.
       MESSAGE 'A queue with this name already exists.' TYPE 'E'.
   ENDTRY.
```

- 有关 API 的详细信息，请参阅适用CreateQueue于 S AP 的AWS SDK ABAP API 参考。

Swift

适用于 Swift 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        let output = try await sqsClient.createQueue(
            input: CreateQueueInput(
                queueName: queueName
            )
        )
```

```
        guard let queueUrl = output.queueUrl else {
            print("No queue URL returned.")
            return
        }
```

- 如需了解 API 的详细信息，请参阅适用CreateQueue于 S wift 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteMessage`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteMessage`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 发送和接收批量消息

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收来自 Amazon SQS 队列的消息，然后删除该消息。

```
        public static async Task Main()
        {
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
```

```
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
        }

        /// <summary>
        /// Retrieve the queue URL for the queue named in the queueName
        /// property using the client object.
        /// </summary>
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
 queueName)
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await
 client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }

        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</
param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
        public static async Task<ReceiveMessageResponse>
 ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
```

```
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };

            var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };

            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteMessage中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
//! Delete a message from an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param messageReceiptHandle: A message receipt handle.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::deleteMessage(const Aws::String &queueUrl,
                                const Aws::String &messageReceiptHandle,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::DeleteMessageRequest request;
    request.SetQueueUrl(queueUrl);
    request.SetReceiptHandle(messageReceiptHandle);

    const Aws::SQS::Model::DeleteMessageOutcome outcome =
 sqsClient.DeleteMessage(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted message from queue " << queueUrl
                  << std::endl;
    }
    else {
        std::cerr << "Error deleting message from queue " << queueUrl << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考DeleteMessage中的。

CLI

AWS CLI

删除消息

此示例将删除指定消息。

命令:

```
aws sqs delete-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

输出：

```
None.
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteMessage中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        try {
            for (Message message : messages) {
                DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .receiptHandle(message.receiptHandle())
                        .build();
                sqsClient.deleteMessage(deleteMessageRequest);
            }
        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteMessage中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收和删除 Amazon SQS 消息。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
```

```
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考DeleteMessage中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收和删除 Amazon SQS 消息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";
```

```
var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
    });
  }
});
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-send-receive-messages.html#sqs-examples-send-receive-messages-receiving。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考DeleteMessage中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.purgeQueue(purgeRequest)
        println("Messages are successfully deleted from $queueUrlVal")
    }
}

suspend fun deleteQueue(queueUrlVal: String) {
    val request =
        DeleteQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
        println("$queueUrlVal was deleted!")
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteMessage于 K otlin 的AWS SDK API 参考。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例删除了指定队列中具有指定接收句柄的消息。

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) DeleteMessage中的。

适用于 PowerShell V5 的工具

示例 1：此示例删除了指定队列中具有指定接收句柄的消息。

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) DeleteMessage中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
def delete_message(message):
    """
    Delete a message from a queue. Clients must delete messages after they
    are received and processed to remove them from the queue.

    :param message: The message to delete. The message's queue URL is contained
 in
                    the message's metadata.
```

```
    :return: None
    """
    try:
        message.delete()
        logger.info("Deleted message: %s", message.message_id)
    except ClientError as error:
        logger.exception("Couldn't delete message: %s", message.message_id)
        raise error
```

- 有关 API 的详细信息，请参阅适用DeleteMessage于 Python 的AWS SDK (Boto3) API 参考。

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    TRY.
        lo_sqs->deletemessage(
           iv_queueurl = iv_queue_url
           iv_receipthandle = iv_receipt_handle ).
        MESSAGE 'Message deleted from SQS queue.' TYPE 'I'.
      CATCH /aws1/cx_sqsinvalididformat.
        MESSAGE 'The specified receipt handle is not valid.' TYPE 'E'.
      CATCH /aws1/cx_sqsreceipthandleisinv.
        MESSAGE 'The specified receipt handle is not valid for the current
version.' TYPE 'E'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用DeleteMessage于 S AP 的AWS SDK ABAP API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteMessageBatch`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteMessageBatch`。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 处理 S3 事件通知
- 将消息发布到队列
- 发送和接收批量消息

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl,
List<Message> messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
```

```
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteMessageBatch中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::SQS::SQSClient sqsClient(clientConfiguration);

            Aws::SQS::Model::DeleteMessageBatchRequest request;
            request.SetQueueUrl(queueURLS[i]);
            int id = 1; // Ids must be unique within a batch delete request.
            for (const Aws::String &receiptHandle: receiptHandles) {
                Aws::SQS::Model::DeleteMessageBatchRequestEntry entry;
                entry.SetId(std::to_string(id));
                ++id;
                entry.SetReceiptHandle(receiptHandle);
                request.AddEntries(entry);
            }
```

```
        Aws::SQS::Model::DeleteMessageBatchOutcome outcome =
                sqsClient.DeleteMessageBatch(request);

        if (outcome.IsSuccess()) {
            std::cout << "The batch deletion of messages was successful."
                      << std::endl;
        }
        else {
            std::cerr << "Error with SQS::DeleteMessageBatch. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUp(topicARN,
                    queueURLS,
                    subscriptionARNS,
                    snsClient,
                    sqsClient);

            return false;
        }
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考DeleteMessageBatch中的。

CLI

AWS CLI

批量删除多条消息

此示例将删除指定消息。

命令:

```
aws sqs delete-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-batch.json
```

输入文件 (delete-message-batch.json)：

```
[
```

```
  {
        "Id": "FirstMessage",
        "ReceiptHandle": "AQEB1mgl...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```

输出：

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteMessageBatch中的。

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// DeleteMessages uses the DeleteMessageBatch action to delete a batch of
 messages from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}
```

- 有关 API 的详细信息，请参阅 适用于 Go 的 AWS SDK API 参考DeleteMessageBatch中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
```

```
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考DeleteMessageBatch中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例删除了指定队列中具有指定接收句柄的 2 条消息。

```
$deleteMessageRequest1 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJSQg=="

$deleteMessageRequest2 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBqOVY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
 $deleteMessageRequest2
```

输出：

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) DeleteMessageBatch中的。

适用于 PowerShell V5 的工具

示例 1：此示例删除了指定队列中具有指定接收句柄的 2 条消息。

```
$deleteMessageRequest1 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJSQg=="

$deleteMessageRequest2 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBqOVY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
 $deleteMessageRequest2
```

输出：

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) DeleteMessageBatch中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
def delete_messages(queue, messages):
    """
    Delete a batch of messages from a queue in a single request.

    :param queue: The queue from which to delete the messages.
    :param messages: The list of messages to delete.
    :return: The response from SQS that contains the list of successful and
 failed
             message deletions.
    """
    try:
        entries = [
            {"Id": str(ind), "ReceiptHandle": msg.receipt_handle}
            for ind, msg in enumerate(messages)
        ]
        response = queue.delete_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info("Deleted %s",
 messages[int(msg_meta["Id"])].receipt_handle)
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Could not delete %s",
 messages[int(msg_meta["Id"])].receipt_handle
                )
    except ClientError:
        logger.exception("Couldn't delete messages from queue %s", queue)
    else:
        return response
```

```
class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)


    def delete_messages(self, queue_url: str, messages: List[Dict[str, Any]]) ->
 bool:
        """
        Delete messages from an SQS queue in batches.

        :param queue_url: The URL of the queue.
        :param messages: List of messages to delete.
        :return: True if successful.
        :raises ClientError: If deleting messages fails.
        """
        try:
            if not messages:
                return True

            # Build delete entries for batch delete
            delete_entries = []
            for i, message in enumerate(messages):
                delete_entries.append({
                    'Id': str(i),
```

```
                'ReceiptHandle': message['ReceiptHandle']
            })

        # Delete messages in batches of 10 (SQS limit)
        batch_size = 10
        for i in range(0, len(delete_entries), batch_size):
            batch = delete_entries[i:i + batch_size]

            response = self.sqs_client.delete_message_batch(
                QueueUrl=queue_url,
                Entries=batch
            )

            # Check for failures
            if 'Failed' in response and response['Failed']:
                for failed in response['Failed']:
                    logger.warning(f"Failed to delete message: {failed}")

        logger.info(f"Deleted {len(messages)} messages from {queue_url}")
        return True

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error deleting messages: {error_code} - {e}")
        raise
```

- 有关 API 的详细信息，请参阅适用DeleteMessageBatch于 Python 的AWS SDK (Boto3) API 参考。

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   TRY.
       oo_result = lo_sqs->deletemessagebatch(       " oo_result is returned for
testing purposes. "
          iv_queueurl = iv_queue_url
          it_entries = it_entries ).
       MESSAGE 'Messages deleted from SQS queue.' TYPE 'I'.
     CATCH /aws1/cx_sqsbtcentidsnotdist00.
       MESSAGE 'Two or more batch entries in the request have the same ID.' TYPE
'E'.
     CATCH /aws1/cx_sqsemptybatchrequest.
       MESSAGE 'The batch request does not contain any entries.' TYPE 'E'.
     CATCH /aws1/cx_sqsinvbatchentryid.
       MESSAGE 'The ID of a batch entry in a batch request is not valid.' TYPE
'E'.
     CATCH /aws1/cx_sqstoomanyentriesin00.
       MESSAGE 'The batch request contains more entries than allowed.' TYPE 'E'.
   ENDTRY.
```

- 有关 API 的详细信息，请参阅适用DeleteMessageBatch于 S AP 的AWS SDK ABAP API 参考。

Swift

适用于 Swift 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        // Create the list of message entries.

        var entries: [SQSClientTypes.DeleteMessageBatchRequestEntry] = []
        var messageNumber = 1
```

```
        for handle in handles {
            let entry = SQSClientTypes.DeleteMessageBatchRequestEntry(
                id: "\(messageNumber)",
                receiptHandle: handle
            )
            entries.append(entry)
            messageNumber += 1
        }

        // Delete the messages.

        let output = try await sqsClient.deleteMessageBatch(
            input: DeleteMessageBatchInput(
                entries: entries,
                queueUrl: queue
            )
        )

        // Get the lists of failed and successful deletions from the output.

        guard let failedEntries = output.failed else {
            print("Failed deletion list is missing!")
            return
        }
        guard let successfulEntries = output.successful else {
            print("Successful deletion list is missing!")
            return
        }

        // Display a list of the failed deletions along with their
        // corresponding explanation messages.

        if failedEntries.count != 0 {
            print("Failed deletions:")

            for entry in failedEntries {
                print("Message #\(entry.id ?? "<unknown>") failed:
\(entry.message ?? "<unknown>")")
            }
        } else {
            print("No failed deletions.")
        }
```

```
        // Output a list of the message numbers that were successfully deleted.

        if successfulEntries.count != 0 {
            var successes = ""

            for entry in successfulEntries {
                if successes.count == 0 {
                    successes = entry.id ?? "<unknown>"
                } else {
                    successes = "\(successes), \(entry.id ?? "<unknown>")"
                }
            }
            print("Succeeded: ", successes)
        } else {
            print("No successful deletions.")
        }
```

- 如需了解 API 的详细信息，请参阅适用DeleteMessageBatch于 S wift 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteQueue`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteQueue`。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 将消息发布到队列
- 发送和接收批量消息
- 使用 Amazon SQS Java Messaging Library 处理 JMS 接口

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用此 URL 删除队列。

```
/// <summary>
/// Delete a queue by its URL.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteQueueByUrl(string queueUrl)
{
    var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
        new DeleteQueueRequest()
        {
            QueueUrl = queueUrl
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteQueue中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Delete an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueURL: An Amazon SQS queue URL.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::deleteQueue(const Aws::String &queueURL,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);
    Aws::SQS::Model::DeleteQueueRequest request;
    request.SetQueueUrl(queueURL);

    const Aws::SQS::Model::DeleteQueueOutcome outcome =
 sqsClient.DeleteQueue(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted queue with url " << queueURL <<
                  std::endl;
    }
    else {
        std::cerr << "Error deleting queue " << queueURL << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考DeleteQueue中的。

CLI

AWS CLI

删除队列

此示例将删除指定队列。

命令:

```
aws sqs delete-queue --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

输出：

```
None.
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteQueue中的。

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// DeleteQueue deletes an Amazon SQS queue.
```

```
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

- 有关 API 的详细信息，请参阅 适用于 Go 的 AWS SDK API 参考DeleteQueue中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteQueue {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:    <queueName>

                Where:
                   queueName - The name of the Amazon SQS queue to delete.

                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String queueName = args[0];
        SqsClient sqs = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        deleteSQSQueue(sqs, queueName);
        sqs.close();
    }

    public static void deleteSQSQueue(SqsClient sqsClient, String queueName) {
        try {
            GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
                    .queueName(queueName)
                    .build();

            String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();
            DeleteQueueRequest deleteQueueRequest = DeleteQueueRequest.builder()
                    .queueUrl(queueUrl)
                    .build();

            sqsClient.deleteQueue(deleteQueueRequest);

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteQueue中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除 Amazon SQS 队列。

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-delete-queue。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考DeleteQueue中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除 Amazon SQS 队列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-delete-queue。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考DeleteQueue中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
```

```
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.purgeQueue(purgeRequest)
        println("Messages are successfully deleted from $queueUrlVal")
    }
}

suspend fun deleteQueue(queueUrlVal: String) {
    val request =
        DeleteQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
        println("$queueUrlVal was deleted!")
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteQueue于 K otlin 的AWS SDK API 参考。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例删除了指定队列。

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) DeleteQueue中的。

适用于 PowerShell V5 的工具

示例 1：此示例删除了指定队列。

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) DeleteQueue中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
def remove_queue(queue):
    """
    Removes an SQS queue. When run against an AWS account, it can take up to
    60 seconds before the queue is actually deleted.

    :param queue: The queue to delete.
    :return: None
    """
    try:
        queue.delete()
        logger.info("Deleted queue with URL=%s.", queue.url)
    except ClientError as error:
        logger.exception("Couldn't delete queue with URL=%s!", queue.url)
        raise error
```

```
class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.
```

```
        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)


    def delete_queue(self, queue_url: str) -> bool:
        """
        Delete an SQS queue.

        :param queue_url: The URL of the queue to delete.
        :return: True if successful.
        :raises ClientError: If the queue deletion fails.
        """
        try:
            self.sqs_client.delete_queue(QueueUrl=queue_url)

            logger.info(f"Deleted queue: {queue_url}")
            return True

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')

            if error_code == 'AWS.SimpleQueueService.NonExistentQueue':
                logger.warning(f"Queue not found: {queue_url}")
                return True  # Already deleted
            else:
                logger.error(f"Error deleting queue: {error_code} - {e}")
                raise
```

- 有关 API 的详细信息，请参阅适用DeleteQueue于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
require 'aws-sdk-sqs' # v2: require 'aws-sdk'
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
sqs = Aws::SQS::Client.new(region: 'us-west-2')

sqs.delete_queue(queue_url: URL)
```

- 有关 API 的详细信息，请参阅 适用于 Ruby 的 AWS SDK API 参考DeleteQueue中的。

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    TRY.
        lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).
        MESSAGE 'SQS queue deleted' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用DeleteQueue于 S AP 的AWS SDK ABAP API 参考。

Swift

适用于 Swift 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        do {
            _ = try await sqsClient.deleteQueue(
                input: DeleteQueueInput(
                    queueUrl: queueUrl
                )
            )
        } catch _ as AWSSQS.QueueDoesNotExist {
            print("Error: The specified queue doesn't exist.")
            return
        }
```

- 如需了解 API 的详细信息，请参阅适用DeleteQueue于 S wift 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetQueueAttributes`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetQueueAttributes`。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 处理 S3 事件通知

- 将消息发布到队列

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };

        var getAttributesResponse = await
_amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetQueueAttributes中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::SQS::SQSClient sqsClient(clientConfiguration);

            Aws::SQS::Model::GetQueueAttributesRequest request;
            request.SetQueueUrl(queueURL);

request.AddAttributeNames(Aws::SQS::Model::QueueAttributeName::QueueArn);

            Aws::SQS::Model::GetQueueAttributesOutcome outcome =
                    sqsClient.GetQueueAttributes(request);

            if (outcome.IsSuccess()) {
                const Aws::Map<Aws::SQS::Model::QueueAttributeName, Aws::String>
&attributes =
                        outcome.GetResult().GetAttributes();
                const auto &iter = attributes.find(
                        Aws::SQS::Model::QueueAttributeName::QueueArn);
                if (iter != attributes.end()) {
                    queueARN = iter->second;
                    std::cout << "The queue ARN '" << queueARN
                              << "' has been retrieved."
                              << std::endl;
                }

            }
            else {
                std::cerr << "Error with SQS::GetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
```

```
        }
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考GetQueueAttributes中的。

CLI

AWS CLI

获取队列的属性

此示例将获取指定队列的所有属性。

命令:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All
```

输出：

```
{
  "Attributes": {
    "ApproximateNumberOfMessagesNotVisible": "0",
    "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":1000}",
    "MessageRetentionPeriod": "345600",
    "ApproximateNumberOfMessagesDelayed": "0",
    "MaximumMessageSize": "262144",
    "CreatedTimestamp": "1442426968",
    "ApproximateNumberOfMessages": "0",
    "ReceiveMessageWaitTimeSeconds": "0",
    "DelaySeconds": "0",
    "VisibilityTimeout": "30",
    "LastModifiedTimestamp": "1442426968",
    "QueueArn": "arn:aws:sqs:us-east-1:80398EXAMPLE:MyNewQueue"
  }
}
```

此示例仅获取指定队列的最大消息大小和可见性超时属性。

命令:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attribute-
names MaximumMessageSize VisibilityTimeout
```

输出：

```
{
  "Attributes": {
    "VisibilityTimeout": "30",
    "MaximumMessageSize": "262144"
  }
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetQueueAttributes中的。

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
```

```
type SqsActions struct {
 SqsClient *sqs.Client
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string)
 (string, error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}
```

- 有关 API 的详细信息，请参阅 适用于 Go 的 AWS SDK API 参考GetQueueAttributes中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";
```

```
const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考[GetQueueAttributes](#)中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例列出了指定队列的所有属性。

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

输出：

```
VisibilityTimeout                       : 30
DelaySeconds                            : 0
```

```
MaximumMessageSize                     : 262144
MessageRetentionPeriod                 : 345600
ApproximateNumberOfMessages            : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed     : 0
CreatedTimestamp                       : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                  : 12/29/2015 2:23:17 PM
QueueARN                               : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                 : {"Version":"2012-10-17",
 "Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/SQSDefaultPolicy","Statement":
[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                       398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                       arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                             : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
                                       [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

示例 2：此示例仅单独列出了指定队列的指定属性。

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

输出：

```
VisibilityTimeout                      : 30
DelaySeconds                           : 0
MaximumMessageSize                     : 262144
MessageRetentionPeriod                 : 345600
ApproximateNumberOfMessages            : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed     : 0
CreatedTimestamp                       : 2/11/2015 5:53:35 PM
```

```
LastModifiedTimestamp                 : 12/29/2015 2:23:17 PM
QueueARN                              : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                : {"Version":"2012-10-17",
 "Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/SQSDefaultPolicy","Statement":
[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                       398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                       arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                            : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) GetQueueAttributes中的。

适用于 PowerShell V5 的工具

示例 1：此示例列出了指定队列的所有属性。

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

输出：

```
VisibilityTimeout                     : 30
DelaySeconds                          : 0
MaximumMessageSize                    : 262144
MessageRetentionPeriod                : 345600
ApproximateNumberOfMessages           : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed    : 0
CreatedTimestamp                      : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                 : 12/29/2015 2:23:17 PM
QueueARN                              : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
```

```
Policy                                : {"Version":"2012-10-17",
 "Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/SQSDefaultPolicy","Statement":
[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                      398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                      arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                            : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
                                      [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

示例 2：此示例仅单独列出了指定队列的指定属性。

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

输出：

```
VisibilityTimeout                     : 30
DelaySeconds                          : 0
MaximumMessageSize                    : 262144
MessageRetentionPeriod                : 345600
ApproximateNumberOfMessages           : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed    : 0
CreatedTimestamp                      : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                 : 12/29/2015 2:23:17 PM
QueueARN                              : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                : {"Version":"2012-10-17",
 "Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/SQSDefaultPolicy","Statement":
[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
```

```
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                          arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                              : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) GetQueueAttributes中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
```

```
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)


    def get_queue_arn(self, queue_url: str) -> str:
        """
        Get the ARN of an SQS queue.

        :param queue_url: The URL of the queue.
        :return: The ARN of the queue.
        :raises ClientError: If getting queue attributes fails.
        """
        try:
            response = self.sqs_client.get_queue_attributes(
                QueueUrl=queue_url,
                AttributeNames=['QueueArn']
            )

            queue_arn = response['Attributes']['QueueArn']
            logger.info(f"Queue ARN for {queue_url}: {queue_arn}")
            return queue_arn

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error getting queue ARN: {error_code} - {e}")
            raise
```

- 有关 API 的详细信息，请参阅适用GetQueueAttributes于 Python 的AWS SDK (Boto3) API 参考。

Swift

适用于 Swift 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        let output = try await sqsClient.getQueueAttributes(
            input: GetQueueAttributesInput(
                attributeNames: [
                    .approximatenumberofmessages,
                    .maximummessagesize
                ],
                queueUrl: url
            )
        )

        guard let attributes = output.attributes else {
            print("No queue attributes returned.")
            return
        }

        for (attr, value) in attributes {
            switch(attr) {
            case "ApproximateNumberOfMessages":
                print("Approximate message count: \(value)")
            case "MaximumMessageSize":
                print("Maximum message size: \(value)kB")
            default:
                continue
            }
        }
```

- 如需了解 API 的详细信息，请参阅适用GetQueueAttributes于 S wift 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetQueueUrl`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetQueueUrl`。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.SQS;
    using Amazon.SQS.Model;

    public class GetQueueUrl
    {
        /// <summary>
        /// Initializes the Amazon SQS client object and then calls the
        /// GetQueueUrlAsync method to retrieve the URL of an Amazon SQS
        /// queue.
        /// </summary>
        public static async Task Main()
        {
            // If the Amazon SQS message queue is not in the same AWS Region as
your
            // default user, you need to provide the AWS Region as a parameter to
the
            // client constructor.
            var client = new AmazonSQSClient();

            string queueName = "New-Example-Queue";

            try
            {
                var response = await client.GetQueueUrlAsync(queueName);

                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine($"The URL for {queueName} is:
{response.QueueUrl}");
                }
```

```
            }
            catch (QueueDoesNotExistException ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine($"The queue {queueName} was not found.");
            }
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetQueueUrl中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Get the URL for an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueName: An Amazon SQS queue name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::getQueueUrl(const Aws::String &queueName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::GetQueueUrlRequest request;
    request.SetQueueName(queueName);
```

```
    const Aws::SQS::Model::GetQueueUrlOutcome outcome =
 sqsClient.GetQueueUrl(request);
    if (outcome.IsSuccess()) {
        std::cout << "Queue " << queueName << " has url " <<
                  outcome.GetResult().GetQueueUrl() << std::endl;
    }
    else {
        std::cerr << "Error getting url for queue " << queueName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考GetQueueUrl中的。

CLI

AWS CLI

获取队列 URL

此示例将获取指定队列的 URL。

命令:

```
aws sqs get-queue-url --queue-name MyQueue
```

输出：

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetQueueUrl中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
            GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
            return getQueueUrlResponse.queueUrl();
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetQueueUrl中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取 Amazon SQS 队列的 URL。

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

  const response = await client.send(command);
  console.log(response);
```

```
  return response;
};
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-get-queue-url。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考GetQueueUrl中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取 Amazon SQS 队列的 URL。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-get-queue-url。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考GetQueueUrl中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例列出了具有指定名称的队列的 URL。

```
Get-SQSQueueUrl -QueueName MyQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) GetQueueUrl中的。

适用于 PowerShell V5 的工具

示例 1：此示例列出了具有指定名称的队列的 URL。

```
Get-SQSQueueUrl -QueueName MyQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) GetQueueUrl中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
def get_queue(name):
    """
    Gets an SQS queue by name.

    :param name: The name that was used to create the queue.
    :return: A Queue object.
    """
    try:
        queue = sqs.get_queue_by_name(QueueName=name)
        logger.info("Got queue '%s' with URL=%s", name, queue.url)
    except ClientError as error:
        logger.exception("Couldn't get queue named %s.", name)
        raise error
    else:
        return queue
```

- 有关 API 的详细信息，请参阅适用GetQueueUrl于 Python 的AWS SDK (Boto3) API 参考。

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    TRY.
        oo_result = lo_sqs->getqueueurl( iv_queuename = iv_queue_name ).        "
 oo_result is returned for testing purposes. "
        MESSAGE 'Queue URL retrieved.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedoesnotexist.
        MESSAGE 'The requested queue does not exist.' TYPE 'E'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用GetQueueUrl于 S AP 的AWS SDK ABAP API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 `ListDeadLetterSourceQueues` 与 CLI 配合使用

以下代码示例演示如何使用 `ListDeadLetterSourceQueues`。

CLI

AWS CLI

列出死信来源队列

此示例列出与指定死信源队列关联的队列。

命令:

```
aws sqs list-dead-letter-source-queues --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

输出：

```
{
  "queueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListDeadLetterSourceQueues中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例列 URLs 出了依赖指定队列作为死信队列的所有队列。

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) ListDeadLetterSourceQueues中的。

适用于 PowerShell V5 的工具

示例 1：此示例列 URLs 出了依赖指定队列作为死信队列的所有队列。

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) ListDeadLetterSourceQueues中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListQueues`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListQueues`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! List the Amazon Simple Queue Service (Amazon SQS) queues within an AWS
 account.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::SQS::listQueues(const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::ListQueuesRequest listQueuesRequest;

    Aws::String nextToken; // Used for pagination.
    Aws::Vector<Aws::String> allQueueUrls;

    do {
        if (!nextToken.empty()) {
            listQueuesRequest.SetNextToken(nextToken);
        }
        const Aws::SQS::Model::ListQueuesOutcome outcome = sqsClient.ListQueues(
                listQueuesRequest);
        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::String> &queueUrls =
 outcome.GetResult().GetQueueUrls();
            allQueueUrls.insert(allQueueUrls.end(),
                                queueUrls.begin(),
                                queueUrls.end());
```

```
            nextToken = outcome.GetResult().GetNextToken();
        }
        else {
            std::cerr << "Error listing queues: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

    } while (!nextToken.empty());

    std::cout << allQueueUrls.size() << " Amazon SQS queue(s) found." <<
 std::endl;
    for (const auto &iter: allQueueUrls) {
        std::cout << " " << iter << std::endl;
    }

    return true;
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考ListQueues中的。

CLI

AWS CLI

列出队列

此示例将列出所有队列。

命令:

```
aws sqs list-queues
```

输出：

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue",
```

```
    "https://queue.amazonaws.com/80398EXAMPLE/TestQueue1",
        "https://queue.amazonaws.com/80398EXAMPLE/TestQueue2"
  ]
}
```

此示例仅列出以“My”开头的队列。

命令:

```
aws sqs list-queues --queue-name-prefix My
```

输出：

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListQueues中的。

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
package main

import (
 "context"
 "fmt"
```

```
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get queues. Here's why: %v\n", err)
   break
  } else {
   queueUrls = append(queueUrls, output.QueueUrls...)
  }
 }
 if len(queueUrls) == 0 {
  fmt.Println("You don't have any queues!")
 } else {
  for _, queueUrl := range queueUrls {
   fmt.Printf("\t%v\n", queueUrl)
  }
 }
}
```

- 有关 API 的详细信息，请参阅 适用于 Go 的 AWS SDK API 参考[ListQueues](#)中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        String prefix = "que";

        try {
            ListQueuesRequest listQueuesRequest =
ListQueuesRequest.builder().queueNamePrefix(prefix).build();
            ListQueuesResponse listQueuesResponse =
sqsClient.listQueues(listQueuesRequest);
            for (String url : listQueuesResponse.queueUrls()) {
                System.out.println(url);
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListQueues中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出 Amazon SQS 队列。

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    for (const url of urls) {
      console.log(url);
    }
  }

  return urls;
};
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-listing-queues。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考ListQueues中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出 Amazon SQS 队列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
```

```
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-listing-queues。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考ListQueues中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun listQueues() {
    println("\nList Queues")

    val prefix = "que"
    val listQueuesRequest =
        ListQueuesRequest {
            queueNamePrefix = prefix
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.listQueues(listQueuesRequest)
        response.queueUrls?.forEach { url ->
            println(url)
```

```
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用ListQueues于 K otlin 的AWS SDK API 参考。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例列出了所有队列。

```
Get-SQSQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/AnotherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/DeadLetterQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

示例 2：此示例列出了具有指定名称的所有队列。

```
Get-SQSQueue -QueueNamePrefix My
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) ListQueues中的。

适用于 PowerShell V5 的工具

示例 1：此示例列出了所有队列。

```
Get-SQSQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/AnotherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/DeadLetterQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

示例 2：此示例列出了具有指定名称的所有队列。

```
Get-SQSQueue -QueueNamePrefix My
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) ListQueues中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
def get_queues(prefix=None):
    """
    Gets a list of SQS queues. When a prefix is specified, only queues with names
    that start with the prefix are returned.

    :param prefix: The prefix used to restrict the list of returned queues.
    :return: A list of Queue objects.
    """
    if prefix:
```

```
        queue_iter = sqs.queues.filter(QueueNamePrefix=prefix)
    else:
        queue_iter = sqs.queues.all()
    queues = list(queue_iter)
    if queues:
        logger.info("Got queues: %s", ", ".join([q.url for q in queues]))
    else:
        logger.warning("No queues found.")
    return queues
```

- 有关 API 的详细信息，请参阅适用ListQueues于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @example
#   list_queue_urls(Aws::SQS::Client.new(region: 'us-west-2'))
def list_queue_urls(sqs_client)
  queues = sqs_client.list_queues

  queues.queue_urls.each do |url|
    puts url
  end
rescue StandardError => e
  puts "Error listing queue URLs: #{e.message}"
end
```

```
# Lists the attributes of a queue in Amazon Simple Queue Service (Amazon SQS).
#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @example
#   list_queue_attributes(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
#   )
def list_queue_attributes(sqs_client, queue_url)
  attributes = sqs_client.get_queue_attributes(
    queue_url: queue_url,
    attribute_names: ['All']
  )

  attributes.attributes.each do |key, value|
    puts "#{key}: #{value}"
  end
rescue StandardError => e
  puts "Error getting queue attributes: #{e.message}"
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'

  sqs_client = Aws::SQS::Client.new(region: region)

  puts 'Listing available queue URLs...'
  list_queue_urls(sqs_client)

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  puts "\nGetting information about queue '#{queue_name}'..."
  list_queue_attributes(sqs_client, queue_url)
end
```

```
```

- 有关 API 的详细信息，请参阅 适用于 Ruby 的 AWS SDK API 参考ListQueues中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

检索该区域中列出的第一个 Amazon SQS 队列。

```
async fn find_first_queue(client: &Client) -> Result<String, Error> {
    let queues = client.list_queues().send().await?;
    let queue_urls = queues.queue_urls();
    Ok(queue_urls
        .first()
        .expect("No queues in this account and Region. Create a queue to
 proceed.")
        .to_string())
}
```

- 有关 API 的详细信息，请参阅适用ListQueues于 Rust 的AWS SDK API 参考。

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    TRY.
```

```
        oo_result = lo_sqs->listqueues( ).        " oo_result is returned for
 testing purposes. "
        MESSAGE 'Retrieved list of queues.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用ListQueues于 S AP 的AWS SDK ABAP API 参考。

Swift

适用于 Swift 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        var queues: [String] = []
        let outputPages = sqsClient.listQueuesPaginated(
            input: ListQueuesInput()
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let urls = output.queueUrls else {
                print("No queues found.")
                return
            }

            // Iterate over the queue URLs listed on this page, adding them
            // to the `queues` array.

            for queueUrl in urls {
                queues.append(queueUrl)
            }
```

```
        }
```

- 如需了解 API 的详细信息，请参阅适用ListQueues于 S wift 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **PurgeQueue** 与 CLI 配合使用

以下代码示例演示如何使用 PurgeQueue。

CLI

AWS CLI

清除队列

此示例删除指定队列中的所有消息。

命令:

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue
```

输出：

```
None.
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考PurgeQueue中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例删除了指定队列中的所有消息。

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) PurgeQueue中的。

适用于 PowerShell V5 的工具

示例 1：此示例删除了指定队列中的所有消息。

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) PurgeQueue中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ReceiveMessage`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ReceiveMessage`。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 使用 S3 管理大型消息
- 处理 S3 事件通知
- 将消息发布到队列
- 发送和接收批量消息

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用队列的 URL 接收来自队列的消息。

```
    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
 maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }
```

从 Amazon SQS 队列接收消息，然后删除该消息。

```
        public static async Task Main()
        {
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
        }

        /// <summary>
        /// Retrieve the queue URL for the queue named in the queueName
```

```
        /// property using the client object.
        /// </summary>
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
 queueName)
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await
 client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }

        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</
param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
        public static async Task<ReceiveMessageResponse>
 ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };
```

```
            var receiveMessageResponse = await
 client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };

            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ReceiveMessage中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Receive a message from an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::receiveMessage(const Aws::String &queueUrl,
```

```
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::ReceiveMessageRequest request;
    request.SetQueueUrl(queueUrl);
    request.SetMaxNumberOfMessages(1);

    const Aws::SQS::Model::ReceiveMessageOutcome outcome =
 sqsClient.ReceiveMessage(
            request);
    if (outcome.IsSuccess()) {

        const Aws::Vector<Aws::SQS::Model::Message> &messages =
                outcome.GetResult().GetMessages();
        if (!messages.empty()) {
            const Aws::SQS::Model::Message &message = messages[0];
            std::cout << "Received message:" << std::endl;
            std::cout << "  MessageId: " << message.GetMessageId() << std::endl;
            std::cout << "  ReceiptHandle: " << message.GetReceiptHandle() <<
 std::endl;
            std::cout << "  Body: " << message.GetBody() << std::endl <<
 std::endl;
        }
        else {
            std::cout << "No messages received from queue " << queueUrl <<
                      std::endl;

        }
    }
    else {
        std::cerr << "Error receiving message from queue " << queueUrl << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考[ReceiveMessage](#)中的。

CLI

AWS CLI

接收消息

此示例最多接收 10 条可用消息，返回所有可用属性。

命令:

```
aws sqs receive-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All --message-attribute-names All --max-number-of-messages 10
```

输出：

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEBzbVv...fqNzFw==",
      "MD5OfBody": "1000f835...a35411fa",
      "MD5OfMessageAttributes": "9424c491...26bc3ae7",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "ApproximateFirstReceiveTimestamp": "1442428276921",
        "SenderId": "AIDAIAZKMSNQ7TEXAMPLE",
        "ApproximateReceiveCount": "5",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        },
        "City": {
          "DataType": "String",
          "StringValue": "Any City"
        }
      }
    }
  ]
}
```

此示例接收下一条可用消息，仅返回 SenderId 和 SentTimestamp 属性以及 m PostalCode essage 属性。

命令:

```
aws sqs receive-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-
names SenderId SentTimestamp --message-attribute-names PostalCode
```

输出：

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEB6nR4...HzlvZQ==",
      "MD5OfBody": "1000f835...a35411fa",
      "MD5OfMessageAttributes": "b8e89563...e088e74f",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "SenderId": "AIDAIAZKMSNQ7TEXAMPLE",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        }
      }
    }
  ]
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ReceiveMessage中的。

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
```

```
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 } else {
  messages = result.Messages
 }
 return messages, err
}
```

- 有关 API 的详细信息，请参阅 适用于 Go 的 AWS SDK API 参考ReceiveMessage中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        try {
            ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .maxNumberOfMessages(5)
                    .build();
            return sqsClient.receiveMessage(receiveMessageRequest).messages();

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ReceiveMessage中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收来自 Amazon SQS 队列的消息。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
```

```
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

使用长轮询支持接收来自 Amazon SQS 队列的消息。

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new ReceiveMessageCommand({
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueUrl,
    // The duration (in seconds) for which the call waits for a message
    // to arrive in the queue before returning. If a message is available,
    // the call returns sooner than WaitTimeSeconds. If no messages are
    // available and the wait time expires, the call returns successfully
    // with an empty list of messages.
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
    WaitTimeSeconds: 20,
  });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考ReceiveMessage中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用长轮询接收来自 Amazon SQS 队列的消息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
```

```
});
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-enable-long-polling.html#sqs-examples-enable-long-polling-on-receive-message。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考ReceiveMessage中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun receiveMessages(queueUrlVal: String?) {
    println("Retrieving messages from $queueUrlVal")

    val receiveMessageRequest =
        ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.receiveMessage(receiveMessageRequest)
        response.messages?.forEach { message ->
            println(message.body)
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用ReceiveMessage于 K otlin 的AWS SDK API 参考。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例列出了指定队列将要接收的消息（最多 10 条）的信息。该信息将包含指定消息属性的值（如果存在）。

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

输出：

```
Attributes             : {[SenderId, AIDAIAZKMSNQ7TEXAMPLE], [SentTimestamp,
 1451495923744]}
Body                   : Information about John Doe's grade.
MD5OfBody              : ea572796e3c231f974fe75d89EXAMPLE
MD5OfMessageAttributes : 48c1ee811f0fe7c4e88fbe0f5EXAMPLE
MessageAttributes      : {[StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
 [StudentName, Amazon.SQS.Model.MessageAttributeValue]}
MessageId              : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle          : AQEBpfGp...20Q5cg==
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) ReceiveMessage中的。

适用于 PowerShell V5 的工具

示例 1：此示例列出了指定队列将要接收的消息（最多 10 条）的信息。该信息将包含指定消息属性的值（如果存在）。

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

输出：

```
Attributes             : {[SenderId, AIDAIAZKMSNQ7TEXAMPLE], [SentTimestamp,
 1451495923744]}
Body                   : Information about John Doe's grade.
MD5OfBody              : ea572796e3c231f974fe75d89EXAMPLE
MD5OfMessageAttributes : 48c1ee811f0fe7c4e88fbe0f5EXAMPLE
```

```
MessageAttributes      : {[StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
 [StudentName, Amazon.SQS.Model.MessageAttributeValue]}
MessageId              : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle          : AQEBpfGp...20Q5cg==
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) ReceiveMessage中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
def receive_messages(queue, max_number, wait_time):
    """
    Receive a batch of messages in a single request from an SQS queue.

    :param queue: The queue from which to receive messages.
    :param max_number: The maximum number of messages to receive. The actual
 number
                       of messages received might be less.
    :param wait_time: The maximum time to wait (in seconds) before returning.
 When
                      this number is greater than zero, long polling is used.
 This
                      can result in reduced costs and fewer false empty
 responses.
    :return: The list of Message objects received. These each contain the body
             of the message and metadata and custom attributes.
    """
    try:
        messages = queue.receive_messages(
            MessageAttributeNames=["All"],
            MaxNumberOfMessages=max_number,
            WaitTimeSeconds=wait_time,
        )
```

```
        for msg in messages:
            logger.info("Received message: %s: %s", msg.message_id, msg.body)
    except ClientError as error:
        logger.exception("Couldn't receive messages from queue: %s", queue)
        raise error
    else:
        return messages
```

```
class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)


    def receive_messages(self, queue_url: str, max_messages: int = 10) ->
 List[Dict[str, Any]]:
        """
        Receive messages from an SQS queue.

        :param queue_url: The URL of the queue to receive messages from.
        :param max_messages: Maximum number of messages to receive (1-10).
        :return: List of received messages.
        :raises ClientError: If receiving messages fails.
        """
```

```
        try:
            # Ensure max_messages is within valid range
            max_messages = max(1, min(10, max_messages))

            response = self.sqs_client.receive_message(
                QueueUrl=queue_url,
                MaxNumberOfMessages=max_messages,
                WaitTimeSeconds=2,  # Short polling
                MessageAttributeNames=['All']
            )

            messages = response.get('Messages', [])
            logger.info(f"Received {len(messages)} messages from {queue_url}")
            return messages

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error receiving messages: {error_code} - {e}")
            raise
```

- 有关 API 的详细信息，请参阅适用ReceiveMessage于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

# Receives messages in a queue in Amazon Simple Queue Service (Amazon SQS).
#
```

```
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param max_number_of_messages [Integer] The maximum number of messages
#   to receive. This number must be 10 or less. The default is 10.
# @example
#   receive_messages(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     10
#   )
def receive_messages(sqs_client, queue_url, max_number_of_messages = 10)
  if max_number_of_messages > 10
    puts 'Maximum number of messages to receive must be 10 or less. ' \
      'Stopping program.'
    return
  end

  response = sqs_client.receive_message(
    queue_url: queue_url,
    max_number_of_messages: max_number_of_messages
  )

  if response.messages.count.zero?
    puts 'No messages to receive, or all messages have already ' \
      'been previously received.'
    return
  end

  response.messages.each do |message|
    puts '-' * 20
    puts "Message body: #{message.body}"
    puts "Message ID:   #{message.message_id}"
  end
rescue StandardError => e
  puts "Error receiving messages: #{e.message}"
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  max_number_of_messages = 10
```

```
  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Receiving messages from queue '#{queue_name}'..."

  receive_messages(sqs_client, queue_url, max_number_of_messages)
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- 有关 API 的详细信息，请参阅 适用于 Ruby 的 AWS SDK API 参考ReceiveMessage中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
async fn receive(client: &Client, queue_url: &String) -> Result<(), Error> {
    let rcv_message_output =
 client.receive_message().queue_url(queue_url).send().await?;

    println!("Messages from queue with url: {}", queue_url);

    for message in rcv_message_output.messages.unwrap_or_default() {
        println!("Got the message: {:#?}", message);
    }

    Ok(())
}
```

- 有关 API 的详细信息，请参阅适用ReceiveMessage于 Rust 的AWS SDK API 参考。

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收来自 Amazon SQS 队列的消息。

```
    TRY.
        oo_result = lo_sqs->receivemessage( iv_queueurl = iv_queue_url ).    "
oo_result is returned for testing purposes. "
        DATA(lt_messages) = oo_result->get_messages( ).
        MESSAGE 'Message received from SQS queue.' TYPE 'I'.
      CATCH /aws1/cx_sqsoverlimit.
        MESSAGE 'Maximum number of in-flight messages reached.' TYPE 'E'.
    ENDTRY.
```

使用长轮询支持接收来自 Amazon SQS 队列的消息。

```
    TRY.
        oo_result = lo_sqs->receivemessage(           " oo_result is returned for
testing purposes. "
                iv_queueurl = iv_queue_url
                iv_waittimeseconds = iv_wait_time ).    " Time in seconds for
long polling, such as how long the call waits for a message to arrive in the
queue before returning. " ).
        DATA(lt_messages) = oo_result->get_messages( ).
        MESSAGE 'Message received from SQS queue.' TYPE 'I'.
      CATCH /aws1/cx_sqsoverlimit.
        MESSAGE 'Maximum number of in-flight messages reached.' TYPE 'E'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用ReceiveMessage于 S AP 的AWS SDK ABAP API 参考。

Swift

适用于 Swift 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        let output = try await sqsClient.receiveMessage(
            input: ReceiveMessageInput(
                maxNumberOfMessages: maxMessages,
                queueUrl: url
            )
        )

        guard let messages = output.messages else {
            print("No messages received.")
            return
        }

        for message in messages {
            print("Message ID:     \(message.messageId ?? "<unknown>")")
            print("Receipt handle: \(message.receiptHandle ?? "<unknown>")")
            print(message.body ?? "<body missing>")
            print("---")
        }
```

- 如需了解 API 的详细信息，请参阅适用ReceiveMessage于 S wift 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **RemovePermission** 与 CLI 配合使用

以下代码示例演示如何使用 RemovePermission。

CLI

AWS CLI

删除权限

此示例从指定队列中移除具有指定标签的权限。

命令:

```
aws sqs remove-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue
```

输出：

```
None.
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考RemovePermission中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例从指定队列中删除具有指定标签的权限设置。

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) RemovePermission中的。

适用于 PowerShell V5 的工具

示例 1：此示例从指定队列中删除具有指定标签的权限设置。

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) RemovePermission中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

SendMessage与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 SendMessage。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 使用 S3 管理大型消息
- 发送和接收批量消息

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 Amazon SQS 队列并向其发送消息。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.SQS;
using Amazon.SQS.Model;

public class CreateSendExample
{
```

```
        // Specify your AWS Region (an example Region is shown).
        private static readonly string QueueName = "Example_Queue";
        private static readonly RegionEndpoint ServiceRegion =
 RegionEndpoint.USWest2;
        private static IAmazonSQS client;

        public static async Task Main()
        {
            client = new AmazonSQSClient(ServiceRegion);
            var createQueueResponse = await CreateQueue(client, QueueName);

            string queueUrl = createQueueResponse.QueueUrl;

            Dictionary<string, MessageAttributeValue> messageAttributes = new
 Dictionary<string, MessageAttributeValue>
            {
                { "Title",   new MessageAttributeValue { DataType = "String",
 StringValue = "The Whistler" } },
                { "Author",  new MessageAttributeValue { DataType = "String",
 StringValue = "John Grisham" } },
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
 StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
 bestseller for week of 12/11/2016.";

            var sendMsgResponse = await SendMessage(client, queueUrl,
 messageBody, messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS
 client, string queueName)
        {
```

```
            var request = new CreateQueueRequest
            {
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

            return response;
        }

        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
```

```
            Console.WriteLine($"Sent a message with id : {response.MessageId}");

            return response;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考SendMessage中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Send a message to an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param messageBody: A message body.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::sendMessage(const Aws::String &queueUrl,
                              const Aws::String &messageBody,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SendMessageRequest request;
    request.SetQueueUrl(queueUrl);
    request.SetMessageBody(messageBody);
```

```
    const Aws::SQS::Model::SendMessageOutcome outcome =
 sqsClient.SendMessage(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully sent message to " << queueUrl <<
                  std::endl;
    }
    else {
        std::cerr << "Error sending message to " << queueUrl << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考SendMessage中的。

CLI

AWS CLI

发送邮件

此示例向指定队列发送一条具有指定消息正文、延迟时间和消息属性的消息。

命令:

```
aws sqs send-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --message-body "Information about the
 largest city in Any Region." --delay-seconds 10 --message-attributes file://
send-message.json
```

输入文件（send-message.json）：

```
{
  "City": {
    "DataType": "String",
    "StringValue": "Any City"
  },
  "Greeting": {
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
  },
```

```
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

输出：

```
{
  "MD5OfMessageBody": "51b0a325...39163aa0",
  "MD5OfMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考SendMessage中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

以下是 SendMessage 操作的两个示例：

- 发送带有正文和延迟的消息
- 发送带有正文和消息属性的消息

发送带有正文和延迟的消息。

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

/**
```

```
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SendMessages {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <queueName> <message>

                Where:
                   queueName - The name of the queue.
                   message - The message to send.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String queueName = args[0];
        String message = args[1];
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();
        sendMessage(sqsClient, queueName, message);
        sqsClient.close();
    }

    public static void sendMessage(SqsClient sqsClient, String queueName, String
 message) {
        try {
            CreateQueueRequest request = CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();
            sqsClient.createQueue(request);

            GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
                    .queueName(queueName)
                    .build();
```

```
            String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();
            SendMessageRequest sendMsgRequest = SendMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageBody(message)
                    .delaySeconds(5)
                    .build();

            sqsClient.sendMessage(sendMsgRequest);

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

发送带有正文和消息属性的消息。

```
    /**
     * <p>This method demonstrates how to add message attributes to a message.
     * Each attribute must specify a name, value, and data type. You use a Java
 Map to supply the attributes. The map's
     * key is the attribute name, and you specify the map's entry value using a
 builder that includes the attribute
     * value and data type.</p>
     *
     * <p>The data type must start with one of "String", "Number" or "Binary".
 You can optionally
     * define a custom extension by using a "." and your extension.</p>
     *
     * <p>The SQS Developer Guide provides more information on @see <a
     * href="https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-message-metadata.html#sqs-message-attributes">message
     * attributes</a>.</p>
     *
     * @param thumbailPath Filesystem path of the image.
     * @param queueUrl     URL of the SQS queue.
     */
    static void sendMessageWithAttributes(Path thumbailPath, String queueUrl) {
        Map<String, MessageAttributeValue> messageAttributeMap;
        try {
```

```
            messageAttributeMap = Map.of(
                    "Name", MessageAttributeValue.builder()
                            .stringValue("Jane Doe")
                            .dataType("String").build(),
                    "Age", MessageAttributeValue.builder()
                            .stringValue("42")
                            .dataType("Number.int").build(),
                    "Image", MessageAttributeValue.builder()

.binaryValue(SdkBytes.fromByteArray(Files.readAllBytes(thumbailPath)))
                            .dataType("Binary.jpg").build()
            );
        } catch (IOException e) {
            LOGGER.error("An I/O exception occurred reading thumbnail image: {}",
e.getMessage(), e);
            throw new RuntimeException(e);
        }

        SendMessageRequest request = SendMessageRequest.builder()
                .queueUrl(queueUrl)
                .messageBody("Hello SQS")
                .messageAttributes(messageAttributeMap)
                .build();
        try {
            SendMessageResponse sendMessageResponse =
SQS_CLIENT.sendMessage(request);
            LOGGER.info("Message ID: {}", sendMessageResponse.messageId());
        } catch (SqsException e) {
            LOGGER.error("Exception occurred sending message: {}",
e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考SendMessage中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

向 Amazon SQS 队列发送消息。

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
    },
    MessageBody:
      "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
```

```
};
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-send-receive-messages.html#sqs-examples-send-receive-messages-sending。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考SendMessage中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

向 Amazon SQS 队列发送消息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
```

```
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  // MessageDeduplicationId: "TheWhistler",  // Required for FIFO queues
  // MessageGroupId: "Group1",  // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-send-receive-messages.html#sqs-examples-send-receive-messages-sending。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考SendMessage中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun sendMessages(
    queueUrlVal: String,
    message: String,
) {
    println("Sending multiple messages")
```

```
    println("\nSend message")
    val sendRequest =
        SendMessageRequest {
            queueUrl = queueUrlVal
            messageBody = message
            delaySeconds = 10
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.sendMessage(sendRequest)
        println("A single message was successfully sent.")
    }
}

suspend fun sendBatchMessages(queueUrlVal: String?) {
    println("Sending multiple messages")

    val msg1 =
        SendMessageBatchRequestEntry {
            id = "id1"
            messageBody = "Hello from msg 1"
        }

    val msg2 =
        SendMessageBatchRequestEntry {
            id = "id2"
            messageBody = "Hello from msg 2"
        }

    val sendMessageBatchRequest =
        SendMessageBatchRequest {
            queueUrl = queueUrlVal
            entries = listOf(msg1, msg2)
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.sendMessageBatch(sendMessageBatchRequest)
        println("Batch message were successfully sent.")
    }
}
```

- 有关 API 的详细信息，请参阅适用SendMessage于 K otlin 的AWS SDK API 参考。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例向指定队列发送一条具有指定属性和消息正文的消息，消息传递延迟 10 秒。

```
$cityAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Send-SQSMessage -DelayInSeconds 10 -MessageAttributes $messageAttributes -
MessageBody "Information about the largest city in Any Region." -QueueUrl
 https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

输出：

```
MD5OfMessageAttributes             MD5OfMessageBody                    MessageId

----------------------             ----------------                    ---------

1d3e51347bc042efbdf6dda31EXAMPLE    51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) SendMessage中的。

适用于 PowerShell V5 的工具

示例 1：此示例向指定队列发送一条具有指定属性和消息正文的消息，消息传递延迟 10 秒。

```
$cityAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
```

```
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Send-SQSMessage -DelayInSeconds 10 -MessageAttributes $messageAttributes -
MessageBody "Information about the largest city in Any Region." -QueueUrl
 https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

输出：

```
MD5OfMessageAttributes             MD5OfMessageBody                    MessageId

----------------------             ----------------                    ---------

1d3e51347bc042efbdf6dda31EXAMPLE   51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) SendMessage中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
def send_message(queue, message_body, message_attributes=None):
    """
    Send a message to an Amazon SQS queue.

    :param queue: The queue that receives the message.
    :param message_body: The body text of the message.
    :param message_attributes: Custom attributes of the message. These are key-
value
```

```
                                pairs that can be whatever you want.
    :return: The response from SQS that contains the assigned message ID.
    """
    if not message_attributes:
        message_attributes = {}

    try:
        response = queue.send_message(
            MessageBody=message_body, MessageAttributes=message_attributes
        )
    except ClientError as error:
        logger.exception("Send message failed: %s", message_body)
        raise error
    else:
        return response
```

- 有关 API 的详细信息，请参阅适用SendMessage于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param message_body [String] The contents of the message to be sent.
# @return [Boolean] true if the message was sent; otherwise, false.
# @example
#   exit 1 unless message_sent?(
#     Aws::SQS::Client.new(region: 'us-west-2'),
```

```
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     'This is my message.'
#   )
def message_sent?(sqs_client, queue_url, message_body)
  sqs_client.send_message(
    queue_url: queue_url,
    message_body: message_body
  )
  true
rescue StandardError => e
  puts "Error sending message: #{e.message}"
  false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  message_body = 'This is my message.'

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Sending a message to the queue named '#{queue_name}'..."

  if message_sent?(sqs_client, queue_url, message_body)
    puts 'Message sent.'
  else
    puts 'Message not sent.'
  end
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- 有关 API 的详细信息，请参阅 适用于 Ruby 的 AWS SDK API 参考SendMessage中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
async fn send(client: &Client, queue_url: &String, message: &SQSMessage) ->
 Result<(), Error> {
    println!("Sending message to queue with URL: {}", queue_url);

    let rsp = client
        .send_message()
        .queue_url(queue_url)
        .message_body(&message.body)
        // If the queue is FIFO, you need to set .message_deduplication_id
        // and message_group_id or configure the queue for
 ContentBasedDeduplication.
        .send()
        .await?;

    println!("Send message to the queue: {:#?}", rsp);

    Ok(())
}
```

- 有关 API 的详细信息，请参阅适用SendMessage于 Rust 的AWS SDK API 参考。

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    TRY.
        oo_result = lo_sqs->sendmessage(              " oo_result is returned for
testing purposes. "
          iv_queueurl = iv_queue_url
          iv_messagebody = iv_message ).
        MESSAGE 'Message sent to SQS queue.' TYPE 'I'.
      CATCH /aws1/cx_sqsinvalidmsgconts.
        MESSAGE 'Message contains non-valid characters.' TYPE 'E'.
      CATCH /aws1/cx_sqsunsupportedop.
        MESSAGE 'Operation not supported.' TYPE 'E'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用SendMessage于 S AP 的AWS SDK ABAP API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

SendMessageBatch与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 SendMessageBatch。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 发送和接收批量消息

CLI

AWS CLI

批量发送多条消息

此示例向指定队列发送 2 条具有指定消息正文、延迟时间和消息属性的消息。

命令:

```
aws sqs send-message-batch --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-
batch.json
```

输入文件 (send-message-batch.json)：

```
[
  {
    "Id": "FuelReport-0001-2015-09-16T140731Z",
        "MessageBody": "Fuel report for account 0001 on 2015-09-16 at 02:07:31
 PM.",
        "DelaySeconds": 10,
        "MessageAttributes": {
          "SellerName": {
            "DataType": "String",
                "StringValue": "Example Store"
     },
         "City": {
        "DataType": "String",
        "StringValue": "Any City"
     },
         "Region": {
           "DataType": "String",
                "StringValue": "WA"
     },
         "PostalCode": {
           "DataType": "String",
                "StringValue": "99065"
         },
         "PricePerGallon": {
           "DataType": "Number",
                "StringValue": "1.99"
     }
       }
 },
 {
    "Id": "FuelReport-0002-2015-09-16T140930Z",
        "MessageBody": "Fuel report for account 0002 on 2015-09-16 at 02:09:30
 PM.",
        "DelaySeconds": 10,
        "MessageAttributes": {
          "SellerName": {
            "DataType": "String",
                "StringValue": "Example Fuels"
     },
         "City": {
        "DataType": "String",
        "StringValue": "North Town"
```

```
      },
          "Region": {
            "DataType": "String",
                "StringValue": "WA"
      },
          "PostalCode": {
            "DataType": "String",
                "StringValue": "99123"
          },
          "PricePerGallon": {
            "DataType": "Number",
                "StringValue": "1.87"
      }
        }
  }
]
```

输出：

```
{
  "Successful": [
    {
      "MD5OfMessageBody": "203c4a38...7943237e",
      "MD5OfMessageAttributes": "10809b55...baf283ef",
      "Id": "FuelReport-0001-2015-09-16T140731Z",
      "MessageId": "d175070c-d6b8-4101-861d-adeb3EXAMPLE"
    },
    {
      "MD5OfMessageBody": "2cf0159a...c1980595",
      "MD5OfMessageAttributes": "55623928...ae354a25",
      "Id": "FuelReport-0002-2015-09-16T140930Z",
      "MessageId": "f9b7d55d-0570-413e-b9c5-a9264EXAMPLE"
    }
  ]
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考SendMessageBatch中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
            SendMessageBatchRequest sendMessageBatchRequest =
SendMessageBatchRequest.builder()
                    .queueUrl(queueUrl)

.entries(SendMessageBatchRequestEntry.builder().id("id1").messageBody("Hello
from msg 1").build(),

SendMessageBatchRequestEntry.builder().id("id2").messageBody("msg
2").delaySeconds(10)
                                        .build())
                    .build();
            sqsClient.sendMessageBatch(sendMessageBatchRequest);
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考SendMessageBatch中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例向指定队列发送 2 条具有指定属性和消息正文的消息。第一条消息的传递延迟了 15 秒，第二条消息的传递延迟了 10 秒。

```
$student1NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1NameAttributeValue.DataType = "String"
$student1NameAttributeValue.StringValue = "John Doe"

$student1GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1GradeAttributeValue.DataType = "Number"
$student1GradeAttributeValue.StringValue = "89"
```

```
$student2NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2NameAttributeValue.DataType = "String"
$student2NameAttributeValue.StringValue = "Jane Doe"

$student2GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2GradeAttributeValue.DataType = "Number"
$student2GradeAttributeValue.StringValue = "93"

$message1 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message1.DelaySeconds = 15
$message1.Id = "FirstMessage"
$message1.MessageAttributes.Add("StudentName", $student1NameAttributeValue)
$message1.MessageAttributes.Add("StudentGrade", $student1GradeAttributeValue)
$message1.MessageBody = "Information about John Doe's grade."

$message2 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message2.DelaySeconds = 10
$message2.Id = "SecondMessage"
$message2.MessageAttributes.Add("StudentName", $student2NameAttributeValue)
$message2.MessageAttributes.Add("StudentGrade", $student2GradeAttributeValue)
$message2.MessageBody = "Information about Jane Doe's grade."

Send-SQSMessageBatch -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -Entry $message1, $message2
```

输出：

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) SendMessageBatch中的。

适用于 PowerShell V5 的工具

示例 1：此示例向指定队列发送 2 条具有指定属性和消息正文的消息。第一条消息的传递延迟了 15 秒，第二条消息的传递延迟了 10 秒。

```
$student1NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1NameAttributeValue.DataType = "String"
```

```
$student1NameAttributeValue.StringValue = "John Doe"

$student1GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1GradeAttributeValue.DataType = "Number"
$student1GradeAttributeValue.StringValue = "89"

$student2NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2NameAttributeValue.DataType = "String"
$student2NameAttributeValue.StringValue = "Jane Doe"

$student2GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2GradeAttributeValue.DataType = "Number"
$student2GradeAttributeValue.StringValue = "93"

$message1 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message1.DelaySeconds = 15
$message1.Id = "FirstMessage"
$message1.MessageAttributes.Add("StudentName", $student1NameAttributeValue)
$message1.MessageAttributes.Add("StudentGrade", $student1GradeAttributeValue)
$message1.MessageBody = "Information about John Doe's grade."

$message2 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message2.DelaySeconds = 10
$message2.Id = "SecondMessage"
$message2.MessageAttributes.Add("StudentName", $student2NameAttributeValue)
$message2.MessageAttributes.Add("StudentGrade", $student2GradeAttributeValue)
$message2.MessageBody = "Information about Jane Doe's grade."

Send-SQSMessageBatch -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -Entry $message1, $message2
```

输出：

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) SendMessageBatch中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
def send_messages(queue, messages):
    """
    Send a batch of messages in a single request to an SQS queue.
    This request may return overall success even when some messages were not
 sent.
    The caller must inspect the Successful and Failed lists in the response and
    resend any failed messages.

    :param queue: The queue to receive the messages.
    :param messages: The messages to send to the queue. These are simplified to
                     contain only the message body and attributes.
    :return: The response from SQS that contains the list of successful and
 failed
             messages.
    """
    try:
        entries = [
            {
                "Id": str(ind),
                "MessageBody": msg["body"],
                "MessageAttributes": msg["attributes"],
            }
            for ind, msg in enumerate(messages)
        ]
        response = queue.send_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info(
                    "Message sent: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
                )
```

```
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Failed to send: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
                )
    except ClientError as error:
        logger.exception("Send messages failed to queue: %s", queue)
        raise error
    else:
        return response
```

- 有关 API 的详细信息，请参阅适用SendMessageBatch于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param entries [Hash] The contents of the messages to be sent,
#   in the correct format.
# @return [Boolean] true if the messages were sent; otherwise, false.
# @example
#   exit 1 unless messages_sent?(
```

```
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     [
#       {
#         id: 'Message1',
#         message_body: 'This is the first message.'
#       },
#       {
#         id: 'Message2',
#         message_body: 'This is the second message.'
#       }
#     ]
#   )
def messages_sent?(sqs_client, queue_url, entries)
  sqs_client.send_message_batch(
    queue_url: queue_url,
    entries: entries
  )
  true
rescue StandardError => e
  puts "Error sending messages: #{e.message}"
  false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  entries = [
    {
      id: 'Message1',
      message_body: 'This is the first message.'
    },
    {
      id: 'Message2',
      message_body: 'This is the second message.'
    }
  ]

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
```

```
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Sending messages to the queue named '#{queue_name}'..."

  if messages_sent?(sqs_client, queue_url, entries)
    puts 'Messages sent.'
  else
    puts 'Messages not sent.'
  end
end
```

- 有关 API 的详细信息，请参阅 适用于 Ruby 的 AWS SDK API 参考SendMessageBatch中的。

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    TRY.
        oo_result = lo_sqs->sendmessagebatch(         " oo_result is returned for
testing purposes. "
          iv_queueurl = iv_queue_url
          it_entries = it_messages ).
        MESSAGE 'Messages sent to SQS queue.' TYPE 'I'.
      CATCH /aws1/cx_sqsbtcentidsnotdist00.
        MESSAGE 'Two or more batch entries in the request have the same ID.' TYPE
'E'.
      CATCH /aws1/cx_sqsbatchreqtoolong.
        MESSAGE 'The length of all the messages put together is more than the
limit.' TYPE 'E'.
```

```
      CATCH /aws1/cx_sqsemptybatchrequest.
        MESSAGE 'The batch request does not contain any entries.' TYPE 'E'.
      CATCH /aws1/cx_sqsinvbatchentryid.
        MESSAGE 'The ID of a batch entry in a batch request is not valid.' TYPE
'E'.
      CATCH /aws1/cx_sqstoomanyentriesin00.
        MESSAGE 'The batch request contains more entries than allowed.' TYPE 'E'.
      CATCH /aws1/cx_sqsunsupportedop.
        MESSAGE 'Operation not supported.' TYPE 'E'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用SendMessageBatch于 S AP 的AWS SDK ABAP API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`SetQueueAttributes`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `SetQueueAttributes`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 将消息发布到队列

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

为主题设置队列的策略属性。

```
    /// <summary>
```

```
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string
topicArn, string queueUrl)
    {
        var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                              "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
                                               $"\"aws:SourceArn\":
\"{topicArn}\"" +
                                            "}" +
                                       "}" +
                                "}]" +
                            "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[SetQueueAttributes](#)中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Set the value for an attribute in an Amazon Simple Queue Service (Amazon SQS)
 queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param attributeName: An attribute name enum.
  \param attribute: The attribute value as a string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::setQueueAttributes(const Aws::String &queueURL,
                                     Aws::SQS::Model::QueueAttributeName
 attributeName,
                                     const Aws::String &attribute,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(queueURL);
    request.AddAttributes(
            attributeName,
            attribute);

    const Aws::SQS::Model::SetQueueAttributesOutcome outcome =
 sqsClient.SetQueueAttributes(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully set the attribute  " <<
```

```
 Aws::SQS::Model::QueueAttributeNameMapper::GetNameForQueueAttributeName(
                          attributeName)
                  << " with value " << attribute << " in queue " <<
                  queueURL << "." << std::endl;
    }
    else {
        std::cout << "Error setting attribute for  queue " <<
                  queueURL << ": " << outcome.GetError().GetMessage() <<
                  std::endl;
    }

    return outcome.IsSuccess();
}
```

配置死信队列。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Connect an Amazon Simple Queue Service (Amazon SQS) queue to an associated
//! dead-letter queue.
/*!
  \param srcQueueUrl: An Amazon SQS queue URL.
  \param deadLetterQueueARN: The Amazon Resource Name (ARN) of an Amazon SQS
 dead-letter queue.
  \param maxReceiveCount: The max receive count of a message before it is sent to
 the dead-letter queue.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::setDeadLetterQueue(const Aws::String &srcQueueUrl,
                                     const Aws::String &deadLetterQueueARN,
                                     int maxReceiveCount,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String redrivePolicy = MakeRedrivePolicy(deadLetterQueueARN,
 maxReceiveCount);

    Aws::SQS::SQSClient sqsClient(clientConfiguration);
```

```
    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(srcQueueUrl);
    request.AddAttributes(
            Aws::SQS::Model::QueueAttributeName::RedrivePolicy,
            redrivePolicy);

    const Aws::SQS::Model::SetQueueAttributesOutcome outcome =
            sqsClient.SetQueueAttributes(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully set dead letter queue for queue  " <<
                  srcQueueUrl << " to " << deadLetterQueueARN << std::endl;
    }
    else {
        std::cerr << "Error setting dead letter queue for queue " <<
                  srcQueueUrl << ": " << outcome.GetError().GetMessage() <<
                  std::endl;
    }

    return outcome.IsSuccess();
}

//! Make a redrive policy for a dead-letter queue.
/*!
  \param queueArn: An Amazon SQS ARN for the dead-letter queue.
  \param maxReceiveCount: The max receive count of a message before it is sent to
 the dead-letter queue.
  \return Aws::String: Policy as JSON string.
 */
Aws::String MakeRedrivePolicy(const Aws::String &queueArn, int maxReceiveCount) {
    Aws::Utils::Json::JsonValue redrive_arn_entry;
    redrive_arn_entry.AsString(queueArn);

    Aws::Utils::Json::JsonValue max_msg_entry;
    max_msg_entry.AsInteger(maxReceiveCount);

    Aws::Utils::Json::JsonValue policy_map;
    policy_map.WithObject("deadLetterTargetArn", redrive_arn_entry);
    policy_map.WithObject("maxReceiveCount", max_msg_entry);

    return policy_map.View().WriteReadable();
}
```

将 Amazon SQS 队列配置为使用长轮询。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Set the wait time for an Amazon Simple Queue Service (Amazon SQS) queue poll.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param pollTimeSeconds: The receive message wait time in seconds.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::setQueueLongPollingAttribute(const Aws::String &queueURL,
                                               const Aws::String
 &pollTimeSeconds,
                                               const
 Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(queueURL);
    request.AddAttributes(
            Aws::SQS::Model::QueueAttributeName::ReceiveMessageWaitTimeSeconds,
            pollTimeSeconds);

    const Aws::SQS::Model::SetQueueAttributesOutcome outcome =
 sqsClient.SetQueueAttributes(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated long polling time for queue " <<
                  queueURL << " to " << pollTimeSeconds << std::endl;
    }
    else {
        std::cout << "Error updating long polling time for queue " <<
                  queueURL << ": " << outcome.GetError().GetMessage() <<
                  std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考SetQueueAttributes中的。

CLI

AWS CLI

设置队列属性

此示例将指定队列的传输延迟设置为 10 秒，最大消息大小为 128 KB（128 KB * 1,024 字节），消息保留期为 3 天（3 天 * 24 小时 * 60 分钟 * 60 秒），接收消息等待时间为 20 秒，默认可见性超时为 60 秒。此示例还将指定的死信队列与最大接收数 1,000 条消息相关联。

命令:

```
aws sqs set-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-
attributes.json
```

输入文件 (set-queue-attributes.json)：

```
{
  "DelaySeconds": "10",
  "MaximumMessageSize": "131072",
  "MessageRetentionPeriod": "259200",
  "ReceiveMessageWaitTimeSeconds": "20",
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "VisibilityTimeout": "60"
}
```

输出：

```
None.
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考SetQueueAttributes中的。

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy
 to an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages
 to the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
```

```
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
   string(types.QueueAttributeNamePolicy): string(policyBytes),
  },
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string
```

- 有关 API 的详细信息，请参阅 适用于 Go 的 AWS SDK API 参考SetQueueAttributes中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

将 Amazon SQS 配置为使用采用自定义 KMS 密钥的服务器端加密（SSE）。

```
   public static void addEncryption(String queueName, String kmsMasterKeyAlias)
{
       SqsClient sqsClient = SqsClient.create();

       GetQueueUrlRequest urlRequest = GetQueueUrlRequest.builder()
               .queueName(queueName)
               .build();

       GetQueueUrlResponse getQueueUrlResponse;
       try {
           getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest);
       } catch (QueueDoesNotExistException e) {
           LOGGER.error(e.getMessage(), e);
           throw new RuntimeException(e);
       }
       String queueUrl = getQueueUrlResponse.queueUrl();


       Map<QueueAttributeName, String> attributes = Map.of(
               QueueAttributeName.KMS_MASTER_KEY_ID, kmsMasterKeyAlias,
               QueueAttributeName.KMS_DATA_KEY_REUSE_PERIOD_SECONDS, "140" //
Set the data key reuse period to 140 seconds.
       );                                                                    //
This is how long SQS can reuse the data key before requesting a new one from
KMS.
```

```
        SetQueueAttributesRequest attRequest =
 SetQueueAttributesRequest.builder()
                .queueUrl(queueUrl)
                .attributes(attributes)
                .build();
        try {
            sqsClient.setQueueAttributes(attRequest);
            LOGGER.info("The attributes have been applied to {}", queueName);
        } catch (InvalidAttributeNameException | InvalidAttributeValueException
 e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        } finally {
            sqsClient.close();
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考SetQueueAttributes中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

将 Amazon SQS 队列配置为使用长轮询。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      ReceiveMessageWaitTimeSeconds: "20",
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

配置死信队列。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";

export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      RedrivePolicy: JSON.stringify({
```

```
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
        // be processed (consumed) successfully.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-dead-letter-queues.html
        deadLetterTargetArn: deadLetterQueueArn,
        maxReceiveCount: "10",
      }),
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考[SetQueueAttributes](#)中的。

PowerShell

适用于 PowerShell V4 的工具

示例 1：此示例展示了如何设置策略，让队列订阅 SNS 主题。发布到主题的消息会被发送到订阅了该主题的队列。

```
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeName
 "QueueArn").QueueARN

# construct the policy and inject arns
$policy = @"
{
  "Version":"2012-10-17",
```

```
  "Id": "$qarn/SQSPOLICY",
  "Statement": [
      {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "SQS:SendMessage",
      "Resource": "$qarn",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "$topicarn"
          }
      }
    }
  ]
}
"@

# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

示例 2：此示例为指定队列设置指定属性。

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 4) SetQueueAttributes中的。

适用于 PowerShell V5 的工具

示例 1：此示例展示了如何设置策略，让队列订阅 SNS 主题。发布到主题的消息会被发送到订阅了该主题的队列。

```
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeName
 "QueueArn").QueueARN
```

```
# construct the policy and inject arns
$policy = @"
{
  "Version":"2012-10-17",
  "Id": "$qarn/SQSPOLICY",
  "Statement": [
      {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "SQS:SendMessage",
      "Resource": "$qarn",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "$topicarn"
          }
      }
    }
  ]
}
"@

# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

示例 2：此示例为指定队列设置指定属性。

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 (V 5) SetQueueAttributes中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

为主题设置队列的策略属性。

```
class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)


    def set_queue_policy_for_topic(self, queue_arn: str, topic_arn: str,
 queue_url: str) -> bool:
        """
        Set the queue policy to allow SNS to send messages to the queue.

        :param queue_arn: The ARN of the SQS queue.
        :param topic_arn: The ARN of the SNS topic.
        :param queue_url: The URL of the SQS queue.
        :return: True if successful.
```

```
        :raises ClientError: If setting the queue policy fails.
        """
        try:
            # Create policy that allows SNS to send messages to the queue
            policy = {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "sns.amazonaws.com"
                        },
                        "Action": "sqs:SendMessage",
                        "Resource": queue_arn,
                        "Condition": {
                            "ArnEquals": {
                                "aws:SourceArn": topic_arn
                            }
                        }
                    }
                ]
            }

            self.sqs_client.set_queue_attributes(
                QueueUrl=queue_url,
                Attributes={
                    'Policy': json.dumps(policy)
                }
            )

            logger.info(f"Set queue policy for {queue_url} to allow messages from
{topic_arn}")
            return True

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error setting queue policy: {error_code} - {e}")
            raise
```

- 有关 API 的详细信息，请参阅适用SetQueueAttributes于 Python 的AWS SDK (Boto3) API 参考。

Swift

适用于 Swift 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        do {
            _ = try await sqsClient.setQueueAttributes(
                input: SetQueueAttributesInput(
                    attributes: [
                        "MaximumMessageSize": "\(maxSize)"
                    ],
                    queueUrl: url
                )
            )
        } catch _ as AWSSQS.InvalidAttributeValue {
            print("Invalid maximum message size: \(maxSize) kB.")
        }
```

- 如需了解 API 的详细信息，请参阅适用SetQueueAttributes于 S wift 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 Amazon SQS 的场景 AWS SDKs

以下代码示例向您展示了如何使用在 Amazon SQS 中实现常见场景。 AWS SDKs这些场景演示了如何通过调用 Amazon SQS 中的多个函数或与其他 AWS 服务结合来完成特定任务。每个场景都包含完整源代码的链接，您可以在其中找到有关如何设置和运行代码的说明。

场景以中等水平的经验为目标，可帮助您结合具体环境了解服务操作。

示例

- 使用 Amazon SQS 创建用于发送和检索消息的 Web 应用程序
- 使用 Step Functions 创建 Messenger 应用程序
- 创建 Amazon Textract 浏览器应用程序
- 使用软件开发工具包创建并发布到 FIFO Amazon SNS 主题 AWS
- 使用 Amazon Rekognition 使用软件开发工具包检测视频中的人物和物体 AWS
- 使用带软件开发工具包的 Amazon S3 管理大型亚马逊 SQS 消息 AWS
- 使用软件开发工具包接收和处理 Amazon AWS S3 事件通知
- 使用软件开发工具包将 Amazon SNS 消息发布到亚马逊 SQS 队列 AWS
- 使用软件开发工具包通过 Amazon SQS 发送和接收批量消息 AWS
- 使用适用于.NET 的 AWS 消息处理框架发布和接收 Amazon SQS 消息
- 使用 Amazon SQS Java Messaging Library 处理 Amazon SQS 的 Java Message Service（JMS）接口
- 使用软件开发工具包处理队列标签和 Amazon SQS AWS

使用 Amazon SQS 创建用于发送和检索消息的 Web 应用程序

以下代码示例显示如何使用 Amazon SQS 创建消息传输应用程序。

Java

适用于 Java 的 SDK 2.x

演示如何使用 Amazon SQS API 开发用于发送和检索消息的 Spring REST API。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Comprehend
- Amazon SQS

Kotlin

适用于 Kotlin 的 SDK

演示如何使用 Amazon SQS API 开发用于发送和检索消息的 Spring REST API。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Comprehend
- Amazon SQS

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 Step Functions 创建 Messenger 应用程序

以下代码示例说明如何创建用于从数据库表中检索消息记录的 AWS Step Functions Messenger 应用程序。

Python

适用于 Python 的 SDK（Boto3）

演示如何使用 with 创建信使应用程序，该应用程序从亚马逊 DynamoDB 表中检索消息记录并适用于 Python (Boto3) 的 AWS SDK 通过 AWS Step Functions 亚马逊简单队列服务 (Amazon SQS) Simple SQUEE Service 将其发送。状态机集成了扫描数据库中是否有未发送消息的 AWS Lambda 功能。

- 创建检索并更新 Amazon DynamoDB 表中的消息记录的状态机。
- 更新状态机定义以便也将消息发送到 Amazon Simple Queue Service (Amazon SQS)。
- 启动和停止状态机运行。
- 使用服务集成从状态机连接到 Lambda、DynamoDB 和 Amazon SQS。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB
- Lambda

- Amazon SQS
- Step Functions

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

创建 Amazon Textract 浏览器应用程序

以下代码示例展示如何通过交互式应用程序探索 Amazon Textract 输出。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

演示如何使用 适用于 JavaScript 的 AWS SDK 来构建 React 应用程序，该应用程序使用 Amazon Textract 从文档图像中提取数据并将其显示在交互式网页中。此示例在 Web 浏览器中运行，需要经过身份验证的 Amazon Cognito 身份才能获得凭证。它使用 Amazon Simple Storage Service（Amazon S3）进行存储；对于通知，它将轮询订阅 Amazon Simple Notification Service（Amazon SNS）主题的 Amazon Simple Queue Service（Amazon SQS）队列。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

适用于 Python 的 SDK（Boto3）

演示如何 适用于 Python (Boto3) 的 AWS SDK 与 Amazon Textract 配合使用来检测文档图像中的文本、表单和表格元素。输入图像和 Amazon Textract 输出在 Tkinter 应用程序中显示，该应用程序可让您探索检测到的元素。

- 将文档图像提交到 Amazon Textract 并探索检测到的元素的输出。
- 将图像直接提交到 Amazon Textract，或通过 Amazon Simple Storage Service（Amazon S3）桶提交图像。
- 使用异步 APIs 启动任务，该任务在任务完成时向亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 主题发布通知。
- 轮询 Amazon Simple Queue Service (Amazon SQS) 队列，以获取任务完成消息并显示结果。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包创建并发布到 FIFO Amazon SNS 主题 AWS

以下代码示例展示如何创建并发布到 FIFO Amazon SNS 主题。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例

- 创建一个 Amazon SNS FIFO 主题、两个 Amazon SQS FIFO 队列和一个标准队列。
- 将队列订阅到主题，发布一条消息到主题。

该测试验证每个队列是否收到消息。完整的示例还显示了添加访问策略，并在最后删除了资源。

```
public class PriceUpdateExample {
    public final static SnsClient snsClient = SnsClient.create();
    public final static SqsClient sqsClient = SqsClient.create();

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "    <topicName> <wholesaleQueueFifoName> <retailQueueFifoName>
 <analyticsQueueName>\n\n" +
            "Where:\n" +
            "   fifoTopicName - The name of the FIFO topic that you want to
 create. \n\n" +
            "   wholesaleQueueARN - The name of a SQS FIFO queue that will be
 created for the wholesale consumer. \n\n"
            +
            "   retailQueueARN - The name of a SQS FIFO queue that will created
 for the retail consumer. \n\n" +
            "   analyticsQueueARN - The name of a SQS standard queue that will be
 created for the analytics consumer. \n\n";
        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        final String fifoTopicName = args[0];
        final String wholeSaleQueueName = args[1];
        final String retailQueueName = args[2];
        final String analyticsQueueName = args[3];

        // For convenience, the QueueData class holds metadata about a queue:
 ARN, URL,
        // name and type.
        List<QueueData> queues = List.of(
            new QueueData(wholeSaleQueueName, QueueType.FIFO),
            new QueueData(retailQueueName, QueueType.FIFO),
            new QueueData(analyticsQueueName, QueueType.Standard));

        // Create queues.
        createQueues(queues);

        // Create a topic.
```

```
        String topicARN = createFIFOTopic(fifoTopicName);

        // Subscribe each queue to the topic.
        subscribeQueues(queues, topicARN);

        // Allow the newly created topic to send messages to the queues.
        addAccessPolicyToQueuesFINAL(queues, topicARN);

        // Publish a sample price update message with payload.
        publishPriceUpdate(topicARN, "{\"product\": 214, \"price\": 79.99}",
"Consumables");

        // Clean up resources.
        deleteSubscriptions(queues);
        deleteQueues(queues);
        deleteTopic(topicARN);
    }

    public static String createFIFOTopic(String topicName) {
        try {
            // Create a FIFO topic by using the SNS service client.
            Map<String, String> topicAttributes = Map.of(
                "FifoTopic", "true",
                "ContentBasedDeduplication", "false",
                "FifoThroughputScope", "MessageGroup");

            CreateTopicRequest topicRequest = CreateTopicRequest.builder()
                .name(topicName)
                .attributes(topicAttributes)
                .build();

            CreateTopicResponse response = snsClient.createTopic(topicRequest);
            String topicArn = response.topicArn();
            System.out.println("The topic ARN is" + topicArn);

            return topicArn;

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
```

```
   public static void subscribeQueues(List<QueueData> queues, String topicARN) {
       queues.forEach(queue -> {
           SubscribeRequest subscribeRequest = SubscribeRequest.builder()
               .topicArn(topicARN)
               .endpoint(queue.queueARN)
               .protocol("sqs")
               .build();

           // Subscribe to the endpoint by using the SNS service client.
           // Only Amazon SQS queues can receive notifications from an Amazon
SNS FIFO
           // topic.
           SubscribeResponse subscribeResponse =
snsClient.subscribe(subscribeRequest);
           System.out.println("The queue [" + queue.queueARN + "] subscribed to
the topic [" + topicARN + "]");
           queue.subscriptionARN = subscribeResponse.subscriptionArn();
       });
   }

   public static void publishPriceUpdate(String topicArn, String payload, String
groupId) {

       try {
           // Create and publish a message that updates the wholesale price.
           String subject = "Price Update";
           String dedupId = UUID.randomUUID().toString();
           String attributeName = "business";
           String attributeValue = "wholesale";

           MessageAttributeValue msgAttValue = MessageAttributeValue.builder()
               .dataType("String")
               .stringValue(attributeValue)
               .build();

           Map<String, MessageAttributeValue> attributes = new HashMap<>();
           attributes.put(attributeName, msgAttValue);
           PublishRequest pubRequest = PublishRequest.builder()
               .topicArn(topicArn)
               .subject(subject)
               .message(payload)
               .messageGroupId(groupId)
               .messageDeduplicationId(dedupId)
               .messageAttributes(attributes)
```

```
                .build();

            final PublishResponse response = snsClient.publish(pubRequest);
            System.out.println(response.messageId());
            System.out.println(response.sequenceNumber());
            System.out.println("Message was published to " + topicArn);

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API Reference》中的以下主题。
  - CreateTopic
  - 发布
  - Subscribe

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 Amazon SNS FIFO 主题，将 Amazon SQS FIFO 队列和标准队列订阅到主题，并发布一条消息到主题。

```
def usage_demo():
    """Shows how to subscribe queues to a FIFO topic."""
    print("-" * 88)
    print("Welcome to the `Subscribe queues to a FIFO topic` demo!")
    print("-" * 88)

    sns = boto3.resource("sns")
    sqs = boto3.resource("sqs")
```

```
    fifo_topic_wrapper = FifoTopicWrapper(sns)
    sns_wrapper = SnsWrapper(sns)

    prefix = "sqs-subscribe-demo-"
    queues = set()
    subscriptions = set()

    wholesale_queue = sqs.create_queue(
        QueueName=prefix + "wholesale.fifo",
        Attributes={
            "MaximumMessageSize": str(4096),
            "ReceiveMessageWaitTimeSeconds": str(10),
            "VisibilityTimeout": str(300),
            "FifoQueue": str(True),
            "ContentBasedDeduplication": str(True),
        },
    )
    queues.add(wholesale_queue)
    print(f"Created FIFO queue with URL: {wholesale_queue.url}.")

    retail_queue = sqs.create_queue(
        QueueName=prefix + "retail.fifo",
        Attributes={
            "MaximumMessageSize": str(4096),
            "ReceiveMessageWaitTimeSeconds": str(10),
            "VisibilityTimeout": str(300),
            "FifoQueue": str(True),
            "ContentBasedDeduplication": str(True),
        },
    )
    queues.add(retail_queue)
    print(f"Created FIFO queue with URL: {retail_queue.url}.")

    analytics_queue = sqs.create_queue(QueueName=prefix + "analytics",
Attributes={})
    queues.add(analytics_queue)
    print(f"Created standard queue with URL: {analytics_queue.url}.")

    topic = fifo_topic_wrapper.create_fifo_topic("price-updates-topic.fifo")
    print(f"Created FIFO topic: {topic.attributes['TopicArn']}.")

    for q in queues:
        fifo_topic_wrapper.add_access_policy(q, topic.attributes["TopicArn"])
```

```
    print(f"Added access policies for topic: {topic.attributes['TopicArn']}.")

    for q in queues:
        sub = fifo_topic_wrapper.subscribe_queue_to_topic(
            topic, q.attributes["QueueArn"]
        )
        subscriptions.add(sub)

    print(f"Subscribed queues to topic: {topic.attributes['TopicArn']}.")

    input("Press Enter to publish a message to the topic.")

    message_id = fifo_topic_wrapper.publish_price_update(
        topic, '{"product": 214, "price": 79.99}', "Consumables"
    )

    print(f"Published price update with message ID: {message_id}.")

    # Clean up the subscriptions, queues, and topic.
    input("Press Enter to clean up resources.")
    for s in subscriptions:
        sns_wrapper.delete_subscription(s)

    sns_wrapper.delete_topic(topic)

    for q in queues:
        fifo_topic_wrapper.delete_queue(q)

    print(f"Deleted subscriptions, queues, and topic.")

    print("Thanks for watching!")
    print("-" * 88)



class FifoTopicWrapper:
    """Encapsulates Amazon SNS FIFO topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource
```

```
    def create_fifo_topic(self, topic_name):
        """
        Create a FIFO topic.
        Topic names must be made up of only uppercase and lowercase ASCII
letters,
        numbers, underscores, and hyphens, and must be between 1 and 256
characters long.
        For a FIFO topic, the name must end with the .fifo suffix.

        :param topic_name: The name for the topic.
        :return: The new topic.
        """
        try:
            topic = self.sns_resource.create_topic(
                Name=topic_name,
                Attributes={
                    "FifoTopic": str(True),
                    "ContentBasedDeduplication": str(False),
                    "FifoThroughputScope": "MessageGroup",
                },
            )
            logger.info("Created FIFO topic with name=%s.", topic_name)
            return topic
        except ClientError as error:
            logger.exception("Couldn't create topic with name=%s!", topic_name)
            raise error


    @staticmethod
    def add_access_policy(queue, topic_arn):
        """
        Add the necessary access policy to a queue, so
        it can receive messages from a topic.

        :param queue: The queue resource.
        :param topic_arn: The ARN of the topic.
        :return: None.
        """
        try:
            queue.set_attributes(
                Attributes={
                    "Policy": json.dumps(
                        {
                            "Version":"2012-10-17",
```

```
                        "Statement": [
                            {
                                "Sid": "test-sid",
                                "Effect": "Allow",
                                "Principal": {"AWS": "*"},
                                "Action": "SQS:SendMessage",
                                "Resource": queue.attributes["QueueArn"],
                                "Condition": {
                                    "ArnLike": {"aws:SourceArn": topic_arn}
                                },
                            }
                        ],
                    }
                )
            }
        )
        logger.info("Added trust policy to the queue.")
    except ClientError as error:
        logger.exception("Couldn't add trust policy to the queue!")
        raise error


@staticmethod
def subscribe_queue_to_topic(topic, queue_arn):
    """
    Subscribe a queue to a topic.

    :param topic: The topic resource.
    :param queue_arn: The ARN of the queue.
    :return: The subscription resource.
    """
    try:
        subscription = topic.subscribe(
            Protocol="sqs",
            Endpoint=queue_arn,
        )
        logger.info("The queue is subscribed to the topic.")
        return subscription
    except ClientError as error:
        logger.exception("Couldn't subscribe queue to topic!")
        raise error


@staticmethod
```

```
    def publish_price_update(topic, payload, group_id):
        """
        Compose and publish a message that updates the wholesale price.

        :param topic: The topic to publish to.
        :param payload: The message to publish.
        :param group_id: The group ID for the message.
        :return: The ID of the message.
        """
        try:
            att_dict = {"business": {"DataType": "String", "StringValue":
"wholesale"}}
            dedup_id = uuid.uuid4()
            response = topic.publish(
                Subject="Price Update",
                Message=payload,
                MessageAttributes=att_dict,
                MessageGroupId=group_id,
                MessageDeduplicationId=str(dedup_id),
            )
            message_id = response["MessageId"]
            logger.info("Published message to topic %s.", topic.arn)
        except ClientError as error:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise error
        return message_id


    @staticmethod
    def delete_queue(queue):
        """
        Removes an SQS queue. When run against an AWS account, it can take up to
        60 seconds before the queue is actually deleted.

        :param queue: The queue to delete.
        :return: None
        """
        try:
            queue.delete()
            logger.info("Deleted queue with URL=%s.", queue.url)
        except ClientError as error:
            logger.exception("Couldn't delete queue with URL=%s!", queue.url)
            raise error
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API Reference》中的以下主题。
  - CreateTopic
  - 发布
  - Subscribe

SAP ABAP

适用于 SAP ABAP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 FIFO 主题并为此订阅 Amazon SQS FIFO 队列，然后向 Amazon SNS 主题发布消息。

```
    " Creates a FIFO topic. "
    DATA lt_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>tt_topicattributesmap.
    DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>ts_topicattributesmap_maprow.
    ls_tpc_attributes-key = 'FifoTopic'.
    ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsmap_w( iv_value =
 'true' ).
    INSERT ls_tpc_attributes INTO TABLE lt_tpc_attributes.

    TRY.
        DATA(lo_create_result) = lo_sns->createtopic(
               iv_name = iv_topic_name
               it_attributes = lt_tpc_attributes ).
        DATA(lv_topic_arn) = lo_create_result->get_topicarn( ).
        ov_topic_arn = lv_topic_arn.                                        "
 ov_topic_arn is returned for testing purposes. "
        MESSAGE 'FIFO topic created' TYPE 'I'.
```

```
      CATCH /aws1/cx_snstopiclimitexcdex.
        MESSAGE 'Unable to create more topics. You have reached the maximum
 number of topics allowed.' TYPE 'E'.
    ENDTRY.

    " Subscribes an endpoint to an Amazon Simple Notification Service (Amazon
 SNS) topic. "
    " Only Amazon Simple Queue Service (Amazon SQS) FIFO queues can be subscribed
 to an SNS FIFO topic. "
    TRY.
        DATA(lo_subscribe_result) = lo_sns->subscribe(
               iv_topicarn = lv_topic_arn
               iv_protocol = 'sqs'
               iv_endpoint = iv_queue_arn ).
        DATA(lv_subscription_arn) = lo_subscribe_result->get_subscriptionarn( ).
        ov_subscription_arn = lv_subscription_arn.                        "
 ov_subscription_arn is returned for testing purposes. "
        MESSAGE 'SQS queue was subscribed to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
      CATCH /aws1/cx_snssubscriptionlmte00.
        MESSAGE 'Unable to create subscriptions. You have reached the maximum
 number of subscriptions allowed.' TYPE 'E'.
    ENDTRY.

    " Publish message to SNS topic. "
    TRY.
        DATA lt_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>tt_messageattributemap.
        DATA ls_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>ts_messageattributemap_maprow.
        ls_msg_attributes-key = 'Importance'.
        ls_msg_attributes-value = NEW /aws1/cl_snsmessageattrvalue( iv_datatype =
 'String'

 iv_stringvalue = 'High' ).
        INSERT ls_msg_attributes INTO TABLE lt_msg_attributes.

        DATA(lo_result) = lo_sns->publish(
             iv_topicarn = lv_topic_arn
             iv_message = 'The price of your mobile plan has been increased from
 $19 to $23'
             iv_subject = 'Changes to mobile plan'
             iv_messagegroupid = 'Update-2'
```

```
            iv_messagededuplicationid = 'Update-2.1'
            it_messageattributes = lt_msg_attributes ).
        ov_message_id = lo_result->get_messageid( ).                    "
ov_message_id is returned for testing purposes. "
        MESSAGE 'Message was published to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- 有关 API 详细信息，请参阅适用于 SAP ABAP 的AWS SDK 的 API 参考中的以下主题。
  - CreateTopic
  - 发布
  - Subscribe

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 Amazon Rekognition 使用软件开发工具包检测视频中的人物和物体 AWS

以下代码示例展示如何使用 Amazon Rekognition 检测视频中的人物和对象。

Java

适用于 Java 的 SDK 2.x

展示如何使用 Amazon Rekognition Java API 创建应用程序，以检测位于 Amazon Simple Storage Service (Amazon S3) 存储桶的视频当中的人脸和对象。该应用程序使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES

- Amazon SNS
- Amazon SQS

Python

适用于 Python 的 SDK（Boto3）

通过启动异步检测任务，使用 Amazon Rekognition 来检测视频中的人脸、对象和人物。此示例还将 Amazon Rekognition 配置为在任务完成时通知 Amazon Simple Notification Service (Amazon SNS) 主题，并订阅该主题的 Amazon Simple Queue Service (Amazon SQS) 队列。当队列收到有关任务的消息时，将检索该任务并输出结果。

最好在上查看此示例 GitHub。有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用带软件开发工具包的 Amazon S3 管理大型亚马逊 SQS 消息 AWS

以下代码示例演示如何使用 Amazon SQS 扩展型客户端库来处理大型 Amazon SQS 消息。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.BucketLifecycleConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.ExpirationStatus;
import software.amazon.awssdk.services.s3.model.LifecycleExpiration;
import software.amazon.awssdk.services.s3.model.LifecycleRule;
import software.amazon.awssdk.services.s3.model.LifecycleRuleFilter;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsRequest;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import
 software.amazon.awssdk.services.s3.model.PutBucketLifecycleConfigurationRequest;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;

import java.util.Arrays;
import java.util.List;
import java.util.UUID;

/**
 * Example of using Amazon SQS Extended Client Library for Java 2.x.
 */
public class SqsExtendedClientExample {
    private static final Logger logger =
 LoggerFactory.getLogger(SqsExtendedClientExample.class);

    private String s3BucketName;
```

```
    private String queueUrl;
    private final String queueName;
    private final S3Client s3Client;
    private final SqsClient sqsExtendedClient;
    private final int messageSize;

    /**
     * Constructor with default clients and message size.
     */
    public SqsExtendedClientExample() {
        this(S3Client.create(), 300000);
    }

    /**
     * Constructor with custom S3 client and message size.
     *
     * @param s3Client The S3 client to use
     * @param messageSize The size of the test message to create
     */
    public SqsExtendedClientExample(S3Client s3Client, int messageSize) {
        this.s3Client = s3Client;
        this.messageSize = messageSize;

        // Generate a unique bucket name.
        this.s3BucketName = UUID.randomUUID() + "-" +
                DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new DateTime());

        // Generate a unique queue name.
        this.queueName = "MyQueue-" + UUID.randomUUID();

        // Configure the SQS extended client.
        final ExtendedClientConfiguration extendedClientConfig = new
ExtendedClientConfiguration()
                .withPayloadSupportEnabled(s3Client, s3BucketName);

        this.sqsExtendedClient = new
AmazonSQSExtendedClient(SqsClient.builder().build(), extendedClientConfig);
    }

    public static void main(String[] args) {
        SqsExtendedClientExample example = new SqsExtendedClientExample();
        try {
            example.setup();
            example.sendAndReceiveMessage();
```

```
        } finally {
            example.cleanup();
        }
    }

    /**
     * Send a large message and receive it back.
     *
     * @return The received message
     */
    public Message sendAndReceiveMessage() {
        try {
            // Create a large message.
            char[] chars = new char[messageSize];
            Arrays.fill(chars, 'x');
            String largeMessage = new String(chars);

            // Send the message.
            final SendMessageRequest sendMessageRequest =
SendMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageBody(largeMessage)
                    .build();

            sqsExtendedClient.sendMessage(sendMessageRequest);
            logger.info("Sent message of size: {}", largeMessage.length());

            // Receive and return the message.
            final ReceiveMessageResponse receiveMessageResponse =
sqsExtendedClient.receiveMessage(
                    ReceiveMessageRequest.builder().queueUrl(queueUrl).build());

            List<Message> messages = receiveMessageResponse.messages();
            if (messages.isEmpty()) {
                throw new RuntimeException("No messages received");
            }

            Message message = messages.getFirst();
            logger.info("\nMessage received.");
            logger.info("  ID: {}", message.messageId());
            logger.info("  Receipt handle: {}", message.receiptHandle());
            logger.info("  Message body size: {}", message.body().length());
            logger.info("  Message body (first 5 characters): {}",
message.body().substring(0, 5));
```

```
            return message;
        } catch (RuntimeException e) {
            logger.error("Error during message processing: {}", e.getMessage(),
e);
            throw e;
        }
    }
```

- 有关更多信息，请参阅《AWS SDK for Java 2.x 开发人员指南》https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-s3-messages.html。
- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API Reference》中的以下主题。
  - CreateBucket
  - PutBucketLifecycleConfiguration
  - ReceiveMessage
  - SendMessage

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包接收和处理 Amazon AWS S3 事件通知

以下代码示例显示了如何以面向对象的方式处理 S3 事件通知。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例显示了如何使用 Amazon SQS 处理 S3 通知事件。

```
    /**
```

```
    * This method receives S3 event notifications by using an SqsAsyncClient.
    * After the client receives the messages it deserializes the JSON payload
 and logs them. It uses
    * the S3EventNotification class (part of the S3 event notification API for
 Java) to deserialize
    * the JSON payload and access the messages in an object-oriented way.
    *
    * @param queueUrl The URL of the AWS SQS queue that receives the S3 event
 notifications.
    * @see <a href="https://sdk.amazonaws.com/java/api/latest/software/amazon/
awssdk/eventnotifications/s3/model/package-summary.html">S3EventNotification
 API</a>.
    * <p>
    * To use S3 event notification serialization/deserialization to objects, add
 the following
    * dependency to your Maven pom.xml file.
    * <dependency>
    * <groupId>software.amazon.awssdk</groupId>
    * <artifactId>s3-event-notifications</artifactId>
    * <version><LATEST></version>
    * </dependency>
    * <p>
    * The S3 event notification API became available with version 2.25.11 of the
 Java SDK.
    * <p>
    * This example shows the use of the API with AWS SQS, but it can be used to
 process S3 event notifications
    * in AWS SNS or AWS Lambda as well.
    * <p>
    * Note: The S3EventNotification class does not work with messages routed
 through AWS EventBridge.
    */
   static void processS3Events(String bucketName, String queueUrl, String
 queueArn) {
       try {
           // Configure the bucket to send Object Created and Object Tagging
 notifications to an existing SQS queue.
           s3Client.putBucketNotificationConfiguration(b -> b
                   .notificationConfiguration(ncb -> ncb
                           .queueConfigurations(qcb -> qcb
                                   .events(Event.S3_OBJECT_CREATED,
 Event.S3_OBJECT_TAGGING)
                                   .queueArn(queueArn)))
                           .bucket(bucketName)
```

```
            ).join();

            triggerS3EventNotifications(bucketName);
            // Wait for event notifications to propagate.
            Thread.sleep(Duration.ofSeconds(5).toMillis());

            boolean didReceiveMessages = true;
            while (didReceiveMessages) {
                // Display the number of messages that are available in the
queue.
                sqsClient.getQueueAttributes(b -> b
                                .queueUrl(queueUrl)

.attributeNames(QueueAttributeName.APPROXIMATE_NUMBER_OF_MESSAGES)
                        ).thenAccept(attributeResponse ->
                                logger.info("Approximate number of messages in
the queue: {}",

attributeResponse.attributes().get(QueueAttributeName.APPROXIMATE_NUMBER_OF_MESSAGES)))
                        .join();

                // Receive the messages.
                ReceiveMessageResponse response = sqsClient.receiveMessage(b -> b
                        .queueUrl(queueUrl)
                ).get();
                logger.info("Count of received messages: {}",
response.messages().size());
                didReceiveMessages = !response.messages().isEmpty();

                // Create a collection to hold the received message for deletion
                // after we log the messages.
                HashSet<DeleteMessageBatchRequestEntry> messagesToDelete = new
HashSet<>();
                // Process each message.
                response.messages().forEach(message -> {
                    logger.info("Message id: {}", message.messageId());
                    // Deserialize JSON message body to a S3EventNotification
object
                    // to access messages in an object-oriented way.
                    S3EventNotification event =
S3EventNotification.fromJson(message.body());

                    // Log the S3 event notification record details.
                    if (event.getRecords() != null) {
```

```
                        event.getRecords().forEach(record -> {
                            String eventName = record.getEventName();
                            String key = record.getS3().getObject().getKey();
                            logger.info(record.toString());
                            logger.info("Event name is {} and key is {}",
eventName, key);
                        });
                    }
                    // Add logged messages to collection for batch deletion.
                    messagesToDelete.add(DeleteMessageBatchRequestEntry.builder()
                            .id(message.messageId())
                            .receiptHandle(message.receiptHandle())
                            .build());
                });
                // Delete messages.
                if (!messagesToDelete.isEmpty()) {

sqsClient.deleteMessageBatch(DeleteMessageBatchRequest.builder()
                            .queueUrl(queueUrl)
                            .entries(messagesToDelete)
                            .build()
                    ).join();
                }
            } // End of while block.
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API Reference》中的以下主题。
  - DeleteMessageBatch
  - GetQueueAttributes
  - PutBucketNotificationConfiguration
  - ReceiveMessage

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包将 Amazon SNS 消息发布到亚马逊 SQS 队列 AWS

以下代码示例演示了如何：

- 创建主题（FIFO 或非 FIFO）。
- 针对主题订阅多个队列，并提供应用筛选条件的选项。
- 将消息发布到主题。
- 轮询队列中是否有收到的消息。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
/// <summary>
/// Console application to run a feature scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
 "sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
```

```
    {
        // Set up dependency injection for Amazon EventBridge.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonSQS>()
                    .AddAWSService<IAmazonSimpleNotificationService>()
                    .AddTransient<SNSWrapper>()
                    .AddTransient<SQSWrapper>()
            )
            .Build();

        ServicesSetup(host);
        PrintDescription();

        await RunScenario();

    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
        SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
    }

    /// <summary>
    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();
```

```
            await SetupQueues();

            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
            await CleanupResources();

            Console.WriteLine("Messaging with topics and queues scenario is
complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this scenario, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
                          $"\r\nYou can select from several options for
configuring the topic and the subscriptions for the 2 queues." +
```

```
                            $"\r\nYou can then post to the topic and see the
 results in the queues.\r\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
                            $"\r\nFIFO topics deliver messages in order and support
 deduplication and message filtering." +
                            $"\r\nYou can then post to the topic and see the
 results in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
 topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
 "example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended
 to the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic,
 deduplication is supported." +
                              $"\r\nDeduplication IDs are either set in the
 message or automatically generated " +
                              $"\r\nfrom content using a hash function.\r\n" +
                              $"\r\nIf a message is successfully published to an
 SNS FIFO topic, any message " +
                              $"\r\npublished and determined to have the same
 deduplication ID, " +
                              $"\r\nwithin the five-minute deduplication
 interval, is accepted but not delivered.\r\n" +
```

```
                                $"\r\nFor more information about deduplication, " +
                                $"\r\nsee https://docs.aws.amazon.com/sns/latest/
dg/fifo-message-dedup.html.");

            _useContentBasedDeduplication = GetYesNoResponse("Use content-based
 deduplication instead of entering a deduplication ID?");
            Console.WriteLine(new string('-', 80));
        }

        _topicArn = await SnsWrapper.CreateTopicWithName(_topicName,
 _useFifoTopic, _useContentBasedDeduplication);

        Console.WriteLine($"Your new topic with the name {_topicName}" +
                          $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
                          $"\r\nhas been created.\r\n");

        Console.WriteLine(new string('-', 80));
        return _topicArn;
    }

    /// <summary>
    /// Set up the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task SetupQueues()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
 Service (Amazon SQS) queues to subscribe to the topic.");

        // Repeat this section for each queue.
        for (int i = 0; i < _queueCount; i++)
        {
            var queueName = GetUserResponse("Enter a name for an Amazon SQS
 queue: ", $"example-queue-{i}");
            if (_useFifoTopic)
            {
                // Only explain this once.
                if (i == 0)
                {
                    Console.WriteLine(
                        "Because you have selected a FIFO topic, '.fifo' must be
 appended to the queue name.");
                }
```

```
                var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

                _queueUrls[i] = queueUrl;

                Console.WriteLine($"Your new queue with the name {queueName}" +
                                  $"\r\nand queue URL {queueUrl}" +
                                  $"\r\nhas been created.\r\n");

                if (i == 0)
                {
                    Console.WriteLine(
                        $"The queue URL is used to retrieve the queue ARN,\r\n" +
                        $"which is used to create a subscription.");
                    Console.WriteLine(new string('-', 80));
                }

                var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

                if (i == 0)
                {
                    Console.WriteLine(
                        $"An AWS Identity and Access Management (IAM) policy must
be attached to an SQS queue, enabling it to receive\r\n" +
                        $"messages from an SNS topic");
                }

                await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

                await SetupFilters(i, queueArn, queueName);
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up filters with user options for a queue.
    /// </summary>
    /// <param name="queueCount">The number of this queue.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="queueName">The name of the queue.</param>
```

```
    /// <returns>Async Task.</returns>
    public static async Task SetupFilters(int queueCount, string queueArn, string
 queueName)
    {
        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            // Only explain this once.
            if (queueCount == 0)
            {
                Console.WriteLine(
                    "Subscriptions to a FIFO topic can have filters." +
                    "If you add a filter to this subscription, then only the
 filtered messages " +
                    "will be received in the queue.");

                Console.WriteLine(
                    "For information about message filtering, " +
                    "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

                Console.WriteLine(
                    "For this example, you can filter messages by a" +
                    "TONE attribute.");
            }

            var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
 subscription to the topic?");

            string? filterPolicy = null;
            if (useFilter)
            {
                filterPolicy = CreateFilterPolicy();
            }
            var subscriptionArn = await
 SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
                queueArn);
            _subscriptionArns[queueCount] = subscriptionArn;

            Console.WriteLine(
                $"The queue {queueName} has been subscribed to the topic
 {_topicName} " +
                $"with the subscription ARN {subscriptionArn}");
            Console.WriteLine(new string('-', 80));
```

```
        }
    }

    /// <summary>
    /// Use user input to create a filter policy for a subscription.
    /// </summary>
    /// <returns>The serialized filter policy.</returns>
    public static string CreateFilterPolicy()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            $"You can filter messages by one or more of the following" +
            $"TONE attributes.");

        List<string> filterSelections = new List<string>();

        var selectionNumber = 0;
        do
        {
            Console.WriteLine(
                $"Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", filterSelections.Any() ? "0" :
 "1");
            int.TryParse(selection, out selectionNumber);
            if (selectionNumber > 0 &&
!filterSelections.Contains(_tones[selectionNumber - 1]))
            {
                filterSelections.Add(_tones[selectionNumber - 1]);
            }
        } while (selectionNumber != 0);

        var filters = new Dictionary<string, List<string>>
        {
            { "tone", filterSelections }
        };
        string filterPolicy = JsonSerializer.Serialize(filters);
        return filterPolicy;
    }
```

```
    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is
a sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must
set a message group ID." +
                                  "\r\nAll messages within the same group will be
received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID
for this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                      "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this
message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID
for this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
```

```
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
                        Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
                    {
                        toneAttribute = _tones[selectionNumber - 1];
                    }
                }

                var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                    _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

                Console.WriteLine($"Message published with id {messageID}.");
            }

            keepSendingMessages = GetYesNoResponse("Send another message?",
false);
        }
    }

    /// <summary>
    /// Poll for the published messages to see the results of the user's choices.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task<List<Message>> PollForMessages(string queueUrl)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
                          "\r\nPress any key to continue.");
        if (UseConsole)
        {
            Console.ReadLine();
        }

        var moreMessages = true;
        var messages = new List<Message>();
```

```
        while (moreMessages)
        {
            var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl,
10);

            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the
queue at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
                              $"\n\t{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message>
messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
```

```
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer =
 true)
    {
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                           ynResponse.Equals("y",
                               StringComparison.InvariantCultureIgnoreCase);
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }

    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }
}
```

创建一个包装 Amazon SQS 操作的类。

```
/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
    }

    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
 useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
```

```
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }

            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
        return createResponse.QueueUrl;
    }

    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };

        var getAttributesResponse = await
_amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }

    /// <summary>
```

```
   /// Set the policy attribute of a queue for a topic.
   /// </summary>
   /// <param name="queueArn">The ARN of the queue.</param>
   /// <param name="topicArn">The ARN of the topic.</param>
   /// <param name="queueUrl">The url for the queue.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> SetQueuePolicyForTopic(string queueArn, string
topicArn, string queueUrl)
   {
       var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                              "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
                                               $"\"aws:SourceArn\":
\"{topicArn}\"" +
                                            "}" +
                                       "}" +
                                "}]" +
                            "}";
       var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
           new SetQueueAttributesRequest()
           {
               QueueUrl = queueUrl,
               Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
           });
       return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
   }

   /// <summary>
   /// Receive messages from a queue by its URL.
   /// </summary>
   /// <param name="queueUrl">The url of the queue.</param>
   /// <returns>The list of messages.</returns>
   public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
```

```
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl,
 List<Message> messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
 _amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }

    /// <summary>
```

```
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

创建一个包装 Amazon SNS 操作的类。

```
/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
```

```
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
            }
        }

        var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
```

```
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
 { { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
 _amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</
param>
    /// <param name="attributeValue">The optional attribute value for the
 message.</param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
```

```
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await
_amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }


    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
```

```
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 发布
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
//! Workflow for messaging with topics and queues using Amazon SNS and Amazon
 SQS.
/*!
 \param clientConfig Aws client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::TopicsAndQueues::messagingWithTopicsAndQueues(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << "Welcome to messaging with topics and queues." << std::endl;
    printAsterisksLine();
    std::cout << "In this workflow, you will create an SNS topic and subscribe "
              << NUMBER_OF_QUEUES <<
              " SQS queues to the topic." << std::endl;
    std::cout
            << "You can select from several options for configuring the topic and
 the subscriptions for the "
            << NUMBER_OF_QUEUES << " queues." << std::endl;
    std::cout << "You can then post to the topic and see the results in the
 queues."
              << std::endl;

    Aws::SNS::SNSClient snsClient(clientConfiguration);

    printAsterisksLine();

    std::cout << "SNS topics can be configured as FIFO (First-In-First-Out)."
              << std::endl;
    std::cout
            << "FIFO topics deliver messages in order and support deduplication
 and message filtering."
            << std::endl;
    bool isFifoTopic = askYesNoQuestion(
            "Would you like to work with FIFO topics? (y/n) ");

    bool contentBasedDeduplication = false;
    Aws::String topicName;
    if (isFifoTopic) {
        printAsterisksLine();
        std::cout << "Because you have chosen a FIFO topic, deduplication is
 supported."
                  << std::endl;
        std::cout
                << "Deduplication IDs are either set in the message or
 automatically generated "
```

```
                << "from content using a hash function." << std::endl;
        std::cout
                << "If a message is successfully published to an SNS FIFO topic,
 any message "
                << "published and determined to have the same deduplication ID, "
                << std::endl;
        std::cout
                << "within the five-minute deduplication interval, is accepted
 but not delivered."
                << std::endl;
        std::cout
                << "For more information about deduplication, "
                << "see https://docs.aws.amazon.com/sns/latest/dg/fifo-message-
dedup.html."
                << std::endl;
        contentBasedDeduplication = askYesNoQuestion(
                "Use content-based deduplication instead of entering a
 deduplication ID? (y/n) ");
    }

    printAsterisksLine();

    Aws::SQS::SQSClient sqsClient(clientConfiguration);
    Aws::Vector<Aws::String> queueURLS;
    Aws::Vector<Aws::String> subscriptionARNS;

    Aws::String topicARN;
    {
        topicName = askQuestion("Enter a name for your SNS topic. ");

        // 1.  Create an Amazon SNS topic, either FIFO or non-FIFO.
        Aws::SNS::Model::CreateTopicRequest request;

        if (isFifoTopic) {
            request.AddAttributes("FifoTopic", "true");
            if (contentBasedDeduplication) {
                request.AddAttributes("ContentBasedDeduplication", "true");
            }
            topicName = topicName + FIFO_SUFFIX;

            std::cout
                    << "Because you have selected a FIFO topic, '.fifo' must be
 appended to the topic name."
                    << std::endl;
```

```
        }

        request.SetName(topicName);

        Aws::SNS::Model::CreateTopicOutcome outcome =
snsClient.CreateTopic(request);

        if (outcome.IsSuccess()) {
            topicARN = outcome.GetResult().GetTopicArn();
            std::cout << "Your new topic with the name '" << topicName
                      << "' and the topic Amazon Resource Name (ARN) " <<
std::endl;
            std::cout << "'" << topicARN << "' has been created." << std::endl;

        }
        else {
            std::cerr << "Error with TopicsAndQueues::CreateTopic. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            cleanUp(topicARN,
                    queueURLS,
                    subscriptionARNS,
                    snsClient,
                    sqsClient);

            return false;
        }
    }

    printAsterisksLine();

    std::cout << "Now you will create " << NUMBER_OF_QUEUES
              << " SQS queues to subscribe to the topic." << std::endl;
    Aws::Vector<Aws::String> queueNames;
    bool filteringMessages = false;
    bool first = true;
    for (int i = 1; i <= NUMBER_OF_QUEUES; ++i) {
        Aws::String queueURL;
        Aws::String queueName;
        {
            printAsterisksLine();
            std::ostringstream ostringstream;
            ostringstream << "Enter a name for " << (first ? "an" : "the next")
```

```
                        << " SQS queue. ";
            queueName = askQuestion(ostringstream.str());

            // 2.  Create an SQS queue.
            Aws::SQS::Model::CreateQueueRequest request;
            if (isFifoTopic) {

request.AddAttributes(Aws::SQS::Model::QueueAttributeName::FifoQueue,
                                    "true");
                queueName = queueName + FIFO_SUFFIX;

                if (first) // Only explain this once.
                {
                    std::cout
                            << "Because you are creating a FIFO SQS queue,
'.fifo' must "
                            << "be appended to the queue name." << std::endl;
                }
            }

            request.SetQueueName(queueName);
            queueNames.push_back(queueName);

            Aws::SQS::Model::CreateQueueOutcome outcome =
                    sqsClient.CreateQueue(request);

            if (outcome.IsSuccess()) {
                queueURL = outcome.GetResult().GetQueueUrl();
                std::cout << "Your new SQS queue with the name '" << queueName
                          << "' and the queue URL " << std::endl;
                std::cout << "'" << queueURL << "' has been created." <<
std::endl;
            }
            else {
                std::cerr << "Error with SQS::CreateQueue. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);
```

```
                return false;
            }
        }
        queueURLS.push_back(queueURL);

        if (first) // Only explain this once.
        {
            std::cout
                    << "The queue URL is used to retrieve the queue ARN, which is
"
                    << "used to create a subscription." << std::endl;
        }

        Aws::String queueARN;
        {
            // 3.  Get the SQS queue ARN attribute.
            Aws::SQS::Model::GetQueueAttributesRequest request;
            request.SetQueueUrl(queueURL);

request.AddAttributeNames(Aws::SQS::Model::QueueAttributeName::QueueArn);

            Aws::SQS::Model::GetQueueAttributesOutcome outcome =
                    sqsClient.GetQueueAttributes(request);

            if (outcome.IsSuccess()) {
                const Aws::Map<Aws::SQS::Model::QueueAttributeName, Aws::String>
&attributes =
                        outcome.GetResult().GetAttributes();
                const auto &iter = attributes.find(
                        Aws::SQS::Model::QueueAttributeName::QueueArn);
                if (iter != attributes.end()) {
                    queueARN = iter->second;
                    std::cout << "The queue ARN '" << queueARN
                              << "' has been retrieved."
                              << std::endl;
                }
                else {
                    std::cerr
                            << "Error ARN attribute not returned by
GetQueueAttribute."
                            << std::endl;

                    cleanUp(topicARN,
                            queueURLS,
```

```
                            subscriptionARNS,
                            snsClient,
                            sqsClient);

                    return false;
                }
            }
            else {
                std::cerr << "Error with SQS::GetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        if (first) {
            std::cout
                    << "An IAM policy must be attached to an SQS queue, enabling
it to receive "
                       "messages from an SNS topic." << std::endl;
        }

        {
            // 4.  Set the SQS queue policy attribute with a policy enabling the
receipt of SNS messages.
            Aws::SQS::Model::SetQueueAttributesRequest request;
            request.SetQueueUrl(queueURL);
            Aws::String policy = createPolicyForQueue(queueARN, topicARN);
            request.AddAttributes(Aws::SQS::Model::QueueAttributeName::Policy,
                                  policy);

            Aws::SQS::Model::SetQueueAttributesOutcome outcome =
                    sqsClient.SetQueueAttributes(request);

            if (outcome.IsSuccess()) {
                std::cout << "The attributes for the queue '" << queueName
                          << "' were successfully updated." << std::endl;
```

```
            }
            else {
                std::cerr << "Error with SQS::SetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        printAsterisksLine();

        {
            // 5.  Subscribe the SQS queue to the SNS topic.
            Aws::SNS::Model::SubscribeRequest request;
            request.SetTopicArn(topicARN);
            request.SetProtocol("sqs");
            request.SetEndpoint(queueARN);
            if (isFifoTopic) {
                if (first) {
                    std::cout << "Subscriptions to a FIFO topic can have
 filters."
                              << std::endl;
                    std::cout
                            << "If you add a filter to this subscription, then
 only the filtered messages "
                            << "will be received in the queue." << std::endl;
                    std::cout << "For information about message filtering, "
                              << "see https://docs.aws.amazon.com/sns/latest/dg/
sns-message-filtering.html"
                              << std::endl;
                    std::cout << "For this example, you can filter messages by a
 \""
                              << TONE_ATTRIBUTE << "\" attribute." << std::endl;
                }

                std::ostringstream ostringstream;
                ostringstream << "Filter messages for \"" << queueName
```

```
                            << "\"'s subscription to the topic \""
                            << topicName << "\"?  (y/n)";

                // Add filter if user answers yes.
                if (askYesNoQuestion(ostringstream.str())) {
                    Aws::String jsonPolicy = getFilterPolicyFromUser();
                    if (!jsonPolicy.empty()) {
                        filteringMessages = true;

                        std::cout << "This is the filter policy for this
subscription."
                                  << std::endl;
                        std::cout << jsonPolicy << std::endl;

                        request.AddAttributes("FilterPolicy", jsonPolicy);
                    }
                    else {
                        std::cout
                                << "Because you did not select any attributes, no
filter "
                                << "will be added to this subscription." <<
std::endl;
                    }
                }
            }  // if (isFifoTopic)
            Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

            if (outcome.IsSuccess()) {
                Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
                std::cout << "The queue '" << queueName
                          << "' has been subscribed to the topic '"
                          << "'" << topicName << "'" << std::endl;
                std::cout << "with the subscription ARN '" << subscriptionARN <<
"."
                          << std::endl;
                subscriptionARNS.push_back(subscriptionARN);
            }
            else {
                std::cerr << "Error with TopicsAndQueues::Subscribe. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
```

```
                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        first = false;
    }

    first = true;
    do {
        printAsterisksLine();

        // 6.  Publish a message to the SNS topic.
        Aws::SNS::Model::PublishRequest request;
        request.SetTopicArn(topicARN);
        Aws::String message = askQuestion("Enter a message text to publish.  ");
        request.SetMessage(message);
        if (isFifoTopic) {
            if (first) {
                std::cout
                        << "Because you are using a FIFO topic, you must set a
message group ID."
                        << std::endl;
                std::cout
                        << "All messages within the same group will be received
in the "
                        << "order they were published." << std::endl;
            }
            Aws::String messageGroupID = askQuestion(
                    "Enter a message group ID for this message. ");
            request.SetMessageGroupId(messageGroupID);
            if (!contentBasedDeduplication) {
                if (first) {
                    std::cout
                            << "Because you are not using content-based
deduplication, "
                            << "you must enter a deduplication ID." << std::endl;
                }
                Aws::String deduplicationID = askQuestion(
```

```
                    "Enter a deduplication ID for this message. ");
            request.SetMessageDeduplicationId(deduplicationID);
        }
    }

    if (filteringMessages && askYesNoQuestion(
            "Add an attribute to this message? (y/n) ")) {
        for (size_t i = 0; i < TONES.size(); ++i) {
            std::cout << "  " << (i + 1) << ". " << TONES[i] << std::endl;
        }
        int selection = askQuestionForIntRange(
                "Enter a number for an attribute. ",
                1, static_cast<int>(TONES.size()));
        Aws::SNS::Model::MessageAttributeValue messageAttributeValue;
        messageAttributeValue.SetDataType("String");
        messageAttributeValue.SetStringValue(TONES[selection - 1]);
        request.AddMessageAttributes(TONE_ATTRIBUTE, messageAttributeValue);
    }

    Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Your message was successfully published." << std::endl;
    }
    else {
        std::cerr << "Error with TopicsAndQueues::Publish. "
                  << outcome.GetError().GetMessage()
                  << std::endl;

        cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

        return false;
    }

    first = false;
} while (askYesNoQuestion("Post another message? (y/n) "));

printAsterisksLine();

std::cout << "Now the SQS queue will be polled to retrieve the messages."
```

```
              << std::endl;
    askQuestion("Press any key to continue...", alwaysTrueTest);

    for (size_t i = 0; i < queueURLS.size(); ++i) {
        // 7.  Poll an SQS queue for its messages.
        std::vector<Aws::String> messages;
        std::vector<Aws::String> receiptHandles;
        while (true) {
            Aws::SQS::Model::ReceiveMessageRequest request;
            request.SetMaxNumberOfMessages(10);
            request.SetQueueUrl(queueURLS[i]);

            // Setting WaitTimeSeconds to non-zero enables long polling.
            // For information about long polling, see
            // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
            request.SetWaitTimeSeconds(1);
            Aws::SQS::Model::ReceiveMessageOutcome outcome =
                    sqsClient.ReceiveMessage(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::SQS::Model::Message> &newMessages =
 outcome.GetResult().GetMessages();
                if (newMessages.empty()) {
                    break;
                }
                else {
                    for (const Aws::SQS::Model::Message &message: newMessages) {
                        messages.push_back(message.GetBody());
                        receiptHandles.push_back(message.GetReceiptHandle());
                    }
                }
            }
            else {
                std::cerr << "Error with SQS::ReceiveMessage. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);
```

```
            return false;
        }
    }

    printAsterisksLine();

    if (messages.empty()) {
        std::cout << "No messages were ";
    }
    else if (messages.size() == 1) {
        std::cout << "One message was ";
    }
    else {
        std::cout << messages.size() << " messages were ";
    }
    std::cout << "received by the queue '" << queueNames[i]
              << "'." << std::endl;
    for (const Aws::String &message: messages) {
        std::cout << "  Message : '" << message << "'."
                  << std::endl;
    }

    // 8.  Delete a batch of messages from an SQS queue.
    if (!receiptHandles.empty()) {
        Aws::SQS::Model::DeleteMessageBatchRequest request;
        request.SetQueueUrl(queueURLS[i]);
        int id = 1; // Ids must be unique within a batch delete request.
        for (const Aws::String &receiptHandle: receiptHandles) {
            Aws::SQS::Model::DeleteMessageBatchRequestEntry entry;
            entry.SetId(std::to_string(id));
            ++id;
            entry.SetReceiptHandle(receiptHandle);
            request.AddEntries(entry);
        }

        Aws::SQS::Model::DeleteMessageBatchOutcome outcome =
                sqsClient.DeleteMessageBatch(request);

        if (outcome.IsSuccess()) {
            std::cout << "The batch deletion of messages was successful."
                      << std::endl;
        }
        else {
            std::cerr << "Error with SQS::DeleteMessageBatch. "
```

```
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }
    }

    return cleanUp(topicARN,
                   queueURLS,
                   subscriptionARNS,
                   snsClient,
                   sqsClient,
                   true); // askUser
}


bool AwsDoc::TopicsAndQueues::cleanUp(const Aws::String &topicARN,
                                      const Aws::Vector<Aws::String> &queueURLS,
                                      const Aws::Vector<Aws::String>
 &subscriptionARNS,
                                      const Aws::SNS::SNSClient &snsClient,
                                      const Aws::SQS::SQSClient &sqsClient,
                                      bool askUser) {
    bool result = true;
    printAsterisksLine();
    if (!queueURLS.empty() && askUser &&
        askYesNoQuestion("Delete the SQS queues? (y/n) ")) {

        for (const auto &queueURL: queueURLS) {
            // 9.  Delete an SQS queue.
            Aws::SQS::Model::DeleteQueueRequest request;
            request.SetQueueUrl(queueURL);

            Aws::SQS::Model::DeleteQueueOutcome outcome =
                    sqsClient.DeleteQueue(request);

            if (outcome.IsSuccess()) {
                std::cout << "The queue with URL '" << queueURL
```

```
                        << "' was successfully deleted." << std::endl;
            }
            else {
                std::cerr << "Error with SQS::DeleteQueue. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }

        for (const auto &subscriptionARN: subscriptionARNS) {
            // 10. Unsubscribe an SNS subscription.
            Aws::SNS::Model::UnsubscribeRequest request;
            request.SetSubscriptionArn(subscriptionARN);

            Aws::SNS::Model::UnsubscribeOutcome outcome =
                    snsClient.Unsubscribe(request);

            if (outcome.IsSuccess()) {
                std::cout << "Unsubscribe of subscription ARN '" <<
subscriptionARN
                          << "' was successful." << std::endl;
            }
            else {
                std::cerr << "Error with TopicsAndQueues::Unsubscribe. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }

    printAsterisksLine();
    if (!topicARN.empty() && askUser &&
        askYesNoQuestion("Delete the SNS topic? (y/n) ")) {

        // 11. Delete an SNS topic.
        Aws::SNS::Model::DeleteTopicRequest request;
        request.SetTopicArn(topicARN);

        Aws::SNS::Model::DeleteTopicOutcome outcome =
snsClient.DeleteTopic(request);

        if (outcome.IsSuccess()) {
```

```
            std::cout << "The topic with ARN '" << topicARN
                      << "' was successfully deleted." << std::endl;
        }
        else {
            std::cerr << "Error with TopicsAndQueues::DeleteTopicRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}

//! Create an IAM policy that gives an SQS queue permission to receive messages
 from an SNS topic.
/*!
 \sa createPolicyForQueue()
 \param queueARN: The SQS queue Amazon Resource Name (ARN).
 \param topicARN: The SNS topic ARN.
 \return Aws::String: The policy as JSON.
 */
Aws::String AwsDoc::TopicsAndQueues::createPolicyForQueue(const Aws::String
 &queueARN,
                                                          const Aws::String
 &topicARN) {
    std::ostringstream policyStream;
    policyStream << R"({
        "Statement": [
        {
            "Effect": "Allow",
                    "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": "sqs:SendMessage",
                    "Resource": ")" << queueARN << R"(",
                    "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": ")" << topicARN << R"("
                }
            }
        }
        ]
    })";
```

```
    return policyStream.str();
}
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API Reference》中的以下主题。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 发布
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"
 "strings"
```

```
 "topics_and_queues/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
 Message string
}

// ScenarioRunner separates the steps of this scenario into individual functions
 so that
// they are simpler to read and understand.
type ScenarioRunner struct {
 questioner demotools.IQuestioner
 snsActor   *actions.SnsActions
 sqsActor   *actions.SqsActions
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string,
 bool, bool) {
 log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or
 standard.\n" +
  "FIFO topics deliver messages in order and support deduplication and message
 filtering.")
 isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
 topics? (y/n) ", "y")

 contentBasedDeduplication := false
 if isFifoTopic {
  log.Println(strings.Repeat("-", 88))
  log.Println("Because you have chosen a FIFO topic, deduplication is supported.
\n" +
   "Deduplication IDs are either set in the message or are automatically
 generated\n" +
```

```
   "from content using a hash function. If a message is successfully published to
\n" +
   "an SNS FIFO topic, any message published and determined to have the same\n" +
   "deduplication ID, within the five-minute deduplication interval, is accepted
\n" +
   "but not delivered. For more information about deduplication, see:\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
  contentBasedDeduplication = runner.questioner.AskBool(
   "\nDo you want to use content-based deduplication instead of entering a
 deduplication ID? (y/n) ", "y")
 }
 log.Println(strings.Repeat("-", 88))

 topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
 if isFifoTopic {
  topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
  log.Printf("Because you have selected a FIFO topic, '%v' must be appended to
\n"+
   "the topic name.", FIFO_SUFFIX)
 }

 topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
 contentBasedDeduplication)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN)
 \n"+
  "'%v' has been created.", topicName, topicArn)

 return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
 isFifoTopic bool) (string, string) {
 queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS
 queue. ", ordinal))
 if isFifoTopic {
  queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
  if ordinal == "first" {
   log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
    "be appended to the queue name.\n", FIFO_SUFFIX)
  }
 }
```

```
 queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
  "'%v' has been created.", queueName, queueUrl)

 return queueName, queueUrl
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
 ctx context.Context, queueName string, queueUrl string, topicName string,
 topicArn string, ordinal string,
 isFifoTopic bool) (string, bool) {

 queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
 if err != nil {
  panic(err)
 }
 log.Printf("The ARN of your queue is: %v.\n", queueArn)

 err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
 if err != nil {
  panic(err)
 }
 log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
  "messages to it.")
 log.Println(strings.Repeat("-", 88))

 var filterPolicy map[string][]string
 if isFifoTopic {
  if ordinal == "first" {
   log.Println("Subscriptions to a FIFO topic can have filters.\n" +
    "If you add a filter to this subscription, then only the filtered messages\n"
 +
    "will be received in the queue.\n" +
    "For information about message filtering, see\n" +
    "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
    "For this example, you can filter messages by a \"tone\" attribute.")
  }

  wantFiltering := runner.questioner.AskBool(
   fmt.Sprintf("Do you want to filter messages that are sent to \"%v\"\n"+
    "from the %v topic? (y/n) ", queueName, topicName), "y")
```

```
  if wantFiltering {
   log.Println("You can filter messages by one or more of the following \"tone\"
 attributes.")

   var toneSelections []string
   askAboutTones := true
   for askAboutTones {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelections = append(toneSelections, ToneChoices[toneIndex])
    askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
 the filter? (y/n) ", "y")
   }
   log.Printf("Your subscription will be filtered to only pass the following
 tones: %v\n", toneSelections)
   filterPolicy = map[string][]string{TONE_KEY: toneSelections}
  }
 }

 subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
 filterPolicy)
 if err != nil {
  panic(err)
 }
 log.Printf("The queue %v is now subscribed to the topic %v with the subscription
 ARN %v.\n",
  queueName, topicName, subscriptionArn)

 return subscriptionArn, filterPolicy != nil
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn
 string, isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
 var message string
 var groupId string
 var dedupId string
 var toneSelection string
 publishMore := true
 for publishMore {
  groupId = ""
  dedupId = ""
  toneSelection = ""
  message = runner.questioner.Ask("Enter a message to publish: ")
  if isFifoTopic {
```

```
    log.Println("Because you are using a FIFO topic, you must set a message group
 ID.\n" +
     "All messages within the same group will be received in the order they were
 published.")
    groupId = runner.questioner.Ask("Enter a message group ID: ")
    if !contentBasedDeduplication {
     log.Println("Because you are not using content-based deduplication,\n" +
      "you must enter a deduplication ID.")
     dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
    }
   }
   if usingFilters {
    if runner.questioner.AskBool("Add a tone attribute so this message can be
 filtered? (y/n) ", "y") {
     toneIndex := runner.questioner.AskChoice(
      "Enter the number of the tone you want to filter by:\n", ToneChoices)
     toneSelection = ToneChoices[toneIndex]
    }
   }

   err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId,
 TONE_KEY, toneSelection)
   if err != nil {
    panic(err)
   }
   log.Println(("Your message was published."))

   publishMore = runner.questioner.AskBool("Do you want to publish another
 messsage? (y/n) ", "y")
  }
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
 []string) {
 log.Println("Polling queues for messages...")
 for _, queueUrl := range queueUrls {
  var messages []types.Message
  for {
   currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
   if err != nil {
    panic(err)
   }
   if len(currentMsgs) == 0 {
    break
```

```
    }
    messages = append(messages, currentMsgs...)
   }
   if len(messages) == 0 {
    log.Printf("No messages were received by queue %v.\n", queueUrl)
   } else if len(messages) == 1 {
    log.Printf("One message was received by queue %v:\n", queueUrl)

   } else {
    log.Printf("%v messages were received by queue %v:\n", len(messages),
  queueUrl)
    }
    for msgIndex, message := range messages {
     messageBody := MessageBody{}
     err := json.Unmarshal([]byte(*message.Body), &messageBody)
     if err != nil {
      panic(err)
     }
     log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
    }

   if len(messages) > 0 {
    log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
    err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
    if err != nil {
     panic(err)
    }
   }
  }
}

// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
  the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
```

```
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunTopicsAndQueuesScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
 resources := Resources{}
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.\n" +
    "Cleaning up any resources that were created...")
   resources.Cleanup(ctx)
  }
 }()
 queueCount := 2

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome to messaging with topics and queues.\n\n"+
  "In this scenario, you will create an SNS topic and subscribe %v SQS queues to
 the\n"+
  "topic. You can select from several options for configuring the topic and the
\n"+
  "subscriptions for the queues. You can then post to the topic and see the
 results\n"+
  "in the queues.\n", queueCount)

 log.Println(strings.Repeat("-", 88))

 runner := ScenarioRunner{
  questioner: questioner,
  snsActor:   &actions.SnsActions{SnsClient: sns.NewFromConfig(sdkConfig)},
  sqsActor:   &actions.SqsActions{SqsClient: sqs.NewFromConfig(sdkConfig)},
 }
 resources.snsActor = runner.snsActor
 resources.sqsActor = runner.sqsActor

 topicName, topicArn, isFifoTopic, contentBasedDeduplication :=
 runner.CreateTopic(ctx)
 resources.topicArn = topicArn
 log.Println(strings.Repeat("-", 88))

 log.Printf("Now you will create %v SQS queues and subscribe them to the topic.
\n", queueCount)
 ordinals := []string{"first", "next"}
 usingFilters := false
```

```
 for _, ordinal := range ordinals {
  queueName, queueUrl := runner.CreateQueue(ctx, ordinal, isFifoTopic)
  resources.queueUrls = append(resources.queueUrls, queueUrl)

  _, filtering := runner.SubscribeQueueToTopic(ctx, queueName, queueUrl,
 topicName, topicArn, ordinal, isFifoTopic)
  usingFilters = usingFilters || filtering
 }

 log.Println(strings.Repeat("-", 88))
 runner.PublishMessages(ctx, topicArn, isFifoTopic, contentBasedDeduplication,
 usingFilters)
 log.Println(strings.Repeat("-", 88))
 runner.PollForMessages(ctx, resources.queueUrls)

 log.Println(strings.Repeat("-", 88))

 wantCleanup := questioner.AskBool("Do you want to remove all AWS resources
 created for this scenario? (y/n) ", "y")
 if wantCleanup {
  log.Println("Cleaning up resources...")
  resources.Cleanup(ctx)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

定义一个结构来包装此示例中使用的 Amazon SNS 操作。

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)
```

```
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
```

```
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
 an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}
```

```
// Publish publishes a message to an Amazon SNS topic. The message is then sent
 to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered
 according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string)
 error {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
 err)
 }
 return err
}
```

定义一个结构来包装此示例中使用的 Amazon SQS 操作。

```
import (
 "context"
 "encoding/json"
```

```
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can
 specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
```

```
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string)
 (string, error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy
 to an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages
 to the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
```

```
    string(types.QueueAttributeNamePolicy): string(policyBytes),
   },
   QueueUrl: aws.String(queueUrl),
  })
  if err != nil {
   log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
  queueUrl, err)
  }
  return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
```

```
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 } else {
  messages = result.Messages
 }
 return messages, err
}



// DeleteMessages uses the DeleteMessageBatch action to delete a batch of
 messages from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}



// DeleteQueue deletes an Amazon SQS queue.
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

清理资源。

```
import (
 "context"
 "fmt"
 "log"
 "topics_and_queues/actions"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 topicArn  string
 queueUrls []string
 snsActor  *actions.SnsActions
 sqsActor  *actions.SqsActions
}

// Cleanup deletes all AWS resources created during an example.
func (resources Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Println("Something went wrong during cleanup. Use the AWS Management
 Console\n" +
    "to remove any remaining resources that were created for this scenario.")
  }
 }()

 var err error
 if resources.topicArn != "" {
  log.Printf("Deleting topic %v.\n", resources.topicArn)
  err = resources.snsActor.DeleteTopic(ctx, resources.topicArn)
  if err != nil {
   panic(err)
  }
 }

 for _, queueUrl := range resources.queueUrls {
  log.Printf("Deleting queue %v.\n", queueUrl)
  err = resources.sqsActor.DeleteQueue(ctx, queueUrl)
```

```
  if err != nil {
   panic(err)
  }
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API Reference》中的以下主题。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 发布
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
package com.example.sns;

import
 software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
```

```
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.MessageAttributeValue;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import
 software.amazon.awssdk.services.sns.model.SetSubscriptionAttributesRequest;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;
import software.amazon.awssdk.services.sns.model.UnsubscribeRequest;
import software.amazon.awssdk.services.sns.model.UnsubscribeResponse;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesResponse;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlResponse;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

import com.google.gson.Gson;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import com.google.gson.JsonPrimitive;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
```

```
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 * <p>
 * This Java example performs these tasks:
 * <p>
 * 1. Gives the user three options to choose from.
 * 2. Creates an Amazon Simple Notification Service (Amazon SNS) topic.
 * 3. Creates an Amazon Simple Queue Service (Amazon SQS) queue.
 * 4. Gets the SQS queue Amazon Resource Name (ARN) attribute.
 * 5. Attaches an AWS Identity and Access Management (IAM) policy to the queue.
 * 6. Subscribes to the SQS queue.
 * 7. Publishes a message to the topic.
 * 8. Displays the messages.
 * 9. Deletes the received message.
 * 10. Unsubscribes from the topic.
 * 11. Deletes the SNS topic.
 */
public class SNSWorkflow {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage:\n" +
            "    <fifoQueueARN>\n\n" +
            "Where:\n" +
            "    accountId - Your AWS account Id value.";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .build();

        SqsClient sqsClient = SqsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .build();
```

```
        Scanner in = new Scanner(System.in);
        String accountId = args[0];
        String useFIFO;
        String duplication = "n";
        String topicName;
        String deduplicationID = null;
        String groupId = null;

        String topicArn;
        String sqsQueueName;
        String sqsQueueUrl;
        String sqsQueueArn;
        String subscriptionArn;
        boolean selectFIFO = false;

        String message;
        List<Message> messageList;
        List<String> filterList = new ArrayList<>();
        String msgAttValue = "";

        System.out.println(DASHES);
        System.out.println("Welcome to messaging with topics and queues.");
        System.out.println("In this scenario, you will create an SNS topic and
 subscribe an SQS queue to the topic.\n" +
            "You can select from several options for configuring the topic and
 the subscriptions for the queue.\n" +
            "You can then post to the topic and see the results in the queue.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("SNS topics can be configured as FIFO (First-In-First-
Out).\n" +
            "FIFO topics deliver messages in order and support deduplication and
 message filtering.\n" +
            "Would you like to work with FIFO topics? (y/n)");
        useFIFO = in.nextLine();
        if (useFIFO.compareTo("y") == 0) {
            selectFIFO = true;
            System.out.println("You have selected FIFO");
            System.out.println(" Because you have chosen a FIFO topic,
 deduplication is supported.\n" +
                "        Deduplication IDs are either set in the message or
 automatically generated from content using a hash function.\n"
```

```
                +
                "        If a message is successfully published to an SNS FIFO
 topic, any message published and determined to have the same deduplication ID,
\n"
                +
                "        within the five-minute deduplication interval, is
 accepted but not delivered.\n" +
                "        For more information about deduplication, see https://
docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.");

            System.out.println(
                "Would you like to use content-based deduplication instead of
 entering a deduplication ID? (y/n)");
            duplication = in.nextLine();
            if (duplication.compareTo("y") == 0) {
                System.out.println("Please enter a group id value");
                groupId = in.nextLine();
            } else {
                System.out.println("Please enter deduplication Id value");
                deduplicationID = in.nextLine();
                System.out.println("Please enter a group id value");
                groupId = in.nextLine();
            }
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a topic.");
        System.out.println("Enter a name for your SNS topic.");
        topicName = in.nextLine();
        if (selectFIFO) {
            System.out.println("Because you have selected a FIFO topic, '.fifo'
 must be appended to the topic name.");
            topicName = topicName + ".fifo";
            System.out.println("The name of the topic is " + topicName);
            topicArn = createFIFO(snsClient, topicName, duplication);
            System.out.println("The ARN of the FIFO topic is " + topicArn);

        } else {
            System.out.println("The name of the topic is " + topicName);
            topicArn = createSNSTopic(snsClient, topicName);
            System.out.println("The ARN of the non-FIFO topic is " + topicArn);

        }
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Create an SQS queue.");
        System.out.println("Enter a name for your SQS queue.");
        sqsQueueName = in.nextLine();
        if (selectFIFO) {
            sqsQueueName = sqsQueueName + ".fifo";
        }
        sqsQueueUrl = createQueue(sqsClient, sqsQueueName, selectFIFO);
        System.out.println("The queue URL is " + sqsQueueUrl);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get the SQS queue ARN attribute.");
        sqsQueueArn = getSQSQueueAttrs(sqsClient, sqsQueueUrl);
        System.out.println("The ARN of the new queue is " + sqsQueueArn);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Attach an IAM policy to the queue.");

        // Define the policy to use. Make sure that you change the REGION if you
are
        // running this code
        // in a different region.
        String policy = """
        {
             "Statement": [
             {
                 "Effect": "Allow",
                         "Principal": {
                     "Service": "sns.amazonaws.com"
                 },
                 "Action": "sqs:SendMessage",
                         "Resource": "arn:aws:sqs:us-east-1:%s:%s",
                         "Condition": {
                     "ArnEquals": {
                         "aws:SourceArn": "arn:aws:sns:us-east-1:%s:%s"
                     }
                 }
             }
             ]
         }
```

```
        """.formatted(accountId, sqsQueueName, accountId, topicName);

        setQueueAttr(sqsClient, sqsQueueUrl, policy);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Subscribe to the SQS queue.");
        if (selectFIFO) {
            System.out.println(
                "If you add a filter to this subscription, then only the filtered
 messages will be received in the queue.\n"
                    +
                    "For information about message filtering, see https://
docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n"
                    +
                    "For this example, you can filter messages by a \"tone\"
 attribute.");
            System.out.println("Would you like to filter messages for " +
 sqsQueueName + "'s subscription to the topic "
                + topicName + "?  (y/n)");
            String filterAns = in.nextLine();
            if (filterAns.compareTo("y") == 0) {
                boolean moreAns = false;
                System.out.println("You can filter messages by one or more of the
 following \"tone\" attributes.");
                System.out.println("1. cheerful");
                System.out.println("2. funny");
                System.out.println("3. serious");
                System.out.println("4. sincere");
                while (!moreAns) {
                    System.out.println("Select a number or choose 0 to end.");
                    String ans = in.nextLine();
                    switch (ans) {
                        case "1":
                            filterList.add("cheerful");
                            break;
                        case "2":
                            filterList.add("funny");
                            break;
                        case "3":
                            filterList.add("serious");
                            break;
                        case "4":
                            filterList.add("sincere");
```

```
                            break;
                        default:
                            moreAns = true;
                            break;
                    }
                }
            }
        }
        subscriptionArn = subQueue(snsClient, topicArn, sqsQueueArn, filterList);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Publish a message to the topic.");
        if (selectFIFO) {
            System.out.println("Would you like to add an attribute to this
message?  (y/n)");
            String msgAns = in.nextLine();
            if (msgAns.compareTo("y") == 0) {
                System.out.println("You can filter messages by one or more of the
following \"tone\" attributes.");
                System.out.println("1. cheerful");
                System.out.println("2. funny");
                System.out.println("3. serious");
                System.out.println("4. sincere");
                System.out.println("Select a number or choose 0 to end.");
                String ans = in.nextLine();
                switch (ans) {
                    case "1":
                        msgAttValue = "cheerful";
                        break;
                    case "2":
                        msgAttValue = "funny";
                        break;
                    case "3":
                        msgAttValue = "serious";
                        break;
                    default:
                        msgAttValue = "sincere";
                        break;
                }

                System.out.println("Selected value is " + msgAttValue);
            }
            System.out.println("Enter a message.");
```

```
            message = in.nextLine();
            pubMessageFIFO(snsClient, message, topicArn, msgAttValue,
duplication, groupId, deduplicationID);

        } else {
            System.out.println("Enter a message.");
            message = in.nextLine();
            pubMessage(snsClient, message, topicArn);
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Display the message. Press any key to continue.");
        in.nextLine();
        messageList = receiveMessages(sqsClient, sqsQueueUrl, msgAttValue);
        for (Message mes : messageList) {
            System.out.println("Message Id: " + mes.messageId());
            System.out.println("Full Message: " + mes.body());
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Delete the received message. Press any key to
continue.");
        in.nextLine();
        deleteMessages(sqsClient, sqsQueueUrl, messageList);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Unsubscribe from the topic and delete the queue.
Press any key to continue.");
        in.nextLine();
        unSub(snsClient, subscriptionArn);
        deleteSQSQueue(sqsClient, sqsQueueName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Delete the topic. Press any key to continue.");
        in.nextLine();
        deleteSNSTopic(snsClient, topicArn);

        System.out.println(DASHES);
        System.out.println("The SNS/SQS workflow has completed successfully.");
        System.out.println(DASHES);
```

```
    }

    public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
        try {
            DeleteTopicRequest request = DeleteTopicRequest.builder()
                .topicArn(topicArn)
                .build();

            DeleteTopicResponse result = snsClient.deleteTopic(request);
            System.out.println("Status was " +
result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteSQSQueue(SqsClient sqsClient, String queueName) {
        try {
            GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
                .queueName(queueName)
                .build();

            String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();
            DeleteQueueRequest deleteQueueRequest = DeleteQueueRequest.builder()
                .queueUrl(queueUrl)
                .build();

            sqsClient.deleteQueue(deleteQueueRequest);
            System.out.println(queueName + " was successfully deleted.");

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void unSub(SnsClient snsClient, String subscriptionArn) {
        try {
            UnsubscribeRequest request = UnsubscribeRequest.builder()
                .subscriptionArn(subscriptionArn)
                .build();
```

```
            UnsubscribeResponse result = snsClient.unsubscribe(request);
            System.out.println("Status was " +
result.sdkHttpResponse().statusCode()
                + "\nSubscription was removed for " + request.subscriptionArn());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {
        try {
            List<DeleteMessageBatchRequestEntry> entries = new ArrayList<>();
            for (Message msg : messages) {
                DeleteMessageBatchRequestEntry entry =
DeleteMessageBatchRequestEntry.builder()
                    .id(msg.messageId())
                    .build();

                entries.add(entry);
            }

            DeleteMessageBatchRequest deleteMessageBatchRequest =
DeleteMessageBatchRequest.builder()
                .queueUrl(queueUrl)
                .entries(entries)
                .build();

            sqsClient.deleteMessageBatch(deleteMessageBatchRequest);
            System.out.println("The batch delete of messages was successful");

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static List<Message> receiveMessages(SqsClient sqsClient, String
queueUrl, String msgAttValue) {
        try {
            if (msgAttValue.isEmpty()) {
```

```
                ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .maxNumberOfMessages(5)
                    .build();
                return
sqsClient.receiveMessage(receiveMessageRequest).messages();
            } else {
                // We know there are filters on the message.
                ReceiveMessageRequest receiveRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageAttributeNames(msgAttValue) // Include other message
attributes if needed.
                    .maxNumberOfMessages(5)
                    .build();

                return sqsClient.receiveMessage(receiveRequest).messages();
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static void pubMessage(SnsClient snsClient, String message, String
topicArn) {
        try {
            PublishRequest request = PublishRequest.builder()
                .message(message)
                .topicArn(topicArn)
                .build();

            PublishResponse result = snsClient.publish(request);
            System.out
                .println(result.messageId() + " Message sent. Status is " +
result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

```
    }

    public static void pubMessageFIFO(SnsClient snsClient,
                                      String message,
                                      String topicArn,
                                      String msgAttValue,
                                      String duplication,
                                      String groupId,
                                      String deduplicationID) {

        try {
            PublishRequest request;
            // Means the user did not choose to use a message attribute.
            if (msgAttValue.isEmpty()) {
                if (duplication.compareTo("y") == 0) {
                    request = PublishRequest.builder()
                        .message(message)
                        .messageGroupId(groupId)
                        .topicArn(topicArn)
                        .build();
                } else {
                    request = PublishRequest.builder()
                        .message(message)
                        .messageDeduplicationId(deduplicationID)
                        .messageGroupId(groupId)
                        .topicArn(topicArn)
                        .build();
                }

            } else {
                Map<String, MessageAttributeValue> messageAttributes = new
HashMap<>();
                messageAttributes.put(msgAttValue,
MessageAttributeValue.builder()
                    .dataType("String")
                    .stringValue("true")
                    .build());

                if (duplication.compareTo("y") == 0) {
                    request = PublishRequest.builder()
                        .message(message)
                        .messageGroupId(groupId)
                        .topicArn(topicArn)
                        .build();
```

```
                } else {
                    // Create a publish request with the message and attributes.
                    request = PublishRequest.builder()
                        .topicArn(topicArn)
                        .message(message)
                        .messageDeduplicationId(deduplicationID)
                        .messageGroupId(groupId)
                        .messageAttributes(messageAttributes)
                        .build();
                }
            }

            // Publish the message to the topic.
            PublishResponse result = snsClient.publish(request);
            System.out
                .println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    // Subscribe to the SQS queue.
    public static String subQueue(SnsClient snsClient, String topicArn, String
queueArn, List<String> filterList) {
        try {
            SubscribeRequest request;
            if (filterList.isEmpty()) {
                // No filter subscription is added.
                request = SubscribeRequest.builder()
                    .protocol("sqs")
                    .endpoint(queueArn)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

                SubscribeResponse result = snsClient.subscribe(request);
                System.out.println("The queue " + queueArn + " has been
subscribed to the topic " + topicArn + "\n" +
                    "with the subscription ARN " + result.subscriptionArn());
                return result.subscriptionArn();
            } else {
```

```
                request = SubscribeRequest.builder()
                    .protocol("sqs")
                    .endpoint(queueArn)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

                SubscribeResponse result = snsClient.subscribe(request);
                System.out.println("The queue " + queueArn + " has been
subscribed to the topic " + topicArn + "\n" +
                    "with the subscription ARN " + result.subscriptionArn());

                String attributeName = "FilterPolicy";
                Gson gson = new Gson();
                String jsonString = "{\"tone\": []}";
                JsonObject jsonObject = gson.fromJson(jsonString,
JsonObject.class);
                JsonArray toneArray = jsonObject.getAsJsonArray("tone");
                for (String value : filterList) {
                    toneArray.add(new JsonPrimitive(value));
                }

                String updatedJsonString = gson.toJson(jsonObject);
                System.out.println(updatedJsonString);
                SetSubscriptionAttributesRequest attRequest =
SetSubscriptionAttributesRequest.builder()
                    .subscriptionArn(result.subscriptionArn())
                    .attributeName(attributeName)
                    .attributeValue(updatedJsonString)
                    .build();

                snsClient.setSubscriptionAttributes(attRequest);
                return result.subscriptionArn();
            }

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    // Attach a policy to the queue.
```

```
   public static void setQueueAttr(SqsClient sqsClient, String queueUrl, String
policy) {
       try {
           Map<software.amazon.awssdk.services.sqs.model.QueueAttributeName,
String> attrMap = new HashMap<>();
           attrMap.put(QueueAttributeName.POLICY, policy);

           SetQueueAttributesRequest attributesRequest =
SetQueueAttributesRequest.builder()
               .queueUrl(queueUrl)
               .attributes(attrMap)
               .build();

           sqsClient.setQueueAttributes(attributesRequest);
           System.out.println("The policy has been successfully attached.");

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static String getSQSQueueAttrs(SqsClient sqsClient, String queueUrl) {
       // Specify the attributes to retrieve.
       List<QueueAttributeName> atts = new ArrayList<>();
       atts.add(QueueAttributeName.QUEUE_ARN);

       GetQueueAttributesRequest attributesRequest =
GetQueueAttributesRequest.builder()
           .queueUrl(queueUrl)
           .attributeNames(atts)
           .build();

       GetQueueAttributesResponse response =
sqsClient.getQueueAttributes(attributesRequest);
       Map<String, String> queueAtts = response.attributesAsStrings();
       for (Map.Entry<String, String> queueAtt : queueAtts.entrySet())
           return queueAtt.getValue();

       return "";
   }

   public static String createQueue(SqsClient sqsClient, String queueName,
Boolean selectFIFO) {
```

```
        try {
            System.out.println("\nCreate Queue");
            if (selectFIFO) {
                Map<QueueAttributeName, String> attrs = new HashMap<>();
                attrs.put(QueueAttributeName.FIFO_QUEUE, "true");
                CreateQueueRequest createQueueRequest =
CreateQueueRequest.builder()
                    .queueName(queueName)
                    .attributes(attrs)
                    .build();

                sqsClient.createQueue(createQueueRequest);
                System.out.println("\nGet queue url");
                GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
                return getQueueUrlResponse.queueUrl();
            } else {
                CreateQueueRequest createQueueRequest =
CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();

                sqsClient.createQueue(createQueueRequest);
                System.out.println("\nGet queue url");
                GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
                return getQueueUrlResponse.queueUrl();
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static String createSNSTopic(SnsClient snsClient, String topicName) {
        CreateTopicResponse result;
        try {
            CreateTopicRequest request = CreateTopicRequest.builder()
                .name(topicName)
                .build();
```

```
            result = snsClient.createTopic(request);
            return result.topicArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static String createFIFO(SnsClient snsClient, String topicName, String
 duplication) {
        try {
            // Create a FIFO topic by using the SNS service client.
            Map<String, String> topicAttributes = new HashMap<>();
            if (duplication.compareTo("n") == 0) {
                topicAttributes.put("FifoTopic", "true");
                topicAttributes.put("ContentBasedDeduplication", "false");
            } else {
                topicAttributes.put("FifoTopic", "true");
                topicAttributes.put("ContentBasedDeduplication", "true");
            }

            CreateTopicRequest topicRequest = CreateTopicRequest.builder()
                .name(topicName)
                .attributes(topicAttributes)
                .build();

            CreateTopicResponse response = snsClient.createTopic(topicRequest);
            return response.topicArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API Reference》中的以下主题。
  - CreateQueue

- CreateTopic
- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- 发布
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

这是此场景的入口点。

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
```

```
};
```

前面的代码提供必要的依赖关系并启动此场景。下一节包含示例的大部分内容。

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../../libs/prompter.js').Prompter} prompter
   * @param {import('../../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
```

```
    this.logger = logger;
  }

  async welcome() {
    await this.logger.log(MESSAGES.description);
  }

  async confirmFifo() {
    await this.logger.log(MESSAGES.snsFifoDescription);
    this.isFifo = await this.prompter.confirm({
      message: MESSAGES.snsFifoPrompt,
    });

    if (this.isFifo) {
      this.logger.logSeparator(MESSAGES.headerDedup);
      await this.logger.log(MESSAGES.deduplicationNotice);
      await this.logger.log(MESSAGES.deduplicationDescription);
      this.autoDedup = await this.prompter.confirm({
        message: MESSAGES.deduplicationPrompt,
      });
    }
  }

  async createTopic() {
    await this.logger.log(MESSAGES.creatingTopics);
    this.topicName = await this.prompter.input({
      message: MESSAGES.topicNamePrompt,
    });
    if (this.isFifo) {
      this.topicName += ".fifo";
      this.logger.logSeparator(MESSAGES.headerFifoNaming);
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );
```

```
    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    const maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
```

```
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
```

```
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
```

```
          message: MESSAGES.fifoFilterSelect.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
```

```
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
```

```
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
```

```
    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
```

```
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- 有关 API 详细信息，请参阅《适用于 JavaScript 的 AWS SDK API Reference》中的以下主题。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 发布
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
package com.example.sns

import aws.sdk.kotlin.services.sns.SnsClient
import aws.sdk.kotlin.services.sns.model.CreateTopicRequest
import aws.sdk.kotlin.services.sns.model.DeleteTopicRequest
import aws.sdk.kotlin.services.sns.model.PublishRequest
import aws.sdk.kotlin.services.sns.model.SetSubscriptionAttributesRequest
import aws.sdk.kotlin.services.sns.model.SubscribeRequest
import aws.sdk.kotlin.services.sns.model.UnsubscribeRequest
import aws.sdk.kotlin.services.sqs.SqsClient
import aws.sdk.kotlin.services.sqs.model.CreateQueueRequest
import aws.sdk.kotlin.services.sqs.model.DeleteMessageBatchRequest
import aws.sdk.kotlin.services.sqs.model.DeleteMessageBatchRequestEntry
import aws.sdk.kotlin.services.sqs.model.DeleteQueueRequest
import aws.sdk.kotlin.services.sqs.model.GetQueueAttributesRequest
import aws.sdk.kotlin.services.sqs.model.GetQueueUrlRequest
import aws.sdk.kotlin.services.sqs.model.Message
import aws.sdk.kotlin.services.sqs.model.QueueAttributeName
import aws.sdk.kotlin.services.sqs.model.ReceiveMessageRequest
import aws.sdk.kotlin.services.sqs.model.SetQueueAttributesRequest
import com.google.gson.Gson
import com.google.gson.JsonObject
import com.google.gson.JsonPrimitive
import java.util.Scanner

/**
Before running this Kotlin code example, set up your development environment,
including your AWS credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This Kotlin example performs the following tasks:

 1. Gives the user three options to choose from.
 2. Creates an Amazon Simple Notification Service (Amazon SNS) topic.
 3. Creates an Amazon Simple Queue Service (Amazon SQS) queue.
 4. Gets the SQS queue Amazon Resource Name (ARN) attribute.
 5. Attaches an AWS Identity and Access Management (IAM) policy to the queue.
 6. Subscribes to the SQS queue.
 7. Publishes a message to the topic.
 8. Displays the messages.
 9. Deletes the received message.
```

```
 10. Unsubscribes from the topic.
 11. Deletes the SNS topic.
 */

val DASHES: String = String(CharArray(80)).replace("\u0000", "-")
suspend fun main() {
    val input = Scanner(System.`in`)
    val useFIFO: String
    var duplication = "n"
    var topicName: String
    var deduplicationID: String? = null
    var groupId: String? = null
    val topicArn: String?
    var sqsQueueName: String
    val sqsQueueUrl: String?
    val sqsQueueArn: String
    val subscriptionArn: String?
    var selectFIFO = false
    val message: String
    val messageList: List<Message?>?
    val filterList = ArrayList<String>()
    var msgAttValue = ""

    println(DASHES)
    println("Welcome to the AWS SDK for Kotlin messaging with topics and
 queues.")
    println(
        """
                In this scenario, you will create an SNS topic and subscribe an
 SQS queue to the topic.
                You can select from several options for configuring the topic and
 the subscriptions for the queue.
                You can then post to the topic and see the results in the queue.
        """.trimIndent(),
    )
    println(DASHES)

    println(DASHES)
    println(
        """
                SNS topics can be configured as FIFO (First-In-First-Out).
                FIFO topics deliver messages in order and support deduplication
 and message filtering.
                Would you like to work with FIFO topics? (y/n)
```

```
        """.trimIndent(),
    )
    useFIFO = input.nextLine()
    if (useFIFO.compareTo("y") == 0) {
        selectFIFO = true
        println("You have selected FIFO")
        println(
            """ Because you have chosen a FIFO topic, deduplication is supported.
        Deduplication IDs are either set in the message or automatically
 generated from content using a hash function.
        If a message is successfully published to an SNS FIFO topic, any message
 published and determined to have the same deduplication ID,
        within the five-minute deduplication interval, is accepted but not
 delivered.
        For more information about deduplication, see https://
docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.""",
        )

        println("Would you like to use content-based deduplication instead of
 entering a deduplication ID? (y/n)")
        duplication = input.nextLine()
        if (duplication.compareTo("y") == 0) {
            println("Enter a group id value")
            groupId = input.nextLine()
        } else {
            println("Enter deduplication Id value")
            deduplicationID = input.nextLine()
            println("Enter a group id value")
            groupId = input.nextLine()
        }
    }
    println(DASHES)

    println(DASHES)
    println("2. Create a topic.")
    println("Enter a name for your SNS topic.")
    topicName = input.nextLine()
    if (selectFIFO) {
        println("Because you have selected a FIFO topic, '.fifo' must be appended
 to the topic name.")
        topicName = "$topicName.fifo"
        println("The name of the topic is $topicName")
        topicArn = createFIFO(topicName, duplication)
        println("The ARN of the FIFO topic is $topicArn")
```

```
    } else {
        println("The name of the topic is $topicName")
        topicArn = createSNSTopic(topicName)
        println("The ARN of the non-FIFO topic is $topicArn")
    }
    println(DASHES)

    println(DASHES)
    println("3. Create an SQS queue.")
    println("Enter a name for your SQS queue.")
    sqsQueueName = input.nextLine()
    if (selectFIFO) {
        sqsQueueName = "$sqsQueueName.fifo"
    }
    sqsQueueUrl = createQueue(sqsQueueName, selectFIFO)
    println("The queue URL is $sqsQueueUrl")
    println(DASHES)

    println(DASHES)
    println("4. Get the SQS queue ARN attribute.")
    sqsQueueArn = getSQSQueueAttrs(sqsQueueUrl)
    println("The ARN of the new queue is $sqsQueueArn")
    println(DASHES)

    println(DASHES)
    println("5. Attach an IAM policy to the queue.")
    // Define the policy to use.
    val policy = """{
     "Statement": [
     {
         "Effect": "Allow",
                 "Principal": {
             "Service": "sns.amazonaws.com"
         },
         "Action": "sqs:SendMessage",
                 "Resource": "$sqsQueueArn",
                 "Condition": {
             "ArnEquals": {
                 "aws:SourceArn": "$topicArn"
             }
         }
     }
     ]
     }"""
```

```
    setQueueAttr(sqsQueueUrl, policy)
    println(DASHES)

    println(DASHES)
    println("6. Subscribe to the SQS queue.")
    if (selectFIFO) {
        println(
            """If you add a filter to this subscription, then only the filtered
 messages will be received in the queue.
For information about message filtering, see https://docs.aws.amazon.com/sns/
latest/dg/sns-message-filtering.html
For this example, you can filter messages by a "tone" attribute.""",
        )
        println("Would you like to filter messages for $sqsQueueName's
 subscription to the topic $topicName?  (y/n)")
        val filterAns: String = input.nextLine()
        if (filterAns.compareTo("y") == 0) {
            var moreAns = false
            println("You can filter messages by using one or more of the
 following \"tone\" attributes.")
            println("1. cheerful")
            println("2. funny")
            println("3. serious")
            println("4. sincere")
            while (!moreAns) {
                println("Select a number or choose 0 to end.")
                val ans: String = input.nextLine()
                when (ans) {
                    "1" -> filterList.add("cheerful")
                    "2" -> filterList.add("funny")
                    "3" -> filterList.add("serious")
                    "4" -> filterList.add("sincere")
                    else -> moreAns = true
                }
            }
        }
    }
    subscriptionArn = subQueue(topicArn, sqsQueueArn, filterList)
    println(DASHES)

    println(DASHES)
    println("7. Publish a message to the topic.")
    if (selectFIFO) {
        println("Would you like to add an attribute to this message?  (y/n)")
```

```
        val msgAns: String = input.nextLine()
        if (msgAns.compareTo("y") == 0) {
            println("You can filter messages by one or more of the following
\"tone\" attributes.")
            println("1. cheerful")
            println("2. funny")
            println("3. serious")
            println("4. sincere")
            println("Select a number or choose 0 to end.")
            val ans: String = input.nextLine()
            msgAttValue = when (ans) {
                "1" -> "cheerful"
                "2" -> "funny"
                "3" -> "serious"
                else -> "sincere"
            }
            println("Selected value is $msgAttValue")
        }
        println("Enter a message.")
        message = input.nextLine()
        pubMessageFIFO(message, topicArn, msgAttValue, duplication, groupId,
deduplicationID)
    } else {
        println("Enter a message.")
        message = input.nextLine()
        pubMessage(message, topicArn)
    }
    println(DASHES)

    println(DASHES)
    println("8. Display the message. Press any key to continue.")
    input.nextLine()
    messageList = receiveMessages(sqsQueueUrl, msgAttValue)
    if (messageList != null) {
        for (mes in messageList) {
            println("Message Id: ${mes.messageId}")
            println("Full Message: ${mes.body}")
        }
    }
    println(DASHES)

    println(DASHES)
    println("9. Delete the received message. Press any key to continue.")
    input.nextLine()
```

```
    if (messageList != null) {
        deleteMessages(sqsQueueUrl, messageList)
    }
    println(DASHES)

    println(DASHES)
    println("10. Unsubscribe from the topic and delete the queue. Press any key
 to continue.")
    input.nextLine()
    unSub(subscriptionArn)
    deleteSQSQueue(sqsQueueName)
    println(DASHES)

    println(DASHES)
    println("11. Delete the topic. Press any key to continue.")
    input.nextLine()
    deleteSNSTopic(topicArn)
    println(DASHES)

    println(DASHES)
    println("The SNS/SQS workflow has completed successfully.")
    println(DASHES)
}

suspend fun deleteSNSTopic(topicArnVal: String?) {
    val request = DeleteTopicRequest {
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was deleted")
    }
}

suspend fun deleteSQSQueue(queueNameVal: String) {
    val getQueueRequest = GetQueueUrlRequest {
        queueName = queueNameVal
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val queueUrlVal = sqsClient.getQueueUrl(getQueueRequest).queueUrl
        val deleteQueueRequest = DeleteQueueRequest {
            queueUrl = queueUrlVal
```

```
        }

        sqsClient.deleteQueue(deleteQueueRequest)
        println("$queueNameVal was successfully deleted.")
    }
}

suspend fun unSub(subscripArn: String?) {
    val request = UnsubscribeRequest {
        subscriptionArn = subscripArn
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.unsubscribe(request)
        println("Subscription was removed for $subscripArn")
    }
}

suspend fun deleteMessages(queueUrlVal: String?, messages: List<Message>) {
    val entriesVal: MutableList<DeleteMessageBatchRequestEntry> = mutableListOf()
    for (msg in messages) {
        val entry = DeleteMessageBatchRequestEntry {
            id = msg.messageId
        }
        entriesVal.add(entry)
    }

    val deleteMessageBatchRequest = DeleteMessageBatchRequest {
        queueUrl = queueUrlVal
        entries = entriesVal
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteMessageBatch(deleteMessageBatchRequest)
        println("The batch delete of messages was successful")
    }
}

suspend fun receiveMessages(queueUrlVal: String?, msgAttValue: String):
 List<Message>? {
    if (msgAttValue.isEmpty()) {
        val request = ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }
```

```
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            return sqsClient.receiveMessage(request).messages
        }
    } else {
        val receiveRequest = ReceiveMessageRequest {
            queueUrl = queueUrlVal
            waitTimeSeconds = 1
            maxNumberOfMessages = 5
        }
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            return sqsClient.receiveMessage(receiveRequest).messages
        }
    }
}

suspend fun pubMessage(messageVal: String?, topicArnVal: String?) {
    val request = PublishRequest {
        message = messageVal
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}

suspend fun pubMessageFIFO(
    messageVal: String?,
    topicArnVal: String?,
    msgAttValue: String,
    duplication: String,
    groupIdVal: String?,
    deduplicationID: String?,
) {
    // Means the user did not choose to use a message attribute.
    if (msgAttValue.isEmpty()) {
        if (duplication.compareTo("y") == 0) {
            val request = PublishRequest {
                message = messageVal
                messageGroupId = groupIdVal
                topicArn = topicArnVal
            }
```

```
            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        } else {
            val request = PublishRequest {
                message = messageVal
                messageDeduplicationId = deduplicationID
                messageGroupId = groupIdVal
                topicArn = topicArnVal
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        }
    } else {
        val messAttr = aws.sdk.kotlin.services.sns.model.MessageAttributeValue {
            dataType = "String"
            stringValue = "true"
        }

        val mapAtt: Map<String,
aws.sdk.kotlin.services.sns.model.MessageAttributeValue> =
            mapOf(msgAttValue to messAttr)
        if (duplication.compareTo("y") == 0) {
            val request = PublishRequest {
                message = messageVal
                messageGroupId = groupIdVal
                topicArn = topicArnVal
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        } else {
            // Create a publish request with the message and attributes.
            val request = PublishRequest {
                topicArn = topicArnVal
                message = messageVal
                messageDeduplicationId = deduplicationID
                messageGroupId = groupIdVal
```

```
                messageAttributes = mapAtt
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        }
    }
}

// Subscribe to the SQS queue.
suspend fun subQueue(topicArnVal: String?, queueArnVal: String, filterList:
 List<String?>): String? {
    val request: SubscribeRequest
    if (filterList.isEmpty()) {
        // No filter subscription is added.
        request = SubscribeRequest {
            protocol = "sqs"
            endpoint = queueArnVal
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.subscribe(request)
            println(
                "The queue " + queueArnVal + " has been subscribed to the topic "
 + topicArnVal + "\n" +
                    "with the subscription ARN " + result.subscriptionArn,
            )
            return result.subscriptionArn
        }
    } else {
        request = SubscribeRequest {
            protocol = "sqs"
            endpoint = queueArnVal
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.subscribe(request)
```

```
            println("The queue $queueArnVal has been subscribed to the topic
 $topicArnVal with the subscription ARN ${result.subscriptionArn}")

            val attributeNameVal = "FilterPolicy"
            val gson = Gson()
            val jsonString = "{\"tone\": []}"
            val jsonObject = gson.fromJson(jsonString, JsonObject::class.java)
            val toneArray = jsonObject.getAsJsonArray("tone")
            for (value: String? in filterList) {
                toneArray.add(JsonPrimitive(value))
            }

            val updatedJsonString: String = gson.toJson(jsonObject)
            println(updatedJsonString)
            val attRequest = SetSubscriptionAttributesRequest {
                subscriptionArn = result.subscriptionArn
                attributeName = attributeNameVal
                attributeValue = updatedJsonString
            }

            snsClient.setSubscriptionAttributes(attRequest)
            return result.subscriptionArn
        }
    }
}

suspend fun setQueueAttr(queueUrlVal: String?, policy: String) {
    val attrMap: MutableMap<String, String> = HashMap()
    attrMap[QueueAttributeName.Policy.toString()] = policy

    val attributesRequest = SetQueueAttributesRequest {
        queueUrl = queueUrlVal
        attributes = attrMap
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        sqsClient.setQueueAttributes(attributesRequest)
        println("The policy has been successfully attached.")
    }
}

suspend fun getSQSQueueAttrs(queueUrlVal: String?): String {
    val atts: MutableList<QueueAttributeName> = ArrayList()
    atts.add(QueueAttributeName.QueueArn)
```

```
    val attributesRequest = GetQueueAttributesRequest {
        queueUrl = queueUrlVal
        attributeNames = atts
    }
    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.getQueueAttributes(attributesRequest)
        val mapAtts = response.attributes
        if (mapAtts != null) {
            mapAtts.forEach { entry ->
                println("${entry.key} : ${entry.value}")
                return entry.value
            }
        }
    }
    return ""
}

suspend fun createQueue(queueNameVal: String?, selectFIFO: Boolean): String? {
    println("\nCreate Queue")
    if (selectFIFO) {
        val attrs = mutableMapOf<String, String>()
        attrs[QueueAttributeName.FifoQueue.toString()] = "true"

        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
            attributes = attrs
        }

        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
            println("\nGet queue url")

            val urlRequest = GetQueueUrlRequest {
                queueName = queueNameVal
            }

            val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
            return getQueueUrlResponse.queueUrl
        }
    } else {
        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
        }
```

```
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
            println("Get queue url")

            val urlRequest = GetQueueUrlRequest {
                queueName = queueNameVal
            }

            val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
            return getQueueUrlResponse.queueUrl
        }
    }
}

suspend fun createSNSTopic(topicName: String?): String? {
    val request = CreateTopicRequest {
        name = topicName
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn
    }
}

suspend fun createFIFO(topicName: String?, duplication: String): String? {
    val topicAttributes: MutableMap<String, String> = HashMap()
    if (duplication.compareTo("n") == 0) {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "false"
    } else {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "true"
    }

    val topicRequest = CreateTopicRequest {
        name = topicName
        attributes = topicAttributes
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.createTopic(topicRequest)
        return response.topicArn
    }
```

```
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API Reference》中的以下主题。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 发布
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
class TopicsAndQueuesScenario:
    """Manages the Topics and Queues feature scenario."""

    DASHES = "-" * 80

    def __init__(self, sns_wrapper: SnsWrapper, sqs_wrapper: SqsWrapper) -> None:
        """
        Initialize the Topics and Queues scenario.
```

```
        :param sns_wrapper: SnsWrapper instance for SNS operations.
        :param sqs_wrapper: SqsWrapper instance for SQS operations.
        """
        self.sns_wrapper = sns_wrapper
        self.sqs_wrapper = sqs_wrapper

        # Scenario state
        self.use_fifo_topic = False
        self.use_content_based_deduplication = False
        self.topic_name = None
        self.topic_arn = None
        self.queue_count = 2
        self.queue_urls = []
        self.subscription_arns = []
        self.tones = ["cheerful", "funny", "serious", "sincere"]

    def run_scenario(self) -> None:
        """Run the Topics and Queues feature scenario."""
        print(self.DASHES)
        print("Welcome to messaging with topics and queues.")
        print(self.DASHES)
        print(f"""
    In this scenario, you will create an SNS topic and subscribe
{self.queue_count} SQS queues to the topic.
    You can select from several options for configuring the topic and the
subscriptions for the queues.
    You can then post to the topic and see the results in the queues.
        """)

        try:
            # Setup Phase
            print(self.DASHES)
            self._setup_topic()
            print(self.DASHES)

            self._setup_queues()
            print(self.DASHES)

            # Demonstration Phase
            self._publish_messages()
            print(self.DASHES)

            # Examination Phase
            self._poll_queues_for_messages()
```

```
            print(self.DASHES)

            # Cleanup Phase
            self._cleanup_resources()
            print(self.DASHES)

        except Exception as e:
            logger.error(f"Scenario failed: {e}")
            print(f"There was a problem with the scenario: {e}")
            print("\nInitiating cleanup...")
            try:
                self._cleanup_resources()
            except Exception as cleanup_error:
                logger.error(f"Error during cleanup: {cleanup_error}")

        print("Messaging with topics and queues scenario is complete.")
        print(self.DASHES)

    def _setup_topic(self) -> None:
        """Set up the SNS topic to be used with the queues."""
        print("SNS topics can be configured as FIFO (First-In-First-Out).")
        print("FIFO topics deliver messages in order and support deduplication
 and message filtering.")
        print()

        self.use_fifo_topic = q.ask("Would you like to work with FIFO topics? (y/
n): ", q.is_yesno)

        if self.use_fifo_topic:
            print(self.DASHES)
            self.topic_name = q.ask("Enter a name for your SNS topic: ",
 q.non_empty)
            print("Because you have selected a FIFO topic, '.fifo' must be
 appended to the topic name.")
            print()

            print(self.DASHES)
            print("""
    Because you have chosen a FIFO topic, deduplication is supported.
    Deduplication IDs are either set in the message or automatically generated
    from content using a hash function.

    If a message is successfully published to an SNS FIFO topic, any message
    published and determined to have the same deduplication ID,
```

```
   within the five-minute deduplication interval, is accepted but not delivered.

   For more information about deduplication,
   see https://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.
           """)

           self.use_content_based_deduplication = q.ask(
               "Use content-based deduplication instead of entering a
deduplication ID? (y/n): ",
               q.is_yesno
           )
       else:
           self.topic_name = q.ask("Enter a name for your SNS topic: ",
q.non_empty)

       print(self.DASHES)

       # Create the topic
       self.topic_arn = self.sns_wrapper.create_topic(
           self.topic_name,
           self.use_fifo_topic,
           self.use_content_based_deduplication
       )

       print(f"Your new topic with the name {self.topic_name}")
       print(f"  and Amazon Resource Name (ARN) {self.topic_arn}")
       print(f"  has been created.")
       print()

   def _setup_queues(self) -> None:
       """Set up the SQS queues and subscribe them to the topic."""
       print(f"Now you will create {self.queue_count} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.")

       for i in range(self.queue_count):
           queue_name = q.ask(f"Enter a name for SQS queue #{i+1}: ",
q.non_empty)

           if self.use_fifo_topic and i == 0:
               print("Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.")

           # Create the queue
```

```
            queue_url = self.sqs_wrapper.create_queue(queue_name,
self.use_fifo_topic)
            self.queue_urls.append(queue_url)

            print(f"Your new queue with the name {queue_name}")
            print(f"  and queue URL {queue_url}")
            print(f"  has been created.")
            print()

            if i == 0:
                print("The queue URL is used to retrieve the queue ARN,")
                print("which is used to create a subscription.")
                print(self.DASHES)

            # Get queue ARN
            queue_arn = self.sqs_wrapper.get_queue_arn(queue_url)

            if i == 0:
                print("An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue,")
                print("enabling it to receive messages from an SNS topic.")

            # Set queue policy to allow SNS to send messages
            self.sqs_wrapper.set_queue_policy_for_topic(queue_arn,
self.topic_arn, queue_url)

            # Set up message filtering if using FIFO
            subscription_arn = self._setup_subscription_with_filter(i, queue_arn,
queue_name)
            self.subscription_arns.append(subscription_arn)

   def _setup_subscription_with_filter(self, queue_index: int, queue_arn: str,
queue_name: str) -> str:
       """Set up subscription with optional message filtering."""
       filter_policy = None

       if self.use_fifo_topic:
           print(self.DASHES)
           if queue_index == 0:
               print("Subscriptions to a FIFO topic can have filters.")
               print("If you add a filter to this subscription, then only the
filtered messages")
               print("will be received in the queue.")
               print()
```

```
                print("For information about message filtering,")
                print("see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html")
                print()
                print("For this example, you can filter messages by a TONE
 attribute.")

            use_filter = q.ask(f"Filter messages for {queue_name}'s subscription
 to the topic? (y/n): ", q.is_yesno)

            if use_filter:
                filter_policy = self._create_filter_policy()

        subscription_arn = self.sns_wrapper.subscribe_queue_to_topic(
            self.topic_arn, queue_arn, filter_policy
        )

        print(f"The queue {queue_name} has been subscribed to the topic
 {self.topic_name}")
        print(f"  with the subscription ARN {subscription_arn}")

        return subscription_arn

    def _create_filter_policy(self) -> str:
        """Create a message filter policy based on user selections."""
        print(self.DASHES)
        print("You can filter messages by one or more of the following TONE
 attributes.")

        filter_selections = []
        selection_number = 0

        while True:
            print("Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.")
            for i, tone in enumerate(self.tones, 1):
                print(f"  {i}. {tone}")

            selection = q.ask("Your choice: ", q.is_int, q.in_range(0,
 len(self.tones)))

            if selection == 0:
                break
```

```
            elif selection > 0 and self.tones[selection - 1] not in
filter_selections:
                filter_selections.append(self.tones[selection - 1])
                print(f"Added '{self.tones[selection - 1]}' to filter list.")

        if filter_selections:
            filters = {"tone": filter_selections}
            return json.dumps(filters)
        return None

    def _publish_messages(self) -> None:
        """Publish messages to the topic with various options."""
        print("Now we can publish messages.")

        keep_sending = True
        while keep_sending:
            print()
            message = q.ask("Enter a message to publish: ", q.non_empty)

            message_group_id = None
            deduplication_id = None
            tone_attribute = None

            if self.use_fifo_topic:
                print("Because you are using a FIFO topic, you must set a message
group ID.")
                print("All messages within the same group will be received in the
order they were published.")
                print()
                message_group_id = q.ask("Enter a message group ID for this
message: ", q.non_empty)

                if not self.use_content_based_deduplication:
                    print("Because you are not using content-based
deduplication,")
                    print("you must enter a deduplication ID.")
                    deduplication_id = q.ask("Enter a deduplication ID for this
message: ", q.non_empty)

                # Ask about tone attribute
                add_attribute = q.ask("Add an attribute to this message? (y/n):
", q.is_yesno)
                if add_attribute:
                    print("Enter a number for an attribute:")
```

```
                    for i, tone in enumerate(self.tones, 1):
                        print(f"  {i}. {tone}")

                    selection = q.ask("Your choice: ", q.is_int, q.in_range(1,
 len(self.tones)))
                    if 1 <= selection <= len(self.tones):
                        tone_attribute = self.tones[selection - 1]

            # Publish the message
            message_id = self.sns_wrapper.publish_message(
                self.topic_arn,
                message,
                tone_attribute,
                deduplication_id,
                message_group_id
            )

            print(f"Message published with ID: {message_id}")

            keep_sending = q.ask("Send another message? (y/n): ", q.is_yesno)

    def _poll_queues_for_messages(self) -> None:
        """Poll all queues for messages and display results."""
        for i, queue_url in enumerate(self.queue_urls):
            print(f"Polling queue #{i+1} at {queue_url} for messages...")

            q.ask("Press Enter to continue...")

            messages = self._poll_queue_for_messages(queue_url)

            if messages:
                print(f"{len(messages)} message(s) were received by queue #{i
+1}")
                for j, message in enumerate(messages, 1):
                    print(f"  Message {j}:")
                    # Parse the SNS message body to get the actual message
                    try:
                        sns_message = json.loads(message['Body'])
                        actual_message = sns_message.get('Message',
 message['Body'])
                        print(f"    {actual_message}")
                    except (json.JSONDecodeError, KeyError):
                        print(f"    {message['Body']}")
```

```
                # Delete the messages
                self.sqs_wrapper.delete_messages(queue_url, messages)
                print(f"Messages deleted from queue #{i+1}")
            else:
                print(f"No messages received by queue #{i+1}")

            print(self.DASHES)

    def _poll_queue_for_messages(self, queue_url: str) -> List[Dict[str, Any]]:
        """Poll a single queue for messages."""
        all_messages = []
        max_polls = 3  # Limit polling to avoid infinite loops

        for poll_count in range(max_polls):
            messages = self.sqs_wrapper.receive_messages(queue_url, 10)

            if messages:
                all_messages.extend(messages)
                print(f"  Received {len(messages)} messages in poll {poll_count +
1}")
                # Small delay between polls
                time.sleep(1)
            else:
                print(f"  No messages in poll {poll_count + 1}")
                break

        return all_messages

    def _cleanup_resources(self) -> None:
        """Clean up all resources created during the scenario."""
        print("Cleaning up resources...")

        # Delete queues
        for i, queue_url in enumerate(self.queue_urls):
            if queue_url:
                delete_queue = q.ask(f"Delete queue #{i+1} with URL {queue_url}?
(y/n): ", q.is_yesno)
                if delete_queue:
                    try:
                        self.sqs_wrapper.delete_queue(queue_url)
                        print(f"Deleted queue #{i+1}")
                    except Exception as e:
                        print(f"Error deleting queue #{i+1}: {e}")
```

```
        # Unsubscribe from topic
        for i, subscription_arn in enumerate(self.subscription_arns):
            if subscription_arn:
                try:
                    self.sns_wrapper.unsubscribe(subscription_arn)
                    print(f"Unsubscribed subscription #{i+1}")
                except Exception as e:
                    print(f"Error unsubscribing #{i+1}: {e}")

        # Delete topic
        if self.topic_arn:
            delete_topic = q.ask(f"Delete topic {self.topic_name}? (y/n): ",
 q.is_yesno)
            if delete_topic:
                try:
                    self.sns_wrapper.delete_topic(self.topic_arn)
                    print(f"Deleted topic {self.topic_name}")
                except Exception as e:
                    print(f"Error deleting topic: {e}")

        print("Resource cleanup complete.")
```

创建包含 Amazon SNS 和 Amazon SQS 操作的类，以便在场景中使用。

```
class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
```

```
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)


    def create_topic(
        self,
        topic_name: str,
        is_fifo: bool = False,
        content_based_deduplication: bool = False
    ) -> str:
        """
        Create an SNS topic.

        :param topic_name: The name of the topic to create.
        :param is_fifo: Whether to create a FIFO topic.
        :param content_based_deduplication: Whether to use content-based
deduplication for FIFO topics.
        :return: The ARN of the created topic.
        :raises ClientError: If the topic creation fails.
        """
        try:
            # Add .fifo suffix for FIFO topics
            if is_fifo and not topic_name.endswith('.fifo'):
                topic_name += '.fifo'

            attributes = {}
            if is_fifo:
                attributes['FifoTopic'] = 'true'
                if content_based_deduplication:
                    attributes['ContentBasedDeduplication'] = 'true'

            response = self.sns_client.create_topic(
                Name=topic_name,
                Attributes=attributes
            )

            topic_arn = response['TopicArn']
            logger.info(f"Created topic: {topic_name} with ARN: {topic_arn}")
            return topic_arn

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
```

```
            logger.error(f"Error creating topic {topic_name}: {error_code} -
{e}")
            raise


    def subscribe_queue_to_topic(
        self,
        topic_arn: str,
        queue_arn: str,
        filter_policy: Optional[str] = None
    ) -> str:
        """
        Subscribe an SQS queue to an SNS topic.

        :param topic_arn: The ARN of the SNS topic.
        :param queue_arn: The ARN of the SQS queue.
        :param filter_policy: Optional JSON filter policy for message filtering.
        :return: The ARN of the subscription.
        :raises ClientError: If the subscription fails.
        """
        try:
            attributes = {}
            if filter_policy:
                attributes['FilterPolicy'] = filter_policy

            response = self.sns_client.subscribe(
                TopicArn=topic_arn,
                Protocol='sqs',
                Endpoint=queue_arn,
                Attributes=attributes
            )

            subscription_arn = response['SubscriptionArn']
            logger.info(f"Subscribed queue {queue_arn} to topic {topic_arn}")
            return subscription_arn

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error subscribing queue to topic: {error_code} - {e}")
            raise


    def publish_message(
        self,
```

```
        topic_arn: str,
        message: str,
        tone_attribute: Optional[str] = None,
        deduplication_id: Optional[str] = None,
        message_group_id: Optional[str] = None
    ) -> str:
        """
        Publish a message to an SNS topic.

        :param topic_arn: The ARN of the SNS topic.
        :param message: The message content to publish.
        :param tone_attribute: Optional tone attribute for message filtering.
        :param deduplication_id: Optional deduplication ID for FIFO topics.
        :param message_group_id: Optional message group ID for FIFO topics.
        :return: The message ID of the published message.
        :raises ClientError: If the message publication fails.
        """
        try:
            publish_args = {
                'TopicArn': topic_arn,
                'Message': message
            }

            # Add message attributes if tone is specified
            if tone_attribute:
                publish_args['MessageAttributes'] = {
                    'tone': {
                        'DataType': 'String',
                        'StringValue': tone_attribute
                    }
                }

            # Add FIFO-specific parameters
            if message_group_id:
                publish_args['MessageGroupId'] = message_group_id

            if deduplication_id:
                publish_args['MessageDeduplicationId'] = deduplication_id

            response = self.sns_client.publish(**publish_args)

            message_id = response['MessageId']
            logger.info(f"Published message to topic {topic_arn} with ID:
{message_id}")
```

```
            return message_id

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error publishing message to topic: {error_code} -
{e}")
            raise


    def unsubscribe(self, subscription_arn: str) -> bool:
        """
        Unsubscribe from an SNS topic.

        :param subscription_arn: The ARN of the subscription to remove.
        :return: True if successful.
        :raises ClientError: If the unsubscribe operation fails.
        """
        try:
            self.sns_client.unsubscribe(SubscriptionArn=subscription_arn)

            logger.info(f"Unsubscribed: {subscription_arn}")
            return True

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')

            if error_code == 'NotFound':
                logger.warning(f"Subscription not found: {subscription_arn}")
                return True  # Already unsubscribed
            else:
                logger.error(f"Error unsubscribing: {error_code} - {e}")
                raise


    def delete_topic(self, topic_arn: str) -> bool:
        """
        Delete an SNS topic.

        :param topic_arn: The ARN of the topic to delete.
        :return: True if successful.
        :raises ClientError: If the topic deletion fails.
        """
        try:
            self.sns_client.delete_topic(TopicArn=topic_arn)
```

```
            logger.info(f"Deleted topic: {topic_arn}")
            return True

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')

            if error_code == 'NotFound':
                logger.warning(f"Topic not found: {topic_arn}")
                return True  # Already deleted
            else:
                logger.error(f"Error deleting topic: {error_code} - {e}")
                raise


    def list_topics(self) -> list:
        """
        List all SNS topics in the account using pagination.

        :return: List of topic ARNs.
        :raises ClientError: If listing topics fails.
        """
        try:
            topics = []
            paginator = self.sns_client.get_paginator('list_topics')

            for page in paginator.paginate():
                topics.extend([topic['TopicArn'] for topic in page.get('Topics',
 [])])

            logger.info(f"Found {len(topics)} topics")
            return topics

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            if error_code == 'AuthorizationError':
                logger.error("Authorization error listing topics - check IAM
 permissions")
            else:
                logger.error(f"Error listing topics: {error_code} - {e}")
            raise


class SqsWrapper:
```

```
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)


    def create_queue(self, queue_name: str, is_fifo: bool = False) -> str:
        """
        Create an SQS queue.

        :param queue_name: The name of the queue to create.
        :param is_fifo: Whether to create a FIFO queue.
        :return: The URL of the created queue.
        :raises ClientError: If the queue creation fails.
        """
        try:
            # Add .fifo suffix for FIFO queues
            if is_fifo and not queue_name.endswith('.fifo'):
                queue_name += '.fifo'

            attributes = {}
            if is_fifo:
                attributes['FifoQueue'] = 'true'

            response = self.sqs_client.create_queue(
                QueueName=queue_name,
                Attributes=attributes
            )
```

```
            queue_url = response['QueueUrl']
            logger.info(f"Created queue: {queue_name} with URL: {queue_url}")
            return queue_url

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error creating queue {queue_name}: {error_code} -
{e}")
            raise


    def get_queue_arn(self, queue_url: str) -> str:
        """
        Get the ARN of an SQS queue.

        :param queue_url: The URL of the queue.
        :return: The ARN of the queue.
        :raises ClientError: If getting queue attributes fails.
        """
        try:
            response = self.sqs_client.get_queue_attributes(
                QueueUrl=queue_url,
                AttributeNames=['QueueArn']
            )

            queue_arn = response['Attributes']['QueueArn']
            logger.info(f"Queue ARN for {queue_url}: {queue_arn}")
            return queue_arn

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error getting queue ARN: {error_code} - {e}")
            raise


    def set_queue_policy_for_topic(self, queue_arn: str, topic_arn: str,
queue_url: str) -> bool:
        """
        Set the queue policy to allow SNS to send messages to the queue.

        :param queue_arn: The ARN of the SQS queue.
        :param topic_arn: The ARN of the SNS topic.
        :param queue_url: The URL of the SQS queue.
        :return: True if successful.
```

```
        :raises ClientError: If setting the queue policy fails.
        """
        try:
            # Create policy that allows SNS to send messages to the queue
            policy = {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "sns.amazonaws.com"
                        },
                        "Action": "sqs:SendMessage",
                        "Resource": queue_arn,
                        "Condition": {
                            "ArnEquals": {
                                "aws:SourceArn": topic_arn
                            }
                        }
                    }
                ]
            }

            self.sqs_client.set_queue_attributes(
                QueueUrl=queue_url,
                Attributes={
                    'Policy': json.dumps(policy)
                }
            )

            logger.info(f"Set queue policy for {queue_url} to allow messages from
{topic_arn}")
            return True

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error setting queue policy: {error_code} - {e}")
            raise


    def receive_messages(self, queue_url: str, max_messages: int = 10) ->
List[Dict[str, Any]]:
        """
        Receive messages from an SQS queue.
```

```
        :param queue_url: The URL of the queue to receive messages from.
        :param max_messages: Maximum number of messages to receive (1-10).
        :return: List of received messages.
        :raises ClientError: If receiving messages fails.
        """
        try:
            # Ensure max_messages is within valid range
            max_messages = max(1, min(10, max_messages))

            response = self.sqs_client.receive_message(
                QueueUrl=queue_url,
                MaxNumberOfMessages=max_messages,
                WaitTimeSeconds=2,  # Short polling
                MessageAttributeNames=['All']
            )

            messages = response.get('Messages', [])
            logger.info(f"Received {len(messages)} messages from {queue_url}")
            return messages

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error receiving messages: {error_code} - {e}")
            raise


    def delete_messages(self, queue_url: str, messages: List[Dict[str, Any]]) ->
bool:
        """
        Delete messages from an SQS queue in batches.

        :param queue_url: The URL of the queue.
        :param messages: List of messages to delete.
        :return: True if successful.
        :raises ClientError: If deleting messages fails.
        """
        try:
            if not messages:
                return True

            # Build delete entries for batch delete
            delete_entries = []
            for i, message in enumerate(messages):
```

```
                delete_entries.append({
                    'Id': str(i),
                    'ReceiptHandle': message['ReceiptHandle']
                })

            # Delete messages in batches of 10 (SQS limit)
            batch_size = 10
            for i in range(0, len(delete_entries), batch_size):
                batch = delete_entries[i:i + batch_size]

                response = self.sqs_client.delete_message_batch(
                    QueueUrl=queue_url,
                    Entries=batch
                )

                # Check for failures
                if 'Failed' in response and response['Failed']:
                    for failed in response['Failed']:
                        logger.warning(f"Failed to delete message: {failed}")

            logger.info(f"Deleted {len(messages)} messages from {queue_url}")
            return True

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error deleting messages: {error_code} - {e}")
            raise


    def delete_queue(self, queue_url: str) -> bool:
        """
        Delete an SQS queue.

        :param queue_url: The URL of the queue to delete.
        :return: True if successful.
        :raises ClientError: If the queue deletion fails.
        """
        try:
            self.sqs_client.delete_queue(QueueUrl=queue_url)

            logger.info(f"Deleted queue: {queue_url}")
            return True

        except ClientError as e:
```

```
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')

            if error_code == 'AWS.SimpleQueueService.NonExistentQueue':
                logger.warning(f"Queue not found: {queue_url}")
                return True  # Already deleted
            else:
                logger.error(f"Error deleting queue: {error_code} - {e}")
                raise


    def list_queues(self, queue_name_prefix: Optional[str] = None) -> List[str]:
        """
        List all SQS queues in the account using pagination.

        :param queue_name_prefix: Optional prefix to filter queue names.
        :return: List of queue URLs.
        :raises ClientError: If listing queues fails.
        """
        try:
            queue_urls = []
            paginator = self.sqs_client.get_paginator('list_queues')

            page_params = {}
            if queue_name_prefix:
                page_params['QueueNamePrefix'] = queue_name_prefix

            for page in paginator.paginate(**page_params):
                queue_urls.extend(page.get('QueueUrls', []))

            logger.info(f"Found {len(queue_urls)} queues")
            return queue_urls

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            if error_code == 'AccessDenied':
                logger.error("Access denied listing queues - check IAM
permissions")
            else:
                logger.error(f"Error listing queues: {error_code} - {e}")
            raise

    def send_message(self, queue_url: str, message_body: str, **kwargs) -> str:
        """
        Send a message to an SQS queue.
```

```
        :param queue_url: The URL of the queue.
        :param message_body: The message content.
        :param kwargs: Additional message parameters (DelaySeconds,
MessageAttributes, etc.).
        :return: The message ID.
        :raises ClientError: If sending the message fails.
        """
        try:
            send_params = {
                'QueueUrl': queue_url,
                'MessageBody': message_body,
                **kwargs
            }

            response = self.sqs_client.send_message(**send_params)

            message_id = response['MessageId']
            logger.info(f"Sent message to {queue_url} with ID: {message_id}")
            return message_id

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error sending message: {error_code} - {e}")
            raise
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API Reference》中的以下主题。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 发布
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe

- Unsubscribe

Swift

适用于 Swift 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import ArgumentParser
import AWSClientRuntime
import AWSSNS
import AWSSQS
import Foundation

struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "queue-scenario",
        abstract: """
        This example interactively demonstrates how to use Amazon Simple
        Notification Service (Amazon SNS) and Amazon Simple Queue Service
        (Amazon SQS) together to publish and receive messages using queues.
        """,
        discussion: """
        Supports filtering using a "tone" attribute.
        """
    )

    /// Prompt for an input string. Only non-empty strings are allowed.
    ///
    /// - Parameter prompt: The prompt to display.
    ///
    /// - Returns: The string input by the user.
    func stringRequest(prompt: String) -> String {
        var str: String?
```

```
        while str == nil {
            print(prompt, terminator: "")
            str = readLine()

            if str != nil && str?.count == 0 {
                str = nil
            }
        }

        return str!
    }

    /// Ask a yes/no question.
    ///
    /// - Parameter prompt: A prompt string to print.
    ///
    /// - Returns: `true` if the user answered "Y", otherwise `false`.
    func yesNoRequest(prompt: String) -> Bool {
        while true {
            let answer = stringRequest(prompt: prompt).lowercased()
            if answer == "y" || answer == "n" {
                return answer == "y"
            }
        }
    }

    /// Display a menu of options then request a selection.
    ///
    /// - Parameters:
    ///   - prompt: A prompt string to display before the menu.
    ///   - options: An array of strings giving the menu options.
    ///
    /// - Returns: The index number of the selected option or 0 if no item was
    ///   selected.
    func menuRequest(prompt: String, options: [String]) -> Int {
        let numOptions = options.count

        if numOptions == 0 {
            return 0
        }

        print(prompt)
```

```
        for (index, value) in options.enumerated() {
            print("(\(index)) \(value)")
        }

        repeat {
            print("Enter your selection (0 - \(numOptions-1)): ", terminator: "")
            if let answer = readLine() {
                guard let answer = Int(answer) else {
                    print("Please enter the number matching your selection.")
                    continue
                }

                if answer >= 0 && answer < numOptions {
                    return answer
                } else {
                    print("Please enter the number matching your selection.")
                }
            }
        } while true
    }

    /// Ask the user too press RETURN. Accepts any input but ignores it.
    ///
    /// - Parameter prompt: The text prompt to display.
    func returnRequest(prompt: String) {
        print(prompt, terminator: "")
        _ = readLine()
    }

    var attrValues = [
        "<none>",
        "cheerful",
        "funny",
        "serious",
        "sincere"
    ]

    /// Ask the user to choose one of the attribute values to use as a filter.
    ///
    /// - Parameters:
    ///   - message: A message to display before the menu of values.
    ///   - attrValues: An array of strings giving the values to choose from.
    ///
    /// - Returns: The string corresponding to the selected option.
```

```
    func askForFilter(message: String, attrValues: [String]) -> String? {
        print(message)
        for (index, value) in attrValues.enumerated() {
            print("  [\(index)] \(value)")
        }

        var answer: Int?
        repeat {
            answer = Int(stringRequest(prompt: "Select an value for the 'tone'
attribute or 0 to end: "))
        } while answer == nil || answer! < 0 || answer! > attrValues.count + 1

        if answer == 0 {
            return nil
        }
        return attrValues[answer!]
    }

    /// Prompts the user for filter terms and constructs the attribute
    /// record that specifies them.
    ///
    /// - Returns: A mapping of "FilterPolicy" to a JSON string representing
    ///   the user-defined filter.
    func buildFilterAttributes() -> [String:String] {
        var attr: [String:String] = [:]
        var filterString = ""

        var first = true

        while let ans = askForFilter(message: "Choose a value to apply to the
'tone' attribute.",
                                     attrValues: attrValues) {
            if !first {
                filterString += ","
            }
            first = false

            filterString += "\"\(ans)\""
        }

        let filterJSON = "{ \"tone\": [\(filterString)]}"
        attr["FilterPolicy"] = filterJSON

        return attr
```

```
    }
    /// Create a queue, returning its URL string.
    ///
    /// - Parameters:
    ///   - prompt: A prompt to ask for the queue name.
    ///   - isFIFO: Whether or not to create a FIFO queue.
    ///
    /// - Returns: The URL of the queue.
    func createQueue(prompt: String, sqsClient: SQSClient, isFIFO: Bool) async
throws -> String? {
        repeat {
            var queueName = stringRequest(prompt: prompt)
            var attributes: [String: String] = [:]

            if isFIFO {
                queueName += ".fifo"
                attributes["FifoQueue"] = "true"
            }

            do {
                let output = try await sqsClient.createQueue(
                    input: CreateQueueInput(
                        attributes: attributes,
                        queueName: queueName
                    )
                )
                guard let url = output.queueUrl else {
                    return nil
                }

                return url
            } catch _ as QueueDeletedRecently {
                print("You need to use a different queue name. A queue by that
name was recently deleted.")
                continue
            }
        } while true
    }

    /// Return the ARN of a queue given its URL.
    ///
    /// - Parameter queueUrl: The URL of the queue for which to return the
    ///   ARN.
    ///
```

```
    /// - Returns: The ARN of the specified queue.
    func getQueueARN(sqsClient: SQSClient, queueUrl: String) async throws ->
String? {
        let output = try await sqsClient.getQueueAttributes(
            input: GetQueueAttributesInput(
                attributeNames: [.queuearn],
                queueUrl: queueUrl
            )
        )

        guard let attributes = output.attributes else {
            return nil
        }

        return attributes["QueueArn"]
    }

    /// Applies the needed policy to the specified queue.
    ///
    /// - Parameters:
    ///   - sqsClient: The Amazon SQS client to use.
    ///   - queueUrl: The queue to apply the policy to.
    ///   - queueArn: The ARN of the queue to apply the policy to.
    ///   - topicArn: The topic that should have access via the policy.
    ///
    /// - Throws: Errors from the SQS `SetQueueAttributes` action.
    func setQueuePolicy(sqsClient: SQSClient, queueUrl: String,
                        queueArn: String, topicArn: String) async throws {
        _ = try await sqsClient.setQueueAttributes(
            input: SetQueueAttributesInput(
                attributes: [
                    "Policy":
                        """
                        {
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Principal": {
                                        "Service": "sns.amazonaws.com"
                                    },
                                    "Action": "sqs:SendMessage",
                                    "Resource": "\(queueArn)",
                                    "Condition": {
                                        "ArnEquals": {
```

```
                                            "aws:SourceArn": "\(topicArn)"
                                        }
                                    }
                                }
                            ]
                        }
                        """

                ],
                queueUrl: queueUrl
            )
        )
    }

    /// Receive the available messages on a queue, outputting them to the
    /// screen. Returns a dictionary you pass to DeleteMessageBatch to delete
    /// all the received messages.
    ///
    /// - Parameters:
    ///   - sqsClient: The Amazon SQS client to use.
    ///   - queueUrl: The SQS queue on which to receive messages.
    ///
    /// - Throws: Errors from `SQSClient.receiveMessage()`
    ///
    /// - Returns: An array of SQSClientTypes.DeleteMessageBatchRequestEntry
    ///   items, each describing one received message in the format needed to
    ///   delete it.
    func receiveAndListMessages(sqsClient: SQSClient, queueUrl: String) async
throws
                                ->
[SQSClientTypes.DeleteMessageBatchRequestEntry] {
        let output = try await sqsClient.receiveMessage(
            input: ReceiveMessageInput(
                maxNumberOfMessages: 10,
                queueUrl: queueUrl
            )
        )

        guard let messages = output.messages else {
            print("No messages received.")
            return []
        }

        var deleteList: [SQSClientTypes.DeleteMessageBatchRequestEntry] = []
```

```
        // Print out all the messages that were received, including their
        // attributes, if any.

        for message in messages {
            print("Message ID:     \(message.messageId ?? "<unknown>")")
            print("Receipt handle: \(message.receiptHandle ?? "<unknown>")")
            print("Message JSON:   \(message.body ?? "<body missing>")")

            if message.receiptHandle != nil {
                deleteList.append(
                    SQSClientTypes.DeleteMessageBatchRequestEntry(
                        id: message.messageId,
                        receiptHandle: message.receiptHandle
                    )
                )
            }
        }

        return deleteList
    }

    /// Delete all the messages in the specified list.
    ///
    /// - Parameters:
    ///   - sqsClient: The Amazon SQS client to use.
    ///   - queueUrl: The SQS queue to delete messages from.
    ///   - deleteList: A list of `DeleteMessageBatchRequestEntry` objects
    ///     describing the messages to delete.
    ///
    /// - Throws: Errors from `SQSClient.deleteMessageBatch()`.
    func deleteMessageList(sqsClient: SQSClient, queueUrl: String,
                           deleteList:
[SQSClientTypes.DeleteMessageBatchRequestEntry]) async throws {
        let output = try await sqsClient.deleteMessageBatch(
            input: DeleteMessageBatchInput(entries: deleteList, queueUrl:
queueUrl)
        )

        if let failed = output.failed {
            print("\(failed.count) errors occurred deleting messages from the
queue.")
            for message in failed {
```

```
                print("---> Failed to delete message \(message.id ?? "<unknown
 ID>") with error: \(message.code ?? "<unknown>") (\(message.message ?? "..."))")
            }
        }
    }

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let rowOfStars = String(repeating: "*", count: 75)

        print("""
              \(rowOfStars)
              Welcome to the cross-service messaging with topics and queues
 example.
              In this workflow, you'll create an SNS topic, then create two SQS
              queues which will be subscribed to that topic.

              You can specify several options for configuring the topic, as well
 as
              the queue subscriptions. You can then post messages to the topic
 and
              receive the results on the queues.
              \(rowOfStars)\n
              """
        )

        // 0. Create SNS and SQS clients.

        let snsConfig = try await SNSClient.SNSClientConfiguration(region:
 region)
        let snsClient = SNSClient(config: snsConfig)

        let sqsConfig = try await SQSClient.SQSClientConfiguration(region:
 region)
        let sqsClient = SQSClient(config: sqsConfig)

        // 1. Ask the user whether to create a FIFO topic. If so, ask whether
        //    to use content-based deduplication instead of requiring a
        //    deduplication ID.

        let isFIFO = yesNoRequest(prompt: "Do you want to create a FIFO topic (Y/
N)? ")
        var isContentBasedDeduplication = false
```

```
        if isFIFO {
            print("""
                  \(rowOfStars)
                  Because you've chosen to create a FIFO topic, deduplication is
                  supported.

                  Deduplication IDs are either set in the message or are
 automatically
                  generated from the content using a hash function.

                  If a message is successfully published to an SNS FIFO topic,
 any
                  message published and found to have the same deduplication ID
                  (within a five-minute deduplication interval), is accepted but
                  not delivered.

                  For more information about deduplication, see:
                  https://docs.aws.amazon.com/sns/latest/dg/fifo-message-
dedup.html.
                  """
            )

            isContentBasedDeduplication = yesNoRequest(
                prompt: "Use content-based deduplication instead of entering a
 deduplication ID (Y/N)? ")
            print(rowOfStars)
        }

        var topicName = stringRequest(prompt: "Enter the name of the topic to
 create: ")

        // 2. Create the topic. Append ".fifo" to the name if FIFO was
        //    requested, and set the "FifoTopic" attribute to "true" if so as
        //    well. Set the "ContentBasedDeduplication" attribute to "true" if
        //    content-based deduplication was requested.

        if isFIFO {
            topicName += ".fifo"
        }

        print("Topic name: \(topicName)")

        var attributes = [
            "FifoTopic": (isFIFO ? "true" : "false")
```

```
        ]

        // If it's a FIFO topic with content-based deduplication, set the
        // "ContentBasedDeduplication" attribute.

        if isContentBasedDeduplication {
            attributes["ContentBasedDeduplication"] = "true"
        }

        // Create the topic and retrieve the ARN.

        let output = try await snsClient.createTopic(
            input: CreateTopicInput(
                attributes: attributes,
                name: topicName
            )
        )

        guard let topicArn = output.topicArn else {
            print("No topic ARN returned!")
            return
        }

        print("""
              Topic '\(topicName) has been created with the
              topic ARN \(topicArn)."
              """
        )

        print(rowOfStars)

        // 3. Create an SQS queue. Append ".fifo" to the name if one of the
        //    FIFO topic configurations was chosen, and set "FifoQueue" to
        //    "true" if the topic is FIFO.

        print("""
              Next, you will create two SQS queues that will be subscribed
              to the topic you just created.\n
              """
        )

        let q1Url = try await createQueue(prompt: "Enter the name of the first
queue: ",
                                          sqsClient: sqsClient, isFIFO: isFIFO)
```

```
        guard let q1Url else {
            print("Unable to create queue 1!")
            return
        }

        // 4. Get the SQS queue's ARN attribute using `GetQueueAttributes`.

        let q1Arn = try await getQueueARN(sqsClient: sqsClient, queueUrl: q1Url)

        guard let q1Arn else {
            print("Unable to get ARN of queue 1!")
            return
        }
        print("Got queue 1 ARN: \(q1Arn)")

        // 5. Attach an AWS IAM policy to the queue using
        //    `SetQueueAttributes`.

        try await setQueuePolicy(sqsClient: sqsClient, queueUrl: q1Url,
                                 queueArn: q1Arn, topicArn: topicArn)

        // 6. Subscribe the SQS queue to the SNS topic. Set the topic ARN in
        //    the request. Set the protocol to "sqs". Set the queue ARN to the
        //    ARN just received in step 5. For FIFO topics, give the option to
        //    apply a filter. A filter allows only matching messages to enter
        //    the queue.

        var q1Attributes: [String:String]? = nil

        if isFIFO {
            print(
                """

                If you add a filter to this subscription, then only the filtered
 messages will
                be received in the queue. For information about message
 filtering, see
                https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html
                For this example, you can filter messages by a 'tone' attribute.

                """
            )
```

```
            let subPrompt = """
                Would you like to filter messages for the first queue's
subscription to the
                topic \(topicName) (Y/N)?
                """
            if (yesNoRequest(prompt: subPrompt)) {
                q1Attributes = buildFilterAttributes()
            }
        }

        let sub1Output = try await snsClient.subscribe(
            input: SubscribeInput(
                attributes: q1Attributes,
                endpoint: q1Arn,
                protocol: "sqs",
                topicArn: topicArn
            )
        )

        guard let q1SubscriptionArn = sub1Output.subscriptionArn else {
            print("Invalid subscription ARN returned for queue 1!")
            return
        }

        // 7. Repeat steps 3-6 for the second queue.

        let q2Url = try await createQueue(prompt: "Enter the name of the second
queue: ",
                                  sqsClient: sqsClient, isFIFO: isFIFO)

        guard let q2Url else {
            print("Unable to create queue 2!")
            return
        }

        let q2Arn = try await getQueueARN(sqsClient: sqsClient, queueUrl: q2Url)

        guard let q2Arn else {
            print("Unable to get ARN of queue 2!")
            return
        }
        print("Got queue 2 ARN: \(q2Arn)")

        try await setQueuePolicy(sqsClient: sqsClient, queueUrl: q2Url,
```

```
                                queueArn: q2Arn, topicArn: topicArn)

        var q2Attributes: [String:String]? = nil

        if isFIFO {
            let subPrompt = """
                Would you like to filter messages for the second queue's
subscription to the
                topic \(topicName) (Y/N)?
                """
            if (yesNoRequest(prompt: subPrompt)) {
                q2Attributes = buildFilterAttributes()
            }
        }

        let sub2Output = try await snsClient.subscribe(
            input: SubscribeInput(
                attributes: q2Attributes,
                endpoint: q2Arn,
                protocol: "sqs",
                topicArn: topicArn
            )
        )

        guard let q2SubscriptionArn = sub2Output.subscriptionArn else {
            print("Invalid subscription ARN returned for queue 1!")
            return
        }

        // 8. Let the user publish messages to the topic, asking for a message
        //    body for each message. Handle the types of topic correctly (SEE
        //    MVP INFORMATION AND FIX THESE COMMENTS!!!

        print("\n\(rowOfStars)\n")

        var first = true

        repeat {
            var publishInput = PublishInput(
                topicArn: topicArn
            )

            publishInput.message = stringRequest(prompt: "Enter message text to
publish: ")
```

```
            // If using a FIFO topic, a message group ID must be set on the
            // message.

            if isFIFO {
                if first {
                    print("""
                        Because you're using a FIFO topic, you must set a message
                        group ID. All messages within the same group will be
                        received in the same order in which they were published.
\n
                        """
                    )
                }
                publishInput.messageGroupId = stringRequest(prompt: "Enter a
 message group ID for this message: ")

                if !isContentBasedDeduplication {
                    if first {
                        print("""
                              Because you're not using content-based
 deduplication, you
                              must enter a deduplication ID. If other messages
 with the
                              same deduplication ID are published within the same
                              deduplication interval, they will not be delivered.
                              """
                        )
                    }
                    publishInput.messageDeduplicationId = stringRequest(prompt:
 "Enter a deduplication ID for this message: ")
                }
            }

            // Allow the user to add a value for the "tone" attribute if they
            // wish to do so.

            var messageAttributes: [String:SNSClientTypes.MessageAttributeValue]
 = [:]
            let attrValSelection = menuRequest(prompt: "Choose a tone to apply to
 this message.", options: attrValues)

            if attrValSelection != 0 {
```

```
                let val = SNSClientTypes.MessageAttributeValue(dataType:
"String", stringValue: attrValues[attrValSelection])
                messageAttributes["tone"] = val
            }

            publishInput.messageAttributes = messageAttributes

            // Publish the message and display its ID.

            let publishOutput = try await snsClient.publish(input: publishInput)

            guard let messageID = publishOutput.messageId else {
                print("Unable to get the published message's ID!")
                return
            }

            print("Message published with ID \(messageID).")
            first = false

            // 9. Repeat step 8 until the user says they don't want to post
            //    another.

        } while (yesNoRequest(prompt: "Post another message (Y/N)? "))

        // 10. Display a list of the messages in each queue by using
        //     `ReceiveMessage`. Show at least the body and the attributes.

        print(rowOfStars)
        print("Contents of queue 1:")
        let q1DeleteList = try await receiveAndListMessages(sqsClient: sqsClient,
queueUrl: q1Url)
        print("\n\nContents of queue 2:")
        let q2DeleteList = try await receiveAndListMessages(sqsClient: sqsClient,
queueUrl: q2Url)
        print(rowOfStars)

        returnRequest(prompt: "\nPress return to clean up: ")

        // 11. Delete the received messages using `DeleteMessageBatch`.

        print("Deleting the messages from queue 1...")
        try await deleteMessageList(sqsClient: sqsClient, queueUrl: q1Url,
deleteList: q1DeleteList)
        print("\nDeleting the messages from queue 2...")
```

```
        try await deleteMessageList(sqsClient: sqsClient, queueUrl: q2Url,
 deleteList: q2DeleteList)

        // 12. Unsubscribe and delete both queues.

        print("\nUnsubscribing from queue 1...")
        _ = try await snsClient.unsubscribe(
            input: UnsubscribeInput(subscriptionArn: q1SubscriptionArn)
        )

        print("Unsubscribing from queue 2...")
        _ = try await snsClient.unsubscribe(
            input: UnsubscribeInput(subscriptionArn: q2SubscriptionArn)
        )

        print("Deleting queue 1...")
        _ = try await sqsClient.deleteQueue(
            input: DeleteQueueInput(queueUrl: q1Url)
        )

        print("Deleting queue 2...")
        _ = try await sqsClient.deleteQueue(
            input: DeleteQueueInput(queueUrl: q2Url)
        )

        // 13. Delete the topic.

        print("Deleting the SNS topic...")
        _ = try await snsClient.deleteTopic(
            input: DeleteTopicInput(topicArn: topicArn)
        )
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
```

```
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Swift API Reference》中的以下主题。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 发布
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包通过 Amazon SQS 发送和接收批量消息 AWS

以下代码示例演示了如何：

- 创建 Amazon SQS 队列。
- 将成批消息发送到队列。
- 从队列中接收成批消息。
- 从队列中删除成批消息。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

如以下示例所示，可以使用 Amazon SQS，通过将两种不同的方法与 AWS SDK for Java 2.x结合使用来处理批量消息操作：

SendRecvBatch.java 使用显式批处理操作。您可以手动创建消息批次并直接调用 `sendMessageBatch()` 和 `deleteMessageBatch()`。还可以处理批量响应，包括任何失败的消息。这种方法使您可以完全控制批量大小和错误处理。但是，它需要更多的代码来管理批处理逻辑。

SimpleProducerConsumer.java 使用高级`SqsAsyncBatchManager`库进行自动请求批处理。您使用与标准客户端相同的方法签名进行单独的 `sendMessage()` 和 `deleteMessage()` 调用。SDK 会自动缓冲这些调用并将其作为批量操作发送。这种方法只需要最少的代码更改，同时提供批处理性能优势。

当您需要对批量组合和错误处理进行精细控制时，请使用显式批处理。如果您想以最少的代码更改来优化性能，请使用自动批处理。

SendRecvBatch.java-对消息使用显式批处理操作。

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.BatchResultErrorEntry;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchResultEntry;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.MessageAttributeValue;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequest;
```

```
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchResultEntry;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

/**
 * This code demonstrates basic message operations in Amazon Simple Queue Service
 (Amazon SQS).
 */

public class SendRecvBatch {
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SendRecvBatch.class);
    private static final SqsClient sqsClient = SqsClient.create();


    public static void main(String[] args) {
        usageDemo();
    }
    /**
     * Send a batch of messages in a single request to an SQS queue.
     * This request may return overall success even when some messages were not
 sent.
```

```
    * The caller must inspect the Successful and Failed lists in the response
and
    * resend any failed messages.
    *
    * @param queueUrl  The URL of the queue to receive the messages.
    * @param messages  The messages to send to the queue. Each message contains
a body and attributes.
    * @return The response from SQS that contains the list of successful and
failed messages.
    */
   public static SendMessageBatchResponse sendMessages(
           String queueUrl, List<MessageEntry> messages) {

       try {
           List<SendMessageBatchRequestEntry> entries = new ArrayList<>();

           for (int i = 0; i < messages.size(); i++) {
               MessageEntry message = messages.get(i);
               entries.add(SendMessageBatchRequestEntry.builder()
                       .id(String.valueOf(i))
                       .messageBody(message.getBody())
                       .messageAttributes(message.getAttributes())
                       .build());
           }

           SendMessageBatchRequest sendBatchRequest =
SendMessageBatchRequest.builder()
                   .queueUrl(queueUrl)
                   .entries(entries)
                   .build();

           SendMessageBatchResponse response =
sqsClient.sendMessageBatch(sendBatchRequest);

           if (!response.successful().isEmpty()) {
               for (SendMessageBatchResultEntry resultEntry :
response.successful()) {
                   LOGGER.info("Message sent: {}: {}", resultEntry.messageId(),

messages.get(Integer.parseInt(resultEntry.id())).getBody());
               }
           }

           if (!response.failed().isEmpty()) {
```

```
                for (BatchResultErrorEntry errorEntry : response.failed()) {
                    LOGGER.warn("Failed to send: {}: {}", errorEntry.id(),

messages.get(Integer.parseInt(errorEntry.id())).getBody());
                }
            }

            return response;

        } catch (SqsException e) {
            LOGGER.error("Send messages failed to queue: {}", queueUrl, e);
            throw e;
        }
    }

    /**
     * Receive a batch of messages in a single request from an SQS queue.
     *
     * @param queueUrl   The URL of the queue from which to receive messages.
     * @param maxNumber  The maximum number of messages to receive (capped at 10
by SQS).
     *                   The actual number of messages received might be less.
     * @param waitTime   The maximum time to wait (in seconds) before returning.
When
     *                   this number is greater than zero, long polling is used.
This
     *                   can result in reduced costs and fewer false empty
responses.
     * @return The list of Message objects received. These each contain the body
     *         of the message and metadata and custom attributes.
     */
    public static List<Message> receiveMessages(String queueUrl, int maxNumber,
int waitTime) {
        try {
            ReceiveMessageRequest receiveRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .maxNumberOfMessages(maxNumber)
                    .waitTimeSeconds(waitTime)
                    .messageAttributeNames("All")
                    .build();

            List<Message> messages =
sqsClient.receiveMessage(receiveRequest).messages();
```

```
            for (Message message : messages) {
                LOGGER.info("Received message: {}: {}", message.messageId(),
message.body());
            }

            return messages;

        } catch (SqsException e) {
            LOGGER.error("Couldn't receive messages from queue: {}", queueUrl,
e);
            throw e;
        }
    }

    /**
     * Delete a batch of messages from a queue in a single request.
     *
     * @param queueUrl  The URL of the queue from which to delete the messages.
     * @param messages  The list of messages to delete.
     * @return The response from SQS that contains the list of successful and
failed
     *         message deletions.
     */
    public static DeleteMessageBatchResponse deleteMessages(String queueUrl,
List<Message> messages) {
        try {
            List<DeleteMessageBatchRequestEntry> entries = new ArrayList<>();

            for (int i = 0; i < messages.size(); i++) {
                entries.add(DeleteMessageBatchRequestEntry.builder()
                        .id(String.valueOf(i))
                        .receiptHandle(messages.get(i).receiptHandle())
                        .build());
            }

            DeleteMessageBatchRequest deleteRequest =
DeleteMessageBatchRequest.builder()
                    .queueUrl(queueUrl)
                    .entries(entries)
                    .build();

            DeleteMessageBatchResponse response =
sqsClient.deleteMessageBatch(deleteRequest);
```

```
            if (!response.successful().isEmpty()) {
                for (DeleteMessageBatchResultEntry resultEntry :
response.successful()) {
                    LOGGER.info("Deleted {}",
messages.get(Integer.parseInt(resultEntry.id())).receiptHandle());
                }
            }

            if (!response.failed().isEmpty()) {
                for (BatchResultErrorEntry errorEntry : response.failed()) {
                    LOGGER.warn("Could not delete {}",
messages.get(Integer.parseInt(errorEntry.id())).receiptHandle());
                }
            }

            return response;

        } catch (SqsException e) {
            LOGGER.error("Couldn't delete messages from queue {}", queueUrl, e);
            throw e;
        }
    }

    /**
     * Helper class to represent a message with body and attributes.
     */
    public static class MessageEntry {
        private final String body;
        private final Map<String, MessageAttributeValue> attributes;

        public MessageEntry(String body, Map<String, MessageAttributeValue>
attributes) {
            this.body = body;
            this.attributes = attributes != null ? attributes : new HashMap<>();
        }

        public String getBody() {
            return body;
        }

        public Map<String, MessageAttributeValue> getAttributes() {
            return attributes;
        }
```

```
    }

    /**
     * Shows how to:
     * * Read the lines from a file and send the lines in
     *   batches of 10 as messages to a queue.
     * * Receive the messages in batches until the queue is empty.
     * * Reassemble the lines of the file and verify they match the original
 file.
     */
    public static void usageDemo() {
        LOGGER.info("-".repeat(88));
        LOGGER.info("Welcome to the Amazon Simple Queue Service (Amazon SQS)
 demo!");
        LOGGER.info("-".repeat(88));

        String queueUrl = null;
        try {
            // Create a queue for the demo.
            String queueName = "sqs-usage-demo-message-wrapper-" +
 System.currentTimeMillis();
            CreateQueueRequest createRequest = CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();
            queueUrl = sqsClient.createQueue(createRequest).queueUrl();
            LOGGER.info("Created queue: {}", queueUrl);

            try (InputStream inputStream =
 SendRecvBatch.class.getResourceAsStream("/log4j2.xml");
                 BufferedReader reader = new BufferedReader(new
 InputStreamReader(inputStream))) {

                List<String> lines = reader.lines().toList();

                // Send file lines in batches.
                int batchSize = 10;
                LOGGER.info("Sending file lines in batches of {} as messages.",
 batchSize);

                for (int i = 0; i < lines.size(); i += batchSize) {
                    List<MessageEntry> messageBatch = new ArrayList<>();

                    for (int j = i; j < Math.min(i + batchSize, lines.size()); j+
+) {
```

```
                        String line = lines.get(j);
                        if (line == null || line.trim().isEmpty()) {
                            continue; // Skip empty lines.
                        }

                        Map<String, MessageAttributeValue> attributes = new
HashMap<>();
                        attributes.put("line", MessageAttributeValue.builder()
                                .dataType("String")
                                .stringValue(String.valueOf(j))
                                .build());

                        messageBatch.add(new MessageEntry(lines.get(j),
attributes));
                    }

                    sendMessages(queueUrl, messageBatch);
                    System.out.print(".");
                    System.out.flush();
                }

                LOGGER.info("\nDone. Sent {} messages.", lines.size());

                // Receive and process messages.
                LOGGER.info("Receiving, handling, and deleting messages in
batches of {}.", batchSize);
                String[] receivedLines = new String[lines.size()];
                boolean moreMessages = true;

                while (moreMessages) {
                    List<Message> receivedMessages = receiveMessages(queueUrl,
batchSize, 5);

                    for (Message message : receivedMessages) {
                        int lineNumber =
Integer.parseInt(message.messageAttributes().get("line").stringValue());
                        receivedLines[lineNumber] = message.body();
                    }

                    if (!receivedMessages.isEmpty()) {
                        deleteMessages(queueUrl, receivedMessages);
                    } else {
                        moreMessages = false;
                    }
```

```
                }

                LOGGER.info("\nDone.");

                // Verify that all lines were received correctly.
                boolean allLinesMatch = true;
                for (int i = 0; i < lines.size(); i++) {
                    String originalLine = lines.get(i);
                    String receivedLine = receivedLines[i] == null ? "" :
receivedLines[i];

                    if (!originalLine.equals(receivedLine)) {
                        allLinesMatch = false;
                        break;
                    }
                }

                if (allLinesMatch) {
                    LOGGER.info("Successfully reassembled all file lines!");
                } else {
                    LOGGER.info("Uh oh, some lines were missed!");
                }
            }
        } catch (SqsException e) {
            LOGGER.error("SQS operation failed", e);
        } catch (RuntimeException | IOException e) {
            LOGGER.error("Unexpected runtime error during demo", e);
        } finally {
            // Clean up by deleting the queue if it was created.
            if (queueUrl != null) {
                try {
                    DeleteQueueRequest deleteQueueRequest =
DeleteQueueRequest.builder()
                            .queueUrl(queueUrl)
                            .build();
                    sqsClient.deleteQueue(deleteQueueRequest);
                    LOGGER.info("Deleted queue: {}", queueUrl);
                } catch (SqsException e) {
                    LOGGER.error("Failed to delete queue: {}", queueUrl, e);
                }
            }
        }

        LOGGER.info("Thanks for watching!");
```

```
        LOGGER.info("-".repeat(88));
    }
}
```

SimpleProducerConsumer.java-使用自动批处理消息。

```
package com.example.sqs;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsAsyncClient;
import software.amazon.awssdk.services.sqs.batchmanager.SqsAsyncBatchManager;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageResponse;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageResponse;
import software.amazon.awssdk.core.exception.SdkException;

import java.math.BigInteger;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Demonstrates the AWS SDK for Java 2.x Automatic Request Batching API for
 Amazon SQS.
 *
 * This example showcases the high-level SqsAsyncBatchManager library that
 provides
 * efficient batching and buffering for SQS operations. The batch manager offers
 * methods that directly mirror SqsAsyncClient methods—sendMessage,
 changeMessageVisibility,
 * deleteMessage, and receiveMessage—making it a drop-in replacement with minimal
 code changes.
```

```
 *
 * Key features of the SqsAsyncBatchManager:
 * - Automatic batching: The SDK automatically buffers individual requests and
 sends them
 *   as batches when maxBatchSize (default: 10) or sendRequestFrequency (default:
 200ms)
 *   thresholds are reached
 * - Familiar API: Method signatures match SqsAsyncClient exactly, requiring no
 learning curve
 * - Background optimization: The batch manager maintains internal buffers and
 handles
 *   batching logic transparently
 * - Asynchronous operations: All methods return CompletableFuture for non-
blocking execution
 *
 * Performance benefits demonstrated:
 * - Reduced API calls: Multiple individual requests are consolidated into single
 batch operations
 * - Lower costs: Fewer API calls result in reduced SQS charges
 * - Higher throughput: Batch operations process more messages per second
 * - Efficient resource utilization: Fewer network round trips and better
 connection reuse
 *
 * This example compares:
 * 1. Single-message operations using SqsAsyncClient directly
 * 2. Batch operations using SqsAsyncBatchManager with identical method calls
 *
 * Usage patterns:
 * - Set batch size to 1 to use SqsAsyncClient for baseline performance
 measurement
 * - Set batch size > 1 to use SqsAsyncBatchManager for optimized batch
 processing
 * - Monitor real-time throughput metrics to observe performance improvements
 *
 * Prerequisites:
 * - AWS SDK for Java 2.x version 2.28.0 or later
 * - An existing SQS queue
 * - Valid AWS credentials configured
 *
 * The program displays real-time metrics showing the dramatic performance
 difference
 * between individual operations and automatic batching.
 */
public class SimpleProducerConsumer {
```

```
   // The maximum runtime of the program.
   private final static int MAX_RUNTIME_MINUTES = 60;
   private final static Logger log =
LoggerFactory.getLogger(SimpleProducerConsumer.class);

   /**
    * Runs the SQS batching demonstration with user-configured parameters.
    *
    * Prompts for queue name, thread counts, batch size, message size, and
runtime.
    * Creates producer and consumer threads to demonstrate batching performance.
    *
    * @param args command line arguments (not used)
    * @throws InterruptedException if thread operations are interrupted
    */
   public static void main(String[] args) throws InterruptedException {

       final Scanner input = new Scanner(System.in);

       System.out.print("Enter the queue name: ");
       final String queueName = input.nextLine();

       System.out.print("Enter the number of producers: ");
       final int producerCount = input.nextInt();

       System.out.print("Enter the number of consumers: ");
       final int consumerCount = input.nextInt();

       System.out.print("Enter the number of messages per batch: ");
       final int batchSize = input.nextInt();

       System.out.print("Enter the message size in bytes: ");
       final int messageSizeByte = input.nextInt();

       System.out.print("Enter the run time in minutes: ");
       final int runTimeMinutes = input.nextInt();

       // Create SQS async client and batch manager for all operations.
       // The SqsAsyncBatchManager is created from the SqsAsyncClient using the
       // batchManager() factory method, which provides default batching
configuration.
       // This high-level library automatically handles request buffering and
batching
```

```
        // while maintaining the same method signatures as SqsAsyncClient.
        final SqsAsyncClient sqsAsyncClient = SqsAsyncClient.create();
        final SqsAsyncBatchManager batchManager = sqsAsyncClient.batchManager();

        final String queueUrl =
sqsAsyncClient.getQueueUrl(GetQueueUrlRequest.builder()
                .queueName(queueName)
                .build()).join().queueUrl();

        // The flag used to stop producer, consumer, and monitor threads.
        final AtomicBoolean stop = new AtomicBoolean(false);

        // Start the producers.
        final AtomicInteger producedCount = new AtomicInteger();
        final Thread[] producers = new Thread[producerCount];
        for (int i = 0; i < producerCount; i++) {
            if (batchSize == 1) {
                producers[i] = new Producer(sqsAsyncClient, queueUrl,
messageSizeByte,
                        producedCount, stop);
            } else {
                producers[i] = new BatchProducer(batchManager, queueUrl,
batchSize,
                        messageSizeByte, producedCount, stop);
            }
            producers[i].start();
        }

        // Start the consumers.
        final AtomicInteger consumedCount = new AtomicInteger();
        final Thread[] consumers = new Thread[consumerCount];
        for (int i = 0; i < consumerCount; i++) {
            if (batchSize == 1) {
                consumers[i] = new Consumer(sqsAsyncClient, queueUrl,
consumedCount, stop);
            } else {
                consumers[i] = new BatchConsumer(batchManager, queueUrl,
batchSize,
                        consumedCount, stop);
            }
            consumers[i].start();
        }

        // Start the monitor thread.
```

```
        final Thread monitor = new Monitor(producedCount, consumedCount, stop);
        monitor.start();

        // Wait for the specified amount of time then stop.
        Thread.sleep(TimeUnit.MINUTES.toMillis(Math.min(runTimeMinutes,
                MAX_RUNTIME_MINUTES)));
        stop.set(true);

        // Join all threads.
        for (int i = 0; i < producerCount; i++) {
            producers[i].join();
        }

        for (int i = 0; i < consumerCount; i++) {
            consumers[i].join();
        }

        monitor.interrupt();
        monitor.join();

        // Close resources
        batchManager.close();
        sqsAsyncClient.close();
    }

    /**
     * Creates a random string of approximately the specified size in bytes.
     *
     * @param sizeByte the target size in bytes for the generated string
     * @return a random string encoded in base-32
     */
    private static String makeRandomString(int sizeByte) {
        final byte[] bs = new byte[(int) Math.ceil(sizeByte * 5 / 8)];
        new Random().nextBytes(bs);
        bs[0] = (byte) ((bs[0] | 64) & 127);
        return new BigInteger(bs).toString(32);
    }

    /**
     * Sends messages individually using SqsAsyncClient for baseline performance
measurement.
     *
     * This producer demonstrates traditional single-message operations without
batching.
```

```
    * Each sendMessage() call results in a separate API request to SQS,
providing
    * a performance baseline for comparison with the batch operations.
    *
    * The sendMessage() method signature is identical to
SqsAsyncBatchManager.sendMessage(),
    * showing how the high-level batching library maintains API compatibility
while
    * adding automatic optimization behind the scenes.
    */
   private static class Producer extends Thread {
       final SqsAsyncClient sqsAsyncClient;
       final String queueUrl;
       final AtomicInteger producedCount;
       final AtomicBoolean stop;
       final String theMessage;

       /**
        * Creates a producer thread for single-message operations.
        *
        * @param sqsAsyncClient the SQS client for sending messages
        * @param queueUrl the URL of the target queue
        * @param messageSizeByte the size of messages to generate
        * @param producedCount shared counter for tracking sent messages
        * @param stop shared flag to signal thread termination
        */
       Producer(SqsAsyncClient sqsAsyncClient, String queueUrl, int
messageSizeByte,
                AtomicInteger producedCount, AtomicBoolean stop) {
           this.sqsAsyncClient = sqsAsyncClient;
           this.queueUrl = queueUrl;
           this.producedCount = producedCount;
           this.stop = stop;
           this.theMessage = makeRandomString(messageSizeByte);
       }

       /**
        * Continuously sends messages until the stop flag is set.
        *
        * Uses SqsAsyncClient.sendMessage() directly, resulting in one API call
per message.
        * This approach provides baseline performance metrics for comparison
with batching.
```

```
         * Each call blocks until the individual message is sent, demonstrating
traditional
         * one-request-per-operation behavior.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    sqsAsyncClient.sendMessage(SendMessageRequest.builder()
                            .queueUrl(queueUrl)
                            .messageBody(theMessage)
                            .build()).join();
                    producedCount.incrementAndGet();
                }
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                // Handle both SdkException and CompletionException from async
operations.
                // If this unlikely condition occurs, stop.
                log.error("Producer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * Sends messages using SqsAsyncBatchManager for automatic request batching
and optimization.
     *
     * This producer demonstrates the AWS SDK for Java 2.x high-level batching
library.
     * The SqsAsyncBatchManager automatically buffers individual sendMessage()
calls and
     * sends them as batches when thresholds are reached:
     * - maxBatchSize: Maximum 10 messages per batch (default)
     * - sendRequestFrequency: 200ms timeout before sending partial batches
(default)
     *
     * Key advantages of the batching approach:
     * - Identical API: batchManager.sendMessage() has the same signature as
sqsAsyncClient.sendMessage()
     * - Automatic optimization: No code changes needed to benefit from batching
     * - Transparent buffering: The SDK handles batching logic internally
     * - Reduced API calls: Multiple messages sent in single batch requests
     * - Lower costs: Fewer API calls result in reduced SQS charges
```

```
     * - Higher throughput: Batch operations process significantly more messages
per second
     */
    private static class BatchProducer extends Thread {
        final SqsAsyncBatchManager batchManager;
        final String queueUrl;
        final int batchSize;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        /**
         * Creates a producer thread for batch operations.
         *
         * @param batchManager the batch manager for efficient message sending
         * @param queueUrl the URL of the target queue
         * @param batchSize the number of messages to send per batch
         * @param messageSizeByte the size of messages to generate
         * @param producedCount shared counter for tracking sent messages
         * @param stop shared flag to signal thread termination
         */
        BatchProducer(SqsAsyncBatchManager batchManager, String queueUrl, int
batchSize,
                      int messageSizeByte, AtomicInteger producedCount,
                      AtomicBoolean stop) {
            this.batchManager = batchManager;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /**
         * Continuously sends batches of messages using the high-level batching
library.
         *
         * Notice how batchManager.sendMessage() uses the exact same method
signature
         * and request builder pattern as SqsAsyncClient.sendMessage(). This
demonstrates
         * the drop-in replacement capability of the SqsAsyncBatchManager.
         *
         * The SDK automatically:
```

```
         * - Buffers individual sendMessage() calls internally
         * - Groups them into batch requests when thresholds are met
         * - Sends SendMessageBatchRequest operations to SQS
         * - Returns individual CompletableFuture responses for each message
         *
         * This transparent batching provides significant performance
improvements
         * without requiring changes to application logic or error handling
patterns.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    // Send multiple messages using the high-level batch manager.
                    // Each batchManager.sendMessage() call uses identical syntax
to
                    // sqsAsyncClient.sendMessage(), demonstrating API
compatibility.
                    // The SDK automatically buffers these calls and sends them
as
                    // batch operations when maxBatchSize (10) or
sendRequestFrequency (200ms)
                    // thresholds are reached, significantly improving
throughput.
                    for (int i = 0; i < batchSize; i++) {
                        CompletableFuture<SendMessageResponse> future =
batchManager.sendMessage(
                                SendMessageRequest.builder()
                                        .queueUrl(queueUrl)
                                        .messageBody(theMessage)
                                        .build());

                        // Handle the response asynchronously
                        future.whenComplete((response, throwable) -> {
                            if (throwable == null) {
                                producedCount.incrementAndGet();
                            } else if (!(throwable instanceof
java.util.concurrent.CancellationException) &&
                                    !(throwable.getMessage() != null &&
throwable.getMessage().contains("executor not accepting a task"))) {
                                log.error("BatchProducer: Failed to send
message", throwable);
                            }
```

```
                            // Ignore CancellationException and executor shutdown
errors - expected during shutdown
                        });
                    }

                    // Small delay to allow batching to occur
                    Thread.sleep(10);
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                log.error("BatchProducer interrupted: " + e.getMessage());
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                log.error("BatchProducer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * Receives and deletes messages individually using SqsAsyncClient for
baseline measurement.
     *
     * This consumer demonstrates traditional single-message operations without
batching.
     * Each receiveMessage() and deleteMessage() call results in separate API
requests,
     * providing a performance baseline for comparison with batch operations.
     *
     * The method signatures are identical to SqsAsyncBatchManager methods:
     * - receiveMessage() matches batchManager.receiveMessage()
     * - deleteMessage() matches batchManager.deleteMessage()
     *
     * This API consistency allows easy migration to the high-level batching
library.
     */
    private static class Consumer extends Thread {
        final SqsAsyncClient sqsAsyncClient;
        final String queueUrl;
        final AtomicInteger consumedCount;
        final AtomicBoolean stop;

        /**
         * Creates a consumer thread for single-message operations.
         *
```

```
         * @param sqsAsyncClient the SQS client for receiving messages
         * @param queueUrl the URL of the source queue
         * @param consumedCount shared counter for tracking processed messages
         * @param stop shared flag to signal thread termination
         */
        Consumer(SqsAsyncClient sqsAsyncClient, String queueUrl, AtomicInteger
 consumedCount,
                 AtomicBoolean stop) {
            this.sqsAsyncClient = sqsAsyncClient;
            this.queueUrl = queueUrl;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /**
         * Continuously receives and deletes messages using traditional single-
request operations.
         *
         * Uses SqsAsyncClient methods directly:
         * - receiveMessage(): One API call per receive operation
         * - deleteMessage(): One API call per delete operation
         *
         * This approach demonstrates the baseline performance without batching
 optimization.
         * Compare these method calls with the identical signatures used in
 BatchConsumer
         * to see how the high-level batching library maintains API
 compatibility.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    try {
                        final ReceiveMessageResponse result =
 sqsAsyncClient.receiveMessage(
                                ReceiveMessageRequest.builder()
                                        .queueUrl(queueUrl)
                                        .build()).join();

                        if (!result.messages().isEmpty()) {
                            final Message m = result.messages().get(0);
                            // Note: deleteMessage() signature identical to
 batchManager.deleteMessage()
```

```
sqsAsyncClient.deleteMessage(DeleteMessageRequest.builder()
                                    .queueUrl(queueUrl)
                                    .receiptHandle(m.receiptHandle())
                                    .build()).join();
                            consumedCount.incrementAndGet();
                        }
                    } catch (SdkException |
java.util.concurrent.CompletionException e) {
                        log.error(e.getMessage());
                    }
                }
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                // Handle both SdkException and CompletionException from async
operations.
                // If this unlikely condition occurs, stop.
                log.error("Consumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * Receives and deletes messages using SqsAsyncBatchManager for automatic
optimization.
     *
     * This consumer demonstrates the AWS SDK for Java 2.x high-level batching
library
     * for message consumption. The SqsAsyncBatchManager provides two key
optimizations:
     *
     * 1. Receive optimization: Maintains an internal buffer of messages fetched
in the
     *    background, so receiveMessage() calls return immediately from the
buffer
     * 2. Delete batching: Automatically buffers deleteMessage() calls and sends
them
     *    as DeleteMessageBatchRequest operations when thresholds are reached
     *
     * Key features:
     * - Identical API: receiveMessage() and deleteMessage() have the same
signatures
     *   as SqsAsyncClient methods, making this a true drop-in replacement
```

```
     * - Background fetching: The batch manager continuously fetches messages to
keep
     *   the internal buffer populated, reducing receive latency
     * - Automatic delete batching: Individual deleteMessage() calls are buffered
and
     *   sent as batch operations (up to 10 per batch, 200ms frequency)
     * - Transparent optimization: No application logic changes needed to benefit
     *
     * Performance benefits:
     * - Reduced API calls through automatic batching of delete operations
     * - Lower latency for receives due to background message buffering
     * - Higher overall throughput with fewer network round trips
     */
    private static class BatchConsumer extends Thread {
        final SqsAsyncBatchManager batchManager;
        final String queueUrl;
        final int batchSize;
        final AtomicInteger consumedCount;
        final AtomicBoolean stop;

        /**
         * Creates a consumer thread for batch operations.
         *
         * @param batchManager the batch manager for efficient message processing
         * @param queueUrl the URL of the source queue
         * @param batchSize the maximum number of messages to receive per batch
         * @param consumedCount shared counter for tracking processed messages
         * @param stop shared flag to signal thread termination
         */
        BatchConsumer(SqsAsyncBatchManager batchManager, String queueUrl, int
batchSize,
                      AtomicInteger consumedCount, AtomicBoolean stop) {
            this.batchManager = batchManager;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /**
         * Continuously receives and deletes messages using the high-level
batching library.
         *
```

```
        * Demonstrates the key advantage of SqsAsyncBatchManager: identical
method signatures
        * with automatic optimization. Notice how:
        *
        * - batchManager.receiveMessage() uses the same syntax as
sqsAsyncClient.receiveMessage()
        * - batchManager.deleteMessage() uses the same syntax as
sqsAsyncClient.deleteMessage()
        *
        * Behind the scenes, the batch manager:
        * 1. Maintains an internal message buffer populated by background
fetching
        * 2. Returns messages immediately from the buffer (reduced latency)
        * 3. Automatically batches deleteMessage() calls into
DeleteMessageBatchRequest operations
        * 4. Sends batch deletes when maxBatchSize (10) or sendRequestFrequency
(200ms) is reached
        *
        * This provides significant performance improvements with zero code
changes
        * compared to traditional SqsAsyncClient usage patterns.
        */
       public void run() {
           try {
               while (!stop.get()) {
                   // Receive messages using the high-level batch manager.
                   // This call uses identical syntax to
sqsAsyncClient.receiveMessage()
                   // but benefits from internal message buffering for improved
performance.
                   final ReceiveMessageResponse result =
batchManager.receiveMessage(
                           ReceiveMessageRequest.builder()
                                   .queueUrl(queueUrl)
                                   .maxNumberOfMessages(Math.min(batchSize, 10))
                                   .build()).join();

                   if (!result.messages().isEmpty()) {
                       final List<Message> messages = result.messages();

                       // Delete messages using the batch manager.
                       // Each deleteMessage() call uses identical syntax to
SqsAsyncClient
```

```
                        // but the SDK automatically buffers these calls and
sends them
                        // as DeleteMessageBatchRequest operations for optimal
performance.
                        for (Message message : messages) {
                            CompletableFuture<DeleteMessageResponse> future =
batchManager.deleteMessage(
                                    DeleteMessageRequest.builder()
                                            .queueUrl(queueUrl)

.receiptHandle(message.receiptHandle())
                                            .build());

                            future.whenComplete((response, throwable) -> {
                                if (throwable == null) {
                                    consumedCount.incrementAndGet();
                                } else if (!(throwable instanceof
java.util.concurrent.CancellationException) &&
                                        !(throwable.getMessage() != null &&
throwable.getMessage().contains("executor not accepting a task"))) {
                                    log.error("BatchConsumer: Failed to delete
message", throwable);
                                }
                                // Ignore CancellationException and executor
shutdown errors - expected during shutdown
                            });
                        }
                    }

                    // Small delay to prevent tight polling
                    Thread.sleep(10);
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                log.error("BatchConsumer interrupted: " + e.getMessage());
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                // Handle both SdkException and CompletionException from async
operations.
                // If this unlikely condition occurs, stop.
                log.error("BatchConsumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }
```

```
    /**
     * Displays real-time throughput statistics every second.
     *
     * This thread logs the current count of produced and consumed messages
     * to help you monitor the performance comparison.
     */
    private static class Monitor extends Thread {
        private final AtomicInteger producedCount;
        private final AtomicInteger consumedCount;
        private final AtomicBoolean stop;

        /**
         * Creates a monitoring thread that displays throughput statistics.
         *
         * @param producedCount shared counter for messages sent
         * @param consumedCount shared counter for messages processed
         * @param stop shared flag to signal thread termination
         */
        Monitor(AtomicInteger producedCount, AtomicInteger consumedCount,
                AtomicBoolean stop) {
            this.producedCount = producedCount;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /**
         * Logs throughput statistics every second until stopped.
         *
         * Displays the current count of produced and consumed messages
         * to help monitor the performance comparison between batching
strategies.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    Thread.sleep(1000);
                    log.info("produced messages = " + producedCount.get()
                            + ", consumed messages = " + consumedCount.get());
                }
            } catch (InterruptedException e) {
                // Allow the thread to exit.
            }
        }
```

```
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API Reference》中的以下主题。
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建函数来包装 Amazon SQS 消息函数。

```
import logging
import sys

import boto3
from botocore.exceptions import ClientError

import queue_wrapper

logger = logging.getLogger(__name__)
sqs = boto3.resource("sqs")

def send_messages(queue, messages):
    """
    Send a batch of messages in a single request to an SQS queue.
```

```
   This request may return overall success even when some messages were not
sent.
   The caller must inspect the Successful and Failed lists in the response and
   resend any failed messages.

   :param queue: The queue to receive the messages.
   :param messages: The messages to send to the queue. These are simplified to
                    contain only the message body and attributes.
   :return: The response from SQS that contains the list of successful and
failed
            messages.
   """
   try:
       entries = [
           {
               "Id": str(ind),
               "MessageBody": msg["body"],
               "MessageAttributes": msg["attributes"],
           }
           for ind, msg in enumerate(messages)
       ]
       response = queue.send_messages(Entries=entries)
       if "Successful" in response:
           for msg_meta in response["Successful"]:
               logger.info(
                   "Message sent: %s: %s",
                   msg_meta["MessageId"],
                   messages[int(msg_meta["Id"])]["body"],
               )
       if "Failed" in response:
           for msg_meta in response["Failed"]:
               logger.warning(
                   "Failed to send: %s: %s",
                   msg_meta["MessageId"],
                   messages[int(msg_meta["Id"])]["body"],
               )
   except ClientError as error:
       logger.exception("Send messages failed to queue: %s", queue)
       raise error
   else:
       return response
```

```
def receive_messages(queue, max_number, wait_time):
    """
    Receive a batch of messages in a single request from an SQS queue.

    :param queue: The queue from which to receive messages.
    :param max_number: The maximum number of messages to receive. The actual
 number
                       of messages received might be less.
    :param wait_time: The maximum time to wait (in seconds) before returning.
 When
                      this number is greater than zero, long polling is used.
 This
                      can result in reduced costs and fewer false empty
 responses.
    :return: The list of Message objects received. These each contain the body
             of the message and metadata and custom attributes.
    """
    try:
        messages = queue.receive_messages(
            MessageAttributeNames=["All"],
            MaxNumberOfMessages=max_number,
            WaitTimeSeconds=wait_time,
        )
        for msg in messages:
            logger.info("Received message: %s: %s", msg.message_id, msg.body)
    except ClientError as error:
        logger.exception("Couldn't receive messages from queue: %s", queue)
        raise error
    else:
        return messages



def delete_messages(queue, messages):
    """
    Delete a batch of messages from a queue in a single request.

    :param queue: The queue from which to delete the messages.
    :param messages: The list of messages to delete.
    :return: The response from SQS that contains the list of successful and
 failed
             message deletions.
    """
    try:
```

```
        entries = [
            {"Id": str(ind), "ReceiptHandle": msg.receipt_handle}
            for ind, msg in enumerate(messages)
        ]
        response = queue.delete_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info("Deleted %s",
 messages[int(msg_meta["Id"])].receipt_handle)
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Could not delete %s",
 messages[int(msg_meta["Id"])].receipt_handle
                )
    except ClientError:
        logger.exception("Couldn't delete messages from queue %s", queue)
    else:
        return response
```

使用包装器函数批量发送和接收消息。

```
def usage_demo():
    """
    Shows how to:
    * Read the lines from this Python file and send the lines in
      batches of 10 as messages to a queue.
    * Receive the messages in batches until the queue is empty.
    * Reassemble the lines of the file and verify they match the original file.
    """

    def pack_message(msg_path, msg_body, msg_line):
        return {
            "body": msg_body,
            "attributes": {
                "path": {"StringValue": msg_path, "DataType": "String"},
                "line": {"StringValue": str(msg_line), "DataType": "String"},
            },
        }
```

```
    def unpack_message(msg):
        return (
            msg.message_attributes["path"]["StringValue"],
            msg.body,
            int(msg.message_attributes["line"]["StringValue"]),
        )

    print("-" * 88)
    print("Welcome to the Amazon Simple Queue Service (Amazon SQS) demo!")
    print("-" * 88)

    queue = queue_wrapper.create_queue("sqs-usage-demo-message-wrapper")

    with open(__file__) as file:
        lines = file.readlines()

    line = 0
    batch_size = 10
    received_lines = [None] * len(lines)
    print(f"Sending file lines in batches of {batch_size} as messages.")
    while line < len(lines):
        messages = [
            pack_message(__file__, lines[index], index)
            for index in range(line, min(line + batch_size, len(lines)))
        ]
        line = line + batch_size
        send_messages(queue, messages)
        print(".", end="")
        sys.stdout.flush()
    print(f"Done. Sent {len(lines) - 1} messages.")

    print(f"Receiving, handling, and deleting messages in batches of
{batch_size}.")
    more_messages = True
    while more_messages:
        received_messages = receive_messages(queue, batch_size, 2)
        print(".", end="")
        sys.stdout.flush()
        for message in received_messages:
            path, body, line = unpack_message(message)
            received_lines[line] = body
        if received_messages:
            delete_messages(queue, received_messages)
        else:
```

```
            more_messages = False
    print("Done.")

    if all([lines[index] == received_lines[index] for index in
range(len(lines))]):
        print(f"Successfully reassembled all file lines!")
    else:
        print(f"Uh oh, some lines were missed!")

    queue.delete()

    print("Thanks for watching!")
    print("-" * 88)
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API Reference》中的以下主题。
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用适用于.NET 的 AWS 消息处理框架发布和接收 Amazon SQS 消息

以下代码示例说明如何使用适用于.NET 的消息处理框架创建发布和接收 Amazon SQS AWS 消息的应用程序。

.NET

适用于 .NET 的 SDK

提供.NET AWS 消息处理框架的教程。本教程创建了一个支持用户发布 Amazon SQS 消息的 Web 应用程序和一个用于接收消息的命令行应用程序。

有关如何设置和运行的完整源代码和说明，请参阅《 适用于 .NET 的 AWS SDK 开发人员指南》中的完整教程和上的示例GitHub。

本示例中使用的服务

- Amazon SQS

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 Amazon SQS Java Messaging Library 处理 Amazon SQS 的 Java Message Service（ JMS ）接口

以下代码示例演示如何使用 Amazon SQS Java Messaging Library 处理 JMS 接口。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

以下示例处理标准 Amazon SQS 队列，包括：

- 发送文本消息。
- 同步接收消息。
- 异步接收消息。
- 使用 CLIENT_ACKNOWLEDGE 模式接收消息。
- 使用 UNORDERED_ACKNOWLEDGE 模式接收消息。
- 使用 Spring 注入依赖关系。

- 一个实用程序类，提供由其它示例使用的常用方法。

有关将 JMS 与 Amazon SQS 结合使用的更多信息，请参阅 Amazon SQS Developer Guide。

发送文本消息。

```
   /**
    * This method establishes a connection to a standard Amazon SQS queue using
the Amazon SQS
    * Java Messaging Library and sends text messages to it. It uses JMS (Java
Message Service) API
    * with automatic acknowledgment mode to ensure reliable message delivery,
and automatically
    * manages all messaging resources.
    *
    * @throws JMSException If there is a problem connecting to or sending
messages to the queue
    */
   public static void doSendTextMessage() throws JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
               SqsClient.create()
       );

       // Create the connection in a try-with-resources statement so that it's
closed automatically.
       try (SQSConnection connection = connectionFactory.createConnection()) {

           // Create the queue if needed.
           SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

           // Create a session that uses the JMS auto-acknowledge mode.
           Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
           MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));

           createAndSendMessages(session, producer);
       } // The connection closes automatically. This also closes the session.
       LOGGER.info("Connection closed");
   }
```

```
   /**
    * This method reads text input from the keyboard and sends each line as a
separate message
    * to a standard Amazon SQS queue using the Amazon SQS Java Messaging
Library. It continues
    * to accept input until the user enters an empty line, using JMS (Java
Message Service) API to
    * handle the message delivery.
    *
    * @param session The JMS session used to create messages
    * @param producer The JMS message producer used to send messages to the
queue
    */
   private static void createAndSendMessages(Session session, MessageProducer
producer) {
       BufferedReader inputReader = new BufferedReader(
               new InputStreamReader(System.in, Charset.defaultCharset()));

       try {
           String input;
           while (true) {
               LOGGER.info("Enter message to send (leave empty to exit): ");
               input = inputReader.readLine();
               if (input == null || input.isEmpty()) break;

               TextMessage message = session.createTextMessage(input);
               producer.send(message);
               LOGGER.info("Send message {}", message.getJMSMessageID());
           }
       } catch (EOFException e) {
           // Just return on EOF
       } catch (IOException e) {
           LOGGER.error("Failed reading input: {}", e.getMessage(), e);
       } catch (JMSException e) {
           LOGGER.error("Failed sending message: {}", e.getMessage(), e);
       }
   }
```

同步接收消息。

```
   /**
```

```
    * This method receives messages from a standard Amazon SQS queue using the
Amazon SQS Java
    * Messaging Library. It creates a connection to the queue using JMS (Java
Message Service),
    * waits for messages to arrive, and processes them one at a time. The method
handles all
    * necessary setup and cleanup of messaging resources.
    *
    * @throws JMSException If there is a problem connecting to or receiving
messages from the queue
    */
   public static void doReceiveMessageSync() throws JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
               SqsClient.create()
       );

       // Create a connection.
       try (SQSConnection connection = connectionFactory.createConnection() ) {

           // Create the queue if needed.
           SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

           // Create a session.
           Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);
           MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

           connection.start();

           receiveMessages(consumer);
       }  // The connection closes automatically. This also closes the session.
       LOGGER.info("Connection closed");
   }

   /**
    * This method continuously checks for new messages from a standard Amazon
SQS queue using
    * the Amazon SQS Java Messaging Library. It waits up to 20 seconds for each
message, processes
```

```
    * it using JMS (Java Message Service), and confirms receipt. The method
stops checking for
    * messages after 20 seconds of no activity.
    *
    * @param consumer The JMS message consumer that receives messages from the
queue
    */
   private static void receiveMessages(MessageConsumer consumer) {
       try {
           while (true) {
               LOGGER.info("Waiting for messages...");
               // Wait 1 minute for a message
               Message message =
consumer.receive(Duration.ofSeconds(20).toMillis());
               if (message == null) {
                   LOGGER.info("Shutting down after 20 seconds of silence.");
                   break;
               }
               SqsJmsExampleUtils.handleMessage(message);
               message.acknowledge();
               LOGGER.info("Acknowledged message {}",
message.getJMSMessageID());
           }
       } catch (JMSException e) {
           LOGGER.error("Error receiving from SQS: {}", e.getMessage(), e);
       }
   }
```

异步接收消息。

```
   /**
    * This method sets up automatic message handling for a standard Amazon SQS
queue using the
    * Amazon SQS Java Messaging Library. It creates a listener that processes
messages as soon
    * as they arrive using JMS (Java Message Service), runs for 5 seconds, then
cleans up all
    * messaging resources.
    *
    * @throws JMSException If there is a problem connecting to or receiving
messages from the queue
    */
```

```
    public static void doReceiveMessageAsync() throws JMSException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        // Create a connection.
        try (SQSConnection connection = connectionFactory.createConnection() ) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

            // Create a session.
            Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

            try {
                // Create a consumer for the queue.
                MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));
                // Provide an implementation of the MessageListener interface,
which has a single 'onMessage' method.
                // We use a lambda expression for the implementation.
                consumer.setMessageListener(message -> {
                    try {
                        SqsJmsExampleUtils.handleMessage(message);
                        message.acknowledge();
                    } catch (JMSException e) {
                        LOGGER.error("Error processing message: {}",
e.getMessage());
                    }
                });
                // Start receiving incoming messages.
                connection.start();
                LOGGER.info("Waiting for messages...");
            } catch (JMSException e) {
                throw new RuntimeException(e);
            }
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
```

```
            }
        }  // The connection closes automatically. This also closes the session.
        LOGGER.info( "Connection closed" );
    }
```

使用 CLIENT_ACKNOWLEDGE 模式接收消息。

```
    /**
     * This method demonstrates how message acknowledgment affects message
processing in a standard
     * Amazon SQS queue using the Amazon SQS Java Messaging Library. It sends
messages to the queue,
     * then shows how JMS (Java Message Service) client acknowledgment mode
handles both explicit
     * and implicit message confirmations, including how acknowledging one
message can automatically
     * acknowledge previous messages.
     *
     * @throws JMSException If there is a problem with the messaging operations
     */
    public static void doReceiveMessagesSyncClientAcknowledge() throws
JMSException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        // Create the connection in a try-with-resources statement so that it's
closed automatically.
        try (SQSConnection connection = connectionFactory.createConnection() ) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
TIME_OUT_SECONDS);

            // Create a session with client acknowledge mode.
            Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

            // Create a producer and consumer.
```

```
            MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));
            MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

            // Open the connection.
            connection.start();

            // Send two text messages.
            sendMessage(producer, session, "Message 1");
            sendMessage(producer, session, "Message 2");

            // Receive a message and don't acknowledge it.
            receiveMessage(consumer, false);

            // Receive another message and acknowledge it.
            receiveMessage(consumer, true);

            // Wait for the visibility time out, so that unacknowledged messages
reappear in the queue,
            LOGGER.info("Waiting for visibility timeout...");
            try {
                Thread.sleep(TIME_OUT_MILLIS);
            } catch (InterruptedException e) {
                LOGGER.error("Interrupted while waiting for visibility timeout",
e);
                Thread.currentThread().interrupt();
                throw new RuntimeException("Processing interrupted", e);
            }

            /*  We will attempt to receive another message, but none will be
available. This is because in
                CLIENT_ACKNOWLEDGE mode, when we acknowledged the second message,
all previous messages were
                automatically acknowledged as well. Therefore, although we never
directly acknowledged the first
                message, it was implicitly acknowledged when we confirmed the
second one. */
            receiveMessage(consumer, true);
        } // The connection closes automatically. This also closes the session.
        LOGGER.info("Connection closed.");

    }
```

```
   /**
    * Sends a text message using the specified JMS MessageProducer and Session.
    *
    * @param producer    The JMS MessageProducer used to send the message
    * @param session     The JMS Session used to create the text message
    * @param messageText The text content to be sent in the message
    * @throws JMSException If there is an error creating or sending the message
    */
   private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSException {
       // Create a text message and send it.
       producer.send(session.createTextMessage(messageText));
   }

   /**
    * Receives and processes a message from a JMS queue using the specified
consumer.
    * The method waits for a message until the configured timeout period is
reached.
    * If a message is received, it is logged and optionally acknowledged based
on the
    * acknowledge parameter.
    *
    * @param consumer    The JMS MessageConsumer used to receive messages from
the queue
    * @param acknowledge Boolean flag indicating whether to acknowledge the
message.
    *                    If true, the message will be acknowledged after
processing
    * @throws JMSException If there is an error receiving, processing, or
acknowledging the message
    */
   private static void receiveMessage(MessageConsumer consumer, boolean
acknowledge) throws JMSException {
       // Receive a message.
       Message message = consumer.receive(TIME_OUT_MILLIS);

       if (message == null) {
           LOGGER.info("Queue is empty!");
       } else {
           // Since this queue has only text messages, cast the message object
and print the text.
```

```
            LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
message).getText(), acknowledge);

            // Acknowledge the message if asked.
            if (acknowledge) message.acknowledge();
        }
    }
```

使用 UNORDERED_ACKNOWLEDGE 模式接收消息。

```
    /**
     * Demonstrates message acknowledgment behavior in UNORDERED_ACKNOWLEDGE mode
with Amazon SQS JMS.
     * In this mode, each message must be explicitly acknowledged regardless of
receive order.
     * Unacknowledged messages return to the queue after the visibility timeout
expires,
     * unlike CLIENT_ACKNOWLEDGE mode where acknowledging one message
acknowledges all previous messages.
     *
     * @throws JMSException         If a JMS-related error occurs during message
operations
     */
    public static void doReceiveMessagesUnorderedAcknowledge() throws
JMSException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        // Create the connection in a try-with-resources statement so that it's
closed automatically.
        try( SQSConnection connection = connectionFactory.createConnection() ) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
TIME_OUT_SECONDS);

            // Create a session with unordered acknowledge mode.
            Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);
```

```
            // Create the producer and consumer.
            MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));
            MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

            // Open a connection.
            connection.start();

            // Send two text messages.
            sendMessage(producer, session, "Message 1");
            sendMessage(producer, session, "Message 2");

            // Receive a message and don't acknowledge it.
            receiveMessage(consumer, false);

            // Receive another message and acknowledge it.
            receiveMessage(consumer, true);

            // Wait for the visibility time out, so that unacknowledged messages
reappear in the queue.
            LOGGER.info("Waiting for visibility timeout...");
            try {
                Thread.sleep(TIME_OUT_MILLIS);
            } catch (InterruptedException e) {
                LOGGER.error("Interrupted while waiting for visibility timeout",
e);
                Thread.currentThread().interrupt();
                throw new RuntimeException("Processing interrupted", e);
            }

            /*  We will attempt to receive another message, and we'll get the
first message again. This occurs
                because in UNORDERED_ACKNOWLEDGE mode, each message requires its
own separate acknowledgment.
                Since we only acknowledged the second message, the first message
remains in the queue for
                redelivery. */
            receiveMessage(consumer, true);

            LOGGER.info("Connection closed.");
        } // The connection closes automatically. This also closes the session.
    }
```

```
   /**
    * Sends a text message to an Amazon SQS queue using JMS.
    *
    * @param producer    The JMS MessageProducer for the queue
    * @param session     The JMS Session for message creation
    * @param messageText The message content
    * @throws JMSException If message creation or sending fails
    */
   private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSException {
       // Create a text message and send it.
       producer.send(session.createTextMessage(messageText));
   }
   /**
    * Synchronously receives a message from an Amazon SQS queue using the JMS
API
    * with an acknowledgment parameter.
    *
    * @param consumer    The JMS MessageConsumer for the queue
    * @param acknowledge If true, acknowledges the message after receipt
    * @throws JMSException If message reception or acknowledgment fails
    */
   private static void receiveMessage(MessageConsumer consumer, boolean
acknowledge) throws JMSException {
       // Receive a message.
       Message message = consumer.receive(TIME_OUT_MILLIS);

       if (message == null) {
           LOGGER.info("Queue is empty!");
       } else {
           // Since this queue has only text messages, cast the message object
and print the text.
           LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
message).getText(), acknowledge);

           // Acknowledge the message if asked.
           if (acknowledge) message.acknowledge();
       }
   }
```

使用 Spring 注入依赖关系。

```
package com.example.sqs.jms.spring;

import com.amazon.sqs.javamessaging.SQSConnection;
import com.example.sqs.jms.SqsJmsExampleUtils;
import jakarta.jms.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.NoSuchBeanDefinitionException;
import org.springframework.context.support.FileSystemXmlApplicationContext;

import java.io.File;
import java.net.URL;
import java.util.concurrent.TimeUnit;

/**
 * Demonstrates how to send and receive messages using the Amazon SQS Java
 Messaging Library
 * with Spring Framework integration. This example connects to a standard Amazon
 SQS message
 * queue using Spring's dependency injection to configure the connection and
 messaging components.
 * The application uses the JMS (Java Message Service) API to handle message
 operations.
 */
public class SpringExample {
    private static final Integer POLLING_SECONDS = 15;
    private static final String SPRING_XML_CONFIG_FILE =
 "SpringExampleConfiguration.xml.txt";
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SpringExample.class);

    /**
     * Demonstrates sending and receiving messages through a standard Amazon SQS
 message queue
     * using Spring Framework configuration. This method loads connection
 settings from an XML file,
     * establishes a messaging session using the Amazon SQS Java Messaging
 Library, and processes
     * messages using JMS (Java Message Service) operations. If the queue doesn't
 exist, it will
     * be created automatically.
     *
     * @param args Command line arguments (not used)
```

```
     */
    public static void main(String[] args) {

        URL resource =
SpringExample.class.getClassLoader().getResource(SPRING_XML_CONFIG_FILE);
        File springFile = new File(resource.getFile());
        if (!springFile.exists() || !springFile.canRead()) {
            LOGGER.error("File " + SPRING_XML_CONFIG_FILE + " doesn't exist or
isn't readable.");
            System.exit(1);
        }

        try (FileSystemXmlApplicationContext context =
                    new FileSystemXmlApplicationContext("file://" +
springFile.getAbsolutePath())) {

            Connection connection;
            try {
                connection = context.getBean(Connection.class);
            } catch (NoSuchBeanDefinitionException e) {
                LOGGER.error("Can't find the JMS connection to use: " +
e.getMessage(), e);
                System.exit(2);
                return;
            }

            String queueName;
            try {
                queueName = context.getBean("queueName", String.class);
            } catch (NoSuchBeanDefinitionException e) {
                LOGGER.error("Can't find the name of the queue to use: " +
e.getMessage(), e);
                System.exit(3);
                return;
            }
            try {
                if (connection instanceof SQSConnection) {
                    SqsJmsExampleUtils.ensureQueueExists((SQSConnection)
connection, queueName, SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);
                }
                // Create the JMS session.
                Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);
```

```
                SqsJmsExampleUtils.sendTextMessage(session, queueName);
                MessageConsumer consumer =
session.createConsumer(session.createQueue(queueName));

                receiveMessages(consumer);
            } catch (JMSException e) {
                LOGGER.error(e.getMessage(), e);
                throw new RuntimeException(e);
            }
        }   // Spring context autocloses. Managed Spring beans that implement
AutoClosable, such as the
        // 'connection' bean, are also closed.
        LOGGER.info("Context closed");
    }

    /**
     * Continuously checks for and processes messages from a standard Amazon SQS
message queue
     * using the Amazon SQS Java Messaging Library underlying the JMS API. This
method waits for incoming messages,
     * processes them when they arrive, and acknowledges their receipt using JMS
(Java Message
     * Service) operations. The method will stop checking for messages after 15
seconds of
     * inactivity.
     *
     * @param consumer The JMS message consumer used to receive messages from the
queue
     */
    private static void receiveMessages(MessageConsumer consumer) {
        try {
            while (true) {
                LOGGER.info("Waiting for messages...");
                // Wait 15 seconds for a message.
                Message message =
consumer.receive(TimeUnit.SECONDS.toMillis(POLLING_SECONDS));
                if (message == null) {
                    LOGGER.info("Shutting down after {} seconds of silence.",
POLLING_SECONDS);
                    break;
                }
                SqsJmsExampleUtils.handleMessage(message);
                message.acknowledge();
                LOGGER.info("Message acknowledged.");
```

```
            }
        } catch (JMSException e) {
            LOGGER.error("Error receiving from SQS.", e);
        }
    }
}
```

Spring bean 定义。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
        xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans-3.0.xsd
        ">
    <!-- Define the AWS Region -->
    <bean id="region" class="software.amazon.awssdk.regions.Region" factory-
method="of">
        <constructor-arg value="us-east-1"/>
    </bean>

    <bean id="credentialsProviderBean"
 class="software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider"
          factory-method="create"/>

    <bean id="clientBuilder"
 class="software.amazon.awssdk.services.sqs.SqsClient" factory-method="builder"/>

    <bean id="regionSetClientBuilder" factory-bean="clientBuilder" factory-
method="region">
        <constructor-arg ref="region"/>
    </bean>

    <!-- Configure the Builder with Credentials Provider -->
    <bean id="sqsClient" factory-bean="regionSetClientBuilder" factory-
method="credentialsProvider">
        <constructor-arg ref="credentialsProviderBean"/>
    </bean>
```

```
    <bean id="providerConfiguration"
 class="com.amazon.sqs.javamessaging.ProviderConfiguration">
        <property name="numberOfMessagesToPrefetch" value="5"/>
    </bean>

    <bean id="connectionFactory"
 class="com.amazon.sqs.javamessaging.SQSConnectionFactory">
        <constructor-arg ref="providerConfiguration"/>
        <constructor-arg ref="clientBuilder"/>
    </bean>

    <bean id="connection"
          factory-bean="connectionFactory"
          factory-method="createConnection"
          init-method="start"
          destroy-method="close"/>

    <bean id="queueName" class="java.lang.String">
        <constructor-arg value="SQSJMSClientExampleQueue"/>
    </bean>
</beans>
```

一个实用程序类，提供由其它示例使用的常用方法。

```
package com.example.sqs.jms;

import com.amazon.sqs.javamessaging.AmazonSQSMessagingClientWrapper;
import com.amazon.sqs.javamessaging.ProviderConfiguration;
import com.amazon.sqs.javamessaging.SQSConnection;
import com.amazon.sqs.javamessaging.SQSConnectionFactory;
import jakarta.jms.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.exception.SdkException;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;

import java.time.Duration;
import java.util.Base64;
import java.util.Map;
```

```
/**
 * This utility class provides helper methods for working with Amazon Simple
 Queue Service (Amazon SQS)
 * through the Java Message Service (JMS) interface. It contains common
 operations for managing message
 * queues and handling message delivery.
 */
public class SqsJmsExampleUtils {
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SqsJmsExampleUtils.class);
    public static final Long QUEUE_VISIBILITY_TIMEOUT = 5L;

    /**
     * This method verifies that a message queue exists and creates it if
 necessary. The method checks for
     * an existing queue first to optimize performance.
     *
     * @param connection The active connection to the messaging service
     * @param queueName The name of the queue to verify or create
     * @param visibilityTimeout The duration in seconds that messages will be
 hidden after being received
     * @throws JMSException If there is an error accessing or creating the queue
     */
    public static void ensureQueueExists(SQSConnection connection, String
 queueName, Long visibilityTimeout) throws JMSException {
        AmazonSQSMessagingClientWrapper client =
 connection.getWrappedAmazonSQSClient();

      /* In most cases, you can do this with just a 'createQueue' call, but
 'getQueueUrl'
      (called by 'queueExists') is a faster operation for the common case where
 the queue
      already exists. Also, many users and roles have permission to call
 'getQueueUrl'
      but don't have permission to call 'createQueue'.
      */
       if( !client.queueExists(queueName) ) {
           CreateQueueRequest createQueueRequest = CreateQueueRequest.builder()
                   .queueName(queueName)
                   .attributes(Map.of(QueueAttributeName.VISIBILITY_TIMEOUT,
 String.valueOf(visibilityTimeout)))
                   .build();
           client.createQueue( createQueueRequest );
```

```
        }
    }

    /**
     * This method sends a simple text message to a specified message queue. It
handles all necessary
     * setup for the message delivery process.
     *
     * @param session The active messaging session used to create and send the
message
     * @param queueName The name of the queue where the message will be sent
     */
    public static void sendTextMessage(Session session, String queueName) {
        // Rest of implementation...

        try {
            MessageProducer producer =
session.createProducer( session.createQueue( queueName) );
            Message message = session.createTextMessage("Hello world!");
            producer.send(message);
        } catch (JMSException e) {
            LOGGER.error( "Error receiving from SQS", e );
        }
    }

    /**
     * This method processes incoming messages and logs their content based on
the message type.
     * It supports text messages, binary data, and Java objects.
     *
     * @param message The message to be processed and logged
     * @throws JMSException If there is an error reading the message content
     */
    public static void handleMessage(Message message) throws JMSException {
        // Rest of implementation...
        LOGGER.info( "Got message {}", message.getJMSMessageID() );
        LOGGER.info( "Content: ");
        if(message instanceof TextMessage txtMessage) {
            LOGGER.info( "\t{}", txtMessage.getText() );
        } else if(message instanceof BytesMessage byteMessage){
            // Assume the length fits in an int - SQS only supports sizes up to
256k so that
            // should be true
            byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
```

```
            byteMessage.readBytes(bytes);
            LOGGER.info( "\t{}", Base64.getEncoder().encodeToString( bytes ) );
        } else if( message instanceof ObjectMessage) {
            ObjectMessage objMessage = (ObjectMessage) message;
            LOGGER.info( "\t{}", objMessage.getObject() );
        }
    }

    /**
     * This method sets up automatic message processing for a specified queue. It
creates a listener
     * that will receive and handle incoming messages without blocking the main
program.
     *
     * @param session The active messaging session
     * @param queueName The name of the queue to monitor
     * @param connection The active connection to the messaging service
     */
    public static void receiveMessagesAsync(Session session, String queueName,
Connection connection) {
        // Rest of implementation...
        try {
            // Create a consumer for the queue.
            MessageConsumer consumer =
session.createConsumer(session.createQueue(queueName));
            // Provide an implementation of the MessageListener interface, which
has a single 'onMessage' method.
            // We use a lambda expression for the implementation.
            consumer.setMessageListener(message -> {
                try {
                    SqsJmsExampleUtils.handleMessage(message);
                    message.acknowledge();
                } catch (JMSException e) {
                    LOGGER.error("Error processing message: {}", e.getMessage());
                }
            });
            // Start receiving incoming messages.
            connection.start();
        } catch (JMSException e) {
            throw new RuntimeException(e);
        }
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
```

```
            throw new RuntimeException(e);
        }
    }


    /**
     * This method performs cleanup operations after message processing is
complete. It receives
     * any messages in the specified queue, removes the message queue and closes
all
     * active connections to prevent resource leaks.
     *
     * @param queueName The name of the queue to be removed
     * @param visibilityTimeout The duration in seconds that messages are hidden
after being received
     * @throws JMSException If there is an error during the cleanup process
     */
    public static void cleanUpExample(String queueName, Long visibilityTimeout)
throws JMSException {
        LOGGER.info("Performing cleanup.");

        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        try (SQSConnection connection = connectionFactory.createConnection() ) {
            ensureQueueExists(connection, queueName, visibilityTimeout);
            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

            receiveMessagesAsync(session, queueName, connection);

            SqsClient sqsClient =
connection.getWrappedAmazonSQSClient().getAmazonSQSClient();
            try {
                String queueUrl = sqsClient.getQueueUrl(b ->
b.queueName(queueName)).queueUrl();
                sqsClient.deleteQueue(b -> b.queueUrl(queueUrl));
                LOGGER.info("Queue deleted: {}", queueUrl);
            } catch (SdkException e) {
                LOGGER.error("Error during SQS operations: ", e);
            }
        }
```

```
        LOGGER.info("Clean up: Connection closed");
    }

    /**
     * This method creates a background task that sends multiple messages to a
specified queue
     * after waiting for a set time period. The task operates independently to
ensure efficient
     * message processing without interrupting other operations.
     *
     * @param queueName The name of the queue where messages will be sent
     * @param secondsToWait The number of seconds to wait before sending messages
     * @param numMessages The number of messages to send
     * @param visibilityTimeout The duration in seconds that messages remain
hidden after being received
     * @return A task that can be executed to send the messages
     */
    public static Runnable sendAMessageAsync(String queueName, Long
secondsToWait, Integer numMessages, Long visibilityTimeout) {
        return () -> {
            try {
                Thread.sleep(Duration.ofSeconds(secondsToWait).toMillis());
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException(e);
            }
            try {
                SQSConnectionFactory connectionFactory = new
SQSConnectionFactory(
                        new ProviderConfiguration(),
                        SqsClient.create()
                );
                try (SQSConnection connection =
connectionFactory.createConnection()) {
                    ensureQueueExists(connection, queueName, visibilityTimeout);
                    Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);
                    for (int i = 1; i <= numMessages; i++) {
                        MessageProducer producer =
session.createProducer(session.createQueue(queueName));
                        producer.send(session.createTextMessage("Hello World " +
i + "!"));
                    }
                }
```

```
            } catch (JMSException e) {
                LOGGER.error(e.getMessage(), e);
                throw new RuntimeException(e);
            }
        };
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API Reference》中的以下主题。
  - CreateQueue
  - DeleteQueue

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包处理队列标签和 Amazon SQS AWS

以下代码示例演示如何使用 Amazon SQS 执行标签操作。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

以下示例为队列创建标签、列出标签和移除标签。

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.ListQueueTagsResponse;
import software.amazon.awssdk.services.sqs.model.QueueDoesNotExistException;
import software.amazon.awssdk.services.sqs.model.SqsException;
```

```
import java.util.Map;
import java.util.UUID;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials. For more
 * information, see the <a href="https://docs.aws.amazon.com/sdk-for-java/latest/
developer-guide/get-started.html">AWS
 * SDK for Java Developer Guide</a>.
 */
public class TagExamples {
    static final SqsClient sqsClient = SqsClient.create();
    static final String queueName = "TagExamples-queue-" +
 UUID.randomUUID().toString().replace("-", "").substring(0, 20);
    private static final Logger LOGGER =
 LoggerFactory.getLogger(TagExamples.class);

    public static void main(String[] args) {
        final String queueUrl;
        try {
            queueUrl = sqsClient.createQueue(b ->
 b.queueName(queueName)).queueUrl();
            LOGGER.info("Queue created. The URL is: {}", queueUrl);
        } catch (RuntimeException e) {
            LOGGER.error("Program ending because queue was not created.");
            throw new RuntimeException(e);
        }
        try {
            addTags(queueUrl);
            listTags(queueUrl);
            removeTags(queueUrl);
        } catch (RuntimeException e) {
            LOGGER.error("Program ending because of an error in a method.");
        } finally {
            try {
                sqsClient.deleteQueue(b -> b.queueUrl(queueUrl));
                LOGGER.info("Queue successfully deleted. Program ending.");
                sqsClient.close();
            } catch (RuntimeException e) {
                LOGGER.error("Program ending.");
            } finally {
                sqsClient.close();
            }
        }
```

```
    }

    /** This method demonstrates how to use a Java Map to a tag a aueue.
     * @param queueUrl The URL of the queue to tag.
     */
    public static void addTags(String queueUrl) {
        // Build a map of the tags.
        final Map<String, String> tagsToAdd = Map.of(
                "Team", "Development",
                "Priority", "Beta",
                "Accounting ID", "456def");

        try {
            // Add tags to the queue using a Consumer<TagQueueRequest.Builder>
parameter.
            sqsClient.tagQueue(b -> b
                    .queueUrl(queueUrl)
                    .tags(tagsToAdd)
            );
        } catch (QueueDoesNotExistException e) {
            LOGGER.error("Queue does not exist: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }

    /** This method demonstrates how to view the tags for a queue.
     * @param queueUrl The URL of the queue whose tags you want to list.
     */
    public static void listTags(String queueUrl) {
        ListQueueTagsResponse response;
        try {
            // Call the listQueueTags method with a
Consumer<ListQueueTagsRequest.Builder> parameter that creates a
ListQueueTagsRequest.
            response = sqsClient.listQueueTags(b -> b
                    .queueUrl(queueUrl));
        } catch (SqsException e) {
            LOGGER.error("Exception thrown: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }

        // Log the tags.
        response.tags()
                .forEach((k, v) ->
```

```
                        LOGGER.info("Key: {} -> Value: {}", k, v));
    }

    /**
     * This method demonstrates how to remove tags from a queue.
     * @param queueUrl The URL of the queue whose tags you want to remove.
     */
    public static void removeTags(String queueUrl) {
        try {
            // Call the untagQueue method with a
 Consumer<UntagQueueRequest.Builder> parameter.
            sqsClient.untagQueue(b -> b
                    .queueUrl(queueUrl)
                    .tagKeys("Accounting ID") // Remove a single tag.
            );
        } catch (SqsException e) {
            LOGGER.error("Exception thrown: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API Reference》中的以下主题。
  - ListQueueTags
  - TagQueue
  - UntagQueue

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

Amazon SQS 无服务器示例

以下代码示例展示了如何将 Amazon SQS 与配合使用。 AWS SDKs

示例

- 通过 Amazon SQS 触发器调用 Lambda 函数
- 报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败

通过 Amazon SQS 触发器调用 Lambda 函数

以下代码示例演示了如何实现一个 Lambda 函数，该函数接收通过接收来自 SQS 队列的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 .NET 将 SQS 事件与 Lambda 结合使用。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
```

```
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Go 将 SQS 事件与 Lambda 结合使用。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)
```

```
func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Java 将 SQS 事件与 Lambda 结合使用。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
```

```
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
        }

    }
}
```

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Lambd JavaScript a 使用 SQS 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
```

```
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

使用 Lambd TypeScript a 使用 SQS 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 PHP 将 SQS 事件与 Lambda 结合使用。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
```

```
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Python 将 SQS 事件与 Lambda 结合使用。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 Ruby 将 SQS 事件与 Lambda 结合使用。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].each do |message|
    process_message(message)
  end
  puts "done"
end

def process_message(message)
  begin
    puts "Processed message #{message['body']}"
    # TODO: Do interesting work based on the new message
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 Rust 将 SQS 事件与 Lambda 结合使用。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
 record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败

以下代码示例显示如何为接收来自 SQS 队列的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 .NET 进行 Lambda SQS 批处理项目失败。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
```

```
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

Go

适用于 Go 的 SDK V2

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 Go 进行 Lambda SQS 批处理项目失败。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
 (map[string]interface{}, error) {
```

```
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {
  if len(message.Body) > 0 {
   // Your message processing condition here
   fmt.Printf("Successfully processed message: %s\n", message.Body)
  } else {
   // Message processing failed
   fmt.Printf("Failed to process message %s\n", message.MessageId)
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 Java 进行 Lambda SQS 批处理项目失败。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
```

```
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;

import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
 SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context) {
         List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<SQSBatchResponse.BatchItemFailure>();

         for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
             try {
                 //process your message
             } catch (Exception e) {
                 //Add failed message identifier to the batchItemFailures list
                 batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(message.getMessageId()));
             }
         }
         return new SQSBatchResponse(batchItemFailures);
     }
}
```

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用报告 Lambda JavaScript 的 SQS 批处理项目失败。

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
```

```
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

使用报告 Lambda TypeScript 的 SQS 批处理项目失败。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord }
 from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 PHP 进行 Lambda SQS 批处理项目失败。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();
```

```
        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 Python 进行 Lambda SQS 批处理项目失败。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}

        for record in event["Records"]:
```

```
            try:
                print(f"Processed message: {record['body']}")
            except Exception as e:
                batch_item_failures.append({"itemIdentifier":
 record['messageId']})

        sqs_batch_response["batchItemFailures"] = batch_item_failures
        return sqs_batch_response
```

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 Ruby 进行 Lambda SQS 批处理项目失败。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
      rescue StandardError => e
        batch_item_failures << {"itemIdentifier" => record['messageId']}
      end
    end

    sqs_batch_response["batchItemFailures"] = batch_item_failures
    return sqs_batch_response
  end
end
```

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 Rust 进行 Lambda SQS 批处理项目失败。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
 Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}
```

```
#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon SQS 与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

Amazon SQS 问题排查

本主题针对您在使用 Amazon SQS 控制台、Amazon SQS API 或其他工具时可能遇到的常见错误和问题提供问题排查建议。如果您发现某个问题未在此处列出，可以使用此页上的 Feedback 按钮来报告。

有关更多故障排除建议和常见支持问题的答案，请访问AWS知识中心。

主题

- 针对 Amazon SQS 中的访问被拒绝错误进行问题排查
- 针对 Amazon SQS API 错误进行问题排查
- 针对 Amazon SQS 死信队列和死信队列重新驱动问题进行排查
- 针对 Amazon SQS 中的 FIFO 节流问题进行排查
- 对 Amazon SQS ReceiveMessage API 调用未返回的消息进行故障排除
- 针对 Amazon SQS 网络错误进行问题排查
- 使用 Amazon 简单队列服务队列疑难解答 AWS X-Ray

针对 Amazon SQS 中的访问被拒绝错误进行问题排查

以下主题涵盖了 Amazon SQS API 调用中出现 `AccessDenied` 或 `AccessDeniedException` 错误的最常见原因。有关如何解决这些错误的更多信息，请参阅如何解决 Amazon SQS API 调用中的 AccessDeniedException “AccessDenied” 或 “” 错误？ 在AWS 知识中心指南中。

错误消息示例：

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to
        the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

–或者–

```
An error occurred (KMS.AccessDeniedException) when calling the SendMessage
        operation: User: arn:aws:iam::xxxxx:user/xxxx is not authorized to perform:
        kms:GenerateDataKey on resource: arn:aws:kms:us-east-1:xxxx:key/xxxx with an
 explicit
        deny.
```

Amazon SQS 队列策略和 IAM 策略

要验证请求者是否拥有执行 Amazon SQS 操作的适当权限，请执行以下操作：

- 确定正在发起 Amazon SQS API 调用的 IAM 主体。如果 IAM 主体属于同一个账户，那么 Amazon SQS 队列策略或 AWS Identity and Access Management（IAM）策略中必须包含明确允许访问该操作的权限。
- 如果主体是 IAM 实体：
  - 您可以通过检查 AWS 管理控制台的右上角或使用 `aws sts get-caller-identity` 命令来识别您的 IAM 用户或角色。
  - 检查与 IAM 用户或角色相关的 IAM policy。您可以使用以下方法之一：
    - 使用 IAM policy simulator 测试 IAM 策略。
    - 查看不同的 IAM policy 类型。
  - 如果需要，编辑您的 IAM 用户策略。
  - 检查队列策略并根据需要进行编辑。
- 如果委托人是一项 AWS 服务，那么 Amazon SQS 队列策略必须明确允许访问。
- 如果主体是跨账户主体，那么 Amazon SQS 队列策略和 IAM 策略都必须明确允许访问。
- 如果策略使用了条件元素，请检查该条件是否限制了访问。

> **Important**
>
> 任何策略中的显式拒绝都会覆盖显式允许。以下是 Amazon SQS 策略的一些基本示例。

AWS Key Management Service 权限

如果您的 Amazon SQS 队列开启了服务器端加密 (SSE) 并由客户管理 AWS KMS key，则必须向生产者和使用者授予权限。要确认队列是否已加密，可以使用 `GetQueueAttributes` API 的 `KmsMasterKeyId` 属性，或者在队列控制台的加密下查看相关信息。

- 创作者所需的权限：

```
{
"Effect": "Allow",
"Action": [
```

```
    "kms:Decrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

- 使用者所需的权限：

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt"
],
"Resource": "<Key ARN>"
}
```

- 用于跨账户访问的权限：

```
{
"Effect": "Allow",
"Action": [
    "kms:DescribeKey",
    "kms:Decrypt",
    "kms:ReEncrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

请选择以下选项之一，为 Amazon SQS 队列启用加密：

- SSE-Amazon SQS（由 Amazon SQS 服务创建和管理的加密密钥。）
- AWS 托管默认密钥 (alias/aws/sqs)
- 客户托管密钥

但是，如果您使用的是 AWS托管的 KMS 密钥，则无法修改默认密钥策略。因此，要提供对其他服务和跨账户的访问权限，请使用客户自主管理型密钥。这样一来，您就可以编辑密钥政策。

VPC 端点策略

如果您通过 Amazon Virtual Private Cloud（Amazon VPC）端点访问 Amazon SQS，则 Amazon SQS VPC 端点策略必须允许访问。您可以为 Amazon SQS 创建 Amazon VPC 端点策略，并在该策略中指定以下内容：

1. 可执行操作的主体。
2. 可执行的操作。
3. 可对其执行操作的资源。

在以下示例中，VPC 终端节点策略指定允许 IAM 用户*MyUser*向 Amazon SQS 队列发送消息。*MyQueue*通过该 VPC 端点的其他操作、IAM 用户和 Amazon SQS 资源的访问请求都会被拒绝。

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

组织服务控制策略

如果您 AWS 账户 属于某个组织，则 AWS Organizations 策略可能会阻止您访问您的 Amazon SQS 队列。默认情况下， AWS Organizations 策略不会阻止任何向 Amazon SQS 发出的请求。但是，请确保您的 AWS Organizations 策略尚未配置为阻止访问 Amazon SQS 队列。有关如何查看您的 AWS Organizations 政策的说明，请参阅《AWS Organizations 用户指南》中的列出所有政策。

针对 Amazon SQS API 错误进行问题排查

以下主题涵盖发出 Amazon SQS API 调用时系统返回的最常见错误，以及如何针对这些错误进行问题排查。

QueueDoesNotExist 错误

当 Amazon SQS 服务无法找到与 Amazon SQS 操作相关的指定队列时，系统将返回此错误。

可能的原因和缓解措施：

- 区域不正确：查看 Amazon SQS 客户端配置，确认您在客户端上配置了正确的区域。如果您没有在客户端上配置区域，SDK 或 AWS CLI 会从配置文件或环境变量中选择区域。如果 SDK 在配置文件中找不到区域，则默认情况下，SDK 会将区域设置为 us-east-1。
- 队列最近可能被删除：如果队列在 API 调用之前已经被删除，那么 API 调用将返回此错误。查看 CloudTrail，确认在错误发生前是否有任何 DeleteQueue 操作。
- 权限问题：如果发出请求的 AWS Identity and Access Management（IAM）用户或角色没有所需的权限，则您可能会收到以下错误：

  ```
  The specified queue does not exist or you do not have access to it.
  ```

  查看权限，然后使用正确的权限进行 API 调用。

有关针对 `QueueDoesNotExist` 错误进行问题排查的更多信息，请参阅《AWS Knowledge Center 指南》中的“如何解决对 Amazon SQS 队列进行 API 调用时出现的 QueueDoesNotExist 错误？”。

InvalidAttributeValue 错误

使用不正确的策略或主体更新 Amazon SQS 队列资源策略或属性时，系统会返回此错误。

可能的原因和缓解措施：

- 资源策略无效：检查资源策略是否包含所有必填字段。有关更多信息，请参阅 IAM JSON 策略元素参考和 IAM 策略验证。您还可以使用 IAM 策略生成器来创建和测试 Amazon SQS 资源策略。确保策略采用 JSON 格式。
- 主体无效：确保资源策略中存在 `Principal` 元素并且其值有效。如果您的 Amazon SQS 资源策略中的 `Principal` 元素包含 IAM 实体，则在使用该策略之前，请确保该实体已经存在。Amazon SQS 会验证资源策略并检查 IAM 实体是否存在。如果 IAM 实体不存在，您会收到错误。要确认 IAM 实体，可以使用 `GetRole` 和 `GetUser` API。

有关针对 `InvalidAttributeValue` 错误进行问题排查的更多信息，请参阅《AWS Knowledge Center 指南》中的“如何解决对 Amazon SQS 队列进行 API 调用时出现的 QueueDoesNotExist 错误？”。

ReceiptHandle 错误

发出 DeleteMessage API 调用时，如果接收句柄不正确或已过期，则系统可能会返回 ReceiptHandleIsInvalid 或 InvalidParameterValue 错误。

- ReceiptHandleIsInvalid 错误：如果接收句柄不正确，您会收到类似以下示例的错误：

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:
 The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- InvalidParameterValue 错误：如果接收句柄已过期，您会收到类似以下示例的错误：

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:
 Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The
 receipt handle has expired.
```

可能的原因和缓解措施：

系统会为每条收到的消息创建接收句柄，并且接收句柄仅在可见性超时时间内有效。当可见性超时时间到期时，消息将在队列中对使用者可见。当您再次收到使用者发送的消息时，会收到一个新的接收句柄。为防止出现接收句柄不正确或过期的错误，请使用正确的接收句柄在 Amazon SQS 队列可见性超时时间内删除消息。

有关如何针对 ReceiptHandle 错误进行问题排查的更多信息，请参阅《AWS Knowledge Center 指南》中的“当我使用 Amazon SQS DeleteMessage API 调用时，如何排除‘ReceiptHandleIsInvalid’或‘InvalidParameterValue’错误？”。

针对 Amazon SQS 死信队列和死信队列重新驱动问题进行排查

以下主题涵盖了 Amazon SQS 死信队列和死信队列重新驱动问题的最常见原因，以及如何进行问题排查。有关更多信息，请参阅《AWS Knowledge Center 指南》中的“如何解决 Amazon SQS DLQ 重新驱动问题？”。

死信队列问题

了解常见的死信队列问题以及如何解决这些问题。

使用控制台查看消息可能会导致消息移至死信队列

在控制台中根据相应队列的重新驱动策略查看消息时，Amazon SQS 将进行计数。因此，如果您在控制台中查看某条消息的次数达到相应队列的重新驱动策略中指定的次数，则该消息将被移动到相应队列的死信队列中。

要调整此行为，您可以执行下列操作之一：

- 针对相应队列的重新驱动策略增大 Maximum Receives 设置。
- 避免在控制台中查看相应队列的消息。

死信队列的 **NumberOfMessagesSent** 和 **NumberOfMessagesReceived** 不匹配

如果您手动向死信队列发送消息，该消息将被 NumberOfMessagesSent 指标捕获。不过，如果因处理尝试失败而发送消息到死信队列，则此指标不会捕获该消息。因此，NumberOfMessagesSent 和 NumberOfMessagesReceived 的值可能不同。

创建和配置死信队列重新驱动

死信队列重新驱动需要您为 Amazon SQS 设置适当的权限，以便接收来自死信队列的消息并将消息发送到目标队列。如果您没有正确的权限，死信队列重新驱动任务可能会失败。您可以查看消息重新驱动任务的状态，以便修复问题，然后重试。

标准队列和 FIFO 队列消息故障处理

标准队列会在保留期到期之前持续处理消息。这种持续处理可以最大限度地减少队列被未使用的消息阻塞的可能性。如果有大量消息未被使用者成功删除，会增加成本并加重硬件负载。为了降低成本，请将处理失败的消息移动到死信队列。

标准队列还支持大量的传输中消息。如果您的大多数消息无法使用且未发送到死信队列，消息的处理速率会变慢。为了保持队列的效率，请确保您的应用程序能够正确处理消息。

FIFO 队列通过按顺序使用消息组中的消息，确保仅处理一次。因此，尽管使用者可以继续检索其他消息组中的有序消息，但在被阻塞的队列的消息得到成功处理或移动到死信队列之前，第一个消息组将保持不可用状态。

此外，FIFO 队列支持少量的传输中消息。为了防止您的 FIFO 队列被消息阻塞，请确保您的应用程序能够正确处理消息。

有关更多信息，请参阅Amazon SQS 消息配额和 Amazon SQS 最佳实践。

死信队列重新驱动问题

了解常见的死信队列重新驱动问题以及如何解决这些问题。

AccessDenied 权限问题

当死信队列重新驱动因为 AWS Identity and Access Management （IAM）实体没有所需权限而失败时，就会发生 AccessDenied 错误。

错误消息示例：

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

发出死信队列重新驱动请求需要以下 API 权限：

启动消息重新驱动：

- 死信队列权限：
  - sqs:StartMessageMoveTask
  - sqs:ReceiveMessage
  - sqs:DeleteMessage
  - sqs:GetQueueAttributes
  - kms:Decrypt：当死信队列或原始源队列被加密时。
- 目标队列权限：
  - sqs:SendMessage
  - kms:GenerateDataKey：当目标队列被加密时。
  - kms:Decrypt：当目标队列被加密时。

取消进行中的消息重新驱动：

- 死信队列权限：
  - sqs:CancelMessageMoveTask
  - sqs:ReceiveMessage
  - sqs:DeleteMessage
  - sqs:GetQueueAttributes
  - kms:Decrypt：当死信队列或原始源队列被加密时。

显示消息移动状态：

- 死信队列权限：
  - `sqs:ListMessageMoveTasks`
  - `sqs:GetQueueAttributes`

`NonExistentQueue`错误

当 Amazon SQS 源队列不存在或已被删除时，就会发生 `NonExistentQueue` 错误。进行检查并重新驱动到现有的 Amazon SQS 队列。

错误消息示例：

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

CouldNotDetermineMessageSource 错误

当您尝试在以下情况下启动死信队列重新驱动时，就会发生 `CouldNotDetermineMessageSource` 错误：

- 使用 `SendMessage` API 直接发送到死信队列的 Amazon SQS 消息。
- 来自配置了 DLQ 的亚马逊简单通知服务 (Amazon SNS) Simple Notification 主题 AWS Lambda 或函数的消息。

要解决此错误，请在启动重新驱动时选择重新驱动到自定义目标。然后，输入 Amazon SQS 队列 ARN，将所有消息从死信队列移动到目标队列。

错误消息示例：

```
Failed: CouldNotDetermineMessageSource
```

针对 Amazon SQS 中的 FIFO 节流问题进行排查

默认情况下，FIFO 队列支持每个 API 操作（`SendMessage`、`ReceiveMessage` 和 `DeleteMessage`）每秒 300 个事务。即使队列中有可用消息，请求超过每秒 300 个事务也会触发 `ThrottlingException` 错误。您可以使用以下方法来缓解这一问题：

- 为 Amazon SQS 中的 FIFO 队列启用高吞吐量.
- 使用 Amazon SQS API 批量操作 SendMessageBatch、DeleteMessageBatch 和 ChangeMessageVisibilityBatch，将每个 API 操作的 TPS 上限提高到每秒最多 3000 条消息，同时降低成本。对于 ReceiveMessage API，设置 MaxNumberofMessages 参数，以便在每个事务中最多接收 10 条消息。有关更多信息，请参阅 Amazon SQS 批处理操作。
- 对于具有高吞吐量的 FIFO 队列，请按照建议优化分区利用率。批量发送具有相同消息组 IDs 的消息。使用来自同一 ReceiveMessage API 请求的接收句柄批量删除消息或更改消息可见性超时值。
- 增加唯一 MessageGroupId 值的数量。这有助于在 FIFO 队列分区之间实现均匀分配。有关更多信息，请参阅“Using the Amazon SQS message group ID”。

有关更多信息，请参阅《AWS Knowledge Center 指南》中的“为什么我的 Amazon SQS FIFO 队列没有返回所有消息或其他消息组中的消息？”。

对 Amazon SQS ReceiveMessage API 调用未返回的消息进行故障排除

以下主题介绍了 Amazon SQS 消息可能无法返回给使用者的最常见原因以及如何进行问题排查。有关更多信息，请参阅《AWS Knowledge Center 指南》中的“为什么我无法从我的 Amazon SQS 队列接收消息？”。

空队列

要确定队列是否为空，请使用长轮询来调用 ReceiveMessage API。您也可以使用ApproximateNumberOfMessagesVisibleApproximateNumberOfMessagesNotVisible、和ApproximateNumberOfMessagesDelayed CloudWatch 指标。如果所有指标的值在几分钟内均为 0，则判断队列为空。

已达到传输中消息数量限制

如果您使用长轮询，并且消息数量已超出队列的传输中消息数量上限（默认值为 120000），Amazon SQS 不会返回提示超出配额限制的错误消息。

消息延迟

如果 Amazon SQS 队列配置为延迟队列，或者消息通过消息计时器发送，则在延迟时间结束之前消息不可见。要验证队列是否配置为延迟队列，请使用 GetQueueAttributes

API `DelaySeconds` 属性，或者从队列控制台的交付延迟下方查看相关信息。检查`ApproximateNumberOfMessagesDelayed` CloudWatch 指标以了解是否有任何消息延迟。

消息正在传输中

如果其他使用者对消息进行了轮询，则该消息将在可见性超时期间内处于传输中或不可见状态。额外的轮询可能返回空结果（即没有可接收的消息）。检查ApproximateNumberOfMessagesVisible CloudWatch指标以了解可接收的消息数量。对于 FIFO 队列，如果具有消息组 ID 的消息正在传输中，除非您删除该消息或该消息变为可见，否则系统不会再返回消息。这是因为消息顺序是在 FIFO 队列中的消息组级别进行维护的。

轮询方法

如果您使用的是短轮询，则（WaitTimeSeconds为 0）Amazon SQS 会对其服务器的子集进行采样，并仅返回来自这些服务器的消息。因此，即使有可接收的消息，您也可能无法接收到它们。后续的轮询请求将会返回消息。

如果您使用长轮询，Amazon SQS 会轮询所有服务器，并在收集至少一条可用消息后，最多返回您指定的最大数量的消息。如果的值 ReceiveMessage WaitTimeSeconds太低，则可能无法收到所有可用消息。

针对 Amazon SQS 网络错误进行问题排查

以下主题介绍了 Amazon SQS 中的网络问题的最常见原因，以及如何进行问题排查。

ETIMEOUT error

当客户端无法建立与 Amazon SQS 端点的 TCP 连接时，就会发生 ETIMEOUT 错误。

故障排除 –

- 检查网络连接

  运行 `telnet` 等命令来测试与 Amazon SQS 的网络连接。

  Example: telnet sqs.us-east-1.amazonaws.com 443
- 检查网络设置
  - 确保您的本地防火墙规则、路由和访问控制列表（ACL）允许您使用的端口上的流量。
  - 安全组出站（出口）规则必须允许到端口 80 或 443 的流量。

  - 网络 ACL 出站（出口）规则必须允许到 TCP 端口 80 或 443 的流量。
  - 网络 ACL 入站（入口）规则必须允许 TCP 端口 1024-65535 上的流量。
  - 连接到公共互联网的 Amazon Elastic Compute Cloud（Amazon EC2）实例必须具有互联网连接。
- Amazon Virtual Private Cloud（Amazon VPC）端点

  如果您通过 Amazon VPC 端点访问 Amazon SQS，则端点安全组必须允许通过端口 443 进入客户端安全组的入站流量。与 VPC 端点的子网关联的网络 ACL 必须具有以下配置：
  - 网络 ACL 出站（出口）规则必须允许 TCP 端口 1024-65535（临时端口）上的流量。
  - 网络 ACL 入站（入口）规则必须允许端口 443 上的流量。

此外，Amazon SQS VPC 端点 AWS Identity and Access Management（IAM）策略必须允许访问。以下示例 VPC 端点策略指定允许 IAM 用户 *MyUser* 将消息发送到 Amazon SQS 队列 *MyQueue*。通过该 VPC 端点的其他操作、IAM 用户和 Amazon SQS 资源的访问请求都会被拒绝。

```
{
    "Statement": [{
        "Action": ["sqs:SendMessage"],
        "Effect": "Allow",
        "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
        "Principal": {
            "AWS": "arn:aws:iam:123456789012:user/MyUser"
        }
    }]
}
```

UnknownHostException error

当无法确定主机 IP 地址时，就会发生 UnknownHostException 错误。

故障排除 –

使用 nslookup 实用程序返回与主机名关联的 IP 地址：

- Windows and Linux OS

```
nslookup sqs.<region>.amazonaws.com
```

- AWS CLI 或者适用于 Python 的 SDK 原有端点：

```
nslookup <region>.queue.amazonaws.com
```

如果收到失败的输出，请按照《AWS Knowledge Center 指南》中的"How does DNS work and how do I troubleshoot partial or intermittent DNS failures?"的说明进行操作。

如果收到有效的输出，则可能是应用程序级别的问题。要解决应用程序级别的问题，可以尝试以下方法：

- 重启您的应用程序。
- 确认您的 Java 应用程序没有错误的 DNS 缓存。如果可能，对您的应用程序进行配置，使其符合 DNS TTL。有关更多信息，请参阅 Setting the JVM TTL for DNS name lookups。

有关如何针对网络错误进行问题排查的更多信息，请参阅《AWS Knowledge Center 指南》中的"How do I troubleshoot Amazon SQS 'ETIMEOUT' and 'UnknownHostException' connection errors?"。

使用 Amazon 简单队列服务队列疑难解答 AWS X-Ray

AWS X-Ray 收集有关您的应用程序所处理的请求的数据，并允许您查看和筛选数据，以确定潜在的问题和优化机会。对于对您的应用程序的任何跟踪请求，您可以查看有关请求、响应以及应用程序对下游 AWS 资源、微服务、数据库和 HTTP Web APIs 的调用的详细信息。

要通过 Amazon SQS 发送 AWS X-Ray 追踪标头，您可以执行以下任一操作：

- 使用 `X-Amzn-Trace-Id` 跟踪标头。
- 使用 `AWSTraceHeader` 消息系统属性。

要收集有关错误和延迟的数据，您必须使用 AWS X-Ray SDK 对 `AmazonSQS` 客户端进行检测。

您可以使用 AWS X-Ray 控制台查看 Amazon SQS 与您的应用程序使用的其他服务之间的连接图。您还可以使用控制台查看指标，例如平均延迟和故障率。有关更多信息，请参阅《AWS X-Ray 开发人员指南》中的 Amazon SQS 和 AWS X-Ray。

Amazon SQS 中的安全性

本节提供有关 Amazon SQS 安全性、身份验证和访问控制以及 Amazon SQS 访问策略语言的信息。

主题

- Amazon SQS 中的数据保护
- Amazon SQS 中的身份和访问管理
- Amazon SQS 中的日志记录和监控
- Amazon SQS 的合规性验证
- Amazon SQS 中的恢复功能
- Amazon SQS 中的基础设施安全性
- Amazon SQS 安全最佳实践

Amazon SQS 中的数据保护

责任 AWS 共担模式适用于亚马逊简单队列服务中的数据保护。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题解答 AWS条款。 有关欧洲数据保护的信息，请参阅通用数据保护条例（GDPR）中心。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》https://aws.amazon.com/compliance/fips/。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您使用控制台、API 或软件开发工具包 AWS 服务 使用 Amazon SQS 或其他软件开发工具包 AWS CLI的情况。 AWS 在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供 URL，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

以下部分提供有关 Amazon SQS 中数据保护的信息。

Amazon SQS 中的数据加密

数据保护指在数据传输（发往和离开 Amazon SQS 时）和处于静态（存储在 Amazon SQS 数据中心的磁盘上时）期间保护数据。您可以使用安全套接字层 (SSL) 或客户端加密保护传输中的数据。默认情况下，Amazon SQS 使用磁盘加密来存储消息和文件。您可以在将消息保存到数据中心的加密文件系统之前请求 Amazon SQS 对其进行加密，从而保护静态数据。Amazon SQS 建议使用 SSE 来优化数据加密。

主题

- Amazon SQS 中的静态加密
- Amazon SQS 密钥管理

Amazon SQS 中的静态加密

Server-side 加密 (SSE) 允许您在加密队列中传输敏感数据。SSE 使用 SQS-managed加密密钥 (SSE-SQS) 或 () 中管理的密钥保护队列中的 AWS Key Management Service 邮件内容。SSE-KMS有关使用管理 SSE 的信息 AWS 管理控制台，请参阅以下内容：

- SSE-SQS 为队列进行配置（控制台）
- SSE-KMS 为队列进行配置（控制台）

有关使用 适用于 Java 的 AWS SDK （以及`CreateQueue`、`SetQueueAttributes`和`GetQueueAttributes`操作）管理 SSE 的信息，请参阅以下示例：

- 在 Amazon SQS 队列中使用服务器端加密
- 为配置 KMS 权限 AWS 服务

一旦 Amazon SQS 收到消息，SSE 就会对消息进行加密。这些消息以加密形式存储，仅当消息发送给授权使用者时，Amazon SQS 才会对消息进行解密。

> **Important**
>
> 针对启用了 SSE 的队列的所有请求都必须使用 HTTPS 和 Signature Version 4。
> 使用默认密钥（Amazon SQS 的AWS 托管 KMS 密钥）的加密队列无法在其他队列中调用 Lambda 函数。 AWS 账户
> 可以使用 AWS Security Token Service `AssumeRole`操作向 Amazon SQS 发送通知的 AWS 服务的某些功能与 SSE 兼容，但仅适用于标准队列：
>
> - Auto Scaling 生命周期挂钩
> - AWS Lambda Dead-Letter 队列
>
> 有关其他服务与加密队列的兼容性的信息，请参阅 配置 KMS 权限 AWS 务 和服务文档。

AWS KMS 将安全、高度可用的硬件和软件相结合，提供可扩展到云端的密钥管理系统。当您将 Amazon SQS 与一起使用时 AWS KMS，加密消息数据的数据密钥也会被加密并与它们保护的数据一起存储。

使用 AWS KMS具有以下好处：

- 您可以自行创建和管理 AWS KMS keys。
- 您也可以使用适用于 Amazon SQS 的 AWS 托管 KMS 密钥，该密钥对于每个账户和地区都是唯一的。
- AWS KMS 安全标准可以帮助您满足与加密相关的合规性要求。

有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的什么是 AWS Key Management Service？

加密范围

SSE 将对 Amazon SQS 队列中的消息正文进行加密。

SSE 不对以下各项进行加密：

- 队列元数据（队列名称和属性）

- 消息元数据（消息 ID、时间戳和属性）
- Per-queue 指标

对消息进行加密将使其内容对未经授权的或匿名的用户不可用。启用 SSE 后，对加密队列的匿名 SendMessage 和 ReceiveMessage 请求将被拒绝。Amazon SQS 安全最佳实践建议不要使用匿名请求。如果您想向 Amazon SQS 队列发送匿名请求，请确保禁用 SSE。这不会影响 Amazon SQS 的正常功能：

- 仅在启用队列加密后发送消息时对其进行加密。Amazon SQS 不会加密积压的消息。
- 任何加密的消息将保持加密状态，即使已禁用其队列的加密。

将消息移至死信队列不会影响其加密：

- 如果 Amazon SQS 将一条消息从加密的源队列移至未加密的死信队列，则该消息将保持加密状态。
- 如果 Amazon SQS 将一条消息从未加密的源队列移至加密的死信队列，则该消息将保持未加密状态。

关键术语

以下关键术语有助于您更好地了解 SSE 的功能。有关详细说明，请参阅 Amazon Simple Queue Service API 参考。

数据密钥

负责加密 Amazon SQS 消息内容的密钥 (DEK)。

有关更多信息，请参阅《AWS Encryption SDK 开发人员指南》中《AWS Key Management Service 开发人员指南》的数据密钥。

数据密钥重用周期

在再次调用之前，Amazon SQS 可以重复使用数据密钥来加密或解密消息的时间长度，以秒为单位。 AWS KMS 一个表示秒数的证书，介于 60 秒（1 分钟）和 86400 秒（24 小时）之间。默认值为 300（5 分钟）。有关更多信息，请参阅 了解数据密钥重用周期。

> **Note**
>
> 万一出现无法访问的情况 AWS KMS，Amazon SQS 将继续使用缓存的数据密钥，直到重新建立连接。

KMS 密钥 ID

您的账户或其他账户中的 AWS 托管 KMS 密钥或自定义 KMS 密钥的别名、别名 ARN、密钥 ID 或密钥 ARN。虽然 Amazon SQS 的 AWS 托管 KMS 密钥的别名始终为`alias/aws/sqs`，但例如，自定义 KMS 密钥的别名可以是。`alias/`*`MyAlias`*您可以利用这些 KMS 密钥保护 Amazon SQS 队列中的消息。

> **Note**
>
> 记住以下内容：
>
> - 如果您未指定自定义 KMS 密钥，Amazon SQS 将使用亚马逊 SQS 的 AWS 托管 KMS 密钥。
> - 首次使用为队列指定亚马逊 SQS 的 AWS 托管 KMS 密钥时， AWS KMS 会为亚马逊 SQS 创建 AWS 托管 KMS 密钥。 AWS 管理控制台
> - 或者，首次对启用了 SSE 的队列使用`SendMessage`或`SendMessageBatch`操作时，AWS KMS 会为 Amazon SQS 创建 AWS 托管 KMS 密钥。

您可以创建 KMS 密钥，定义控制如何使用 KMS 密钥的策略，并使用控制 AWS KMS 台的 “客户托管密钥” 部分或`CreateKey` AWS KMS 操作来审计 KMS 密钥的使用情况。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 KMS 密钥和创建密钥。有关 KMS 密钥标识符的更多示例，请参阅 AWS Key Management Service API 参考KeyId中的。有关查找 KMS 密钥标识符的信息，请参阅《AWS Key Management Service 开发人员指南》中的查找密钥 ID 和 ARN。

> **Important**
>
> 使用需要支付额外费用 AWS KMS。有关更多信息，请参阅 估计 AWS KMS 成本 和 AWS Key Management Service 定价。

信封加密

加密的数据的安全性部分取决于如何保护可解密该数据的数据密钥。Amazon SQS 使用 KMS 密钥对数据密钥进行加密，然后加密的数据密钥与加密的消息一起存储。这种使用 KMS 密钥加密数据密钥的做法称为信封加密。

有关封装加密的更多信息，请参阅 AWS Encryption SDK 开发人员指南的封装加密。

Amazon SQS 密钥管理

Amazon SQS 与 AWS Key Management Service (KMS) 集成，用于管理服务器端加密 (SSE) 的 KMS 密钥。有关 SSE 信息和密钥管理定义，请参阅Amazon SQS 中的静态加密。Amazon SQS 使用 KMS 密钥来验证和保护用于加密和解密消息的数据密钥。以下部分提供有关在 Amazon SQS 服务中使用 KMS 密钥和数据密钥的信息。

配置 AWS KMS 权限

每个 KMS 密钥都必须有一个密钥政策。请注意，您无法修改 Amazon SQS 的 AWS 托管 KMS 密钥的密钥策略。此 KMS 密钥的政策包括该账户（获授权可使用 Amazon SQS）中所有主体使用加密队列的权限。

Amazon SQS 根据来电者的 AWS 证书对来电者进行区分，无论他们使用的是不同的 AWS 账户、IAM 用户还是 IAM 角色。此外，具有不同作用策略范围的相同 IAM 角色将被视为不同的请求方。但是，在使用 IAM 角色会话时，仅更改 `RoleSessionName` 而保持相同的 IAM 角色和策略范围不会创建不同的请求方。因此，在指定 AWS KMS 关键策略主体时，请避免仅仅依赖`RoleSessionName`差异，因为这些会话将被视为同一个请求者。

或者，您可以在分配给主体的 IAM 策略中指定所需的权限，而这些主体可以创建和使用加密消息。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的将 IAM 策略用于 AWS KMS。

> **Note**
>
> 虽然您可以配置全局权限以向 Amazon SQS 发送和接收，但 AWS KMS 需要在 IAM 策略`Resource`部分中明确命名特定区域的 KMS 密钥的完整 ARN。

配置 KMS 权限 AWS 务

有几项 AWS 服务充当事件源，可以向 Amazon SQS 队列发送事件。要允许这些事件源使用加密队列，您必须创建客户托管的 KMS 密钥，并在密钥策略中添加权限以使用所需 AWS KMS 的 API 方法。执行以下步骤来配置权限。

> **Warning**
>
> 请注意，更改用于加密 Amazon SQS 消息的 KMS 密钥时，使用旧 KMS 密钥加密的现有消息仍将使用该密钥进行加密。要解密这些消息，您必须保留旧的 KMS 密钥，并确保其密钥政策授予 Amazon SQS 执行 `kms:Decrypt` 和 `kms:GenerateDataKey` 的权限。在将加密新消息的密钥更新为新的 KMS 密钥后，请确保处理所有使用旧的 KMS 密钥加密的现有消息并将其从队列中删除，然后再删除或禁用旧的 KMS 密钥。

1. 创建客户托管的 KMS 密钥。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的创建密钥。
2. 要允许 AWS 服务事件源使用`kms:Decrypt`和 `kms:GenerateDataKey` API 方法，请在 KMS 密钥策略中添加以下语句。

   JSON

```
{
   "Version":"2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Principal": {
            "Service": "service.amazonaws.com"
         },
         "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
         ],
         "Resource": "*"
       }]
}
```

   将上述示例中的“服务”替换为事件源的服务名称。事件源包括以下服务。

事件源	服务名称
亚马逊 CloudWatch 活动	events.amazonaws.com
Amazon S3 事件通知	s3.amazonaws.com
Amazon SNS 主题订阅	sns.amazonaws.com

3. 使用 KMS 密钥的 ARN 配置现有 SSE 队列。
4. 向事件源提供加密队列的 ARN。

配置 AWS KMS 制片人的权限

当数据密钥重用周期过期时，创建者下次调用 SendMessage 或 SendMessageBatch 时也会触发对 kms:Decrypt 和 kms:GenerateDataKey 的调用。对 kms:Decrypt 的调用是在使用新数据密钥之前验证它的完整性。因此，创建者必须具有 KMS 密钥的 kms:Decrypt 和 kms:GenerateDataKey 权限。

将以下语句添加到创建者的 IAM 策略中。请记住，为密钥资源和队列资源使用正确的 ARN 值。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/111112222233333"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
```

```
        }
    ]
}
```

配置 AWS KMS 消费者的权限

当数据密钥重用周期过期时，使用者下一次调用 ReceiveMessage 时也会触发对 kms:Decrypt 的调用，以便在使用新数据密钥之前验证它的完整性。因此，使用者必须对用于加密指定队列中消息的任何 KMS 密钥拥有 kms:Decrypt 权限。如果队列充当死信队列，则使用者还必须具有用于加密源队列中消息的任何 KMS 密钥的 kms:Decrypt 权限。将以下语句添加到使用者的 IAM 策略中。请记住，为密钥资源和队列资源使用正确的 ARN 值。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/111112222233333"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage"
            ],
            "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
        }
    ]
}
```

配置 AWS KMS 权限与副手保护混乱

当密钥政策语句中的主体为AWS 服务主体时，您可以使用 aws:SourceArn 或 aws:SourceAccount 全局条件键以防止出现混淆代理问题。要使用这些条件键，请将值设置为要加密的资源的 Amazon 资源名称 (ARN)。如果您不知道资源的 ARN，请改用 aws:SourceAccount。

在此 KMS 密钥政策中，允许账户 111122223333 拥有的服务中的特定资源调用 KMS 进行 Decrypt 和 GenerateDataKey 操作，这些操作在 Amazon SQS 使用 SSE 期间发生。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "sqs.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:sqs:us-west-1:111122223333:resource"
                    ]
                }
            }
        }
    ]
}
```

使用启用 SSE 的 Amazon SQS 队列时，以下服务支持 aws:SourceArn：

- Amazon SNS
- Amazon S3
- CloudWatch 活动
- AWS Lambda
- CodeBuild
- Connect Customer Profiles
- AWS Auto Scaling

- Amazon Chime

了解数据密钥重用周期

数据密钥重用周期定义 Amazon SQS 重用相同数据密钥的最长持续时间。当数据密钥重用周期结束时，Amazon SQS 会生成一个新的数据密钥。请注意以下有关此重用周期的准则。

- 较短的重复使用期可以提高安全性，但会导致更多的拨打电话 AWS KMS，这可能会产生超出免费套餐的费用。
- 尽管用于加密和解密的数据密钥是单独缓存的，重用周期仍将应用于数据密钥的两个副本。
- 当数据密钥重用期结束时，下一次调用`SendMessage`或`SendMessageBatch`通常会触发对 AWS KMS `GenerateDataKey`方法的调用以获取新的数据密钥。此外，接下来对`SendMessage`和的调用都`ReceiveMessage`将触发对的调用 AWS KMS `Decrypt`，以验证数据密钥的完整性，然后再使用它。
- 委托人（AWS 账户 或用户）不共享数据密钥（由唯一委托人发送的消息始终会获得唯一的数据密钥）。因此，对的调用量 AWS KMS 是数据密钥重复使用期间使用的唯一主体数量的倍数。

估计 AWS KMS 成本

为了预测成本并更好地了解您的 AWS 账单，您可能需要了解 Amazon SQS 使用您的 KMS 密钥的频率。

> **Note**
>
> 尽管以下公式可让您很好地了解预计成本，但由于 Amazon SQS 的分布式特性，实际成本可能更高。

要计算每个队列的 API 请求数 (R)，请使用以下公式：

```
R = (B / D) * (2 * P + C)
```

B 是账单周期（以秒为单位）。

D 是数据密钥重用周期（以秒为单位）。

P 是发送到 Amazon SQS 队列的生成主体的数量。

C 是从 Amazon SQS 队列接收的使用主体数量。

> **Important**
>
> 通常，创建主体产生的费用是使用主体的两倍。有关更多信息，请参阅 了解数据密钥重用周期。
> 如果创建者和使用者具有不同的用户，则费用会增加。

以下是一些示例计算。有关准确的定价信息，请参阅 AWS Key Management Service 定价。

示例 1：计算数量 AWS KMS API 调用 2 个主体和 1 个队列

此示例假定：

- 账单周期为 1 月 1 日 - 31 日（2678400 秒）。
- 数据密钥重用周期设置为 5 分钟（300 秒）。
- 有 1 个队列。
- 有 1 个创建主体和 1 个使用主体。

```
(2,678,400 / 300) * (2 * 1 + 1) = 26,784
```

示例 2：计算数量 AWS KMS 针对多个生产者和使用者以及 2 个队列的 API 调用

此示例假定：

- 账单周期为 2 月 1 日 - 28 日（2419200 秒）。
- 数据密钥重用周期设置为 24 小时（86400 秒）。
- 有 2 个队列。
- 第一个队列有 3 个创建主体和 1 个使用主体。
- 第二个队列有 5 个创建主体和 2 个使用主体。

```
(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532
```

AWS KMS 错误

当您使用 Amazon SQS 和时 AWS KMS，可能会遇到错误。以下参考描述错误和可能的故障排除解决方案。

- 常见 AWS KMS 错误
- AWS KMS Decrypt 错误
- AWS KMS GenerateDataKey 错误

Amazon SQS 中的互联网络流量隐私保护

Amazon SQS 的 Amazon Virtual Private Cloud (Amazon VPC) 端点是 VPC 内的逻辑实体，仅允许连接到 Amazon SQS。VPC 将请求路由到 Amazon SQS 并将响应路由回 VPC。以下部分提供有关使用 VPC 端点和创建 VPC 端点策略的信息。

Amazon SQS 的 Amazon Virtual Private Cloud 端点

如果您使用 Amazon VPC 来托管您的 AWS 资源，则可以在您的 VPC 和亚马逊 SQS 之间建立连接。您可以使用此连接将消息发送到 Amazon SQS 队列，而无需跨公共 Internet。

Amazon VPC 允许您在自定义虚拟网络中启动 AWS 资源。可以使用 VPC 控制您的网络设置，例如 IP 地址范围、子网、路由表和网络网关。有关 VPC 的更多信息，请参阅 Amazon VPC 用户指南。

要将 VPC 连接到 Amazon SQS，您必须先定义一个接口 VPC 端点，该端点可让您将 VPC 连接到其他 AWS 服务。该端点提供了到 Amazon SQS 的可靠、可扩展的连接，无需互联网网关、网络地址转换 (NAT) 实例或 VPN 连接。有关更多信息，请参阅本指南中的教程：从 Amazon Virtual Private Cloud 将消息发送到 Amazon SQS 队列和示例 5：如果不是来自 VPC 端点，则拒绝访问，以及《Amazon VPC 用户指南》中的接口 VPC 端点 (AWS PrivateLink)。

Important

- 您只能将 Amazon Virtual Private Cloud 与 HTTPS Amazon SQS 端点一起使用。
- 当您将 Amazon SQS 配置为从 Amazon VPC 发送消息时，必须启用私有 DNS，并为双栈端点指定格式为 `sqs.us-east-2.amazonaws.com` 或 `sqs.us-east-2.api.aws` 的端点。
- Amazon SQS 还通过 PrivateLink 使用终端节点服务支持 FIPS 终端节点。`com.amazonaws.region.sqs-fips`您可以连接到格式为 `sqs-fips.region.amazonaws.com` 的 FIPS 端点。
- 在 Amazon Virtual Private Cloud 中使用双栈端点时，请求将使用 IPv4 和 IPv6 发送。
- 私有 DNS 不支持传统端点，例如 `queue.amazonaws.com` 或 *us-east-2*`.queue.amazonaws.com`。

为 Amazon SQS 创建 Amazon VPC 端点策略

您可以为 Amazon SQS 创建 Amazon VPC 端点策略，在该策略中指定以下内容：

- 可执行操作的主体。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 端点控制对服务的访问

以下 VPC 端点策略示例指定允许用户 `MyUser` 将消息发送到 Amazon SQS 队列 `MyQueue`。

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

以下各项将被拒绝：

- 其他 Amazon SQS API 操作，例如 `sqs:CreateQueue` 和 `sqs:DeleteQueue`。
- 其他尝试使用该 VPC 端点的用户和规则。
- `MyUser` 将消息发送至不同的 Amazon SQS 队列。

> **Note**
>
> 该用户仍然可以从 VPC 外部使用其他 Amazon SQS API 操作。有关更多信息，请参阅 示例 5：如果不是来自 VPC 端点，则拒绝访问。

使用 Dual-stack （IPv4 和 IPv6）终端节点连接亚马逊 SQS

Dual-stack 端点同时支持 IPv4 和 IPv6 流量。当您向双栈端点发出请求时，端点 URL 会解析为 IPv4 或 IPv6 地址。有关双栈和 FIPS 端点的更多信息，请参阅《SDK 参考指南》。

Amazon SQS 支持区域双栈终端节点，这意味着您必须在终端节点名称中指定 AWS 区域。 Dual-stack 端点名称使用以下命名约定：`sqs.Region.amazonaws.com`。例如，`eu-west-1` 区域的双堆栈端点名称是 `sqs.eu-west-1.amazonaws.com`。

有关 Amazon SQS 端点的完整列表，请参阅 AWS General Reference。

Amazon SQS 中的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可以帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以通过身份验证（登录）和获得授权（具有权限）来使用 Amazon SQS 资源。IAM AWS 服务 是您可以免费使用的。

受众

您的使用方式 AWS Identity and Access Management (IAM) 因您的角色而异：

- 服务用户：如果您无法访问功能，请从管理员处请求权限（请参阅Amazon Simple Queue Service 身份和访问权限故障排查）
- 服务管理员：确定用户访问权限并提交权限请求（请参阅Amazon Simple Queue Service 如何与 IAM 结合使用）
- IAM 管理员：编写用于管理访问权限的策略（请参阅策略最佳实践）

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 AWS 账户根用户、IAM 用户身份或通过代入 IAM 角色进行身份验证。

您可以使用来自 AWS IAM Identity Center （IAM 身份中心）、单点登录身份验证或证书等身份源的 Google/Facebook 证书以联合身份身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的如何登录您的 AWS 账户。

对于编程访问， AWS 提供 SDK 和 CLI 以加密方式签署请求。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的AWS 签名版本 4。

AWS 账户 根用户

创建时 AWS 账户，首先要使用一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关需要根用户凭证的任务，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

最佳做法是，要求人类用户使用与身份提供商的联合身份验证来 AWS 服务 使用临时证书进行访问。

联合身份是来自您的企业目录、Web 身份提供商的用户 Directory Service ，或者 AWS 服务 使用来自身份源的证书进行访问的用户。联合身份代入可提供临时凭证的角色。

要集中管理访问权限，建议使用。 AWS IAM Identity Center有关更多信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？。

IAM 用户和群组

IAM 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户 AWS 使用联合身份提供商以使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。 AWS 委托人提出请求时评估这些政策。大多数策略以 JSON 文档的 AWS 形式存储在。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

Identity-based 政策

Identity-based 策略是您附加到身份（用户、群组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

Identity-based 策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

Resource-based 政策

Resource-based 策略是您附加到资源的 JSON 策略文档。示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。您必须在基于资源的策略中指定主体。

Resource-based 策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 的 AWS 托管策略。

其他策略类型

AWS 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，请参阅《 IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略（SCP）– 指定 AWS Organizations中组织或组织单元的最大权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略（RCP）– 设置对账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的资源控制策略（RCP）。
- 会话策略 – 在为角色或联合用户创建临时会话时，作为参数传递的高级策略。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解如何在涉及多种策略类型时 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

管理 Amazon SQS 中的访问权限概述

每个 AWS 资源都归属于 AWS 账户，创建或访问资源的权限受权限策略的约束。账户管理员可以向 IAM 身份（用户、组和角色）附加权限策略，某些服务（如 Amazon SQS）也支持向资源附加权限策略。

> **Note**
>
> 账户管理员（或管理员用户）是具有管理权限的用户。有关更多信息，请参阅《IAM 用户指南》中的 IAM 最佳实操。

在授予权限时，由您指定哪些用户获得权限，获得对哪些资源的权限，以及您允许对这些资源执行哪些具体操作。

Amazon Simple Queue Service 资源和操作

在 Amazon SQS 中，唯一的资源是队列。在策略中，可使用 Amazon 资源名称 (ARN) 标识策略应用到的资源。以下资源具有与之关联的唯一 ARN：

资源类型	ARN 格式
队列	`arn:aws:sqs:region:account_id:queue_name`

以下是队列的 ARN 格式的示例：

- 在美国东部（俄亥俄州）my_queue地区命名的队列的 ARN，属于 AWS 账户 123456789012：

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- Amazon SQS 支持的每个不同区域中名为 my_queue 的队列的 ARN：

```
arn:aws:sqs:*:123456789012:my_queue
```

- 使用 ? 或 * 作为队列名称通配符的 ARN。在以下示例中，ARN 匹配前缀为 my_prefix_ 的所有队列：

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

调用 GetQueueAttributes 操作可以获取现有队列的 ARN 值。QueueArn 属性的值即为队列的 ARN。有关 ARN 的更多信息，请参阅《IAM 用户指南》中的 IAM ARN。

Amazon SQS 提供用于处理队列资源的一组操作。有关更多信息，请参阅 Amazon SQS API 权限：操作和资源参考。

了解资源所有权

无论谁创建了资源，都 AWS 账户 拥有在账户中创建的资源。具体而言，资源所有者是对资源创建请求进行身份验证的主体实体（即根账户、用户或 IAM 角色）的 AWS 账户 。以下示例说明了它的工作原理：

- 如果您使用您的 AWS 账户 根账户证书创建 Amazon SQS 队列，则您 AWS 账户 是该资源的所有者（在 Amazon SQS 中，资源是亚马逊 SQS 队列）。
- 如果您在中创建用户 AWS 账户 并向该用户授予创建队列的权限，则该用户可以创建队列。但是，该用户所属的 AWS 账户 是此队列资源的所有者。
- 如果您在 AWS 账户 拥有创建 Amazon SQS 队列权限的 IAM 角色中创建，则任何能够代入该角色的人都可以创建队列。您的 AWS 账户 （角色所属的）拥有队列资源。

管理对 资源的访问

权限策略描述了授予给账户的权限。下一节介绍创建权限策略时的可用选项。

> **Note**
>
> 本节讨论如何在 Amazon SQS 范围内使用 IAM。这里不提供有关 IAM 服务的详细信息。有关完整的 IAM 文档，请参阅 IAM 用户指南中的什么是 IAM？。有关 IAM 策略语法和说明的信息，请参阅《IAM 用户指南》中的 AWS IAM 策略参考。

附加到 IAM 身份的策略称作基于身份 的策略（IAM 策略），附加到资源的策略称作基于资源 的策略。

Identity-based 政策

可通过两种方式向用户授予访问 Amazon SQS 队列的权限：使用 Amazon SQS 策略系统和使用 IAM 策略系统。您可以使用任一系统或这两种系统来将策略附加到用户或角色。在大多数情况下，使用任一系统都能获得相同的结果。例如，您可以执行以下操作：

- 将权限策略附加到账户中的用户或组 - 要向用户授予创建 Amazon SQS 队列的权限，请将权限策略附加到用户或用户所属的组。
- 将权限策略附加到另一个用户中 AWS 账户-您可以将权限策略附加到另一个用户中， AWS 账户 以允许他们与 Amazon SQS 队列进行交互。但跨账户权限不能应用于以下操作：

  Cross-account 权限不适用于以下操作：

  - AddPermission
  - CancelMessageMoveTask
  - CreateQueue
  - DeleteQueue
  - ListMessageMoveTask
  - ListQueues
  - ListQueueTags
  - RemovePermission
  - SetQueueAttributes
  - StartMessageMoveTask
  - TagQueue
  - UntagQueue

  要授予对这些操作的访问权限，用户必须属于拥有该 Amazon SQS 队列的同一 AWS 账户 。
- 将权限策略附加到角色（授予跨账户权限） – 要向 SQS 队列授予跨账户权限，您必须同时结合使用 IAM 策略和基于资源的策略：

  1. 在账户 A（队列所有者）中:
     - 将基于资源的策略附加到 SQS 队列。该策略必须向账户 B 中的主体（例如 IAM 角色）显式授予所需权限（例如 SendMessage、ReceiveMessage）。
  2. 在账户 A 中，创建一个 IAM 角色：
     - 一个信任策略，允许账户 B 或 AWS 服务 代入该角色。

> **Note**
>
> 如果您想让 AWS 服务 （例如 Lambda 或EventBridge ）代入该角色，请在信任策略中指定服务委托人（例如lambda.amazonaws.com ）。

- 一个基于身份的策略，向该代入角色授予与队列交互的权限。

3. 在账户 B 中，授予在账户 A 中代入角色的权限。

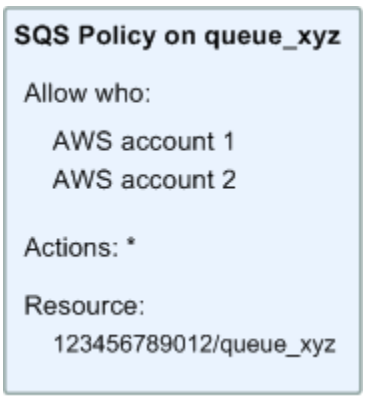
您必须将队列的访问策略配置为允许跨账户主体。仅使用基于 IAM 身份的策略不足以实现跨账户访问 SQS 队列。

有关使用 IAM 委托权限的更多信息，请参阅《IAM 用户指南》中的访问权限管理。

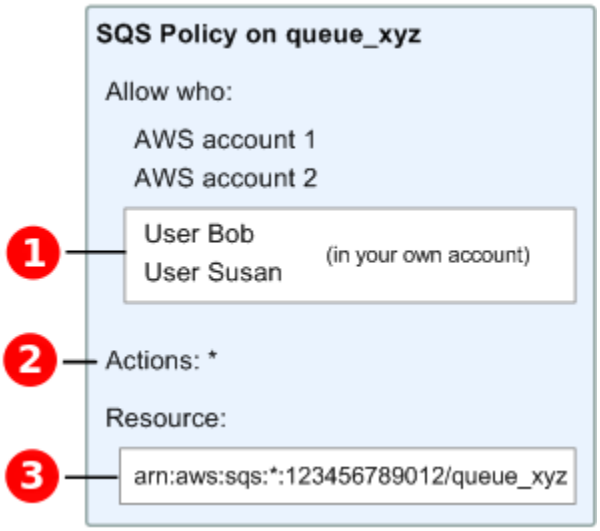
尽管 Amazon SQS 使用 IAM 策略，但它拥有自己的策略基础架构。您可以对队列使用 Amazon SQS 策略来指定哪些 AWS 账户有权访问该队列。您可以指定访问类型和条件（例如，条件是如果请求早于 2010 年 12 月 31 日，即授予使用 `SendMessage` 和 `ReceiveMessage` 的权限）。您可以为其授予权限的特定操作是整个 Amazon SQS 操作列表的子集。如果编写 Amazon SQS 策略并指定 * 以“允许所有 Amazon SQS 操作”，则意味着用户可以在此子集中执行所有操作。

下图说明了这些基本 Amazon SQS 策略中涵盖操作子集的一个策略的概念。该政策适用于`queue_xyz`，它授予 AWS 账户 1 和 AWS 账户 2 对指定队列使用任何允许操作的权限。

> **Note**
>
> 策略中的资源指定为`123456789012/queue_xyz`，其中`123456789012`是拥有该 AWS 队列的账户的账户 ID。



随着 IAM 以及用户和 Amazon 资源名称 (ARN) 概念的推出，SQS 策略发生了一些变化。以下示意图和表格描述了这些变化。



1 有关向不同账户中的用户授予权限的信息，请参阅 IAM 用户指南中的教程：使用 IAM 角色跨 AWS 账户委派访问权限。

2 * 中包含的操作子集已扩展。有关允许的操作的列表，请参阅Amazon SQS API 权限：操作和资源参考。

3 您可以使用 Amazon 资源名称 (ARN) 指定资源，这是在 IAM 策略中指定资源的标准方法。有关 Amazon SQS 队列的 ARN 格式的信息，请参阅 Amazon Simple Queue Service 资源和操作。

例如，根据上图中的 Amazon SQS 政策，任何拥有 AWS 账户 1 或 AWS 账户 2 安全证书的人都可以访问。queue_xyz此外，您自己的 AWS 账户（ID 为 123456789012）中的用户 Bob 和 Susan 也可以访问该队列。

在推出 IAM 之前，Amazon SQS 会自动向某个队列的创建者授予对该队列的完全控制权限（即访问该队列中所有可能的 Amazon SQS 操作）。现在，除非创建者使用 AWS 安全凭证，否则上述情况将不再出现。如果任何有权创建队列的用户希望对所创建的队列执行任何操作，还必须拥有使用其他 Amazon SQS 操作的权限。

下面是一个示例策略，该策略允许用户使用所有 Amazon SQS 操作，但只能对其名称的前缀为文字字符串 bob_queue_ 的队列使用。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
   }]
}
```

有关更多信息，请参阅《IAM 用户指南》中的将策略用于 Amazon SQS以及身份（用户、组和角色）。

指定策略元素：操作、效果、资源和主体

对于每种 Amazon Simple Queue Service 资源，该服务都定义一组操作。为授予这些操作的权限，Amazon SQS 定义一组可以在策略中指定的操作。

> **Note**
>
> 执行一个 操作可能需要多个操作的权限。在授予特定操作的权限时，您也可以标识允许或拒绝对其执行操作的资源。

以下是最基本的策略元素：

- Resource（资源）- 在策略中，您可以使用 Amazon 资源名称 (ARN) 标识策略应用到的资源。
- 操作 - 您可以使用操作关键字标识要允许或拒绝的资源操作。例如，`sqs:CreateQueue` 权限允许用户执行 Amazon Simple Queue Service `CreateQueue` 操作。
- 效果：您可以指定当用户请求特定操作（可以是允许或拒绝）时的效果。如果您没有显式授予对资源的访问权限，则隐式拒绝访问。您也可明确拒绝对资源的访问，这样可确保用户无法访问该资源，即使有其他策略授予了访问权限的情况下也是如此。
- 主体 – 在基于身份的策略（IAM 策略）中，附加了策略的用户是隐式主体。对于基于资源的策略，您可以指定要接收权限的用户、账户、服务或其他实体（仅适用于基于资源的策略）。

要详细了解 Amazon SQS 策略语法和描述，请参阅《IAM 用户指南》中的 AWS IAM 策略参考。

有关所有 Amazon Simple Queue Service 操作及其适用资源的表格，请参阅 Amazon SQS API 权限：操作和资源参考。

Amazon Simple Queue Service 如何与 IAM 结合使用

在使用 IAM 管理对 Amazon SQS 的访问权限之前，您应该了解哪些 IAM 特征可用于 Amazon SQS。

可与 Amazon Simple Queue Service 结合使用的 IAM 特征

IAM 功能	Amazon SQS 支持
Identity-based 政策	是
Resource-based 政策	是
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否
ABAC（策略中的标签）	部分
临时凭证	是
转发访问会话（FAS）	是
服务角色	是
Service-linked 角色	否

要详细了解 Amazon SQS 和其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS 服务。

访问控制

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，但它们不使用 JSON 策略文档格式。

亚马逊 S3 和亚马逊 VPC 是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

> **Note**
>
> 重要的是要明白，所有人 AWS 账户 都可以将权限委托给其账户下的用户。 Cross-account access 允许您共享 AWS 资源访问权限，而无需管理其他用户。有关使用跨账户访问的信息，请参阅 IAM 用户指南中的启用 Cross-Account 访问权限。
> 有关 Amazon SQS 自定义策略中跨内容权限和条件键的更多详细信息，请参阅Amazon SQS 自定义策略的限制。

Identity-based 亚马逊 SQS 的政策

支持基于身份的策略：是

Identity-based 策略是 JSON 权限策略文档，您可以将其附加到身份，例如 IAM 用户、用户组或角色。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

Identity-based 亚马逊 SQS 的策略示例

要查看 Amazon SQS 基于身份的策略示例，请参阅策略最佳实践。

Resource-based 亚马逊 SQS 内部的政策

支持基于资源的策略：是

Resource-based 策略是您附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户访问，您可以将整个账户或其它账户中的 IAM 实体指定为基于资源的策略中的主体。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

Amazon SQS 的策略操作

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问哪些内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。在策略中包含操作以授予执行关联操作的权限。

要查看 Amazon SQS 操作的列表，请参阅服务授权参考中的 Amazon Simple Queue Service 定义的资源。

Amazon SQS 中的策略操作在操作前面使用以下前缀：

```
sqs
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "sqs:action1",
      "sqs:action2"
         ]
```

要查看 Amazon SQS 基于身份的策略示例，请参阅策略最佳实践。

Amazon SQS 的策略资源

支持策略资源：是

管理员可以使用 AWS JSON 策略来指定谁有权访问哪些内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

`Resource` JSON 策略元素指定要向其应用操作的一个或多个对象。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于不支持资源级权限的操作，请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

要查看 Amazon SQS 资源类型及其 ARN 的列表，请参阅服务授权参考中的 Amazon Simple Queue Service 定义的操作。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 Amazon Simple Queue Service 定义的资源。

要查看 Amazon SQS 基于身份的策略示例，请参阅策略最佳实践。

Amazon SQS 的策略条件键

支持特定于服务的策略条件键：是

管理员可以使用 AWS JSON 策略来指定谁有权访问哪些内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

`Condition` 元素根据定义的条件指定语句何时执行。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。要查看所有 AWS 全局条件密钥，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

要查看 Amazon SQS 条件键的列表，请参阅服务授权参考中的 Amazon Simple Queue Service 的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon Simple Queue Service 定义的资源。

要查看 Amazon SQS 基于身份的策略示例，请参阅策略最佳实践。

Amazon SQS 中的 ACL

支持 ACL：否

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，但它们不使用 JSON 策略文档格式。

ABAC 与 Amazon SQS

支持 ABAC（策略中的标签）：部分支持

Attribute-based 访问控制 (ABAC) 是一种授权策略，它根据称为标签的属性定义权限。您可以将标签附加到 IAM 实体和 AWS 资源，然后设计 ABAC 策略，允许在委托人的标签与资源上的标签匹配时进行操作。

要基于标签控制访问，您需要使用 `aws:ResourceTag/`*`key-name`*`aws:RequestTag/`*`key-name`* 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的使用 ABAC 授权定义权限。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

将临时凭证用于 Amazon SQS

支持临时凭证：是

临时证书提供对 AWS 资源的短期访问权限，并在您使用联合或切换角色时自动创建。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 IAM 临时安全凭证和使用 IAM 的AWS 服务。

Amazon SQS 的转发访问会话

支持转发访问会话（FAS）：是

前向访问会话 (FAS) 使用调用主体的权限 AWS 服务，同时请求 AWS 服务 向下游服务发出请求。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

Amazon SQS 的服务角色

支持服务角色：是

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会破坏 Amazon SQS 的功能。仅当 Amazon SQS 提供相关指导时才编辑服务角色。

Service-linked 亚马逊 SQS 的角色

支持服务相关角色：否

服务相关角色是一种链接到的服务角色。 AWS 服务该服务可以代入代表您执行操作的角色。 Service-linked 角色显示在您的中 AWS 账户 且归服务所有。IAM 管理员可以查看但不能编辑服务关联角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找Service-linked 角色列Yes中包含的服务。选择是链接以查看该服务的服务相关角色文档。

亚马逊 SQS 更新至 AWS 托管策略

要向用户、组和角色添加权限，与自己编写策略相比，使用 AWS 托管式策略更简单。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管式策略。这些策略涵盖常见使用案例，可在您的 AWS 账户中使用。有关 AWS 托管策略的更多信息，请参阅 IAM 用户指南中的AWS 托管策略。

AWS 服务维护和更新 AWS 托管策略。您无法更改 AWS 托管策略中的权限。服务偶尔会向 AWS 托管策略添加额外的权限以支持新功能。此类更新会影响附加策略的所有身份（用户、组和角色）。当新功能启动或新操作可用时，服务最有可能更新 AWS 托管策略。服务不会从 AWS 托管策略中移除权限，因此策略更新不会破坏您的现有权限。

此外， AWS 支持跨多个服务的工作职能的托管策略。例如，ReadOnlyAccess AWS 托管策略提供对所有 AWS 服务和资源的只读访问权限。当服务启动新功能时， AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅 IAM 用户指南中的适用于工作职能的AWS 托管式策略。

AWS 管理策略： AmazonSQSFullAccess

您可以将 `AmazonSQSFullAccess` 策略附加到 Amazon SQS 身份。此策略授予允许完全访问 Amazon SQS 的权限。

要查看此策略的权限，请参阅《AWS 托管式策略参考》中的 AmazonSQSFullAccess。

AWS 管理策略： AmazonSQSReadOnlyAccess

您可以将 `AmazonSQSReadOnlyAccess` 策略附加到 Amazon SQS 身份。此策略授予允许对 Amazon SQS 进行只读访问的权限。

要查看此策略的权限，请参阅《AWS 托管式策略参考》中的 AmazonSQSReadOnlyAccess。

AWS 管理策略： SQSUnlockQueuePolicy

如果您错误地将成员账户的队列策略配置为拒绝所有用户访问您的 Amazon SQS 队列，则可以使用`SQSUnlockQueuePolicy` AWS 托管策略解锁队列。

有关如何删除拒绝所有委托人访问 Amazon SQS 队列的错误队列策略的更多信息，请参阅 IAM 用户指南中的对 AWS Organizations 成员账户执行特权任务。

亚马逊 SQS 更新至 AWS 托管策略

查看自该服务开始跟踪这些更改以来 Amazon SQS AWS 托管策略更新的详细信息。有关此页面更改的自动提示，请订阅 Amazon SQS 文档历史记录页面上的 RSS 源。

更改	描述	日期
SQSUnlockQueuePolicy	Amazon SQS 添加了一项名为的新 AWS托管策略，该策略SQSUnlockQueuePolicy 旨在解锁队列并删除拒绝所有委托人访问 Amazon SQS 队列的错误配置的队列策略。	2024 年 11 月 15 日
AmazonSQSReadOnlyAccess	Amazon SQS 添加了 ListQueueTags 操作，该操作用于检索与指定 Amazon SQS 队列关联的所有标签。它让您能够查看出于组织或元数据目的分配给队列的键值对。此操作与 ListQueueTags API 操作关联。	2024 年 6 月 20 日
AmazonSQSReadOnlyAccess	Amazon SQS 添加了一个新操作，允许您列出特定源队列下最新的消息移动任务（最多 10 个）。此操作与 ListMessageMoveTasks API 操作关联。	2023 年 6 月 9 日

Amazon Simple Queue Service 身份和访问权限故障排查

您可以使用以下信息，帮助诊断和修复在使用 Amazon SQS 和 IAM 时可能遇到的常见问题。

我无权在 Amazon SQS 中执行操作

如果您收到一个错误，指明您无权执行某个操作，则必须更新策略以允许您执行该操作。

当 mateojackson 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 sqs:*GetWidget* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 sqs:GetWidget on resource: my-example-widget
```

在此情况下，Mateo 的策略必须更新以允许其使用 sqs:*GetWidget* 操作访问 *my-example-widget* 资源。

如果需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到一个错误，指明您无权执行 iam:PassRole 操作，则必须更新策略以允许您将角色传递给 Amazon SQS。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 Amazon SQS 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人进来 AWS 账户 访问我的 Amazon SQS 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon SQS 是否支持这些特征，请参阅Amazon Simple Queue Service 如何与 IAM 结合使用。
- 要了解如何提供对您所拥有资源的访问权限，请参阅 IAM 用户指南中的向您拥有的另一 AWS 账户个 IAM 用户提供访问权限。 AWS 账户
- 要了解如何向第三方提供对资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方 AWS 账户 拥有的提供访问权限。
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

我想解锁我的队列

如果您 AWS 账户 属于某个组织，则 AWS Organizations 策略可能会阻止您访问 Amazon SQS 资源。默认情况下， AWS Organizations 策略不会阻止向 Amazon SQS 发出的任何请求。但是，请确保您的 AWS Organizations 策略未配置为阻止访问 Amazon SQS 队列。有关如何检查 AWS Organizations 政策的说明，请参阅《AWS Organizations 用户指南》中列出所有策略。

此外，如果您错误地将成员账户的队列策略配置为拒绝所有用户访问您的 Amazon SQS 队列，则可以通过在 IAM 中为该成员账户启动特权会话来解锁队列。启动特权会话后，您可以删除配置错误的队列策略以重新获得对队列的访问权限。有关更多信息，请参阅 IAM 用户指南中的对 AWS Organizations 成员账户执行特权任务。

将策略用于 Amazon SQS

本主题提供基于身份的策略示例，在这些示例中，账户管理员可以向 IAM 身份（用户、组和角色）附加权限策略。

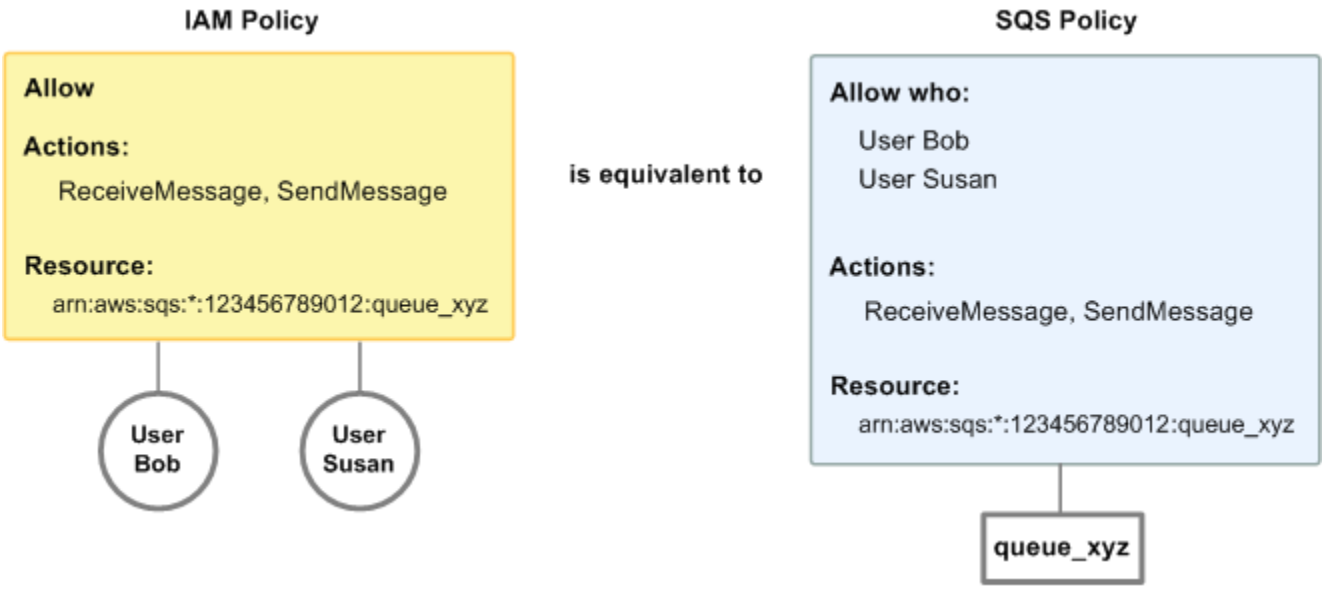
> **Important**
>
> 我们建议您首先阅读以下介绍性主题，这些主题讲解了管理 Amazon Simple Queue Service 资源访问权限的基本概念和选项。有关更多信息，请参阅 管理 Amazon SQS 中的访问权限概述。
>
> 除 `ListQueues` 外，所有 Amazon SQS 操作均支持资源级权限。有关更多信息，请参阅 Amazon SQS API 权限：操作和资源参考。

使用 Amazon SQS 和 IAM 策略

您可以通过两种方式向用户授予访问 Amazon SQS 资源的权限：使用 Amazon SQS 策略系统（基于资源的策略）和使用 IAM 策略系统（基于身份的策略）。您可以使用一种或两种方法，但 `ListQueues` 操作除外，这是一种区域性权限，只能在 IAM 策略中设置。

例如，下图显示了等效的 IAM 策略和 Amazon SQS 策略。IAM 政策授予您 AWS 账户`queue_xyz`中调用的队列的 Amazon SQS `ReceiveMessage` 和`SendMessage`操作的权限，该政策附加到名为 Bob 和 Susan 的用户（Bob 和 Susan 拥有政策中规定的权限）。此 Amazon SQS 策略还向 Bob 和 Susan 授予对同一队列进行 `ReceiveMessage` 和 `SendMessage` 操作的权限。

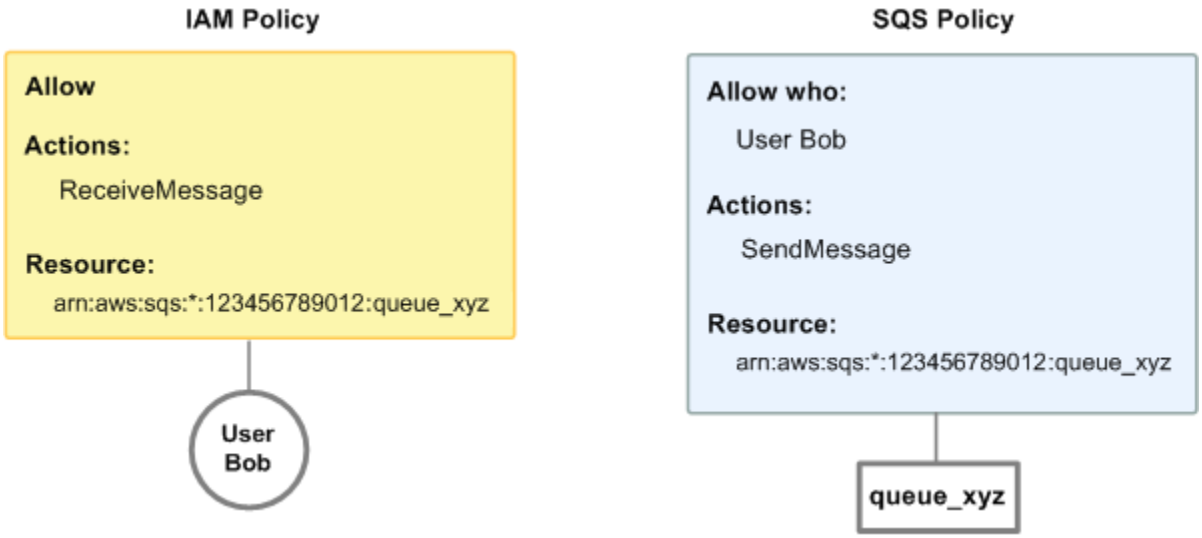
> **Note**
>
> 以下示例显示了不带条件的简单策略。您可以在上述任一策略中指定特定条件，并获得同样的结果。



IAM 和 Amazon SQS 策略之间有一个主要区别：亚马逊 SQS 策略系统允许您向其他 AWS 账户授予权限，而 IAM 不能。

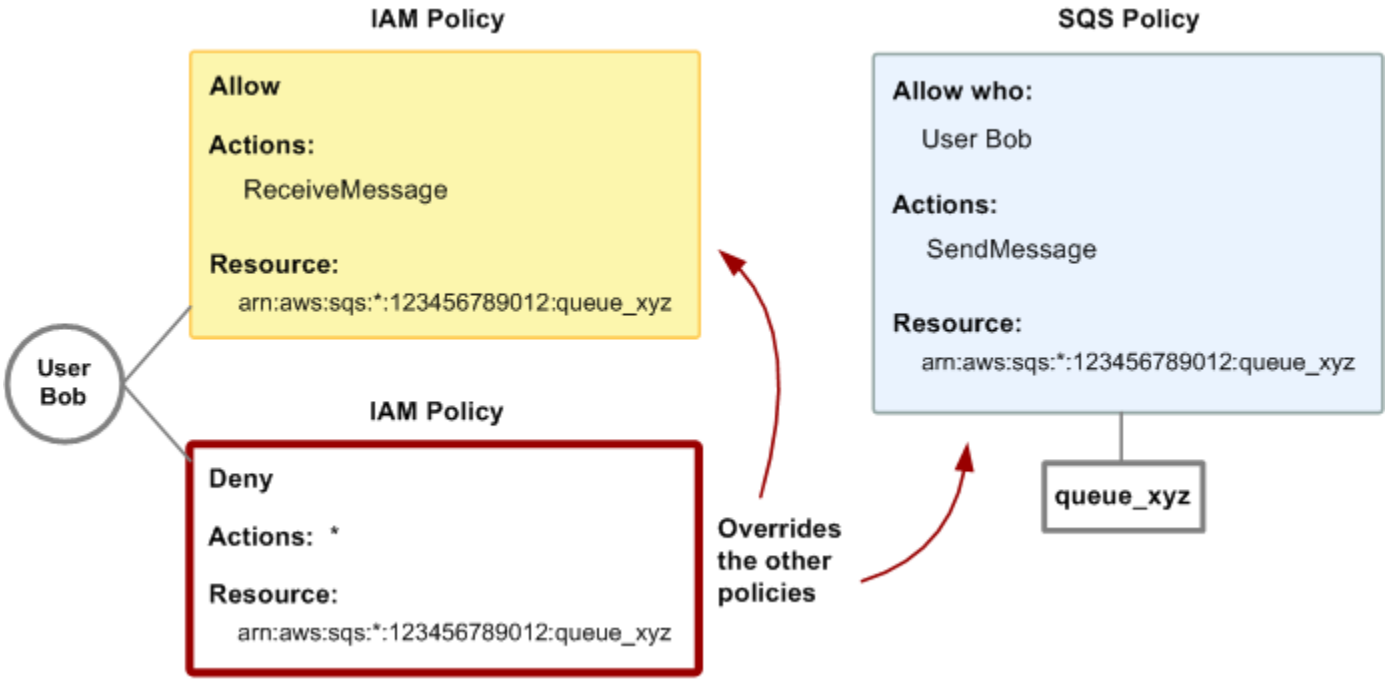
您可以自行决定如何综合使用上述两种系统来管理您的权限。以下示例展示这两种策略系统是如何共同运行的。

- 在第一个示例中，Bob 同时拥有 IAM 策略和适用于其账户的 Amazon SQS 策略。IAM 策略向其账户授予对 queue_xyz 执行 ReceiveMessage 操作的权限，而 Amazon SQS 策略向其账户授予对同一队列执行 SendMessage 操作的权限。下图阐明了这一概念。



如果 Bob 向 queue_xyz 发送 ReceiveMessage 请求，则 IAM 策略将允许执行该操作。如果 Bob 向 queue_xyz 发送 SendMessage 请求，则 Amazon SQS 策略将允许执行该操作。

- 在第二个示例中，Bob 滥用他对 queue_xyz 的访问权限，因此有必要删除他对该队列的所有访问权限。最简单的方法是添加一个策略，拒绝他访问该队列的所有操作。此策略会覆盖另外两个策略，因为显式 deny 始终覆盖 allow。有关策略评估逻辑的更多信息，请参阅[将自定义策略与 Amazon SQS 访问策略语言配合使用]。下图阐明了这一概念。



您还可以向 Amazon SQS 策略中添加一条额外语句，拒绝 Bob 以任何方式访问该队列。添加一条 IAM 策略拒绝 Bob 访问该队列也具有同样的效果。有关涉及 Amazon SQS 操作和资源的策略示例，请参阅 Amazon SQS 策略的基本示例。有关编写 Amazon SQS 策略的更多信息，请参阅将自定义策略与 Amazon SQS 访问策略语言配合使用。

使用 Amazon SQS 控制台所需的权限

希望使用 Amazon SQS 控制台的用户必须具有在用户的 AWS 账户中使用 Amazon SQS 队列的最小权限集。例如，用户必须具有调用 `ListQueues` 操作的权限才能列出队列，或者必须具有调用 `CreateQueue` 操作的权限才能创建队列。要将 Amazon SQS 队列订阅到 Amazon SNS 主题，则除了 Amazon SQS 权限之外，控制台还需要针对 Amazon SNS 操作的权限。

如果创建比必需的最低权限更为严格的 IAM 策略，对于附加了该 IAM 策略的用户，控制台可能无法按预期方式运行。

您无需为仅调用 AWS CLI 或 Amazon SQS 操作的用户提供最低控制台权限。

Identity-based 亚马逊 SQS 的策略示例

默认情况下，用户和角色没有创建或修改 Amazon SQS 资源的权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略（控制台）。

有关 Amazon SQS 定义的操作和资源类型的详细信息，包括每种资源类型的 ARN 格式，请参阅服务授权参考中的 Amazon Simple Queue Service 的操作、资源和条件键。

> **Note**
>
> 在为 Amazon EC2 Auto Scaling 配置生命周期挂钩时，您无需编写策略将消息发送至 Amazon SQS 队列。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 Auto Scaling Lifecycle Hooks。

策略最佳实践

Identity-based 策略决定是否有人可以在您的账户中创建、访问或删除 Amazon SQS 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授权，请使用为许多常见用例授予权限的AWS 托管策略。它们可以在您的 AWS 账户. 我们建议您通过定义特定于您的用例的 AWS 客户管理策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略或工作职能的AWS 托管策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性：IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA) — 如果您的场景需要 IAM 用户或根用户 AWS 账户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 Amazon SQS 控制台

要访问 Amazon Simple Queue Service 控制台，您必须具有一组最低的权限。这些权限必须允许您在中列出和查看有关 Amazon SQS 资源的详细信息。 AWS 账户如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

您无需为仅调用 AWS CLI 或 AWS API 的用户提供最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以使用 Amazon SQS 控制台，还要将 Amazon SQS `AmazonSQSReadOnlyAccess` AWS 托管策略附加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上完成此操作或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

允许用户创建队列

在以下示例中，我们为 Bob 创建了一条策略，允许他访问所有 Amazon SQS 操作，但是仅限于名称前缀为文本字符串 `alice_queue_` 的队列。

Amazon SQS 不会自动向队列创建者授予使用该队列的权限。因此，除了 IAM 策略中的 `CreateQueue` 操作，我们还必须向 Bob 显式授予使用所有 Amazon SQS 操作的权限。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
   }]
}
```

允许开发人员向共享队列写入消息

在以下示例中，我们为开发人员创建了一个群组并附加了一项策略，允许该群组使用 Amazon SQS SendMessage 操作，但只能使用属于指定 AWS 账户 且命名的MyCompanyQueue队列。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"
   }]
}
```

您可以使用 *（而不是 SendMessage）向主体授予对共享队列执行以下操作的权限：ChangeMessageVisibility、DeleteMessage、GetQueueAttributes、GetQueueUrl、Recei和 SendMessage。

> **Note**
>
> 虽然 * 包含其他权限类型提供的访问权限，但 Amazon SQS 会单独考虑这些权限。例如，可以向用户同时授予 * 和 SendMessage 权限，即使 * 包含 SendMessage 提供的访问权限，也是如此。
> 此概念在您删除权限时也适用。如果主体只有 * 权限，则请求删除 SendMessage 权限不会 为主体留下除此以外的一切 权限。相反，该请求不起作用，因为主体不具

有显式 SendMessage 权限。要只为主体留下 ReceiveMessage 权限，请先添加 ReceiveMessage 权限，然后删除 * 权限。

允许管理人员获取队列的一般大小

在以下示例中，我们为经理创建了一个群组并附加了一项策略，允许该群组对属于指定 AWS 账户的所有队列使用 Amazon SQS GetQueueAttributes 操作。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:GetQueueAttributes",
      "Resource": "*"
   }]
}
```

允许合作伙伴向指定队列发送消息

您可以使用 Amazon SQS 策略或 IAM 策略完成此任务。如果您的合作伙伴有 AWS 账户，则使用 Amazon SQS 政策可能会更容易。但是，合作伙伴公司中拥有 AWS 安全证书的任何用户都可以向队列发送消息。如果需要仅向特定用户或应用程序授予访问权限，则必须像对待公司内部的用户那样对待合作伙伴，使用 IAM 策略而不是 Amazon SQS 策略。

本示例将执行以下操作：

1. 创建一个名 WidgetCo 为代表合作伙伴公司的群组。
2. 为合作伙伴公司中需要访问权限的特定用户或应用程序创建用户。
3. 将 用户添加到 组。
4. 挂载一条策略，仅允许该组对名为 SendMessage 的队列执行 WidgetPartnerQueue 操作。

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement": [{
         "Effect": "Allow",
         "Action": "sqs:SendMessage",
         "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
   }]
}
```

Amazon SQS 策略的基本示例

本部分显示了常见 Amazon SQS 使用案例的示例策略。

在将每个策略附加到用户时，可使用控制台验证该策略的效果。最初，用户没有权限并且无法在控制台中执行任何操作。在将策略附加到用户时，可以验证用户是否能在控制台中执行各种操作。

> **Note**
>
> 我们建议您使用两个浏览器窗口：一个用于授予权限，另一个用于在向用户授予权限时 AWS 管理控制台 使用用户的凭据登录，以验证权限。

示例 1：向其中一个授予一个权限 AWS 账户

以下示例策略向 AWS 账户 编号111122223333授予在美国东部（俄亥俄州）区域命名的444455556666/queue1队列的SendMessage权限。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_SendMessage",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": "sqs:SendMessage",
```

```
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue1"
   }]
}
```

示例 2：向其中一个授予两个权限 AWS 账户

以下示例策略为名为的队列授予 AWS 账户 数字111122223333SendMessage和ReceiveMessage权限444455556666/queue1。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_Send_Receive",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:*:444455556666:queue1"
   }]
}
```

示例 3：将所有权限授予两个 AWS 账户

以下示例策略授予两种不同的 AWS 账户 数字（111122223333和444455556666）权限，允许他们使用 Amazon SQS 允许共享访问在美国东部（俄亥俄州）区域命名的123456789012/queue1队列的所有操作。

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AllActions",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333",
            "444455556666"
         ]
      },
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:queue1"
   }]
}
```

示例 4：向角色和用户名授予跨账户权限

以下示例策略授予role1111122223333跨账户使用所有允许共享访问美国东部（俄亥俄州）区域命名的123456789012/queue1队列的 Amazon SQS 操作的权限。username1 AWS 账户

Cross-account 权限不适用于以下操作：

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTask
- ListQueues
- ListQueueTags
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

JSON

```
{
   "Version":"2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AllActions",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "arn:aws:iam::111122223333:role/role1",
            "arn:aws:iam::111122223333:user/username1"
         ]
      },
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:queue1"
   }]
}
```

示例 5：向所有用户授予一项权限

以下示例策略向所有用户（匿名用户）授予对名为 111122223333/queue1 的队列的 ReceiveMessage 权限。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_ReceiveMessage",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:ReceiveMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1"
   }]
}
```

示例 6：向所有用户授予有时间限制的权限

以下示例策略向所有用户（匿名用户）授予对名为 111122223333/queue1 的队列的 ReceiveMessage 权限，但有效时间仅限于 2009 年 1 月 31 日中午 12:00 至下午 3:00 期间。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_ReceiveMessage_TimeLimit",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:ReceiveMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "DateGreaterThan" : {
            "aws:CurrentTime":"2009-01-31T12:00Z"
         },
         "DateLessThan" : {
            "aws:CurrentTime":"2009-01-31T15:00Z"
         }
      }
   }]
}
```

示例 7：向 CIDR 范围内的所有用户授予所有权限

以下示例策略向所有用户（匿名用户）授予对名为 111122223333/queue1 的队列使用可以共享的所有可能的 Amazon SQS 操作的权限，但条件是请求必须来自于 192.0.2.0/24 CIDR 范围。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_AllActions_AllowlistIP",
      "Effect": "Allow",
```

```
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"192.0.2.0/24"
         }
      }
   }]
}
```

示例 8：不同 CIDR 范围内用户的允许列表和阻止列表权限

以下策略示例具有两项陈述：

- 第一个语句向 192.0.2.0/24 CIDR 范围（192.0.2.188 除外）内的所有用户（匿名用户）授予对名为 111122223333/queue1 的队列使用 SendMessage 操作的权限。
- 第二个语句阻止 12.148.72.0/23 CIDR 范围内的所有用户（匿名用户）使用该队列。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_SendMessage_IPLimit",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"192.0.2.0/24"
         },
         "NotIpAddress" : {
            "aws:SourceIp":"192.0.2.188/32"
         }
      }
   }, {
      "Sid":"Queue1_AnonymousAccess_AllActions_IPLimit_Deny",
```

```
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"12.148.72.0/23"
         }
      }
   }]
}
```

将自定义策略与 Amazon SQS 访问策略语言配合使用

要仅根据 AWS 账户 ID 授予基本权限（例如SendMessage或 ReceiveMessage），您无需编写自定义策略。只需使用 Amazon SQS AddPermission 操作即可。

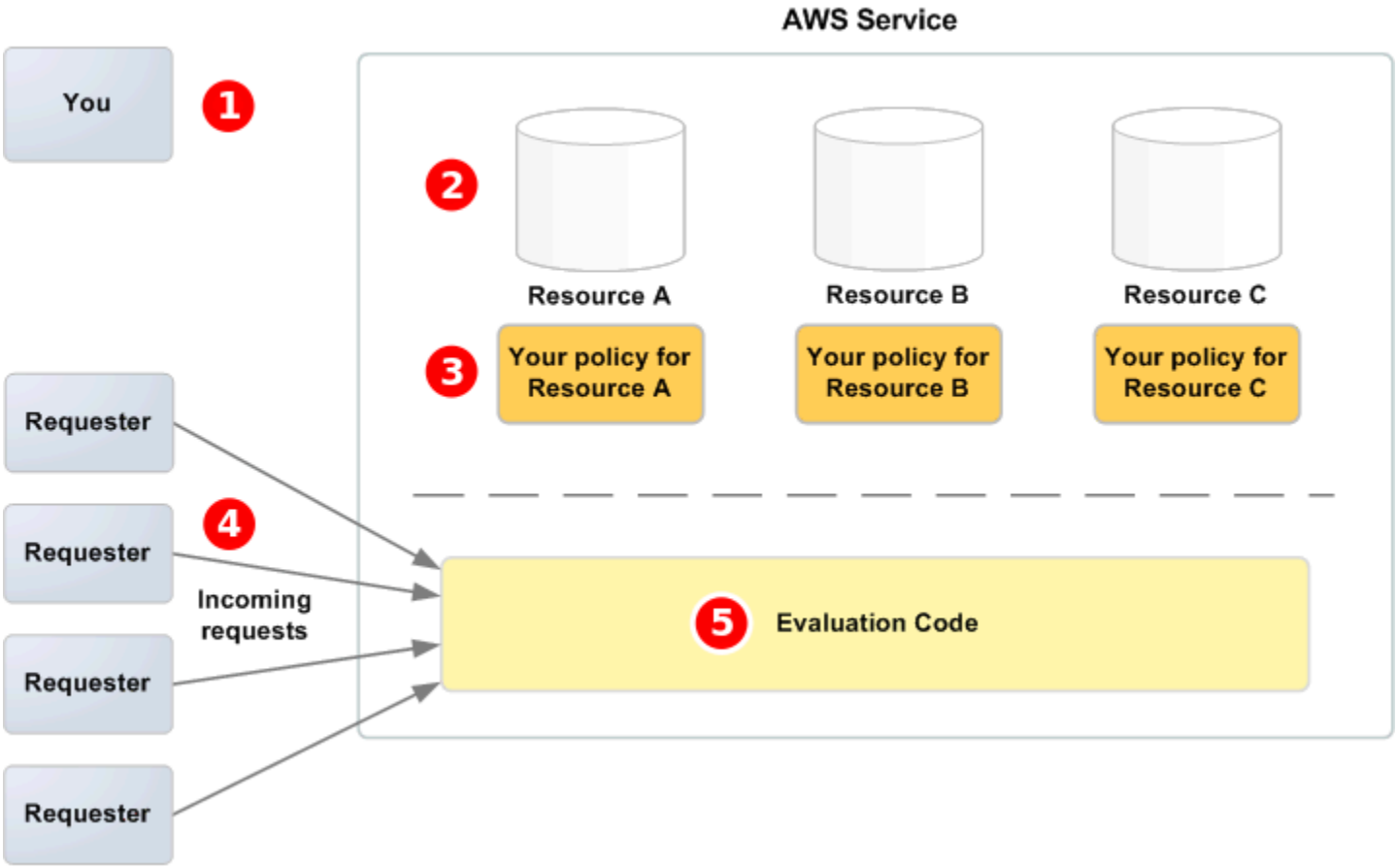
要根据请求时间或请求者的 IP 地址等特定条件允许或拒绝访问，您必须创建自定义 Amazon SQS 策略并使用操作进行上传。 SetQueueAttributes

主题

- Amazon SQS 访问控制架构
- Amazon SQS 访问控制处理工作流程
- Amazon SQS 访问策略语言关键概念
- Amazon SQS 访问策略语言评估逻辑
- Amazon SQS 访问策略语言中显式拒绝和默认拒绝之间的关系
- Amazon SQS 自定义策略的限制
- 自定义 Amazon SQS 访问策略语言示例

Amazon SQS 访问控制架构

下图介绍了 Amazon SQS 资源的访问控制。



1

您，资源所有者。

2

您

在 AWS 服务中包含的资源（例如，亚马逊 SQS 队列）。

3

您的策略。比较好的做法是为每个资源提供一个策略。该 AWS 服务提供一个 API，供您上传和管理您的政策。

4

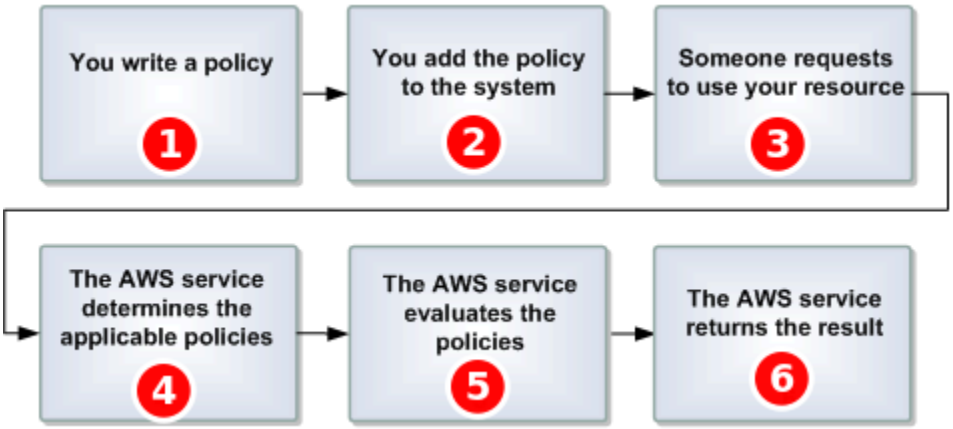
请求者和他们向 AWS 服务传入的请求。

5

访问策略语言评估代码。这是 AWS 服务中的一组代码，用于根据适用的策略评估传入的请求，并确定是否允许请求者访问资源。

Amazon SQS 访问控制处理工作流程

下图介绍了使用 Amazon SQS 访问策略语言进行访问控制的一般工作流程。



1

您为您的队列编写一个 Amazon SQS 策略。

2

您将政策上传到 AWS。该 AWS 服务提供了一个 API，您可以使用它来上传您的政策。例如，您使用 Amazon SQS `SetQueueAttributes` 操作来为特定 Amazon SQS 队列上传策略。

3

某人向您发出使用 Amazon SQS 队列的请求。

4

Amazon SQS 检查所有可用的 Amazon SQS 策略并决定哪些策略适用。

5

Amazon SQS 对这些策略进行评估，确定是否允许请求者使用您的队列。

6

根据策略评估结果，Amazon SQS 向请求者返回 `Access denied` 错误消息，或者继续处理该请求。

Amazon SQS 访问策略语言关键概念

要自行编写策略，您必须熟悉 JSON 和一些关键概念。

允许

将Statement设为 效果 时`allow`的结果。

Action

主体 拥有权限可以执行的活动，通常是对 AWS的请求。

Default-deny

没有 允许 或 Explicit-deny 设置的 Statement 的结果。

Condition

有关许可的任何限制或详细信息。典型的条件与日期、时间和 IP 地址相关。

效果

您希望一个策略的Statement在评估期间返回的结果。您编写策略语句时可以指定 deny 或 allow 值。在策略评估期间有三种可能的结果：Default-deny、允许和Explicit-deny。

Explicit-deny

将Statement设为 效果 时deny的结果。

评估

Amazon SQS 用来根据 策略 确定是否拒绝或允许传入请求的过程。

发布者

编写策略以对资源授予权限的用户。顾名思义，颁发者始终是资源所有者。 AWS 不允许 Amazon SQS 用户为他们不拥有的资源创建策略。

键

设置访问限制指定的特性。

许可

使用Condition和键允许或拒绝对资源的访问的概念。

策略

充当一条或多条语句容器的文档。



Amazon SQS 使用策略确定是否授予用户访问资源的权限。

主体

收到许可中策略的用户。

资源

主体请求访问的对象。

Statement

单个权限的正式描述，以访问策略语言编写，作为更大的 策略 文档的一部分。

请求者

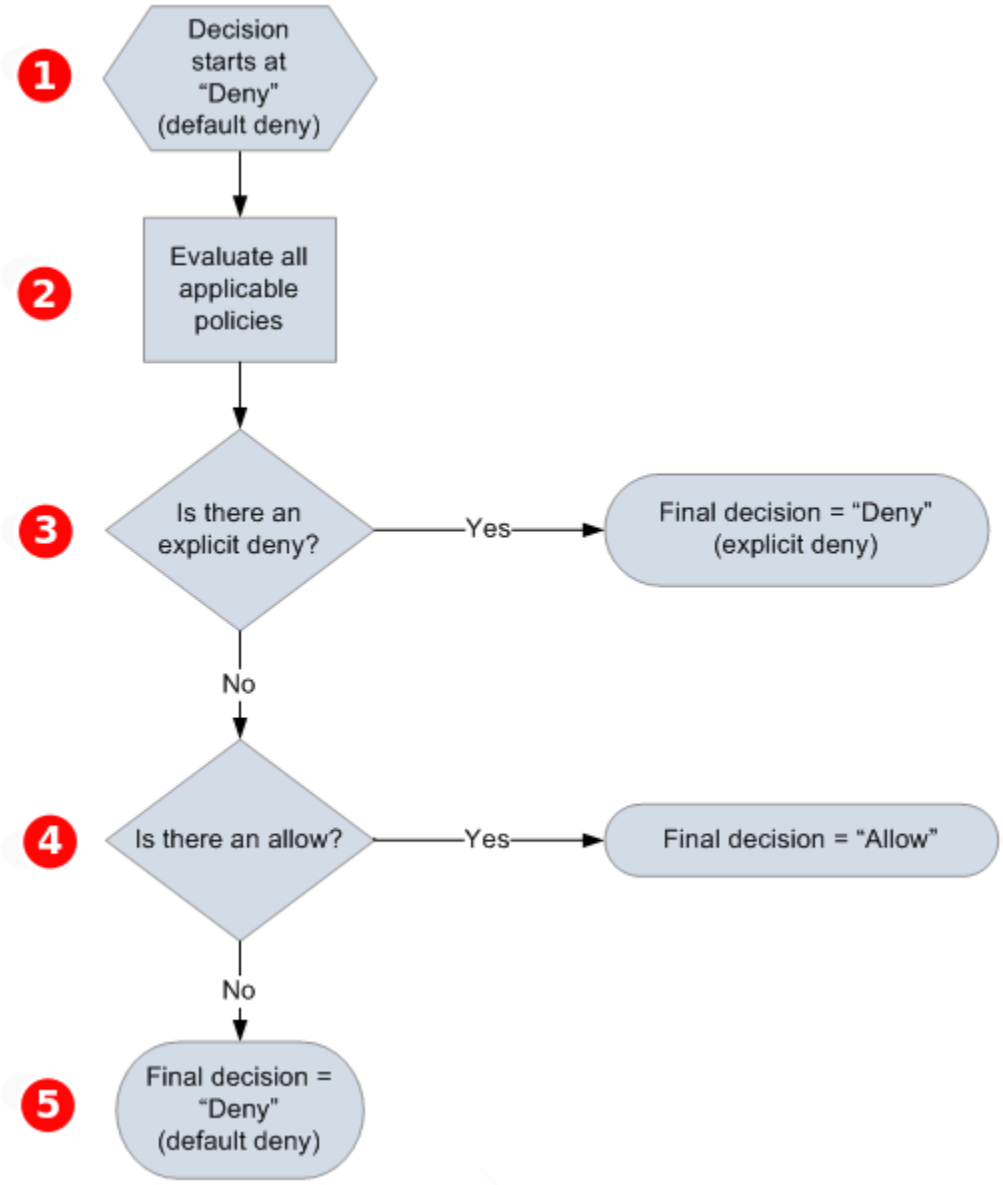
发送访问资源请求的用户。

Amazon SQS 访问策略语言评估逻辑

在评估期间，Amazon SQS 确定应允许还是拒绝资源所有者以外的某人发送的请求。评估逻辑遵循多个基本规则：

- 在默认情况下，您拒绝任何人发送的所有使用资源的请求。
- 允许 将覆盖任意Default-deny。
- Explicit-deny 将覆盖任意允许。
- 策略评估的顺序并不重要。

下图详述了 Amazon SQS 如何评估有关访问权限的决策。



1

该决策开始是一个默认拒绝。

2

执行代码评估适用于请求的所有策略（根据资源、主体、操作和条件）。执行代码评估策略的顺序并不重要。

3

执行代码寻找应用于请求的显式拒绝指令。即使执行代码只找到了一个，也会返回拒绝决策，整个过程完成。

4

如果没有找到显式拒绝指令，那么执行代码将寻找应用于请求的任何允许指令。即使执行代码只找到了一个，也会返回允许决策，整个过程完成（服务将继续处理该请求）。

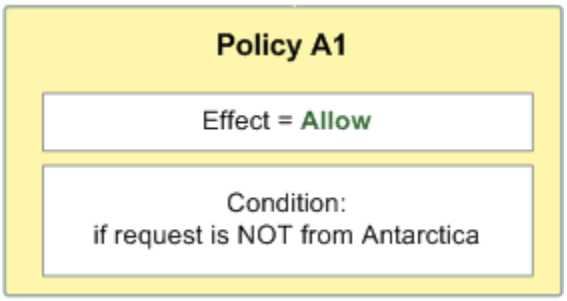
5

如果未找到任何允许指令，那么最终决策将是拒绝（因为没有显式拒绝或允许，所以这将被视为一个默认拒绝）。

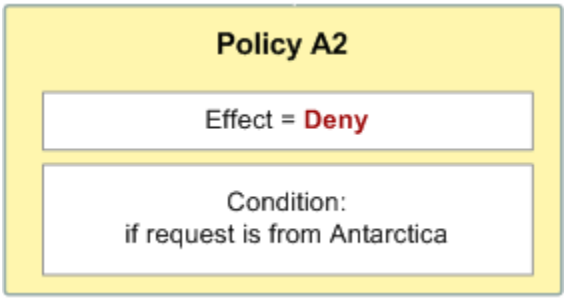
Amazon SQS 访问策略语言中显式拒绝和默认拒绝之间的关系

如果 Amazon SQS 策略不直接应用于请求，则该请求会导致 Default-deny。例如，如果用户请求使用 Amazon SQS 的权限，但应用于该用户的唯一策略可使用 DynamoDB，则该请求会导致 default-deny。

如果未能满足语句中的某个条件，则该请求的结果为默认拒绝。如果满足了语句中的所有条件，那么根据策略中 效果 元素的值，该请求的结果可能为 允许 或 Explicit-deny。如果策略没有指定如何处理未满足某个条件的情况，那么在这种情况下默认结果为默认拒绝。例如，您想要阻止来自南极洲的请求。您编写了策略 A1，只要请求不是来自于南极洲，就接受请求。下图说明了 Amazon SQS 策略。

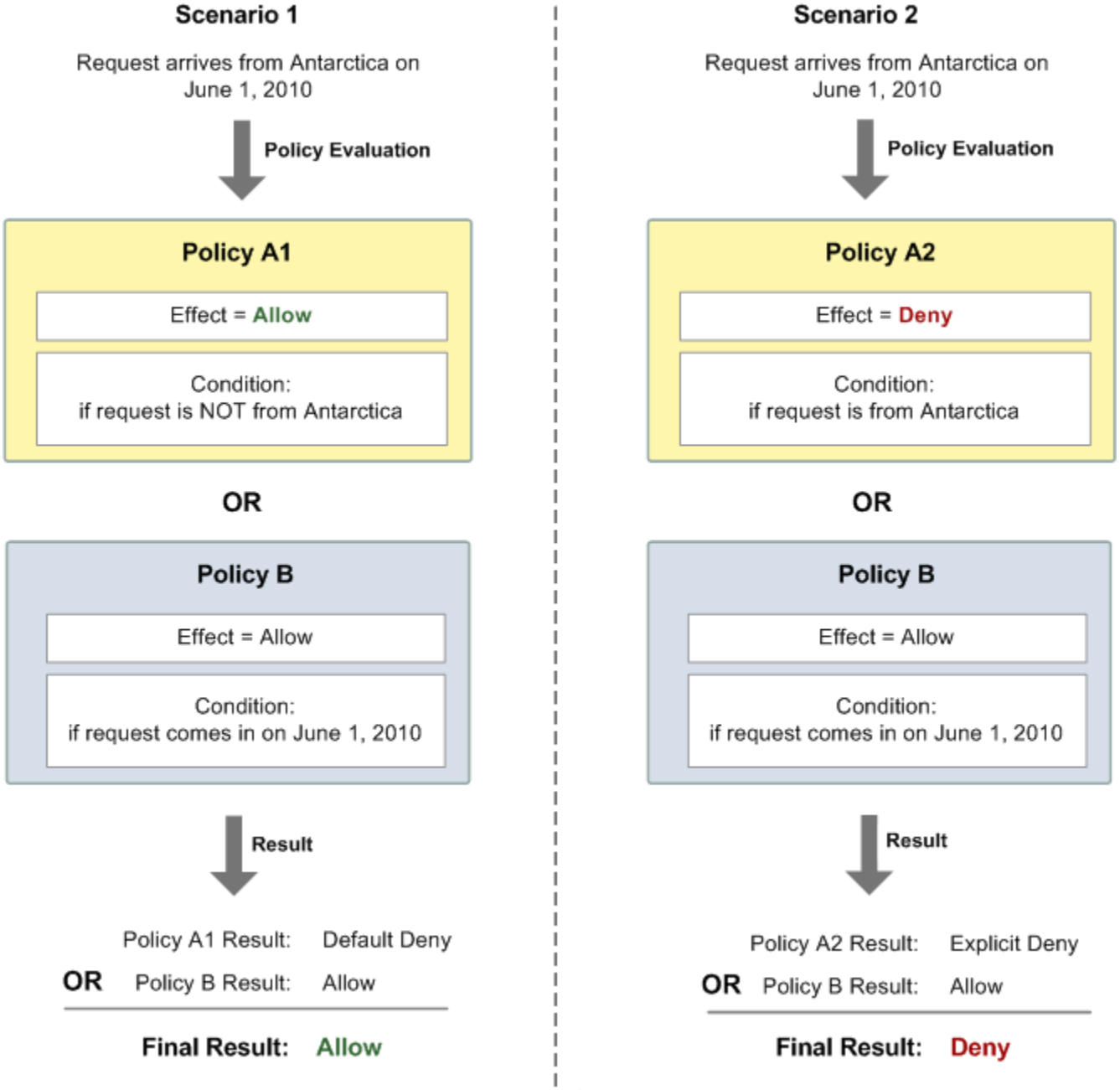


如果用户从 U.S. 发送请求，则条件得到满足（该请求不是来自南极洲），并且该请求会导致允许。 但是，如果用户从南极洲发出请求，那么条件未满足，该请求的默认结果为默认拒绝。您可以编写策略 A2，显式拒绝来自南极洲的请求，这样结果就变为显式拒绝。下列示意图说明了该策略。



如果用户从南极洲发出请求，那么条件已经满足，则该请求的结果为显式拒绝。

默认拒绝和显式拒绝之间的区别很重要，因为允许可以覆盖前者但不能覆盖后者。例如，策略 B 允许在 2010 年 6 月 1 日到达的请求。下图比较了将此策略与策略 A1 和策略 A2 进行组合的情况。



在场景 1 中，策略 A1 的结果为默认拒绝，策略 B 的结果为允许，因为该策略允许 2010 年 6 月 1 日传入的请求。策略 B 返回的允许结果将覆盖策略 A1 的默认拒绝结果，因此，请求获得允许。

在场景 2 中，策略 B2 的结果为显式拒绝，策略 B 的结果为允许。从策略 A2 发出的显式拒绝将覆盖从策略 B 发出的允许，因此该请求会被拒绝。

Amazon SQS 自定义策略的限制

Cross-account 访问

Cross-account 权限不适用于以下操作：

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTask
- ListQueues
- ListQueueTags
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

条件键

Amazon SQS 目前只支持 IAM 中可用条件键的有限子集。有关更多信息，请参阅 Amazon SQS API 权限：操作和资源参考。

自定义 Amazon SQS 访问策略语言示例

以下示例是典型的 Amazon SQS 访问策略。

示例 1：为一个账户授予权限

以下示例 Amazon SQS 策略为 AWS 账户 111122223333 授予权限，允许从 AWS 账户 444455556666 拥有的 queue2 中发送和接收请求。

JSON

```
{
   "Version":"2012-10-17",
```

```
   "Id": "UseCase1",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2"
   }]
}
```

示例 2：对一个或多个账户授予权限

以下示例 Amazon SQS 策略为您的账户在特定时间段内拥有的队列提供了一个或多个 AWS 账户 访问权限。有必要编写此策略并使用 SetQueueAttributes 操作将其上传到 Amazon SQS，因为 AddPermission 操作不允许在授予队列访问权限时指定时间限制。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "UseCase2",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333",
            "444455556666"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
```

```
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
      "Condition": {
         "DateLessThan": {
            "AWS:CurrentTime": "2009-06-30T12:00Z"
         }
      }
   }]
}
```

示例 3：对来自 Amazon EC2 实例的请求授予权限

以下示例 Amazon SQS 策略对来自 Amazon EC2 实例的请求授予访问权限。此示例根据“示例 2：对一个或多个账户授予权限”示例编写：它将访问时间限制在 2009 年 6 月 30 日中午 12 点 (UTC) 之前，将访问 IP 的范围限制在 `203.0.113.0/24`。有必要编写此策略并使用 `SetQueueAttributes` 操作将其上传到 Amazon SQS，因为 `AddPermission` 操作不允许在授予队列访问权限时指定 IP 地址限制。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "UseCase3",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
      "Condition": {
         "DateLessThan": {
            "AWS:CurrentTime": "2009-06-30T12:00Z"
         },
         "IpAddress": {
            "AWS:SourceIp": "203.0.113.0/24"
```

```
         }
      }
   }]
}
```

示例 4：拒绝特定账户的访问

以下示例 Amazon SQS 策略拒绝对您的队列进行特定 AWS 账户 访问。此示例以 “示例 1：为一个账户授予权限” 示例为基础：它拒绝访问指定的 AWS 账户。有必要编写此策略并使用 SetQueueAttributes 操作将其上传到 Amazon SQS，因为 AddPermission 操作不允许拒绝对队列的访问权限（它只允许授予对队列的访问权限）。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "UseCase4",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Deny",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2"
   }]
}
```

示例 5：如果不是来自 VPC 端点，则拒绝访问

以下示例 Amazon SQS 策略限制对 queue1 的访问权限：111122223333 只能通过 VPC 端点 ID vpce-1a2b3c4d（使用 aws:sourceVpce 条件指定）执行 SendMessage 和 ReceiveMessage 操作。有关更多信息，请参阅 Amazon SQS 的 Amazon Virtual Private Cloud 端点。

> **Note**
>
> - `aws:sourceVpce` 条件不需要 VPC 端点资源的 ARN，而只需要 VPC 端点 ID。
> - 您可以通过在第二个语句中拒绝所有 Amazon SQS 操作 (`sqs:*`)，修改以下示例，以将所有操作限制到特定 VPC 端点。但是，此类策略声明将规定所有操作（包括修改队列权限所需的管理操作）必须通过在策略中定义的特定 VPC 端点进行，这可能会阻止用户以后修改队列权限。

JSON

```
{
   "Version":"2012-10-17",
   "Id": "UseCase5",
   "Statement": [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
         "Resource": "arn:aws:sqs:us-east-2:111122223333:queue1"
      },
      {
         "Sid": "2",
         "Effect": "Deny",
         "Principal": "*",
         "Action": [
            "sqs:SendMessage",
            "sqs:ReceiveMessage"
         ],
         "Resource": "arn:aws:sqs:us-east-2:111122223333:queue1",
         "Condition": {
            "StringNotEquals": {
               "aws:sourceVpce": "vpce-1a2b3c4d"
```

```
            }
         }
      }
   ]
}
```

将临时安全凭证用于 Amazon SQS

除了使用自己的安全证书创建用户外，IAM 还允许您向任何用户授予临时安全证书，从而允许该用户访问您的 AWS 服务和资源。您可以管理拥有 AWS 账户的用户。您还可以管理系统中没有的用户 AWS 账户 （联合用户）。此外，您为访问 AWS 资源而创建的应用程序也可以被视为 “用户”。

您可以使用上述临时安全凭证对 Amazon SQS 发出请求。API 库会使用这些凭证计算必要的签名值，以便对您的请求进行身份验证。如果您使用过期的凭证发送请求，则 Amazon S3 会拒绝请求。

> **Note**
>
> 您不能基于临时凭证设置策略。

先决条件

1. 使用 IAM 创建临时安全凭证：
   - 安全令牌
   - 访问密钥 ID
   - 秘密访问密钥
2. 使用临时访问密钥 ID 和安全令牌准备待签字符串。
3. 使用临时秘密访问密钥（而不是您自己的秘密访问密钥）对您的查询 API 请求签名。

> **Note**
>
> 在提交已签名的查询 API 请求时，请使用临时访问密钥 ID（而不是您自己的访问密钥 ID），并且包含安全令牌。有关 IAM 对临时安全证书支持的更多信息，请参阅 IA M 用户指南中的授予对 AWS 资源的临时访问权限。

使用临时安全凭证调用 Amazon SQS 查询 API 操作

1. 使用请求临时安全令牌 AWS Identity and Access Management。有关更多信息，请参阅《IAM 用户指南》中的创建临时安全凭证以便为 IAM 用户启用访问权限。

   IAM 会返回一个安全令牌、一个访问密钥 ID 和一个秘密访问密钥。

2. 使用临时访问密钥 ID（而不是您自己的访问密钥 ID）准备查询，并且包含安全令牌。使用临时秘密访问密钥（而不是您自己的秘密访问密钥）对请求签名。

3. 提交已使用临时访问密钥 ID 和安全令牌签名的查询字符串。

   以下示例说明如何使用临时安全凭证对 Amazon SQS 请求进行身份验证。*AUTHPARAMS* 的结构取决于 API 请求的签名。有关更多信息，请参阅《亚马逊云服务一般参考》中的 “签署 AWS API 请求”。

```
https://sqs.us-east-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Attribute.1.Name=VisibilityTimeout
&Attribute.1.Value=40
&Expires=2020-12-18T22%3A52%3A43PST
&SecurityToken=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

   以下示例使用了临时安全凭证来通过 SendMessageBatch 操作发送两条消息。

```
https://sqs.us-east-2.amazonaws.com/
?Action=SendMessageBatch
&SendMessageBatchRequestEntry.1.Id=test_msg_001
&SendMessageBatchRequestEntry.1.MessageBody=test%20message%20body%201
&SendMessageBatchRequestEntry.2.Id=test_msg_002
&SendMessageBatchRequestEntry.2.MessageBody=test%20message%20body%202
&SendMessageBatchRequestEntry.2.DelaySeconds=60
&Expires=2020-12-18T22%3A52%3A43PST
&SecurityToken=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
&AWSAccessKeyId=AKIAI44QH8DHBEXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

使用最低权限策略管理加密 Amazon SQS 队列的访问权限

您可以使用 Amazon SQS 通过与 AWS Key Management Service (KMS) 集成的服务器端加密 (SSE) 在应用程序之间交换敏感数据。通过集成 Amazon SQS 和 AWS KMS，您可以集中管理保护 Amazon SQS 的密钥以及保护您的其他资源的密钥。 AWS

多个 AWS 服务可以充当向 Amazon SQS 发送事件的事件源。要使事件源能够访问加密的 Amazon SQS 队列，您需要使用客户 AWS KMS 管理的密钥配置队列。然后，使用密钥策略允许服务使用所需的 AWS KMS API 方法。该服务还需要对访问进行身份验证的权限才能使队列发送事件。您可以使用 Amazon SQS 策略来实现这一目标，该策略是一种基于资源的策略，可用于控制对 Amazon SQS 队列及其数据的访问。

以下各节提供有关如何通过亚马逊 SQS 策略和 AWS KMS 密钥策略控制对加密的 Amazon SQS 队列的访问权限的信息。本指南中的策略将帮助您实现最低权限。

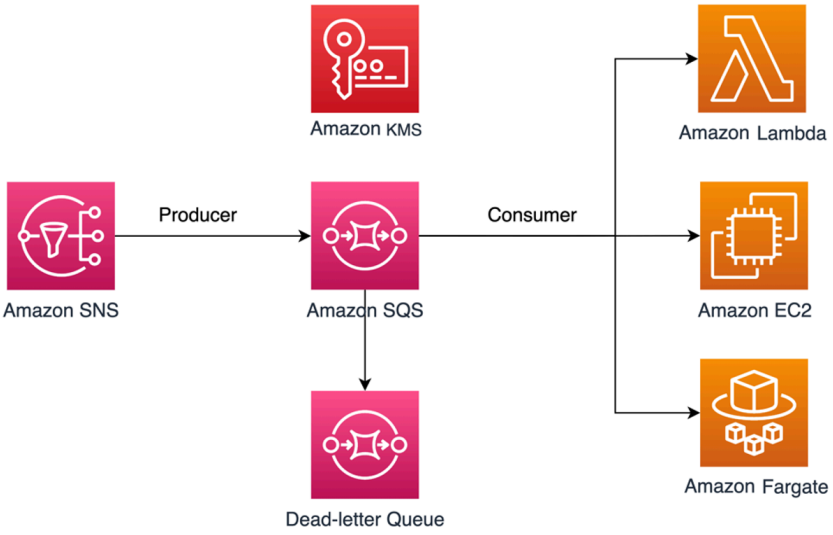
本指南还介绍了基于资源的策略如何使用 `aws:SourceArn`、`aws:SourceAccount` 和 `aws:PrincipalOrgID` 全局 IAM 条件上下文键来解决混淆代理问题。

主题

- 概述
- 最低权限 Amazon SQS 密钥政策
- 死信队列的 Amazon SQS 策略语句
- 防范跨服务混淆代理问题
- 使用 IAM Access Analyzer 查看跨账户访问权限

概述

在本主题中，我们将为您介绍一个常见使用案例，以说明如何构建密钥政策和 Amazon SQS 队列策略。此使用案例如下图所示。



在此示例中，消息创建者是 Amazon Simple Notification Service (SNS) 主题，该主题配置为将消息扇出到您的加密 Amazon SQS 队列。消息使用者是一项计算服务，例如 AWS Lambda 函数、Amazon Elastic Compute Cloud（EC2）实例或 AWS Fargate 容器。然后，将您的 Amazon SQS 队列配置为将失败的消息发送到死信队列 (DLQ)。这对于调试应用程序或消息传递系统非常有用 DLQs ，因为您可以隔离未使用的消息，以确定其处理失败的原因。在本主题中定义的解决方案中，使用 Lambda 函数等计算服务来处理存储在 Amazon SQS 队列中的消息。如果消息使用者位于虚拟私有云（VPC）中，则本指南中包含的 `DenyReceivingIfNotThroughVPCE` 策略语句允许您将消息接收限制在该特定 VPC 上。

> **Note**
>
> 本指南仅以策略语句的形式包含所需的 IAM 权限。要制定政策，您需要将声明添加到您的 Amazon SQS 政策或 AWS KMS 密钥策略中。本指南不提供有关如何创建 Amazon SQS 队列或密钥的 AWS KMS 说明。有关如何创建这些资源的说明，请参阅创建 Amazon SQS 队列和创建密钥。
>
> 本指南中定义的 Amazon SQS 策略不支持将消息直接重新发送到相同或不同的 Amazon SQS 队列。

最低权限 Amazon SQS 密钥政策

在本节中，我们将介绍 AWS KMS 用于加密 Amazon SQS 队列的客户管理密钥所需的最低权限权限。使用这些权限，您可以将访问权限限制为仅限目标实体，同时实施最低权限。密钥政策必须包含以下策略语句，我们将通过以下资源详细描述这些语句：

- 向 AWS KMS 密钥授予管理员权限

- 授予对密钥元数据的只读访问权限
- 授予 Amazon SNS KMS 对 Amazon SNS 向队列发布消息的权限
- 允许使用者解密来自队列的消息

向 AWS KMS 密钥授予管理员权限

要创建 AWS KMS 密钥，您需要为用于部署密 AWS KMS 钥的 IAM 角色提供 AWS KMS 管理员权限。这些管理员权限在以下 `AllowKeyAdminPermissions` 策略语句中进行了定义。在 AWS KMS 密钥策略中添加此声明时，请务必使用用于部署密钥、管理密 AWS KMS 钥或两者兼而有之的 IAM 角色的 Amazon 资源名称 (ARN) 替换*<admin-role ARN>*。 AWS KMS 这可以是您部署管道的 IAM 角色，也可以是 AWS Organizations 中您组织的管理员角色。

```
{
  "Sid": "AllowKeyAdminPermissions",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<admin-role ARN>"
    ]
  },
  "Action": [
    "kms:Create*",
    "kms:Describe*",
    "kms:Enable*",
    "kms:List*",
    "kms:Put*",
    "kms:Update*",
    "kms:Revoke*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:TagResource",
    "kms:UntagResource",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

> **Note**
>
> 在 AWS KMS 密钥策略中，Resource元素的值必须是*，这意味着 “这个 AWS KMS 密钥”。星号 (*) 标识 AWS KMS 密钥策略所关联的密钥。

授予对密钥元数据的只读访问权限

要向其他 IAM 角色授予对您密钥元数据的只读访问权限，请将 AllowReadAccessToKeyMetaData 语句添加到您的密钥政策中。例如，以下语句允许您列出账户中的所有 AWS KMS 密钥以供审计。此语句授予 AWS 根用户对密钥元数据的只读访问权限。因此，账户中的任何 IAM 主体只要其基于身份的策略具有以下语句中列出的权限，就可以访问密钥元数据：kms:Describe*、kms:Get* 和 kms:List*。请务必*<account-ID>*用您自己的信息替换。

```
{
  "Sid": "AllowReadAcesssToKeyMetaData",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
    ]
  },
  "Action": [
    "kms:Describe*",
    "kms:Get*",
    "kms:List*"
  ],
  "Resource": "*"
}
```

授予 Amazon SNS KMS 对 Amazon SNS 向队列发布消息的权限

要允许您的 Amazon SNS 主题向加密的 Amazon SQS 队列发布消息，请将 AllowSNSToSendToSQS 策略语句添加到您的密钥政策中。此声明授予亚马逊 SNS 使用该 AWS KMS 密钥发布到您的亚马逊 SQS 队列的权限。请务必*<account-ID>*用您自己的信息替换。

> **Note**
>
> 声明Condition中的限制只能在同一 AWS 账户中使用 Amazon SNS 服务。

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "<account-id>"
    }
  }
}
```

允许使用者解密来自队列的消息

以下 AllowConsumersToReceiveFromTheQueue 语句向 Amazon SQS 消息使用者授予解密从加密 Amazon SQS 队列收到的消息所需的权限。附加策略声明时，请替换为消息*<consumer's runtime role ARN>*使用者的 IAM 运行时角色 ARN。

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

最低权限 Amazon SQS 策略

本节将引导您了解本指南所涵盖的使用案例的最低权限 Amazon SQS 队列策略（例如，从 Amazon SNS 到 Amazon SQS）。定义的策略旨在通过混合使用 `Deny` 和 `Allow` 语句来防止意外访问。`Allow` 语句会向目标实体或实体授予访问权限。`Deny` 语句可防止其他意外实体访问 Amazon SQS 队列，同时将目标实体排除在策略条件之外。

Amazon SQS 策略包括以下语句，我们在下面对其进行了详细描述：

- 限制 Amazon SQS 管理权限
- 限制指定组织的 Amazon SQS 队列操作
- 向使用者授予 Amazon SQS 权限
- 实施传输中加密
- 将消息传输限制为特定的 Amazon SNS 主题
- （可选）将消息接收限制为特定 VPC 端点

限制 Amazon SQS 管理权限

以下 `RestrictAdminQueueActions` 策略语句将 Amazon SQS 管理权限限制为仅限于您用于部署队列、管理队列或两者兼而有之的 IAM 角色。确保将 *<placeholder values>* 替换为您自己的信息。指定用于部署 Amazon SQS 队列的 IAM 角色的 ARN，以及ARNs 应具有 Amazon SQS 管理权限的任何管理员角色的 ARN。

```
{
  "Sid": "RestrictAdminQueueActions",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:AddPermission",
    "sqs:DeleteQueue",
    "sqs:RemovePermission",
    "sqs:SetQueueAttributes"
  ],
  "Resource": "<SQS Queue ARN>",
  "Condition": {
    "StringNotLike": {
      "aws:PrincipalARN": [
        "arn:aws:iam::<account-id>:role/<deployment-role-name>",
```

```
        "<admin-role ARN>"
      ]
    }
  }
}
```

限制指定组织的 Amazon SQS 队列操作

要帮助保护您的 Amazon SQS 资源不被外部访问（由 AWS 组织外部实体访问），请使用以下语句。此语句会限制您在 Condition 中指定的组织访问 Amazon SQS 队列。请务必*<SQS queue ARN>*替换为用于部署 Amazon SQS 队列的 IAM 角色的 ARN；并使用您的组织 *<org-id>* ID 替换。

```
{
  "Sid": "DenyQueueActionsOutsideOrg",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:AddPermission",
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteQueue",
    "sqs:RemovePermission",
    "sqs:SetQueueAttributes",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:PrincipalOrgID": [
        "<org-id>"
      ]
    }
  }
}
```

向使用者授予 Amazon SQS 权限

要接收来自 Amazon SQS 队列的消息，您需要向消息使用者提供必要的权限。以下策略语句会向您指定的使用者授予使用来自 Amazon SQS 队列的消息所需的权限。将该声明添加到您的 Amazon SQS 策略时，请务必*<consumer's IAM runtime role ARN>*替换为使用者使用的 IAM 运行时角色的 ARN；以及*<SQS queue ARN>*替换为用于部署 Amazon SQS 队列的 IAM 角色的 ARN。

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": "<consumer's IAM execution role ARN>"
  },
  "Action": [
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteMessage",
    "sqs:GetQueueAttributes",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>"
}
```

要防止其他实体接收来自 Amazon SQS 队列的消息，请将 DenyOtherConsumersFromReceiving 语句添加到 Amazon SQS 队列策略中。此语句会限制消息仅供您指定的使用者使用，不允许其他使用者访问，即使他们的身份权限会授予他们访问权限，也是如此。请务必*<consumer's runtime role ARN>*用您自己的信息替换*<SQS queue ARN>*和。

```
{
  "Sid": "DenyOtherConsumersFromReceiving",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteMessage",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotLike": {
      "aws:PrincipalARN": "<consumer's execution role ARN>"
    }
  }
}
```

实施传输中加密

以下 DenyUnsecureTransport 策略语句强制使用者和创建者使用安全通道（TLS 连接）发送和接收来自 Amazon SQS 队列的消息。请务必*<SQS queue ARN>*替换为用于部署 Amazon SQS 队列的 IAM 角色的 ARN。

```
{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "Bool": {
      "aws:SecureTransport": "false"
    }
  }
}
```

将消息传输限制为特定的 Amazon SNS 主题

以下 AllowSNSToSendToTheQueue 策略语句允许指定的 Amazon SNS 主题向 Amazon SQS 队列发送消息。请务必*<SQS queue ARN>*替换为用于部署亚马逊 SQS 队列的 IAM 角色的 ARN；*<SNS topic ARN>*并替换为亚马逊 SNS 主题 ARN。

```
{
  "Sid": "AllowSNSToSendToTheQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnLike": {
```

```
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

以下 DenyAllProducersExceptSNSFromSending 策略语句禁止其他创建者向队列发送消息。将 *<SQS queue ARN>* 和 *<SNS topic ARN>* 替换为您自己的信息。

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

（可选）将消息接收限制为特定 VPC 端点

要限制仅向特定的 VPC 端点发送消息，请在您的 Amazon SQS 队列策略中添加以下策略语句。除非消息来自所需的 VPC 端点，否则此语句可防止消息使用者接收来自队列的消息。*<SQS queue ARN>*替换为用于部署 Amazon SQS 队列的 IAM 角色的 ARN *<vpce_id>* 和 VPC 终端节点的 ID。

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
```

```
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

死信队列的 Amazon SQS 策略语句

将以下策略语句（由其语句 ID 标识）添加到您的 DLQ 访问策略中：

- RestrictAdminQueueActions
- DenyQueueActionsOutsideOrg
- AllowConsumersToReceiveFromTheQueue
- DenyOtherConsumersFromReceiving
- DenyUnsecureTransport

除了将上述策略语句添加到 DLQ 访问策略外，您还应添加一条语句，以限制向 Amazon SQS 队列传输消息，如下一节所述。

限制向 Amazon SQS 队列传输消息

要限制同一账户只能访问 Amazon SQS 队列，请在 DLQ 队列策略中添加以下 DenyAnyProducersExceptSQS 策略语句。此语句不会将消息传输限制到特定队列，因为您需要在创建主队列之前部署 DLQ，因此在创建 DLQ 时不会知道 Amazon SQS ARN。如果您需要将访问权限限制为仅一个 Amazon SQS 队列，请在知道时使用您的 Amazon SQS 源队列的 ARN 修改 Condition 中的 aws:SourceArn。

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
    }
  }
}
```

> **Important**
>
> 本指南中定义的 Amazon SQS 队列策略不会将 `sqs:PurgeQueue` 操作限制在某个 IAM 角色或多个角色。`sqs:PurgeQueue` 操作允许您删除 Amazon SQS 队列中的所有消息。您也可以使用此操作更改消息格式，而无需替换 Amazon SQS 队列。调试应用程序时，您可以清除 Amazon SQS 队列以删除可能错误的消息。测试应用程序时，您可以通过 Amazon SQS 队列发送大量消息，然后在进入生产环境之前清除队列以重新开始。之所以不将此操作限制为某个角色，是因为在部署 Amazon SQS 队列时可能不知道此角色。您需要将此权限添加到角色基于身份的策略中，才能清除队列。

防范跨服务混淆代理问题

混淆代理问题是一个安全问题，即没有执行操作权限的实体可能会迫使更具权限的实体执行该操作。为了防止这种情况， AWS 如果您向第三方（称为跨账户）或其他 AWS 服务（称为跨服务）提供对您账户中资源的访问权限，则提供可帮助您保护账户的工具。本节中的策略语句可以帮助您防范跨服务混淆代理问题。

一个服务（呼叫服务）调用另一项服务（所谓的服务)时，可能会发生跨服务模拟。可以操纵调用服务以使用其权限对另一个客户的资源进行操作，否则该服务不应有访问权限。为了帮助防范此问题，本文中定义的基于资源的策略使用 `aws:SourceArn`、`aws:SourceAccount` 和 `aws:PrincipalOrgID` 全局 IAM 条件上下文键。这限制了服务对Organizations中特定资源、特定账户或特定 AWS 组织的权限。

使用 IAM Access Analyzer 查看跨账户访问权限

您可以使用 AWS IAM Access Analy zer 查看您的 Amazon SQS 队列策略和 AWS KMS 密钥策略，并在亚马逊 SQS 队列或密钥授予外部 AWS KMS 实体访问权限时提醒您。IAM Access Analyzer 帮助标识您组织中的资源和与信任区域之外的实体共享的账户。此信任区域可以是您在启用 IAM Access Analyzer 时指定的 AWS 账户或组织。 AWS

IAM Access Analyzer 使用基于逻辑的推理来分析环境中基于资源的策略，从而识别与外部委托人共享的资源。 AWS 对于在您的信任区域外共享的资源的每个实例，Access Analyzer 都会生成一个调查结果。调查结果包括有关访问权限以及该访问权限授予到的外部主体的信息。您可以查看调查结果，以确定该访问权限是按计划授予且安全，还是计划外的访问权限并存在安全风险。对于任何意外访问，请查看受影响的策略并进行修复。有关 AWS IAM Access Analyzer 如何识别对您的 AWS 资源的意外访问的更多信息，请参阅此博客文章。

有关 AWS IAM 访问分析器的更多信息，请参阅 AWS IAM 访问分析器文档。

Amazon SQS API 权限：操作和资源参考

在设置 访问控制 和编写您可附加到 IAM 身份的权限策略时，可以使用下表作为参考。该列出了每个 Amazon Simple Queue Service 操作、您可以授予执行该操作权限的相应操作以及您可以授予权限的 AWS 资源。

在策略的 `Action` 字段中指定操作，并在策略的 `Resource` 字段中指定资源值。要指定操作，请在 操作名称之前使用 `sqs:` 前缀（例如，`sqs:CreateQueue`）。

目前，Amazon SQS 支持 IAM 中提供的全局条件上下文键。

Amazon Simple Queue Service API 和操作所需的权限

AddPermission

操作： `sqs:AddPermission`

资源：`arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`

ChangeMessageVisibility

操作： `sqs:ChangeMessageVisibility`

资源：`arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`

ChangeMessageVisibilityBatch

操作： `sqs:ChangeMessageVisibilityBatch`

资源：`arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`

CreateQueue

操作： `sqs:CreateQueue`

资源：`arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`

DeleteMessage

操作： `sqs:DeleteMessage`

资源：`arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`

DeleteMessageBatch

操作： `sqs:DeleteMessageBatch`

资源：`arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`

DeleteQueue

操作：sqs:DeleteQueue

资源：arn:aws:sqs:*region*:*account_id*:*queue_name*

GetQueueAttributes

操作：sqs:GetQueueAttributes

资源：arn:aws:sqs:*region*:*account_id*:*queue_name*

GetQueueUrl

操作：sqs:GetQueueUrl

资源：arn:aws:sqs:*region*:*account_id*:*queue_name*

ListDeadLetterSourceQueues

操作：sqs:ListDeadLetterSourceQueues

资源：arn:aws:sqs:*region*:*account_id*:*queue_name*

ListQueues

操作：sqs:ListQueues

资源：arn:aws:sqs:*region*:*account_id*:*queue_name*

ListQueueTags

操作：sqs:ListQueueTags

资源：arn:aws:sqs:*region*:*account_id*:*queue_name*

PurgeQueue

操作：sqs:PurgeQueue

资源：arn:aws:sqs:*region*:*account_id*:*queue_name*

ReceiveMessage

操作：sqs:ReceiveMessage

资源：arn:aws:sqs:*region*:*account_id*:*queue_name*

RemovePermission

操作：sqs:RemovePermission

资源：`arn:aws:sqs:region:account_id:queue_name`

SendMessage 和 SendMessageBatch

操作： `sqs:SendMessage`

资源：`arn:aws:sqs:region:account_id:queue_name`

SetQueueAttributes

操作： `sqs:SetQueueAttributes`

资源：`arn:aws:sqs:region:account_id:queue_name`

TagQueue

操作： `sqs:TagQueue`

资源：`arn:aws:sqs:region:account_id:queue_name`

UntagQueue

操作： `sqs:UntagQueue`

资源：`arn:aws:sqs:region:account_id:queue_name`

Amazon SQS 中的日志记录和监控

Amazon Simple Queue Service 与 AWS CloudTrail 集成，后者是一项服务，可提供由用户、角色或 AWS 服务 服务所采取操作的记录。CloudTrail 将 Amazon SQS 的所有 API 调用作为事件捕获。捕获调用中包括通过 Amazon SQS 控制台的调用和对 Amazon SQS API 操作的代码调用。借助通过 CloudTrail 收集的信息，您可以确定向 Amazon SQS 发出的请求、发出请求的 IP 地址、请求的发出时间以及其他详细信息。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根用户凭证还是用户凭证发出的。
- 请求是否代表 IAM Identity Center 用户发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务 发出。

当您创建账户并可以自动访问 CloudTrail 事件历史记录时，CloudTrail 在您的 AWS 账户 中处于活动状态。CloudTrail 事件历史记录提供对 AWS 区域 中过去 90 天的已记录管理事件的可查看、可搜索、

可下载和不可变记录。有关更多信息，请参见《AWS CloudTrail 用户指南》的 使用 CloudTrail 事件历史记录。查看事件历史记录不会收取 CloudTrail 费用。

要持续记录您的 AWS 账户 过去 90 天的事件，请创建跟踪或 CloudTrail Lake 事件数据存储。

Amazon CloudWatch 警报

在您指定的时间段监控单个指标，并根据该指标在多个时间段内相对于设定阈值的数值，采取一项或多项操作。例如，您可以将 CloudWatch 警报配置为向 Amazon SNS 主题发送通知，或触发向 Amazon SQS 队列发送消息的操作。CloudWatch 警报不会仅仅因为处于特定状态就执行操作；状态必须发生变化并在定义的若干个时间段内保持不变。

有关更多信息，请参阅为 Amazon SQS 指标创建 CloudWatch 警报和使用 Amazon 为死信队列创建警报 CloudWatch。

Amazon CloudWatch Logs

通过配置处理消息的应用程序或 Lambda 函数，将日志发送到 CloudWatch Logs，监控、存储和访问与 Amazon SQS 相关的日志文件。您可以利用这些日志分析消息处理过程、调试问题并监控 Amazon SQS 工作流的性能。

有关更多信息，请参阅 使用记录亚马逊简单队列服务 API 调用 AWS CloudTrail。

Amazon CloudWatch Events

使用 Amazon CloudWatch Events 检测 AWS 环境中的变更或特定事件，并将它们路由到 Amazon SQS 队列。这让您能够捕获事件数据、触发工作流或存储事件以供后续处理。

有关更多信息，请参阅本指南中的 使用亚马逊自动将 AWS 服务发送到亚马逊 SQS 的通知 EventBridge以及《Amazon EventBridge 用户指南》中的 EventBridge is the evolution of Amazon CloudWatch Events。

AWS CloudTrail 日志

CloudTrail 会详细记录用户、角色或 AWS 服务 服务在 Amazon SQS 上执行的操作。借助这些日志，您可以跟踪 API 调用（如 `SendMessage`、`ReceiveMessage` 或 `DeleteQueue`），并提供关键信息，包括请求发起者、发生时间以及来源 IP 地址。

有关更多信息，请参阅 使用记录亚马逊简单队列服务 API 调用 AWS CloudTrail。

AWS Trusted Advisor

Trusted Advisor 使用为 AWS 客户服务的最佳实践，帮助您优化 Amazon SQS 使用。它会审查您的 Amazon SQS 队列并提供切实可行的建议，以增强安全性、提高消息处理可靠性并降低成本。例如，它可能会建议启用死信队列或改进队列访问策略，以确保安全操作。

有关更多信息，请参阅《支持 用户指南》中的 AWS Trusted Advisor。

CloudTrail 跟踪

通过跟踪记录，CloudTrail 可将日志文件传送至 Simple Storage Service (Amazon S3) 存储桶。使用 AWS 管理控制台 创建的所有跟踪均具有多区域属性。您可以通过使用 AWS CLI 创建单区域或多区域跟踪。建议创建多区域跟踪，因为您可记录您账户中的所有 AWS 区域 的活动。如果您创建单区域跟踪，则只能查看跟踪的 AWS 区域 中记录的事件。有关跟踪的更多信息，请参阅《AWS CloudTrail 用户指南》中的为您的 AWS 账户 创建跟踪和为组织创建跟踪。

通过创建跟踪，您可以从 CloudTrail 免费向您的 Amazon S3 存储桶传送一份正在进行的管理事件的副本，但会收取 Amazon S3 存储费用。有关 CloudTrail 定价的更多信息，请参阅 AWS CloudTrail 定价。有关 Amazon S3 定价的信息，请参阅 Amazon S3 定价。

CloudTrail Lake 事件数据存储

CloudTrail Lake 允许您对事件运行基于 SQL 的查询。CloudTrail Lake 可将基于行的 JSON 格式的现有事件转换为 Apache ORC 格式。ORC 是一种针对快速检索数据进行优化的列式存储格式。事件将被聚合到事件数据存储中，它是基于您通过应用高级事件选择器选择的条件的不可变的事件集合。应用于事件数据存储的选择器用于控制哪些事件持续存在并可供您查询。有关 CloudTrail Lake 的更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 AWS CloudTrail Lake。

CloudTrail Lake 事件数据存储和查询会产生费用。创建事件数据存储时，您可以选择要用于事件数据存储的定价选项。定价选项决定了摄取和存储事件的成本，以及事件数据存储的默认和最长保留期。有关 CloudTrail 定价的更多信息，请参阅 AWS CloudTrail 定价。

使用记录亚马逊简单队列服务 API 调用 AWS CloudTrail

CloudTrail 允许您使用两种事件类型记录和监控 Amazon SQS 操作：数据事件和管理事件。这使您能够轻松跟踪和审计您账户中的 Amazon SQS 活动。

中的亚马逊 SQS 数据事件 CloudTrail

数据事件提供对资源或在资源中所执行资源操作（例如，向 Amazon SQS 对象发送消息）的相关信息。这些也称为数据面板操作。数据事件通常是高容量活动。默认情况下， CloudTrail 不记录数据事件。 CloudTrail 事件历史记录不记录数据事件。

记录数据事件将收取额外费用。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

您可以使用 CloudTrail 控制台或 CloudTrail API 操作记录 Amazon SQS 资源类型的数据事件。 AWS CLI有关如何记录数据事件的更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 AWS 管理控制台记录数据事件和使用 AWS Command Line Interface记录数据事件。

要记录 Amazon SQS 数据事件 CloudTrail，您必须使用高级事件选择器来配置要记录的特定 Amazon SQS 资源或操作。请添加 `AWS::SQS::Queue` 资源类型，以捕获队列相关操作。您可以使用 `eventName`（例如 `SendMessage` 事件）等筛选条件进一步优化日志记录首选项。有关更多信息，请参阅《CloudTrail API Reference》中的 AdvancedEventSelector。

数据事件类型（控制台）	resources.type 值	数据 APIs 已记录到 CloudTrail
Amazon SQS 队列	`AWS::SQS::Queue`	• ChangeMessageVisibility • ChangeMessageVisibilityBatch • DeleteMessage • DeleteMessageBatch • GetQueueAttributes • GetQueueUrl • ListDeadLetterSourceQueues • ListQueues • ListQueueTags • ReceiveMessage • SendMessage • SendMessageBatch

请使用高级事件选择器来筛选字段并仅记录重要事件。有关这些字段的更多信息，请参阅《AWS CloudTrail API 参考》中的 AdvancedFieldSelector。

中的亚马逊 SQS 管理事件 CloudTrail

管理事件提供有关对中的资源执行的管理操作的信息 AWS 账户。这些也称为控制面板操作。默认情况下， CloudTrail 记录管理事件。

Amazon SQS 将以下控制平面操作记录 CloudTrail 为管理事件。

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTasks
- PurgeQueue
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

Amazon SQS 事件示例

事件代表来自任何来源的单个请求，包括有关所请求的 API 操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此事件不会按任何特定顺序出现。

以下示例显示了一个演示该SendMessage操作 CloudTrail 的事件。

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EXAMPLE_PRINCIPAL_ID",
    "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/SessionName",
    "accountId": "123456789012",
    "accessKeyId": "ACCESS_KEY_ID",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed",
        "accountId": "123456789012",
        "userName": "RoleToBeAssumed"
      },
      "attributes": {
        "creationDate": "2023-11-07T22:13:06Z",
        "mfaAuthenticated": "false"
```

```
      }
    }
  },
  "eventTime": "2023-11-07T23:59:11Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "SendMessage",
  "awsRegion": "ap-southeast-4",
  "sourceIPAddress": "10.0.118.80",
  "userAgent": "aws-cli/1.29.16 md/Botocore#1.31.16 ua/2.0 os/
linux#5.4.250-173.369.amzn2int.x86_64 md/arch#x86_64 lang/python#3.8.17 md/
pyimpl#CPython cfg/retry-mode#legacy botocore/1.31.16",
  "requestParameters": {
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue",
    "messageBody": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "messageDeduplicationId": "MsgDedupIdSdk1ae1958f2-bbe8-4442-83e7-4916e3b035aa",
    "messageGroupId": "MsgGroupIdSdk16"
  },
  "responseElements": {
    "mD5OfMessageBody": "9a4e3f7a614d9dd9f8722092dbda17a2",
    "mD5OfMessageSystemAttributes": "f88f0587f951b7f5551f18ae699c3a9d",
    "messageId": "93bb6e2d-1090-416c-81b0-31eb1faa8cd8",
    "sequenceNumber": "18881790870905840128"
  },
  "requestID": "c4584600-fe8a-5aa3-a5ba-1bc42f055fae",
  "eventID": "98c735d8-70e0-4644-9432-b6ced4d791b1",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::SQS::Queue",
      "ARN": "arn:aws:sqs:ap-southeast-4:123456789012:MyQueue"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
    "clientProvidedHostHeader": "sqs.ap-southeast-4.amazonaws.com"
  }
```

> **Note**
>
> `ListQueues` 操作是一个独特的案例，因为它不作用于特定的资源。因此，ARN 字段不包含队列名称，而是使用通配符（`*`）。

有关 CloudTrail 录音内容的信息，请参阅《AWS CloudTrail 用户指南》中的CloudTrail录制内容。

使用监控 Amazon SQS 队列 CloudWatch

亚马逊 SQS 和亚马逊 CloudWatch 已集成，因此您可以使用它 CloudWatch 来查看和分析亚马逊 SQS 队列的指标。您可以从 Amazon SQS 控制台、控制CloudWatch 台、使用或 API 查看和分析队列的CloudWatch 指标。AWS CLI您还可以为亚马逊 SQS 指标设置 CloudWatch 警报。

CloudWatch 系统会自动收集您的 Amazon SQS 队列的指标，并以一分钟为间隔推送到 CloudWatch 。这些指标是在符合活跃 CloudWatch 指导方针的所有队列上收集的。 CloudWatch 如果队列包含任何消息，或者如果有任何操作可以访问该队列，则认为该队列处于活动状态的时间长达六个小时。

当 Amazon SQS 队列处于非活动状态超过六个小时时，Amazon SQS 服务被视为休眠状态并停止向该 CloudWatch 服务提供指标。在您的 Amazon SQS 队列处于非活动状态的时间段内，无法在 Amazon SQS CloudWatch 指标中显示缺失的数据或表示零的数据。

> **Note**
>
> - 如果针对某个 Amazon SQS 队列调用 API 的用户未获得授权且请求失败，系统会激活该 Amazon SQS 队列。
> - 队列页面打开时，Amazon SQS 控制台会执行 `GetQueueAttributes` API 调用。`GetQueueAttributes` API 请求会激活队列。
> - 当队列从非活动状态激活时， CloudWatch 指标最多会延迟 15 分钟。
> - 中报告的亚马逊 SQS 指标不收取任何费用。 CloudWatch它们作为 Amazon SQS 服务的一部分提供。
> - CloudWatch 标准队列和 FIFO 队列均支持指标。

访问亚马逊 SQS 的 CloudWatch 指标

亚马逊 SQS 和亚马逊 CloudWatch 已集成，因此您可以使用它 CloudWatch 来查看和分析亚马逊 SQS 队列的指标。您可以从 Amazon SQS 控制台、控制CloudWatch 台、使用或 API 查看和分析队列的CloudWatch 指标。AWS CLI您还可以为亚马逊 SQS 指标设置 CloudWatch 警报。

使用 Amazon SQS 控制台

使用 Amazon SQS 控制台访问和分析最多 10 个 Amazon SQS 队列的指标。

1. 登录 Amazon SQS 控制台。
2. 在队列列表中，选择（选中）与您要访问其指标的队列相对应的框。您最多可以显示 10 个队列的指标。
3. 选择监控选项卡。

   将会在 SQS metrics 部分中显示多个图形。
4. 要了解特定图形表示的内容，请将光标悬停在所需图形旁的

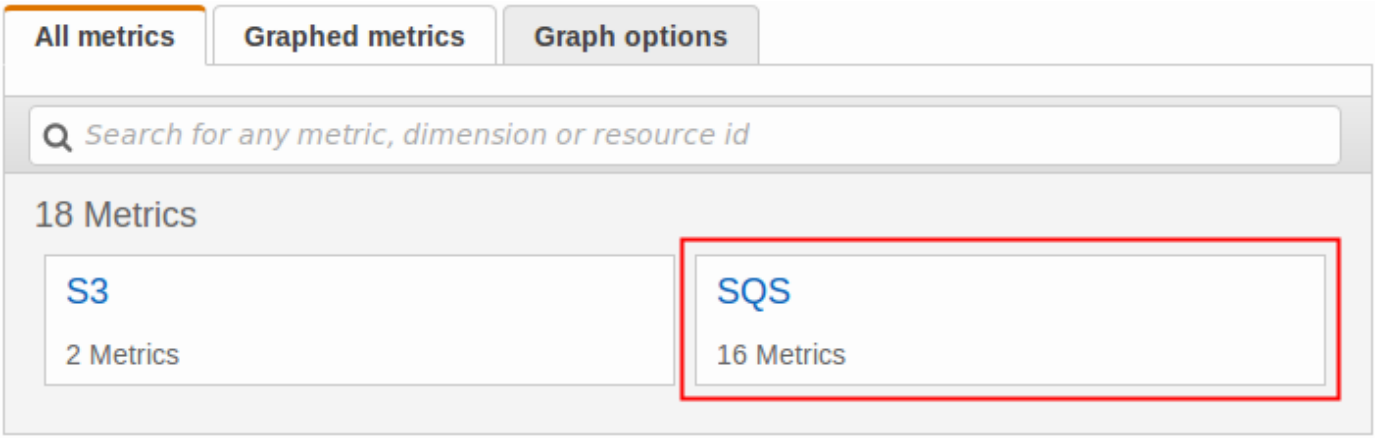
   上，或参阅 亚马逊 SQS 的可用 CloudWatch 指标。
5. 要同时更改所有图形的时间范围，请针对 Time Range 选择所需时间范围（例如，Last Hour）。
6. 要查看单个图形的其他统计数据，请选择该图形。
7. 在 CloudWatch 监控详细信息对话框中，选择一种统计数据（例如，总计）。有关受支持的统计数据的列表，请参阅亚马逊 SQS 的可用 CloudWatch 指标。
8. 要更改单个图形显示的时间范围和时间间隔（例如，显示最近 24 小时的时间范围而不是前 5 分钟，或者显示每小时时间段而不是每 5 分钟），并且仍然显示图形的对话框，则对于 Time Range，选择所需时间范围（例如，Last 24 Hours）。对于 Period，在指定的时间范围内选择所需时间段（例如，1 Hour）。查看完图形后，请选择 Close。
9. （可选）要使用其他 CloudWatch 功能，请在 “监控” 选项卡上选择 “查看所有 CloudWatch指标”，然后按照使用亚马逊 CloudWatch 控制台过程中的说明进行操作。

使用亚马逊 CloudWatch 控制台

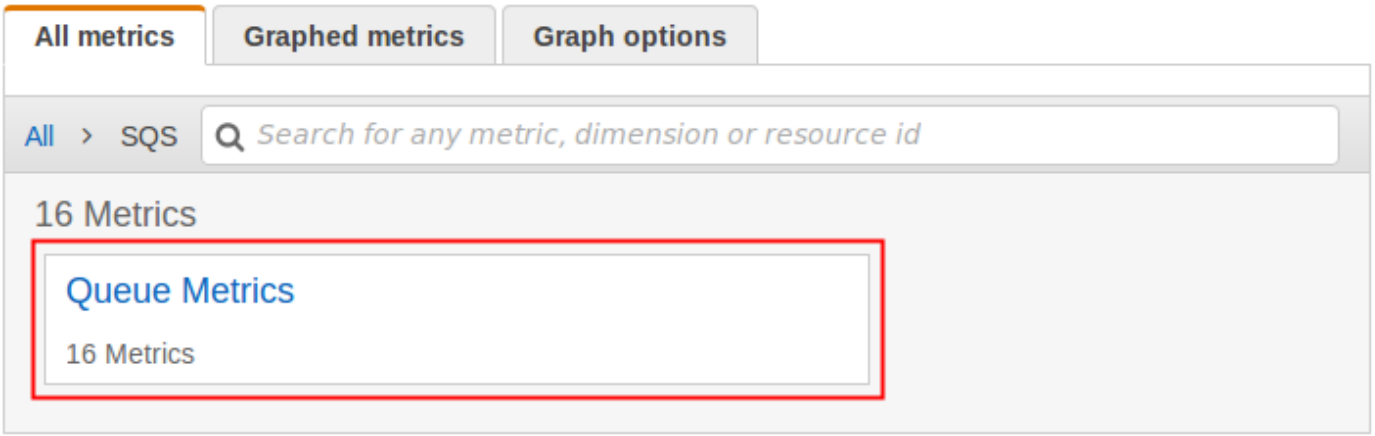
使用 CloudWatch 控制台访问和分析 Amazon SQS 指标。

1. 登录 CloudWatch 控制台。
2. 在导航面板上，选择 Metrics。

3. 选择 SQS 指标命名空间。

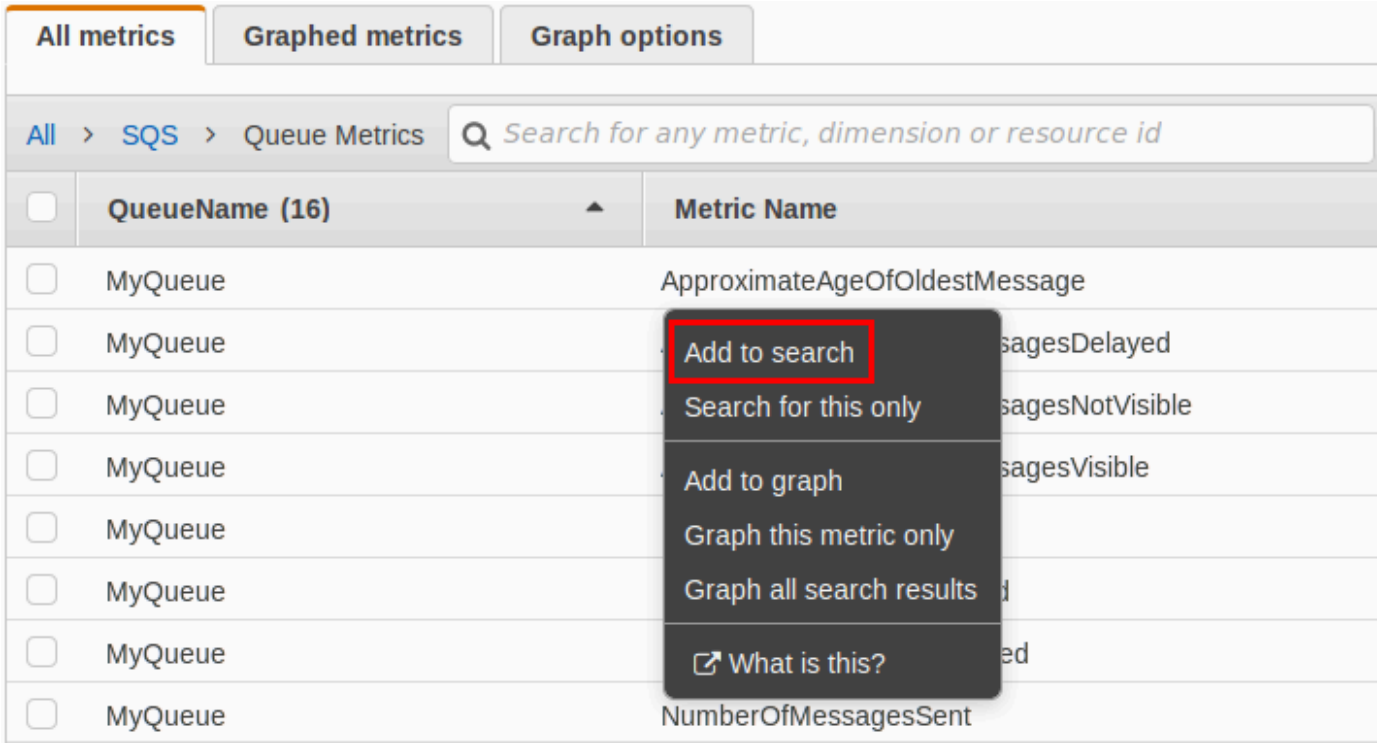


4. 选择 Queue Metrics 指标维度。



5. 现在可检查 Amazon SQS 指标：

   - 要对指标进行排序，请使用列标题。
   - 要为指标绘制图表，请选中该指标旁的复选框。
   - 要按指标进行筛选，请选择指标名称，然后选择 Add to search。



有关更多信息和其他选项，请参阅《亚马逊 CloudWatch 用户指南》中的绘制指标和使用亚马逊 CloudWatch 控制面板。

使用 AWS Command Line Interface

要使用访问亚马逊 SQS 指标 AWS CLI，请运行`get-metric-statistics`命令。

有关更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的获取指标统计数据。

使用 CloudWatch API

要使用 CloudWatch API 访问亚马逊 SQS 指标，请使用`GetMetricStatistics`操作。

有关更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的获取指标统计数据。

为 Amazon SQS 指标创建 CloudWatch 警报

CloudWatch 允许您在指标达到指定阈值时触发警报。例如，您可以为 `NumberOfMessagesSent` 指标创建警报。例如，如果在 1 个小时内向 `MyQueue` 队列发送了 100 条以上的消息，则会发出电子邮件通知。有关更多信息，请参阅亚马逊 CloudWatch 用户指南中的创建亚马逊 CloudWatch 警报。

1. 登录 AWS 管理控制台 并打开 CloudWatch 控制台，地址为https://console.aws.amazon.com/cloudwatch/。
2. 依次选择 Alarms 和 Create Alarm。
3. 在创建警报对话框的选择指标部分中，依次选择浏览指标和 SQS。
4. 在 SQS > 队列指标中，选择要设置警报的QueueName和指标名称，然后选择下一步。有关可用指标的列表，请参阅 亚马逊 SQS 的可用 CloudWatch 指标。

   在以下示例中，选择是针对 `NumberOfMessagesSent` 队列的 `MyQueue` 指标的警报。当发送的消息数超过 100 时，将触发警报。
5. 在创建警报对话框的定义警报部分中，执行以下操作：

   a. 在警报阈值下，为警报键入名称和描述。

   b. 设置 is 为 > 100。

   c. 将对于设置为 1 个数据点中的 1 个数据点。

   d. 在警报预览下，将周期设置为 1 小时。

   e. 将统计数据设置为标准和总计。

   f. 在操作下，将每当此警报设置为状态为“警报”。

   如果您 CloudWatch 想在警报触发时发送通知，请选择现有的 Amazon SNS 主题或选择 “新建列表”，然后输入以逗号分隔的电子邮件地址。

   > **Note**
   >
   > 如果创建一个新 Amazon SNS 主题，则电子邮件地址在接收任何通知之前必须通过验证。如果在验证电子邮件地址之前警报状态发生了变化，则不会发送通知。
6. 选择 Create Alarm（创建警报）。

   将创建警报。

亚马逊 SQS 的可用 CloudWatch 指标

亚马逊 SQS 将以下指标发送至。 CloudWatch

> **Note**
>
> 由于 Amazon SQS 采用分布式架构，某些指标的结果为近似值。在大多数情况下，该计数应接近队列中的实际消息数。

Amazon SQS 指标

亚马逊 SQS 会在AWS/SQS命名空间 CloudWatch下自动向亚马逊发布运营指标。这些指标可帮助您监控队列运行状况和性能。由于 SQS 的分布式特性，许多值都是近似值，但对于大多数运营决策来说足够准确。

> **Note**
>
> - 仅当队列处于活动状态时，所有指标才会输出非负值。
> - 有些指标（例如 `SentMessageSize`）要等到至少发送一条消息后才会输出。

指标	说明	单位	报告行为	关键说明
`ApproximateAgeOfOldestMessage`	队列中最早的未处理消息的期限。	秒	报告队列中是否至少包含一条活动消息。	• 对于标准队列，如果一条消息被接收三次或以上但未删除，SQS 会将其移到队列的后面。然后，该指标会反映下一条未超过接收阈值的消息的期限。即使实施了重新驱动策略，也会发生这种重新排序。 • Poison-pill 消息（重复收到但从未删除的消息）在成功处理之前不包括在此指标中。 • 当消息超过 `maxReceiveCount` 后被移到 DLQ 时，期限将重置。在这种情况下，DLQ 的指标反映的是

指标	说明	单位	报告行为	关键说明
				消息被移动的时间，而不是最初发送的时间。 • FIFO 队列不会为了保持顺序而对消息进行重新排序。失败的消息会阻止其消息组，直至被删除或到期。如果配置了 DLQ，则消息将在达到接收阈值后发送到 DLQ。
ApproximateNumberOfGroupsWithInflightMessages	仅适用于 FIFO。包含一条或多条传输中消息的消息组数量。	计数	报告 FIFO 队列是否处于活动状态。	• 传输中消息是指已被使用者从队列中接收但尚未删除或到期的消息。 • 此指标可帮助您排除故障和优化 FIFO 队列吞吐量。值越高通常表示并发性越强。 • 如果队列中有大量积压消息并且该值很低，请考虑扩大使用者规模或增加活动消息组的数量。 • 有关吞吐量和传输中消息数量上限，请参阅 Amazon SQS 配额。
ApproximateNumberOfMessagesDelayed	队列中延迟且无法立即检索的消息数量。	计数	报告队列中是否存在延迟消息。	• 适用于配置了默认延迟的队列以及使用 DelaySeconds 参数发送的单条消息。 • 延迟消息在延迟期到期之前一直对使用者隐藏，这可能会影响感知到的队列积压情况或吞吐量。

指标	说明	单位	报告行为	关键说明
ApproximateNumberOfMessagesNotVisible	已接收但尚未删除或到期的传输中消息数量。	计数	报告是否存在传输中消息。	• 消息通过 ReceiveMessage API 发送给使用者后即进入传输中状态。 • 这些消息在可见性超时窗口内会暂时向其他使用者隐藏。 • 使用此指标来跟踪消息处理延迟或卡住的使用者。 • 在极少数情况下，当 Amazon SQS 报告队列指标时，Amazon SQS 用来存储您的消息的服务器之一可能不可用。发生这种情况时，此指标可能会报告非零值，包括队列为空或没有ReceiveMessage 呼叫的情况。它通常会在几分钟内解决，因此，如果您为此指标设置警报，请评估多个连续的数据点，而不是单个非零值。
ApproximateNumberOfMessagesVisible	当前可供检索和处理的消息数量。	计数	报告队列是否处于活动状态。	• 反映队列中当前的处理积压情况。 • 可以累积的消息数量没有硬性限制，但它们受队列配置的保留期的限制。 • 值持续较高可能表示使用者预置不足或处理逻辑卡滞。

指标	说明	单位	报告行为	关键说明
NumberOfEmptyReceives [1]	未返回消息的 ReceiveMessage API 调用数量。	计数	在接收操作期间报告。	• 此指标有助于识别轮询行为中的低效问题或未充分利用的使用者实例。 • 当队列为空、使用者使用短轮询或者消息处理速度快于生成速度时，可能会出现高值。 • 这并不能准确反映队列状态。它反映的是服务端行为，可能包括重试。
NumberOfDeduplicatedSentMessages	仅适用于 FIFO。经过重复数据删除且未添加到队列的已发送消息数量。	计数	报告是否检测到重复的 MessageDeduplicationId 值或内容。	• SQS 根据 MessageDeduplicationId 或基于内容的哈希（如果启用）对消息进行重复数据删除。 • 值较高可能表示创建者在 5 分钟的重复数据删除窗口内反复发送同一消息。 • 使用此指标排除冗余创建者逻辑问题或确认重复数据删除是否按预期运行。

指标	说明	单位	报告行为	关键说明

NumberOfMessagesDeleted [1]	从队列中成功删除的消息数量。	计数	报告每个经过有效接收处理的删除请求。	• 此指标会计算所有成功的删除操作，即使同一条消息被删除多次也是如此。
				• 值高于预期的常见原因包括：
				  • 在可见性超时到期并再次收到消息后，使用不同的接收处理方式多次删除同一消息。
				  • 使用相同的接收处理方式重复删除，仍会返回成功状态并增加指标值。
				• 使用此指标来跟踪消息处理成功率，但不要将其视为唯一已删除消息的确切计数。

NumberOfMessagesReceived [1]	ReceiveMessageAPI 返回的消息数量。	计数	在接收操作期间报告。	• 这包括返回给使用者的所有消息，包括那些由于可见性超时到期而随后返回到队列的消息。 • 如果某条消息未被删除，则可能会被多次接收，这可能导致此指标超过发送的消息数量。 • 使用此指标来跟踪使用者活动，但不要将其视为唯一已处理消息的计数。

指标	说明	单位	报告行为	关键说明
NumberOfMessagesSent [1]	成功添加到队列的消息数量。	计数	报告每次成功手动发送。	• 手动调用 SendMessage 或 SendMessageBatch 都会计算在内，包括那些直接定向 DLQ 的调用。 • 超过 maxReceiveCount 后自动移到 DLQ 的消息不包含在此指标中。 • 因此，NumberOfMessagesSent 可能低于 NumberOfMessagesReceived ，尤其是在重新驱动策略在后台将许多消息移到 DLQ 的情况下。
SentMessageSize [1]	成功发送到队列的消息的大小。	字节	至少发送一条消息后才会输出。	• 直到队列收到第一条消息，该指标才会出现在 CloudWatch 控制台中。 • 使用此指标来跟踪每条消息的大小（以字节为单位）。这对于分析有效载荷趋势或估算吞吐量成本非常有用。 • SQS 的最大消息大小为 1 MiB。
ApproximateNumberOfNoisyGroups	公平队列中被视为嘈杂邻居的消息组的数量。嘈杂消息组表示多租户队列中的嘈杂邻居租户。	计数	如果队列处于活动状态，则报告非负值。	• 通过跟踪使用过多资源的消息组，帮助识别多租户环境中潜在的嘈杂邻居问题。 • 使用此指标来设置警报，当嘈杂组的数量超过可接受的阈值时触发，表示可能存在队列公平性问题。

指标	说明	单位	报告行为	关键说明
ApproximateNumberOfMessagesVisibleInQuietGroups	可见消息数量，不包括来自嘈杂消息组的消息。	计数	如果队列处于活动状态，则报告非负值。	• 能够查看标准速率消息组的队列积压情况，不包括来自嘈杂邻居的消息。 • 通过筛除嘈杂邻居的影响，帮助确定典型消息组的真实处理积压情况。
ApproximateNumberOfMessagesNotVisibleInQuietGroups	传输中消息数量，不包括来自嘈杂消息组的消息。	计数	如果队列处于活动状态，则报告非负值。	• 跟踪来自行为良好的消息组的传输中消息（正在处理但尚未删除）。 • 使用此指标来监控常规消息组的处理吞吐量，并检测不是由嘈杂邻居引起的处理瓶颈。
ApproximateNumberOfMessagesDelayedInQuietGroups	延迟且无法立即读取的消息数量，不包括来自嘈杂消息组的消息。如果队列被配置为延迟队列，或者发送消息时使用了延迟参数，则会出现延迟消息。	计数	如果队列处于活动状态，则报告非负值。	• 帮助监控来自具有正常或预期吞吐模式的消息组（而不是高容量或嘈杂组）的延迟消息积压情况 • 有助于了解未来处理需求并为典型工作负载进行容量规划。

指标	说明	单位	报告行为	关键说明
ApproximateAgeOfOldestMessageInQuietGroups	队列中最早的未删除消息的期限，不包括来自嘈杂消息组的消息。	秒	如果队列处于活动状态，则报告非负值。	• 用于监控 SLA 合规性并检测具有正常或预期吞吐量模式的消息组（而不是高容量或嘈杂消息组，否则可能会歪曲指标）中的处理瓶颈。 • 使用此指标设置消息处理超时警报，忽略来自嘈杂邻居的人为旧消息。

[1] 这些指标反映的是系统级别的活动，可能包括重试、重复或延迟消息。如果不考虑消息生命周期行为，请勿使用原始计数来估算实时队列状态。

Dead-letter 队列 (DLQ) 和指标 CloudWatch

使用 DLQ 时，了解 Amazon SQS 指标的行为非常重要：

- **NumberOfMessagesSent** – 对于 DLQ，此指标的行为有所不同：
  - 手动发送 – 此指标将捕获手动发送到 DLQ 的消息。
  - 自动重新驱动 – 此指标不捕获由于处理失败而自动移到 DLQ 的消息。因此，对于 DLQ，NumberOfMessagesSent 和 NumberOfMessagesReceived 指标可能存在差异。
- DLQ 的推荐指标 – 要监控 DLQ 的状态，请使用 ApproximateNumberOfMessagesVisible 指标。此指标表示 DLQ 中当前可供处理的消息数量。

公平的队列和指标 CloudWatch

当您使用公平队列时，Amazon SQS 会输出以下额外指标：

- ApproximateNumberOfNoisyGroups
- ApproximateNumberOfMessagesVisibleInQuietGroups
- ApproximateNumberOfMessagesNotVisibleInQuietGroups
- ApproximateNumberOfMessagesDelayedInQuietGroups
- ApproximateAgeOfOldestMessageInQuietGroups

> **Note**
>
> 每个 QuietGroup 指标都是等效标准队列级别 Approximate 指标的子集，但不包括来自嘈杂邻居组的消息。

嘈杂组

嘈杂消息组表示多租户队列中的嘈杂邻居租户。

安静组

嘈杂组以外的消息组。

观察 SQS 公平队列行为

要监控 Amazon SQS 公平队列的影响，您可以将 Approximate..InQuietGroups 指标与标准队列级别指标进行比较。在特定租户的流量激增期间，队列级别的一般指标可能会显示积压的消息增加或消息停留时间延长。但是，如果单独查看安静组，您可以确定大多数非嘈杂消息组或租户都没有受到影响，并估算出受影响的消息组总数。

虽然通过这些新指标可以了解 Amazon SQS 公平队列行为，但了解哪个特定租户导致了负载问题可能会有所帮助。亚马逊 CloudWatch 贡献者见解使您可以查看有关前 N 位贡献者、独立贡献者总数及其使用情况的指标。这在处理数千个租户的场景中特别有用，否则在输出传统指标时会导致高基数数据（和成本）。

有关公平队列监控配置的示例，请参阅上的GitHub示例。

Amazon SQS 指标的维度

CloudWatch 使用单一维度的亚马逊 SQS 指标：**QueueName**。所有指标数据均按队列名称进行分组和筛选。

监控提示

使用关键指标和 CloudWatch 警报有效监控 SQS，以检测队列积压、优化性能并保持在服务限制范围内。

- 根据设置CloudWatch 警报ApproximateNumberOfMessagesVisible以捕捉待办事项的增长。
- 通过监控 NumberOfEmptyReceives 来调整轮询频率并降低 API 成本。

- 在 FIFO 队列中使用 `ApproximateNumberOfGroupsWithInflightMessages` 来诊断吞吐量限制。
- 查看 SQS 配额以了解指标阈值和服务限制。

Amazon SQS 的合规性验证

我们的新 AWS 注册体验不是为受监管的工作负载设计的。如果您正在使用我们的全新 AWS 注册体验，但想 AWS 用于受监管的工作负载，则可以注册 AWS （高级）或激活 AWS 环境的高级功能。

要了解是否属于特定合规计划的范围，请参阅 “按合规计划 AWS 服务 中的 “按范围划分的合规计划”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于数据的敏感性、贵公司的合规目标以及适用的法律法规。有关您在使用时的合规责任的更多信息 AWS 服务，请参阅AWS 安全文档。

Amazon SQS 中的恢复功能

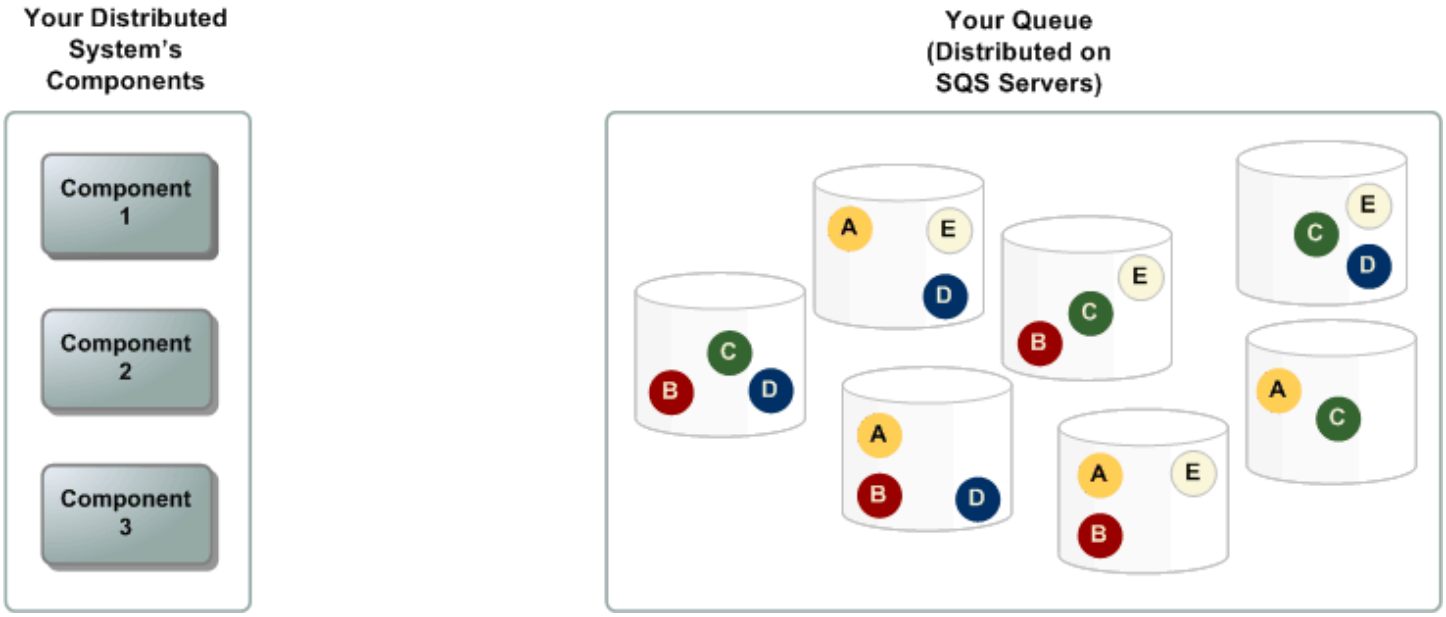
AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错能力和可扩展性。有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

除了 AWS 全球基础设施外，Amazon SQS 还提供分布式队列。

分布式队列

分布式消息收发系统有三个主要部分：分布式系统的组件、队列（分布在 Amazon SQS 服务器上）以及队列中的消息。

在以下场景中，您的系统有多个创建者（向队列发送消息的组件）和使用者（从队列接收消息的组件）。队列（保存从 A 到 E 的消息）在多个 Amazon SQS 服务器上冗余存储消息。



Amazon SQS 中的基础设施安全性

作为一项托管服务，Amazon SQS 受亚马逊网络服务：安全流程概述白皮书中描述的 AWS 全球网络安全程序的保护。

您可以使用 AWS 已发布的 API 操作通过网络访问亚马逊 SQS。客户端必须支持传输层安全性协议（TLS）1.2 或更高版本。客户端还必须支持具有完全向前保密 (PFS) 的密码套件，例如 Ephemeral Diffie-Hellman (DHE) 或 Elliptic Curve Ephemeral Diffie-Hellman (ECDHE)。

您必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service (AWS STS) 生成用于签名请求的临时安全凭证。

您可以从任何网络位置调用这些 API 操作，但 Amazon SQS 支持基于资源的访问策略，其中可以包含基于源 IP 地址的限制。您还可以使用 Amazon SQS 策略来控制来自特定亚马逊 VPC 终端节点或特定终端节点的访问。 VPCs这可以有效地将对给定 Amazon SQS 队列的网络访问与网络中的 AWS 特定 VPC 隔离开来。有关更多信息，请参阅 示例 5：如果不是来自 VPC 端点，则拒绝访问。

Amazon SQS 安全最佳实践

AWS 为 Amazon SQS 提供了许多安全功能，您应根据自己的安全策略查看这些功能。以下是针对 Amazon SQS 的预防性安全最佳实践。

> **Note**
>
> 提供的具体实施指南适用于常见的使用案例和实施。我们建议您在特定使用案例、架构和威胁模型的上下文中查看这些最佳实践。

确保队列不可公开访问

除非您明确要求互联网上的任何人能够读取或写入您的 Amazon SQS 队列，否则应确保您的队列不可公开访问（世界上所有人或任何经过身份验证的 AWS 用户都可以访问）。

- 避免创建 `Principal` 设置为 `""` 的策略。
- 避免使用通配符 (*)。相反，对一个特定用户或多个用户命名。

实施最低权限访问

授予权限后，您可以决定这些权限的接收者、权限所针对的队列以及要允许这些队列使用的特定 API 操作。实施最低权限对于降低安全风险并减少错误或恶意意图的影响至关重要。

遵循授予最低权限的标准安全建议。也就是说，只授予执行特定任务所需的权限。您可以使用安全策略的组合来实现这一点。

Amazon SQS 使用创建者-使用者模型，并且需要以下三类用户账户访问权限：

- 管理员 - 用于创建、修改和删除队列的访问权限。管理员还控制队列策略。
- 创建者 - 用于将消息发送到队列的访问权限。
- 使用者 - 用于接收和删除来自队列的消息的访问权限。

有关详细信息，请参阅以下章节：

- Amazon SQS 中的身份和访问管理
- Amazon SQS API 权限：操作和资源参考
- 将自定义策略与 Amazon SQS 访问策略语言配合使用

对需要 Amazon SQS 访问权限的应用程序和 AWS 服务使用 IAM 角色

要使诸如 Amazon EC2 之类的应用程序或 AWS 服务访问亚马逊 SQS 队列，它们必须在 AWS API AWS 请求中使用有效的证书。由于这些证书不会自动轮换，因此您不应将 AWS 证书直接存储在应用程序或 EC2 实例中。

您应该使用 IAM 角色来管理需要访问 Amazon SQS 的应用程序或服务的临时凭证。使用角色时，无需向 EC2 实例或 AWS 服务分配长期证书（例如用户名、密码和访问密钥），例如 AWS Lambda。相反，该角色提供临时权限，供应用程序在调用其他 AWS 资源时使用。

有关更多信息，请参阅 IAM 用户指南中的 IAM 角色和角色的常见场景：用户、应用程序和服务。

实施服务器端加密

要减少数据泄漏问题，可以通过静态加密，使用存储在与消息存储位置不同的位置的密钥来对消息进行加密。服务器端加密 (SSE) 提供静态数据加密。Amazon SQS 在存储消息时将在消息级别对数据进行加密，并在您访问消息时为您解密消息。SSE 使用中管理的密钥 AWS Key Management Service。只要您对请求进行身份验证并具有访问权限，访问加密队列和未加密队列之间就没有区别。

有关更多信息，请参阅Amazon SQS 中的静态加密和Amazon SQS 密钥管理。

实施传输中数据加密

如果没有 HTTPS (TLS)，基于网络的攻击者可以使用诸如之类的攻击窃听网络流量或对其进行操纵。man-in-the-middle仅允许使用队列策略中的 `aws:SecureTransport` 条件通过 HTTPS (TLS) 加密连接以强制请求使用 SSL。

考虑使用 VPC 端点来访问 Amazon SQS

如果您的队列必须能够与之交互，但是绝对不能暴露在 Internet 上，则可使用 VPC 端点来仅为对特定 VPC 中的主机的访问进行排队。您可以使用队列策略来控制从特定 Amazon VPC 终端节点或特定终端节点访问队列 VPCs。

Amazon SQS VPC 端点提供两种方式来控制对消息的访问：

- 您可以控制允许通过特定 VPC 端点访问的请求、用户或组。
- 您可以使用队列策略控制哪些 VPCs 或 VPC 终端节点可以访问您的队列。

有关更多信息，请参阅Amazon SQS 的 Amazon Virtual Private Cloud 端点和为 Amazon SQS 创建 Amazon VPC 端点策略。

Amazon SQS 的相关资源

下表列出了在您使用此服务时可能为您提供帮助的相关资源。

资源	描述
Amazon Simple Queue Service API 参考	操作、参数和数据类型的说明，以及该服务返回的错误的列表。
AWS CLI 命令参考中的 Amazon SQS	可用于使用队列的 AWS CLI 命令的说明。
区域和端点	提供有关 Amazon SQS 区域和端点的信息
产品页面	Amazon SQS 相关信息的主网页。
开发论坛	一个基于社区的论坛，供开发人员讨论与 Amazon SQS 相关的技术问题。
AWS Premium Support 信息	提供有关 AWS Premium Support 信息的主网页，它是一种一对一、快速响应的支持渠道，可帮助您在 AWS 基础设施服务上构建和运行应用程序。

文档历史记录

下表介绍了 2019 年 1 月之后对《Amazon Simple Queue Service 开发人员指南》的重要更改。如需有关本文档更新的通知，您可以订阅 RSS 源。

服务功能有时会逐步推广到提供服务的 AWS 区域。我们仅在第一次发布时更新了此文档。我们不提供有关区域可用性的信息，也不会宣布后续区域支持情况。有关服务功能的地区可用性以及订阅更新通知的信息，请参阅新增内容 AWS？ 。

变更	说明	日期
公平队列支持	为标准队列增加了公平队列支持。	2025 年 7 月 21 日
增加了对双堆栈（IPv4 和 IPv6）端点的支持	Amazon SQS 现在支持双堆栈（IPv4 和 IPv6）终端节点，允许通过两个 IP 协议访问队列。	2025 年 4 月 17 日
CloudTrail 适用于所有 Amazon SQS 的集成 APIs	CloudTrail 为所有 Amazon SQ APIs S 添加了集成。	2025 年 1 月 10 日
SQSUnlockQueuePolicy	Amazon SQS 添加了一个名为的新 AWS托管策略，该策略SQSUnlockQueuePolicy 用于解锁队列并删除一个配置错误的队列策略，该策略拒绝所有委托人访问 Amazon SQS 队列。	2024 年 11 月 15 日
AWS kms:Decrypt	Amazon SQS 不再需要 SendMessage API 的 kms:Decrypt 权限。客户现在只需要获得对用于加密队列的 KMS 密钥的 kms:GenerateDataKey 权限，但仍然需要 kms:Decrypt 权限来调用 ReceiveMessage 。	2024 年 7 月 24 日

FIFO 指标更新	在 Amazon SQS FIFO 指标中增加了对 `NumberOfDeduplicatedSentMessages` 和 `ApproximateNumberOfGroupsWithInflightMessages` 的支持。	2024 年 7 月 3 日
`ListQueueTags` Amazon SQSRead OnlyAccess 托管政策支持的操作	亚马逊SQSReadOnlyAccess 托管策略支持`ListQueueTags` 检索与指定 Amazon SQS 队列关联的所有标签。	2024 年 5 月 2 日
AWS JSON 协议	使用 AWS JSON 协议发出 API 请求。	2023 年 7 月 27 日
新章节介绍了 Amazon SQS 的 AWS 托管策略以及这些政策的更新	Amazon SQS 添加了一个新操作，允许您列出特定源队列下最新的消息移动任务（最多 10 个）。此操作与 `ListMessageMoveTasks` API 操作关联。	2023 年 6 月 7 日
使用 dead letter 队列重新驱动 APIs	使用 Amazon SQS 配置死信队列重新驱动器。 APIs	2023 年 6 月 7 日
Amazon SQS 的 ABAC	基于属性的访问控制 (ABAC) 使用队列标签来获得灵活且可扩展的访问权限。	2022 年 11 月 10 日
FIFO 高吞吐量限制增加	增加了商业区域中 FIFO 高吞吐量模式的默认配额，以及 FIFO 高吞吐量文档优化。	2022 年 10 月 20 日
提供默认服务器端加密 (SSE)	默认情况下，使用 SQS 拥有的加密 (SSE-SQS) 进行服务器端加密 (SSE)。	2022 年 9 月 26 日

提供 Amazon SQS 混淆代理保护支持	混淆代理保护允许您在请求中指定新的标头，系统会在使用 Amazon SQS 托管 SSE 时根据 KMS 策略中的条件检查这些标头。	2021 年 12 月 29 日
提供托管 SSE	Amazon SQS 托管 SSE (SSE-SQS) 是一种托管式服务器端加密，它使用 Amazon SQS 拥有的加密密钥来保护通过消息队列发送的敏感数据。	2021 年 11 月 23 日
提供死信队列重新驱动	Amazon SQS 支持标准队列的死信队列重新驱动。	2021 年 11 月 10 日
提供 FIFO 队列中消息的高吞吐量	Amazon SQS FIFO 队列的高吞吐量为 FIFO 列中的消息提供了更高的每秒事务数 (TPS)。有关吞吐量配额的信息，请参阅与消息相关的配额。	2021 年 5 月 27 日
提供预览版 FIFO 队列中消息的高吞吐量	Amazon SQS FIFO 队列的高吞吐量提供预览版，可能会有所变化。此特征为 FIFO 队列中的消息提供更高的每秒事务数 (TPS)。有关吞吐量配额的信息，请参阅与消息相关的配额。	2020 年 12 月 17 日
全新 Amazon SQS 控制台设计	为了简化开发和生产工作流，Amazon SQS 控制台提供了全新的用户体验。	2020 年 7 月 8 日
Amazon SQS 支持对列表队列进行分页和 listDeadLetterSourceQueues	您可以指定从列表队列或listDeadLetterSourceQueues请求返回的最大结果数。	2020 年 6 月 22 日

Amazon SQS 在除 AWS GovCloud 了（美国）地区以外 AWS 的所有地区都支持 1 分钟的 CloudWatch 亚马逊指标	Amazon SQS 的一分钟 CloudWatch 指标适用于除 AWS GovCloud (US) 区域之外的所有区域。	2020 年 1 月 9 日
亚马逊 SQS 支持 1 分钟指标 CloudWatch	Amazon SQS 的一分钟 CloudWatch 指标目前仅在以下地区可用：美国东部（俄亥俄州）、欧洲（爱尔兰）、欧洲（斯德哥尔摩）和亚太地区（东京）。	2019 年 11 月 25 日
AWS Lambda Amazon SQS FIFO 队列的触发器可用	您可以配置 FIFO 队列中抵达的消息来触发 Lambda 函数。	2019 年 11 月 25 日
Amazon SNS 的服务器端加密 (SSE) 在中国区域可用	Amazon SQS 的 SSE 在中国区域可用。	2019 年 11 月 13 日
FIFO 队列在中东（巴林）区域可用	FIFO 队列在中东（巴林）区域可用。	2019 年 10 月 10 日
适用于亚马逊 SQS 的亚马逊 Virtual Private Cloud（亚马逊 VPC）终端节点在 AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）地区可用	您可以从 AWS GovCloud（美国东部）和（美国西部）区域的 Amazon VPC 向您的 Amazon SQS 队列发送消息。AWS GovCloud	2019 年 9 月 5 日

Amazon SQS 允许 AWS X-Ray 使用消息系统属性对队列进行故障排除	您可以对通过使用 X-Ray 的 Amazon SQS 队列传递的消息进行问题排查。此版本为 `SendMessage` 和 `SendMessageBatch` API 操作（可让您通过 Amazon SQS 发送 X-Ray 跟踪标头）添加了 `MessageSystemAttribute` 请求参数，为 `ReceiveMessage` API 操作添加了 `AWSTraceHeader` 属性，另外还增加了 `MessageSystemAttributeValue` 数据类型。	2019 年 8 月 28 日
您可以在创建时标记 Amazon SQS 队列	您可以使用单个 Amazon SQS API 调用、 AWS SDK 函数或 AWS Command Line Interface (AWS CLI) 命令来同时创建队列并指定其标签。此外，Amazon SQS 还支持 `aws:TagKeys` 和 `aws:RequestTag` AWS Identity and Access Management (IAM) 密钥。	2019 年 8 月 22 日
适用于 Amazon SQS 的临时队列客户端现已推出	使用 request-response 等常见消息模式时，临时队列可帮助您节省开发时间和部署成本。您可以使用临时队列客户端创建高吞吐量、经济实惠、由应用程序管理的临时队列。	2019 年 7 月 25 日
适用于 Amazon SQS 的 SSE 已在 AWS GovCloud（美国东部）地区推出	适用于 Amazon SQS 的服务器端加密 (SSE) 在（美国东部）地区 AWS GovCloud 提供。	2019 年 6 月 20 日

在亚太地区（香港）、中国（北京）、（美国东部）和 AWS GovCloud AWS GovCloud （美国西部）地区提供 FIFO 队列	在亚太地区（香港）、中国（北京）、（美国东部）和 AWS GovCloud AWS GovCloud （美国西部）地区提供 FIFO 队列。	2019 年 5 月 15 日
Amazon VPC 端点策略适用于 Amazon SQS	您可以为 Amazon SQS 创建 Amazon VPC 端点策略。	2019 年 4 月 4 日
FIFO 队列在欧洲地区（斯德哥尔摩）和中国（宁夏）区域推出	FIFO 队列在欧洲地区（斯德哥尔摩）和中国（宁夏）区域推出。	2019 年 3 月 14 日
FIFO 队列在所有提供 Amazon SQS 的区域推出	FIFO 队列在以下区域推出：美国东部（俄亥俄州）、美国东部（弗吉尼亚州北部）、美国西部（北加利福尼亚）、美国西部（俄勒冈州）、亚太地区（孟买）、亚太地区（首尔）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲地区（法兰克福）、欧洲地区（爱尔兰）、欧洲地区（伦敦）、欧洲地区（巴黎）和南美洲（圣保罗）。	2019 年 2 月 7 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。