



开发人员指南

Amazon Simple Queue Service



Amazon Simple Queue Service: 开发人员指南

Copyright © 2024 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	2
消息生命周期	2
Amazon SQS、Amazon MQ 和 Amazon SNS 之间的区别	4
设置	6
步骤 1：创建 AWS 账户 和 IAM 用户	6
注册获取 AWS 账户	6
创建具有管理访问权限的用户	7
第 2 步：授权以编程方式访问	8
第 3 步：为使用示例代码做好准备	9
后续步骤	10
开始使用	11
先决条件	11
了解 Amazon SQS 控制台	11
队列类型	12
在 Amazon SQS 中实现请求-响应系统	13
创建标准队列	14
创建队列	14
发送消息	15
创建 FIFO 队列	16
创建队列	16
发送消息	18
常见任务	19
管理队列	20
先决条件	11
了解 Amazon SQS 控制台	11
编辑队列	21
接收和删除消息	21
确认队列为空	23
删除队列	24
清除队列	25
标准队列	26

亚马逊 SQS at-least-once 配送	26
队列和消息标识符	26
标准队列的标识符	27
FIFO 队列	29
FIFO 队列关键术语	30
FIFO 传送逻辑	31
仅处理一次	32
从标准队列移至 FIFO 队列	33
FIFO 队列和 Lambda 并发行为	34
FIFO 队列消息分组	34
与 FIFO 队列的 Lambda 并发性	34
使用案例示例	35
FIFO 队列的高吞吐量	35
用例	36
分区和数据分布	36
为 FIFO 队列启用高吞吐量	38
队列和消息标识符	39
FIFO 队列的标识符	27
FIFO 队列的其他标识符	40
配额	42
FIFO 队列配额	42
Amazon SQS 配额	42
标准队列配额	43
邮件配额	45
策略配额	49
特征和功能	50
死信队列	50
对死信队列使用策略	51
了解死信队列的消息保留期	51
配置死信队列	51
配置死信队列重新驱动	52
CloudTrail 更新和权限要求	58
使用 Amazon 为死信队列创建警报 CloudWatch	61
亚马逊 SQS 的消息元数据	62
消息属性	62
消息系统属性	66

处理消息所需的资源	66
列表队列分页	67
成本分配标签	67
短轮询和长轮询	68
通过短轮询来使用消息	69
通过长轮询来使用消息	69
长轮询和短轮询之间的区别	70
可见性超时	70
传输中消息	71
设置可见性超时	72
更改消息的可见性超时	73
终止消息的可见性超时	73
延迟队列	74
临时队列	74
虚拟队列	75
请求-响应消息收发模式 (虚拟队列)	76
示例场景：处理登录请求	77
清除队列	79
消息定时器	80
访问 EventBridge 管道	80
管理大消息	81
使用适用于 Java 的扩展客户端库	82
使用适用于 Python 的扩展客户端库	91
配置 Amazon SQS	95
Amazon SQS 的 ABAC	95
什么是 ABAC？	95
为什么应该在 Amazon SQS 中使用 ABAC？	96
访问控制的标记	96
创建 IAM 用户和 Amazon SQS 队列	97
测试基于属性的访问控制	101
配置队列参数	102
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

配置 Lambda 触发器	108
先决条件	108
使用自动发送通知 EventBridge	109
消息属性	110
最佳实践	112
错误处理和有问题的消息	112
处理 Amazon SQS 中的请求错误	112
在 Amazon SQS 中捕获有问题的消息	112
在 Amazon SQS 中设置死信队列保留	113
消息去重和分组	113
避免 Amazon SQS 中的消息处理不一致	113
使用消息重复数据删除 ID	114
使用消息组 ID	115
使用接收请求尝试 ID	116
消息处理和定时	116
在 Amazon SQS 中及时处理消息	117
在 Amazon SQS 中设置长时间投票	117
在 Amazon SQS 中使用适当的轮询模式	118
Java SDK 示例	119
使用服务器端加密	119
向现有队列添加 SSE	119
为队列禁用 SSE	120
使用 SSE 创建队列	120
检索 SSE 属性	121
配置标签	121
列出标签	121
添加或更新标签	122
删除标签	123
发送消息属性	123
定义属性	123
发送带有属性的消息	125
使用 API	126
使用 AWS JSON协议发出查询API请求	127
构造端点	127
提出POST请求	128
解读亚马逊的SQSJSONAPI回应	129

亚马逊SQS AWS JSON协议 FAQs	130
使用查询协议发出 AWS 查询 API 请求	133
构造端点	133
提出 GET 请求	134
提出 POST 请求	128
解释 Amazon SQS XML API 响应	135
对请求进行身份验证	137
使用 HMAC-SHA 的基本身份验证过程	137
第 1 部分：来自用户的请求	138
第 2 部分：来自的回应 AWS	139
批处理操作	140
批处理消息操作	141
使用 Amazon SQS 启用客户端缓冲和请求批处理	141
通过 Amazon SQS 使用水平扩展和操作批处理来提高吞吐量	147
使用 AWS 软件开发工具包	160
使用 JMS	162
先决条件	162
使用 Java 消息库	163
创建 JMS 连接	164
创建 Amazon SQS 队列	164
同步发送消息	165
同步接收消息	166
异步接收消息	168
使用客户端确认模式	169
使用无序确认模式	170
将 JMS 客户端与其他 Amazon SQS 客户端配合使用	171
在标准队列中使用 JMS 的 Java 工作示例	172
ExampleConfiguration.java	172
TextMessageSender.java	175
SyncMessageReceiver.java	177
AsyncMessageReceiver.java	178
SyncMessageReceiverClientAcknowledge.java	180
SyncMessageReceiverUnorderedAcknowledge.java	184
SpringExampleConfiguration.xml	188
SpringExample.java	189
ExampleCommon.java	191

支持的 JMS 1.1 实施	193
支持的常用接口	193
支持的消息类型	193
支持的消息确认模式	194
JMS 定义的标头和预留属性	194
教程	196
使用创建 Amazon SQS 队列 AWS CloudFormation	196
从 VPC 发送消息	198
步骤 1：创建 Amazon EC2 密钥对	198
步骤 2：创建 AWS 资源	199
步骤 3：确认您的 EC2 实例不可公开访问	200
步骤 4：为 Amazon SQS 创建 Amazon VPC 端点	201
步骤 5：向 Amazon SQS 队列发送消息	202
代码示例	203
操作	213
AddPermission	214
ChangeMessageVisibility	215
ChangeMessageVisibilityBatch	221
CreateQueue	223
DeleteMessage	243
DeleteMessageBatch	252
DeleteQueue	260
GetQueueAttributes	270
GetQueueUrl	277
ListDeadLetterSourceQueues	284
ListQueues	285
PurgeQueue	297
ReceiveMessage	298
RemovePermission	315
SendMessage	316
SendMessageBatch	332
SetQueueAttributes	340
场景	351
创建并发布到FIFO主题	351
处理 S3 事件通知	364
将消息发布到队列	367

发送和接收批量消息	463
无服务器示例	468
从亚马逊触发器调用 Lambda 函数 SQS	469
使用亚马逊触发器报告 Lambda 函数的批量项目失败 SQS	477
跨服务示例	487
创建消息收发应用程序	487
创建 Messenger 应用程序	488
创建 Amazon Textract 浏览器应用程序	489
检测视频中的人物和对象	490
将 AWS 消息处理框架用于 .NET 与亚马逊合作 SQS	491
故障排除	492
访问被拒绝错误	492
Amazon SQS 队列策略和 IAM 政策	493
AWS Key Management Service (AWS KMS) 权限	493
VPC 端点策略	495
组织服务控制政策	495
API 错误	495
QueueDoesNotExist 错误	496
InvalidAttributeValue 错误	496
ReceiptHandle 错误	497
DLQ 还有 DLQ 重启问题	497
DLQ 问题	498
DLQ-重新驱动问题	499
FIFO 限制问题	501
ReceiveMessage API 调用未返回的消息	502
空队列	502
已达到飞行限制	502
消息延迟	502
消息正在传送中	502
轮询方法	503
网络错误	503
ETIMEDOUT error	503
UnknownHostException error	504
使用 X-Ray 排查队列问题	505
安全性	506
数据保护	506

数据加密	507
互连网络流量隐私保护	517
Identity and Access Management	519
受众	519
使用身份进行身份验证	520
使用策略管理访问	522
概述	524
Amazon 简单队列服务的工作原理 IAM	530
AWS 托管策略	536
故障排除	537
使用 策略	539
日记账记录和监控	581
使用记录 API 调用 CloudTrail	581
使用监控队列 CloudWatch	594
合规性验证	606
弹性	607
分布式队列	608
基础设施安全性	608
最佳实践	609
确保队列不可公开访问	609
实施最低权限访问	609
对需要 Amazon SQS 访问权限的应用程序和 AWS 服务使用 IAM 角色	610
实施服务器端加密	610
实施传输中数据加密	610
考虑使用 VPC 端点来访问 Amazon SQS	611
相关资源	612
文档历史记录	613
.....	dcxviii

什么是 Amazon 简单队列服务

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，您可以使用该 AWS 软件开发工具包支持的任何编程语言进行访问。

主题

- [使用 Amazon SQS 的优势](#)
- [基本 Amazon SQS 架构](#)
- [Amazon SQS、Amazon MQ 和 Amazon SNS 之间的区别](#)

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 at-least-once 消息传送](#)，[FIFO 队列支持精确一次的消息处理和高吞吐量模式](#)。
- 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 256 KB 的消息内容，Amazon SQS 保留指向 Amazon S3 对象的指针，您也可以将一条大消息拆分为几个小消息。

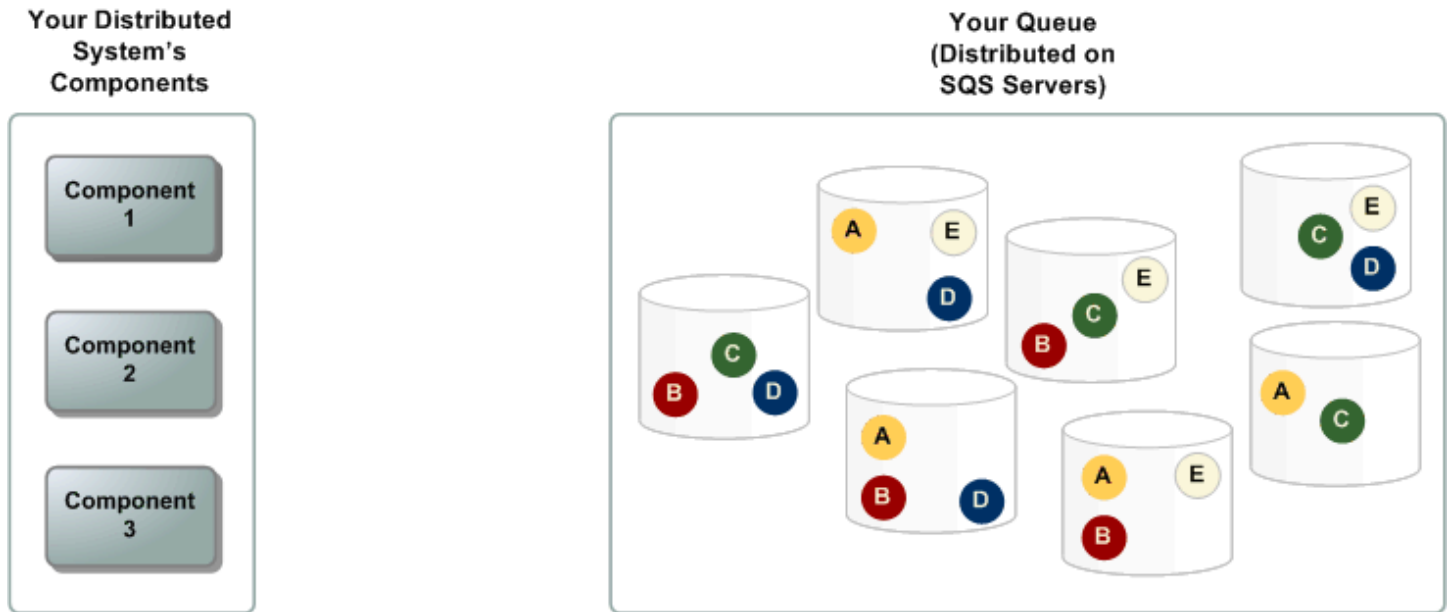
基本 Amazon SQS 架构

本节简要介绍分布式消息传送系统的组成部分并说明 Amazon SQS 消息的生命周期。

分布式队列

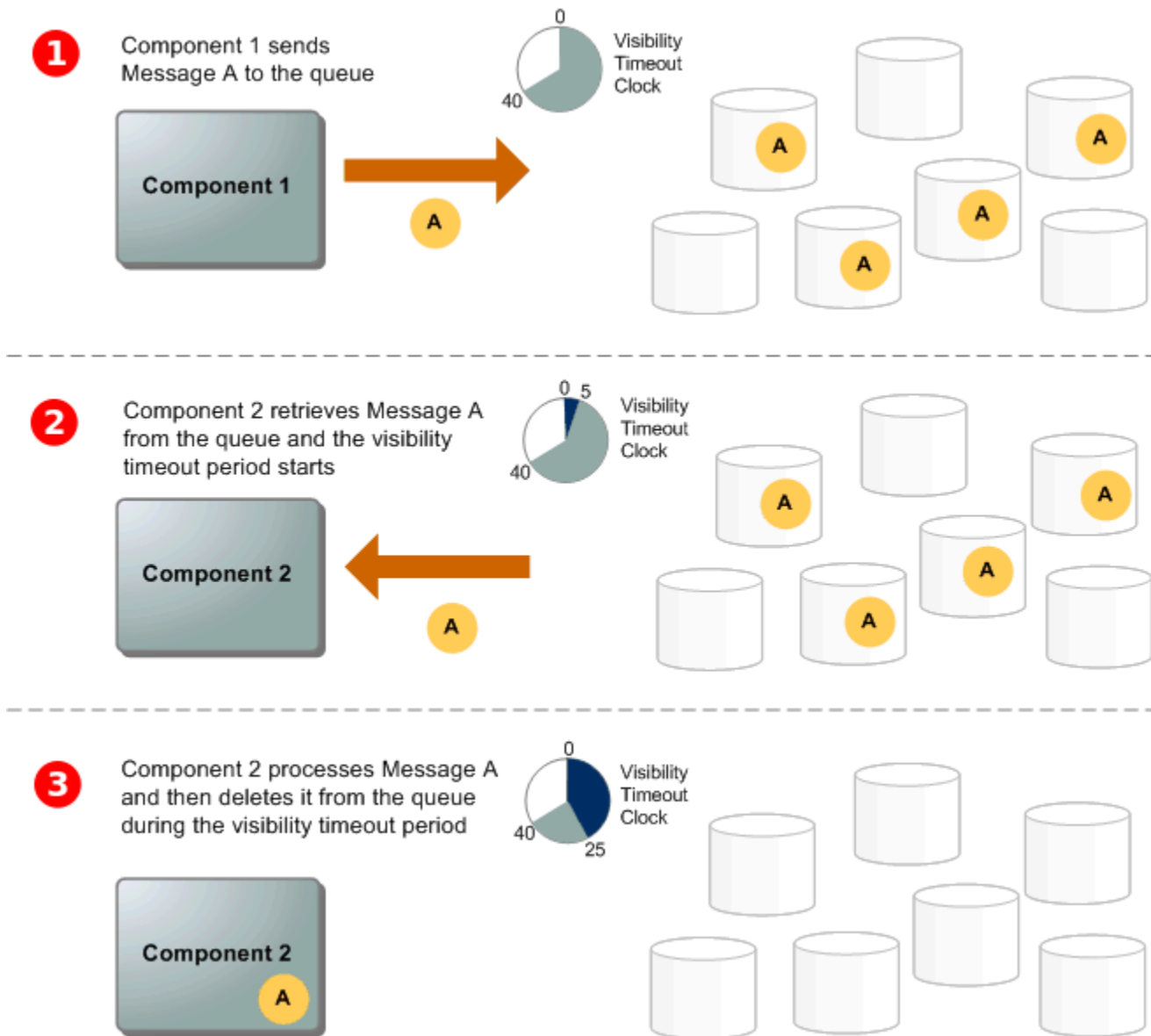
分布式消息传送系统有三个主要组成部分：分布式系统的组件、队列（分布在 Amazon SQS 服务器上）以及队列中的消息。

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（组件 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（组件 2）准备好处理消息时，就从队列使用消息，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（组件 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 easy-to-use](#) 和 Amazon [MQ](#) 提供高度可扩展的托管消息服务，每种服务都是为分布式系统中的特定角色设计的。以下是对这些服务之间差异的增强概述：

Amazon SQS 将分布式软件系统和组件分离并扩展为队列服务。它通常通过单个订阅者处理消息，非常适合订单和丢失防护至关重要的工作流程。为了实现更广泛的分发，将 Amazon SQS 与 Amazon SNS 集成，可以实现分[发消息传送](#)模式，有效地同时向多个订阅者推送消息。

Amazon SNS 允许发布者通过主题向多个订阅者发送消息，主题充当沟通渠道。订阅者使用支持的终端节点类型接收已发布的消息，例如 [Amazon SQS Amazon Data Firehose](#)、[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。为了防止订阅者离线时消息丢失，将 Amazon SNS 与 Amazon SQS 队列消息集成，可确保一致传送。

Amazon MQ [最适合希望从传统消息代理迁移的企业，这些代理支持 AMQP 和 MQTT 等标准消息传递协议，以及 Apache ActiveMQ 和 RabbitMQ。](#)它与需要稳定、可靠消息传递的传统系统兼容，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
发布/订阅消息收发	是	否	是
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。他们通常以其 pay-as-you-go 定价为大批量应用程序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 或高级消息队列协议 (AMQP)、MQTT 和简单文本消息协议 (STOMP) 等协议 (STOMP) 等协议兼容的现有消息代理中迁移应用程序。OpenWire

设置 Amazon SQS

在首次使用 Amazon SQS 之前，您必须完成以下步骤。

主题

- [步骤 1：创建 AWS 账户和 IAM 用户](#)
- [第 2 步：授权以编程方式访问](#)
- [第 3 步：为使用示例代码做好准备](#)
- [后续步骤](#)

步骤 1：创建 AWS 账户和 IAM 用户

要访问任何 AWS 服务，您首先需要创建一个[AWS 账户](#)可以使用 AWS 商品的 Amazon.com 账户。您可以使用 AWS 账户来查看您的活动和使用情况报告，以及管理身份验证和访问权限。

为避免使用您的 AWS 账户根用户执行 Amazon SQS 操作，最佳做法是为每位需要访问 Amazon SQS 的管理权限的人创建一个 IAM 用户。

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开 <https://portal.aws.amazon.com/billing/signup>。
2. 按照屏幕上的说明进行操作。

在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

当您注册时 AWS 账户，就会创建 AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务和资源。作为安全最佳实践，请为用户分配管理访问权限，并且只使用根用户来执行[需要根用户访问权限的任务](#)。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 <https://aws.amazon.com/> 并选择我的账户来查看当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就可以不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。[AWS Management Console](#)在下一页上，输入您的密码。

要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的[以根用户身份登录](#)。

2. 为您的根用户启用多重身份验证 (MFA)。

有关说明，请参阅 [IAM 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备 \(控制台\)](#)。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center

有关说明，请参阅《AWS IAM Identity Center 用户指南》中的[启用 AWS IAM Identity Center](#)。

2. 在 IAM Identity Center 中，为用户授予管理访问权限。

有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《[用户指南](#)》[IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限](#)。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

有关使用 IAM Identity Center 用户[登录的帮助](#)，请参阅[AWS 登录 用户指南中的登录 AWS 访问门户](#)。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

有关说明，请参阅《AWS IAM Identity Center 用户指南》中的[创建权限集](#)。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

有关说明，请参阅《AWS IAM Identity Center 用户指南》中的[添加组](#)。

第 2 步：授权以编程方式访问

要使用 Amazon SQS 操作（例如，使用 Java 或通过 AWS Command Line Interface），您需要一个访问密钥 ID 和一个私有访问密钥。

Note

访问密钥 ID 和私有访问密钥是特定的 AWS Identity and Access Management。不要将它们与其他 AWS 服务（例如 Amazon EC2 密钥对）的证书混淆。

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份 (在 IAM Identity Center 中管理的用户)	使用临时证书签署向 AWS CLI、AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 <ul style="list-style-type: none">有关的 AWS CLI，请参阅 《AWS Command Line Interface 用户指南》AWS IAM Identity Center 中的“配置 AWS CLI 要使用”。有关 AWS 软件开发工具包、工具和 AWS API，请参阅 《软件开发工具包和 AWS 工具参考指南》中的 IAM 身份中心身份验证。

哪个用户需要编程式访问权限？	目的	方式
IAM	使用临时证书签署向 AWS CLI、AWS 软件开发工具包或 AWS API 发出的编程请求。	按照 IAM 用户指南中的 将临时证书与 AWS 资源配合使用 中的说明进行操作。
IAM	(不推荐使用) 使用长期凭证签署向 AWS CLI、AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 <ul style="list-style-type: none"> 有关信息 AWS CLI，请参阅用户指南中的使用 IAM 用户证书进行身份验证。AWS Command Line Interface 有关 AWS SDK 和工具，请参阅 S AWS DK 和工具参考指南中的使用长期凭证进行身份验证。 有关 AWS API，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

第 3 步：为使用示例代码做好准备

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。您可以在[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 Management Console 管理 Amazon SQS 队列和消息了。

Amazon SQS 入门

在本节中，您将学习如何使用 Amazon SQS 控制台创建标准队列或 FIFO 队列。

主题

- [先决条件](#)
- [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](#)

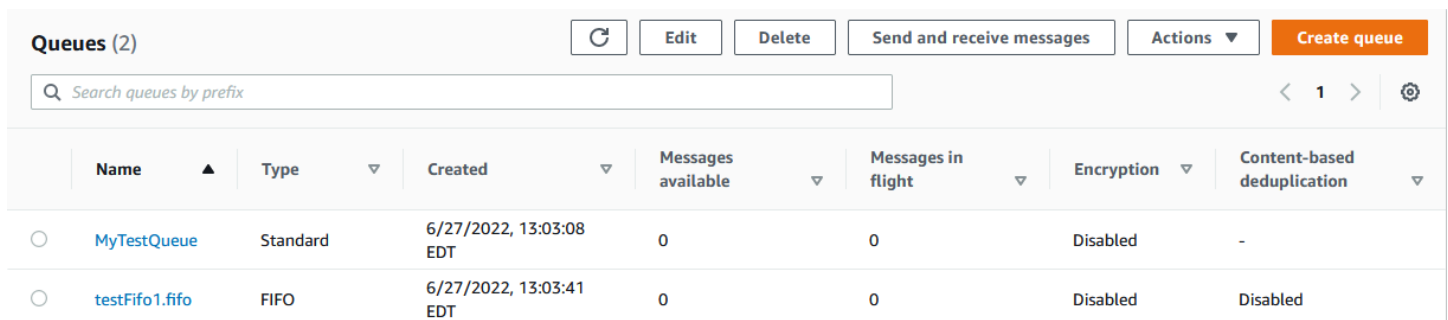
先决条件

在开始之前，请完成 [设置 Amazon SQS](#) 中的步骤。

了解 Amazon SQS 控制台

打开 Amazon SQS 控制台时，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](#)针对最大吞吐量和尽力而为消息排序进行了优化，与[先进先出 \(FIFO\)](#)队列区分开来，后者优先考虑需要严格消息排序的应用程序的消息顺序和唯一性。

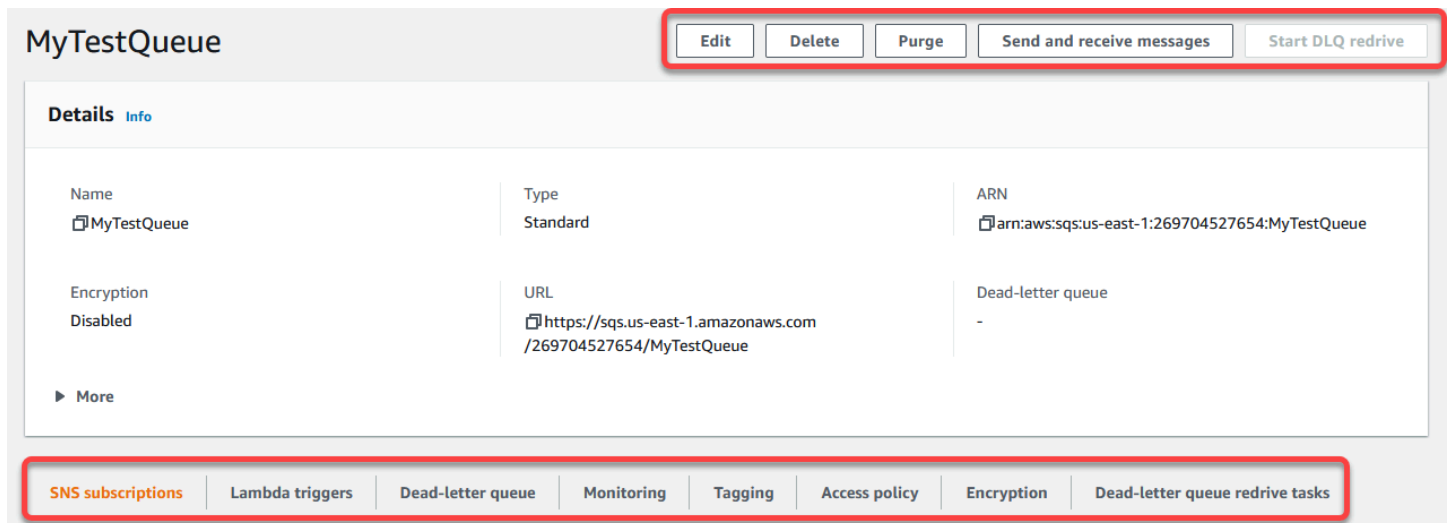


Queues (2)		Refresh	Edit	Delete	Send and receive messages	Actions	Create queue
Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication	
MyTestQueue	Standard	6/27/2022, 13:03:08 EDT	0	0	Disabled	-	
testFifo1.fifo	FIFO	6/27/2022, 13:03:41 EDT	0	0	Disabled	Disabled	

交互式元素和动作

在“队列”页面中，您可以通过多种选项来管理队列：

1. 快速操作-在每个队列名称旁边有一个下拉菜单，可快速访问常见操作，例如发送消息、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-单击队列名称将打开其详细信息页面，您可以在其中更深入地研究队列设置和配置。在这里，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

确保您处于正确状态 AWS 区域，以便有效地访问和管理队列。此外，可以考虑使用资源标签来组织和分类队列，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。

通过利用 Amazon SQS 控制台中提供的特性和功能，您可以高效地管理消息基础设施，优化队列性能，并确保应用程序的消息传输可靠。

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。使用下表中的信息选择适合您情况的队列。要详细了解 Amazon SQS 队列，请参阅[Amazon SQS 标准队列](#)和[Amazon SQS FIFO 队列](#)。

标准队列	FIFO 队列
无限制吞吐量 - 标准队列支持每个 API 操作 (SendMessage、ReceiveMessage 或 DeleteMessage) 每秒几乎无限次的 API 调用。	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

标准队列	FIFO 队列
<p>至少一次传递 – 消息至少传送一次，但偶尔会传送消息的多个副本。</p> <p>最大努力排序 – 消息偶尔可能按不同于其发送时的顺序传送。</p>	<p>DeleteMessageBatch) 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。要申请提高配额，请提交支持请求。在不使用批处理的情况下，FIFO 队列的每个 API 方法 (SendMessage 、 ReceiveMessage 或 DeleteMessage) 每秒最多支持 300 个 API 调用。</p> <p>仅处理一次 – 消息传递一次并在使用者处理并删除它之前保持可用。不会将重复项引入到队列中。</p> <p>先进先出传递 – 严格保持消息的发送和接收顺序。</p>
	
<p>当吞吐量很重要时，请使用此队列在应用程序之间发送数据，例如：</p> <ul style="list-style-type: none"> • 将实时用户请求从密集型后台工作中分离：让用户在调整媒体大小或对媒体编码时上传媒体。 • 将任务分配给多个工作程序节点：处理大量信用卡验证请求。 • 将消息分批以便进一步处理：计划要添加到数据库的多个条目。 	<p>当事件的顺序很重要时，请使用此队列在应用程序之间发送数据，例如：</p> <ul style="list-style-type: none"> • 确保按正确的顺序运行用户输入的命令。 • 通过按正确的顺序发送价格修改来显示正确的产品价格。 • 防止学员在注册账户之前参加课程。

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：

- 启动时创建回复队列：与其为每条消息创建回复队列，不如在启动时按制作者创建回复队列。使用关联 ID 消息属性有效地将回复映射到请求。
- 避免在制作者之间共享回复队列：确保每个制作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发送给其他制作者的响应消息。

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 [请求-响应消息收发模式 \(虚拟队列\)](#)。

创建 Amazon SQS 标准队列并发送消息

这是为 Amazon SQS 创建标准队列的方法。

使用 Amazon SQS 控制台创建队列

您可以使用 Amazon SQS 控制台创建[标准队列](#)。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 标准队列

1. 通过以下网址打开 Amazon SQS 控制台：<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 KB 到 256 KB。默认值为 256 KB。
 - 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。

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 Send message (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

这是为 Amazon SQS 创建 FIFO 队列的方法。

创建队列

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. 通过以下网址打开 Amazon SQS 控制台：<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 KB 到 256 KB。默认值为 256 KB。
- 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 中先进先出队列的高吞吐量](#) 和 [亚马逊 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：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。

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. 对于先进先出 (FIFO) 队列，输入消息组 ID。有关更多信息，请参阅 [亚马逊 SQS 中的 FIFO 队列传送逻辑](#)。
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 [亚马逊 SQS 中的 FIFO 队列传送逻辑](#)。
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 Send message (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，您已创建了队列，并学会如何发送、接收和删除消息，以及如何删除队列，您可能想要尝试以下操作：

- 要触发 Lambda 函数，请参阅[配置 Amazon SQS 队列以触发函数 AWS Lambda](#)。
- 了解如何[配置队列，包括 SSE 和其他特征](#)。
- 了解如何[发送带有属性的消息](#)。
- 了解如何[从 VPC 发送消息](#)。
- 要探索 Amazon SQS 的功能和架构，请参阅[Amazon SQS 队列类型](#)和[基本 Amazon SQS 架构](#)。
- 要查找有助于您充分利用 Amazon SQS 的准则和注意事项，请参阅[亚马逊 SQS 最佳实践](#)。
- [浏览其中一个软件开发工具包的 Amazon AWS SQS 示例，例如开发者指南。AWS SDK for Java 2.x](#)
- 要了解有关 Amazon SQS AWS CLI 命令的信息，请参阅[AWS CLI 命令参考](#)。
- 要了解 Amazon SQS 操作，请参阅[Amazon Simple Queue Service API 参考](#)。
- [了解如何以编程方式与 Amazon SQS 交互：阅读使用 API 并探索开发中心：AWS](#)
 - [Java](#)
 - [JavaScript](#)
 - [PHP](#)
 - [Python](#)
 - [Ruby](#)
 - [Windows 和 .NET](#)
- 在[对 Amazon SQS 中的问题进行故障排除](#)部分中了解如何监控成本和资源。
- 在[安全性](#)部分中了解如何保护和访问您的数据。
- 要了解有关亚马逊 SQS 工作流程的更多信息，请参阅[亚马逊 SQS 访问控制](#)流程工作流程部分。

管理 Amazon SQS 队列

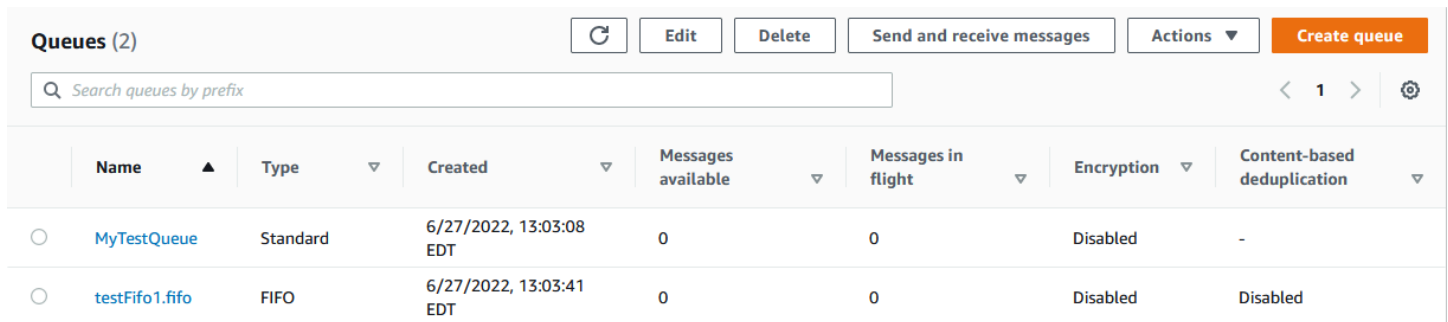
本节介绍如何使用 Amazon SQS 控制台管理队列和消息，从而帮助您进一步熟悉 Amazon SQS。

先决条件

在开始之前，请完成 [设置 Amazon SQS](#) 中的步骤。

了解 Amazon SQS 控制台

打开控制台时，从导航窗格中选择队列，以显示队列页面。队列页面提供有关活动区域中所有队列的信息。

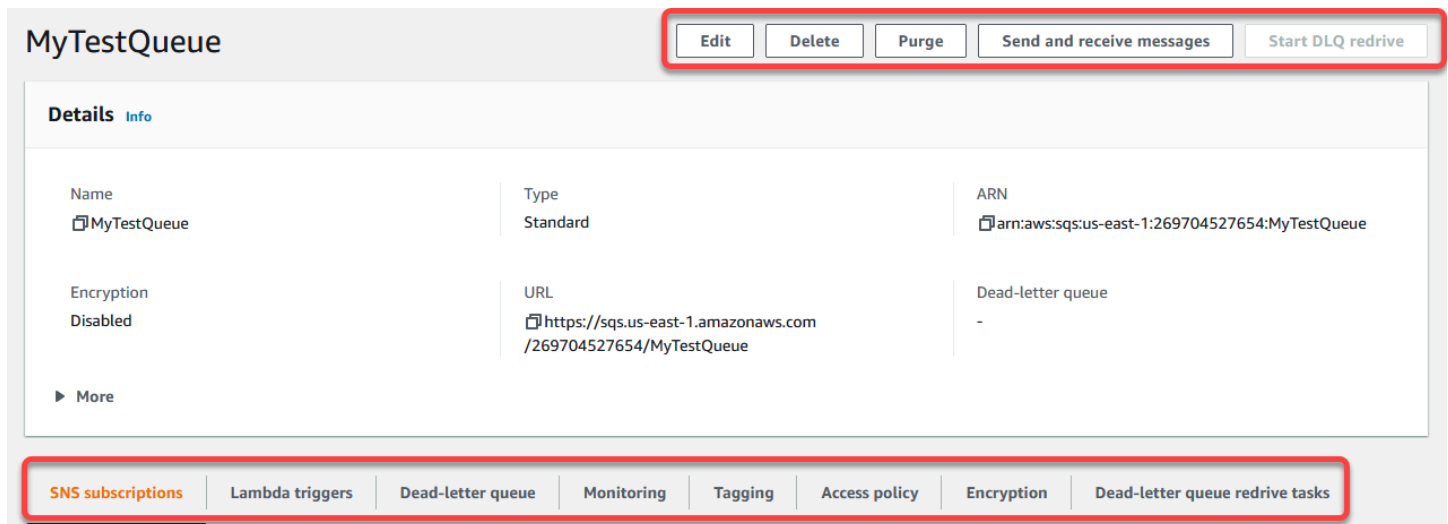


Queues (2)		Refresh	Edit	Delete	Send and receive messages	Actions	Create queue
Search queues by prefix						< 1 >	Settings
Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication	
MyTestQueue	Standard	6/27/2022, 13:03:08 EDT	0	0	Disabled	-	
testFifo1.fifo	FIFO	6/27/2022, 13:03:41 EDT	0	0	Disabled	Disabled	

每个队列的条目显示队列类型和有关队列的其他信息。类型列可帮助您一目了然地区分标准队列和先进先出 (FIFO) 队列。

在队列页面中，有两种方法可以对队列执行操作。您可以选择队列名称旁边的选项，然后选择要对队列执行的操作。

您也可以选择队列名称，这将打开队列的详细信息页面。详细信息页面包含与队列页面相同的操作。此外，您可以选择详细信息部分下方的选项卡之一，以查看其他配置详细信息和操作。



使用控制台编辑 Amazon SQS 队列

您可以使用 Amazon SQS 控制台编辑任何队列配置参数（队列类型除外）以及添加或删除队列特征。

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，您可以选择接收和删除这些消息。从队列请求消息时，不能指定单个消息。取而代之的是，您可以确定要检索的最大消息数，上限为 10。

Amazon SQS 作为分布式系统运行，当从包含少量消息的队列中检索消息时，偶尔会导致响应为空。如果发生这种情况，只需重新运行您的请求即可。要优化消息检索并最大限度地减少空响应，请考虑使用[长轮询](#)。长轮询会延迟响应，直到消息可用或民意调查超时，从而减少不必要的轮询成本并提高效率。

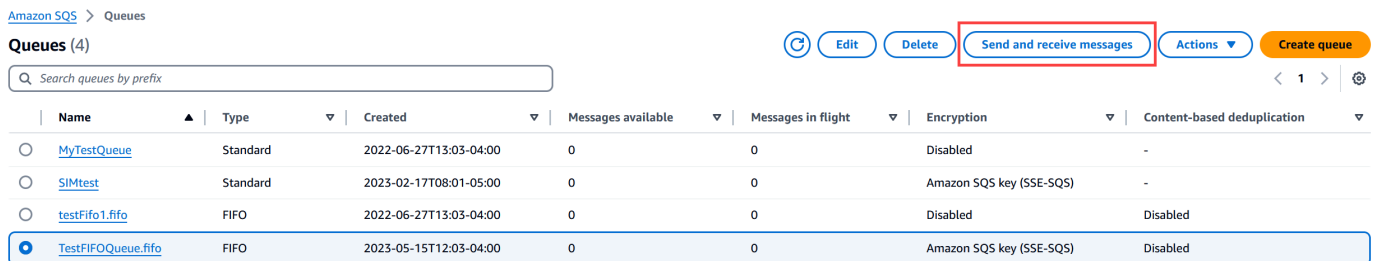
消息在检索后不会自动删除，因为 Amazon SQS 可确保您不会因为处理故障（例如应用程序问题或网络中断）而失去对消息的访问权限。要从队列中永久删除邮件，您必须在处理完邮件后明确发送删除请求以确认成功接收和处理。

通过 Amazon SQS 控制台检索消息时，它们会立即重新显示以供重新检索。此默认行为可确保消息不会在手动操作期间无意中丢失，但可能会导致重复处理。在自动化环境中，调整可见性超时设置以控制检索消息后对其他用户不可见的时间长度。此设置对于协调多个使用者的消息处理以及确保消息只处理一次至关重要。

有关接收和删除消息的更多详细操作，请参阅[Amazon SQS API 参考指南](#)。本指南提供有关 API 端点的全面信息，包括有效管理复杂消息处理场景的参数。

使用控制台接收和删除消息

1. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues（队列）。
3. 在“队列”页面上，选择一个队列，然后选择“发送和接收消息”。



4. 在“发送和接收消息”页面上，选择“轮询留言”。

Amazon SQS 开始轮询队列中的消息。接收消息部分右侧的进度条显示轮询的持续时间。

消息部分显示已接收消息的列表。对于每条消息，列表会显示消息 ID、发送日期、大小和接收计数。

5. 要删除消息，请选择要删除的消息，然后选择删除。
6. 在“删除消息”对话框中，选择“删除”。

确认亚马逊 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。

确认队列是否为空 (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](#)
 - AWS API: [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 Amazon CloudWatch 指标，请确保连续看到多个零数据点，然后再考虑该队列为空。有关 CloudWatch 指标的更多信息，请参阅[亚马逊 SQS 的可用 CloudWatch 指标](#)。

删除亚马逊 SQS 队列

如果您不再使用 Amazon SQS 队列，并且预计近期也不会使用该队列，建议将其删除。

Tip

如果要在删除队列之前验证队列是否为空，请参阅[确认亚马逊 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. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (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)

您可以使用以下命令之一删除队列：

- AWS CLI: [aws sqs delete-queue](#)
- AWS API: [DeleteQueue](#)

使用 Amazon SQS 控制台从队列中清除消息

如果不想删除 Amazon SQS 队列，但需要删除队列中的所有消息，可使用清除队列。消息删除过程最多需要 60 秒。我们建议您等待 60 秒，无论您的队列的大小如何。

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. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (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，每秒支持、和等 [SendMessage](#) 操作的 API 调用次数几乎不受限制。[ReceiveMessageDeleteMessage](#) 标准队列可确保 at-least-once 消息传送，但由于高度分散的架构，可能会传送一条消息的多个副本，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 (AZ) 中。这种冗余可确保任何一台计算机、网络或可用区故障都不会导致消息无法访问。

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 控制台创建队列](#)。有关特定于 Java 的示例，请参见 [Amazon SQS Java SDK 示例](#)

标准队列的用例

标准消息队列适用于各种场景，前提是您的应用程序可以处理可能多次到达或不按顺序到达的消息。示例包括：

- 将 @@ 实时用户请求与密集的后台工作分开 — 用户可以在系统在后台调整大小或编码媒体时上传媒体。
- 将任务分配给多个工作节点-例如，处理大量的信用卡验证请求。
- 批量处理消息以备将来处理-安排多个条目稍后添加到数据库中。

有关与标准队列相关的配额的信息，请参阅 [亚马逊 SQS 标准队列配额](#)。

有关使用标准队列的最佳实践，请参阅 [亚马逊 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。

如果发生这种情况，则不会删除服务器上不可用的邮件副本，并且在收到消息时您可能会再次获得该消息副本。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

此部分将介绍标准和 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。

亚马逊 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。每当您要对该队列执行操作时，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）。

以下是接收句柄的示例（跨三条线分解）。

```
MbZj6wDWli+JvwWJaBV+3dcjk2YW2vA3+STFF1jTM8tJJg6HRG6PYSasuWXPJB+Cw  
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq20KpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+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 队列最重要的功能是 FIFO \(先进先出\) 交付和精确一次处理](#)：

- 严格保留消息的发送和接收顺序，消息只传送一次，在消费者处理和删除消息之前一直不可用。
- 不会将重复项引入到队列中。

此外，FIFO 队列支持允许在单个队列中使用多个有序消息组的消息组。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 中先进先出队列的高吞吐量](#)。有关吞吐量配额的信息，请参阅 [the section called “邮件配额”](#)。

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[亚马逊 SQS 最佳实践](#)。

亚马逊 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

用于对已发送消息进行重复数据删除的令牌。如果成功发送了带有特定消息重复数据删除 ID 的消息，则所有使用相同消息重复数据删除 ID 发送的消息都将被成功接受，但不会在 5 分钟的重复数据删除间隔内传输。

Note

Amazon SQS 继续跟踪消息重复数据删除 ID，即使在消息被接收和删除之后也是如此。

消息组 ID

指定消息属于特定消息组的标签。属于同一消息组的消息按相对于消息组的严格顺序逐个处理（但是，属于不同消息组的消息可能会不按顺序处理）。

接收请求尝试 ID

用于对 ReceiveMessage 调用进行重复数据删除的令牌。

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。

亚马逊 SQS 中的 FIFO 队列传送逻辑

以下概念有助于您更好地了解发送和接收 FIFO 消息。

发送消息

如果将多条消息相继发送到 FIFO 队列，并且每条消息具有不同的消息重复数据删除 ID，则 Amazon SQS 将存储这些消息并确认传输。然后，可按传输每条消息的确切顺序接收和处理消息。

在 FIFO 队列中，消息基于消息组 ID 进行排序。如果多台主机（或同一主机上的不同线程）将具有相同消息组 ID 的消息发送到 FIFO 队列，则 Amazon SQS 将按消息到达以供处理的顺序存储消息。为了确保 Amazon SQS 保留发送和接收消息的顺序，每位创建者均应使用唯一的消息组 ID 来发送其所有消息。

FIFO 队列逻辑仅应用于每个消息组 ID。每个消息组 ID 表示 Amazon SQS 队列中不同的有序消息组。对于每一个消息组 ID，所有消息的发送和接收均严格遵循一定的顺序。但是，具有不同的消息组 ID 值的消息可能不会按顺序发送和接收。您必须将消息组 ID 与消息关联。如果您未提供消息组 ID，此操作将失败。如果需要一组有序的消息，请为要将消息发送到的 FIFO 队列提供相同的消息组 ID。

接收消息

您无法请求接收具有特定消息组 ID 的消息。

当接收来自具有多个消息组 ID 的 FIFO 队列的消息时，Amazon SQS 首先会尝试尽可能多地返回具有同一消息组 ID 的消息。这使其他使用者能处理具有不同消息组 ID 的消息。当您收到带有消息组 ID 的消息时，除非您删除该消息或该消息变为可见，否则不会再返回具有同一消息组 ID 的消息。

Note

可使用 `MaxNumberOfMessages` 操作的 [ReceiveMessage](#) 请求参数在单次调用中接收多达 10 条消息。这些消息将保留其 FIFO 顺序且可具有相同的消息组 ID。因此，如果具有相同消息组 ID 的消息少于 10 条，则您可能在同一批次的 10 条消息中，接收具有其他消息组 ID 的消息，但仍按 FIFO 顺序处理。

多次重试

FIFO 队列允许创建者或使用者尝试多次重试：

- 如果创建者检测到 `SendMessage` 操作失败，则可以根据需要使用相同的消息重复数据删除 ID 重试发送多次。假设创建者在重复数据删除间隔到期之前至少收到一次确认，则多次重试既不会影响消息的顺序，也不会引入重复消息。
- 如果使用者检测到 `ReceiveMessage` 操作失败，则可以根据需要使用相同的接收请求尝试 ID 重试多次。假设使用者在可见性超时到期之前至少收到一次确认，则多次重试不会影响消息的顺序。
- 当您收到带有消息组 ID 的消息时，除非您删除该消息或该消息变为可见，否则不会再返回具有同一消息组 ID 的消息。

在 Amazon SQS 中精确处理一次

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 中从标准队列移动到 FIFO 队列

如果您的现有应用程序使用标准队列，并且您想利用 FIFO 队列的排序或精确一次处理功能，则需要正确配置队列和应用程序。

重要注意事项：

- 创建 FIFO 队列：您无法将现有的标准队列转换为 FIFO 队列。您必须为应用程序创建新的 FIFO 队列，或者删除现有的标准队列并将其重新创建为 FIFO 队列。
- 延迟参数：FIFO 队列不支持每条消息的延迟，仅支持每队列的延迟。如果您的应用程序在每条消息上都设置了 DelaySeconds 参数，则必须将其修改为在整个队列 DelaySeconds 上设置。
- 消息组 ID：为每封已发送的 [消息提供消息组 ID](#)。此 ID 允许并行处理消息，同时保持消息的相应顺序。使用精细的业务维度作为消息组 ID，以便更好地扩展 FIFO 队列。您向其分发消息的消息组 ID 越多，可供使用的消息数量就越多。
- 高吞吐量模式：对 FIFO 队列使用推荐 [的高吞吐量模式](#) 以提高吞吐量。有关消息配额的更多信息，请参阅 [亚马逊 SQS 消息配额](#)。

移至 FIFO 队列的清单

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

- 修改您的应用程序以消除每条消息的延迟。
- 在整个队列上设置 DelaySeconds 参数。

2. 设置消息组 ID

- 根据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

- 如果您的应用程序可以发送具有相同消息正文的消息，请为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带有唯一消息正文的消息，请启用基于内容的重复数据删除。

4. 配置消费者

- 通常，消费者无需更改代码。

- 如果处理消息需要很长时间并且可见性超时设置得很高，请考虑为每个ReceiveMessage操作添加一个接收请求尝试 ID。这有助于在网络出现故障时重试接收尝试，并防止队列因接收尝试失败而暂停。

通过执行这些步骤，您可以确保您的应用程序在FIFO队列中正常运行，充分利用其排序和精确一次处理功能。有关更多详细信息，请参阅 [Amazon 简单队列服务 API 参考](#)。

亚马逊 SQS FIFO 队列和 Lambda 并发行为

通过将 FIFO (先进先出) 队列与 Lambda 配合使用，您可以确保对每个消息组中的消息进行有序处理。Lambda 函数不会同时为同一个消息组运行多个实例，从而保持顺序。但是，它可以向上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 Amazon SQS FIFO 队列的消息时与消息组 ID 相关的行为：

- 每个消息组单个实例：在任何时候，只有一个 Lambda 实例将处理来自特定消息组 ID 的消息。这样可以确保按顺序处理同一组中的消息，从而保持FIFO序列的完整性。
- 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 ID 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 ID 的消息，利用 Lambda 的并发功能并行处理多个组。

FIFO 队列消息分组

FIFO 队列可确保按发送消息的确切顺序进行处理。他们使用消息组 ID 对应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，并且每次仅处理每个组中的一条消息以保持此顺序。

与 FIFO 队列的 Lambda 并行性

创建队列后，您可以向该队列发送消息。

当您设置 Lambda 函数来处理来自亚马逊 SQS FIFO 队列的消息时，Lambda 会遵守先进先出队列提供的排序保证。以下几点描述了在使用消息组 ID 处理来自 Amazon SQS FIFO 队列的消息时，Lambda 函数在并行性和扩展方面的行为。

- 消息组内的并行性：一次只有一个 Lambda 实例处理特定消息组 ID 的消息。这样可以确保按顺序处理群组内的消息。

- 扩展和多个消息组：虽然 Lambda 可以向上扩展以同时处理消息，但这种扩展发生在不同的消息组中。如果您有多个消息组，Lambda 可以并行处理多个组，每个组由单独的 Lambda 实例处理。

有关更多信息，请参阅《操作员指南》中的 [Lambda 中的扩展和并发性](#) 或 [AWS 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）。

如果正在处理来自组 ID “A” 的消息，而另一封来自组 ID 为 “A” 的消息到达，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 ID “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 中先进先出队列的高吞吐量

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送的场景。

在严格的消息排序并不重要、传入消息量相对较低或零星的情况下，不需要 Amazon SQS 高吞吐量 FIFO 队列。例如，如果您有一个处理不频繁或非顺序消息的小型应用程序，那么与高吞吐量 FIFO 队列相关的复杂性和成本可能不合理。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，更易于管理。

为了增强高吞吐量 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 [亚马逊 SQS 消息配额](#) 和 [SQS FIFO 队列高吞吐量的分区和数据分布](#)。

主题

- [Amazon SQS FIFO 队列高吞吐量的用例](#)
- [SQS FIFO 队列高吞吐量的分区和数据分布](#)
- [在 Amazon SQS 中为 FIFO 队列启用高吞吐量](#)

Amazon SQS FIFO 队列高吞吐量的用例

以下用例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. **实时数据处理**：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，同时保留消息顺序以进行准确分析。
2. **电子商务订单处理**：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，即使在购物旺季也是如此。
3. **金融服务**：处理高频交易或交易数据的金融机构依靠高吞吐量 FIFO 队列以最小的延迟处理市场数据和交易，同时遵守严格的报文排序监管要求。
4. **媒体流**：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交付，确保用户获得流畅的播放体验，同时保持正确的内容交付顺序。

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，该队列在一个 AWS 区域内的多个可用区之间自动复制。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：

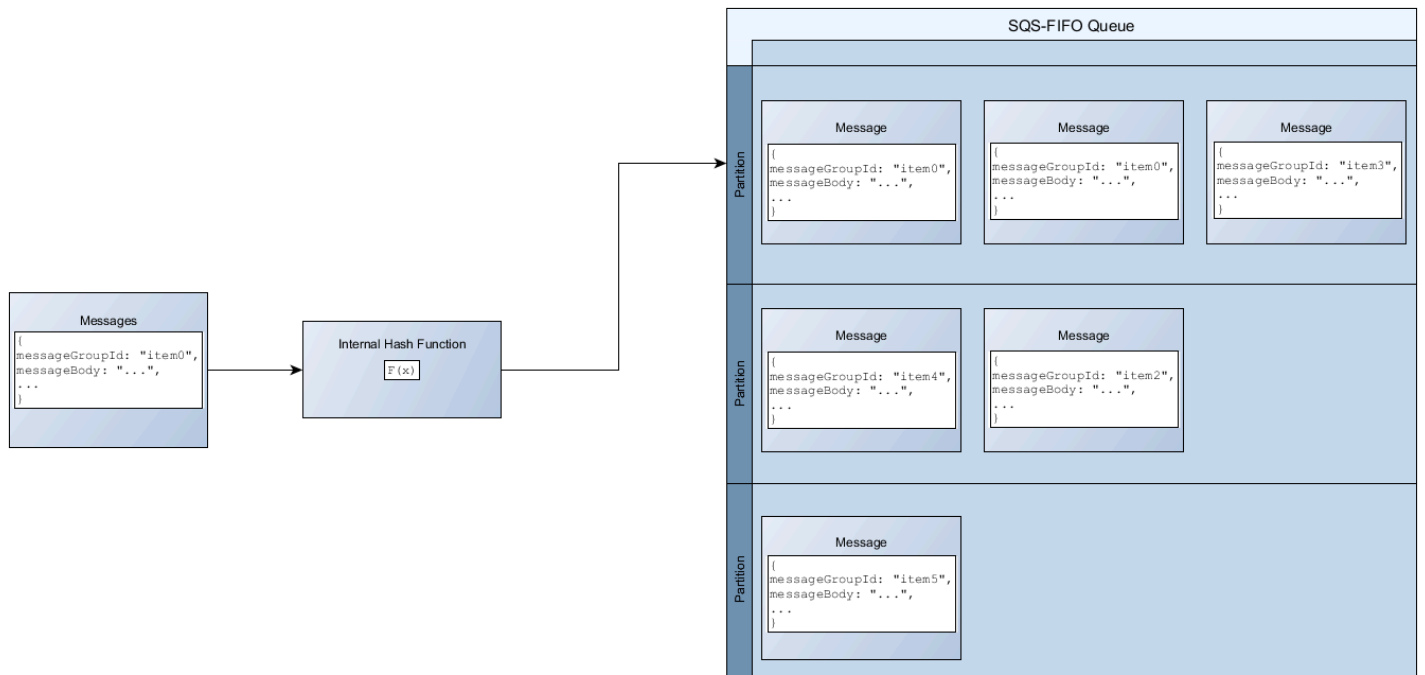
- 如果当前请求速率接近或超过现有分区所能支持的速率，则会分配其他分区，直到队列达到区域配额。有关配额的信息，请参阅[亚马逊 SQS 消息配额](#)。
- 如果当前分区的利用率较低，则分区的数量可能会减少。

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，可以在 FIFO 队列的分区中均匀分配项目，无论分区数量如何。AWS 建议您使用可能具有大量不同值的消息组 ID。

优化分区利用率

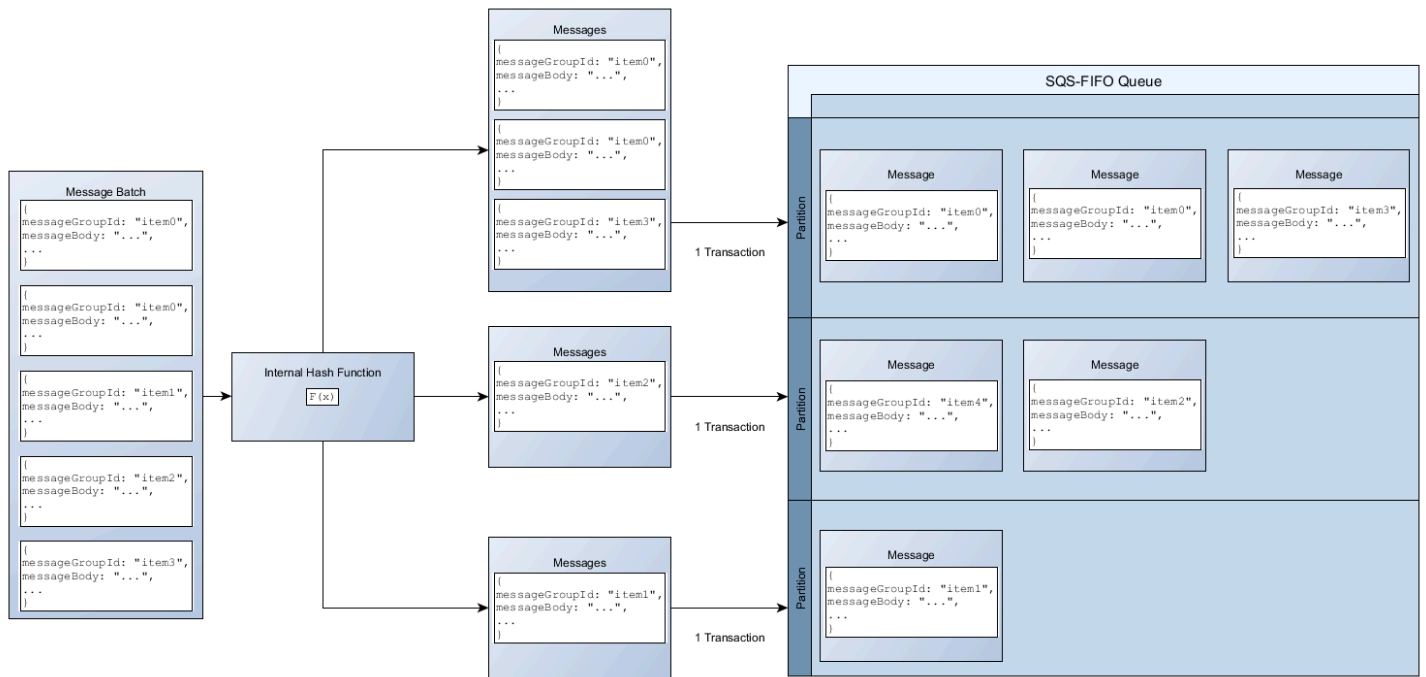
在支持区域，每个分区支持每秒最多 3000 条消息进行批处理，或者支持每秒最多 300 条消息用于发送、接收和删除操作。有关高吞吐量消息配额的更多信息，请参阅《Amazon Web Services 一般参考》中的 [Amazon SQS 服务限额](#)。

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 建议尽可能使用相同的消息组 ID 对消息进行批处理。

要优化 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接收句柄进行批处理。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。

在 Amazon SQS 中为 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[亚马逊 SQS 消息配额](#)中的消息吞吐量。

亚马逊 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。

主题

- [亚马逊 SQS 中先进先出队列的标识符](#)
- [Amazon SQS FIFO 队列的其他标识符](#)

亚马逊 SQS 中先进先出队列的标识符

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。每当您要对该队列执行操作时，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 AIFO 队列的队列 URL，该队列由 MyQueue 拥有 AWS 账号 123456789012 的用户拥有。

```
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）。

以下是接收句柄的示例（跨三条线分解）。

```
MbZj6wDW1i+JvwWJaBV+3dcjk2YW2vA3+STFF1jTM8tJJg6HRG6PYSasuWXPJB+Cw  
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq20KpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+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，请参阅在 [Amazon SQS 中精确处理一次](#) 和 [Amazon Simple Queue Service API 参考](#)。

消息重复数据删除 ID

用于对已发送消息进行重复数据删除的令牌。如果成功发送了带有特定消息重复数据删除 ID 的消息，则所有使用相同消息重复数据删除 ID 发送的消息都将被成功接受，但不会在 5 分钟的重复数据删除间隔内传输。

消息组 ID

指定消息属于特定消息组的标签。属于同一消息组的消息按相对于消息组的严格顺序逐个处理（但是，属于不同消息组的消息可能会不按顺序处理）。

序列号

Amazon SQS 为每条消息分配的大型非连续数字。

Amazon SQS 配额

本主题将列出 Amazon Simple Queue Service (Amazon SQS) 中的配额。

主题

- [亚马逊 SQS FIFO 队列配额](#)
- [亚马逊 SQS 标准队列配额](#)
- [亚马逊 SQS 消息配额](#)
- [亚马逊 SQS 政策配额](#)

亚马逊 SQS FIFO 队列配额

Amazon SQS 配额

下表列出了与 FIFO 队列相关的配额。

限额	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)	对于 FIFO 队列，最多可以有 20000 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)。如果您达到此配额，Amazon SQS 将不会返回任何错误消息。
队列名称	FIFO 队列名称必须以 <code>.fifo</code> 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。

限额	描述
队列标签	<p>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。</p> <p>标签 Key 是必需的，而标签 Value 是可选的。</p> <p>标签 Key 和标签 Value 区分大小写。</p> <p>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： _ . : / = + - @</p> <p>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）。</p> <p>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。</p> <p>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。</p> <p>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。</p>

亚马逊 SQS 标准队列配额

下表列出了与标准队列相关的配额。

限额	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。

限额	描述
每个队列的消息数 (传输中)	<p>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 <code>OverLimit</code> 错误消息。如果您使用长轮询 , 则 Amazon SQS 不返回任何错误消息。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。</p>
队列名称	<p>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 (_) 。</p> <div data-bbox="688 768 1507 982" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p> Note</p> <p>队列名称区分大小写 (例如 <code>Test-queue</code> 和 <code>test-queue</code> 是不同的队列) 。</p> </div>
队列标签	<p>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。</p> <p>标签 Key 是必需的 , 而标签 Value 是可选的。</p> <p>标签 Key 和标签 Value 区分大小写。</p> <p>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 : <code>_ . : / = + - @</code></p> <p>标签 Key 或 Value 不得包含预留前缀 <code>aws:</code> (您不能删除带此前缀的标签键或值) 。</p> <p>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 <code>null</code> 。</p> <p>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 <code>null</code> 。</p>

限额	描述
	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 。


亚马逊 SQS 消息配额

下表列出了与消息相关的配额。

限额	描述
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 AmazonSQS BufferedAsyncClient 。
消息内容	<p>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</p> <p>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。</p>
消息组 ID	<p>处理积压的消息，以避免积压大量具有相同消息组 ID 的消息。</p> <p>MessageGroupId 是 FIFO 队列所必需的。您不能将其用于标准队列。</p> <p>您必须将非空 MessageGroupId 与消息相关联。如果您未提供 MessageGroupId，操作将失败。</p>

限额	描述
	<p>MessageGroupId 的长度为 128 个字符。有效值：字母数字字符和标点符号 (!"#\$%&'()*+,-./:;<=>?@[\]^_`{ }~)。</p>
消息保留	<p>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)。</p>
消息吞吐量	<p>标准队列的每个 API 操作 (SendMessage、ReceiveMessage 或 DeleteMessage) 每秒支持接近无限的 API 调用。</p> <div data-bbox="667 674 1521 1171" style="background-color: #f0f0f0; padding: 10px;"> <p>FIFO 队列</p> <ul style="list-style-type: none"> • FIFO 队列支持每个 API 操作 (SendMessage、ReceiveMessage 和 DeleteMessage) 每秒 300 个事务的配额。 • 如果您使用批处理，则 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。 </div>

限额	描述
	<p data-bbox="686 226 992 260"><u>FIFO 队列的高吞吐量</u></p> <ul data-bbox="686 306 1503 1293" style="list-style-type: none"><li data-bbox="686 306 1503 533">• 如果不使用批处理 (<code>SendMessage</code> 、 <code>ReceiveMessage</code> 和 <code>DeleteMessage</code>) , 在美国东部 (弗吉尼亚州北部) 、 美国西部 (俄勒冈州) 和欧洲地区 (爱尔兰) 区域中 , FIFO 队列的高吞吐量每次 API 操作每秒处理多达 7 万个事务。<li data-bbox="686 554 1503 638">• 对于美国东部 (俄亥俄州) 和欧洲地区 (法兰克福) 区域 , 默认吞吐量为每个 API 操作每秒 1.8 万个事务。<li data-bbox="686 659 1503 785">• 对于亚太地区 (孟买) 、 亚太地区 (新加坡) 、 亚太地区 (悉尼) 和亚太地区 (东京) 区域 , 默认吞吐量为每个 API 操作每秒 9000 个事务。<li data-bbox="686 806 1503 890">• 对于欧洲地区 (伦敦) 和南美洲 (圣保罗) 区域 , 默认吞吐量为每个 API 操作每秒 4500 个事务。<li data-bbox="686 911 1503 995">• 为了最大限度地提高吞吐量 , 请在不使用批处理的情况下 , 增加用于已发送消息的消息组 ID 的数量。<li data-bbox="686 1016 1503 1293">• 您可以通过在美国东部 (弗吉尼亚州北部) 、 美国西部 (俄勒冈州) 和欧洲地区 (爱尔兰) 区域使用批处理 API (<code>SendMessageBatch</code> 和 <code>DeleteMessageBatch</code>) , 将吞吐量提高到每秒 70 万条消息。每秒 70 万条消息代表每秒 7 万个事务 , 每个事务带有包含 10 条消息的一个批处理。 <p data-bbox="716 1339 1503 1520">对于欧洲地区 (法兰克福) 和美国东部 (俄亥俄州) 区域 , 您可以通过使用批处理 API 每秒最多处理 18 万条消息。每秒 18 万条消息代表每秒 1.8 万个事务 , 每个事务带有包含 10 条消息的一个批处理。</p> <p data-bbox="716 1566 1503 1839">对于亚太地区 (孟买) 、 亚太地区 (新加坡) 、 亚太地区 (悉尼) 和亚太地区 (东京) 区域 , 您可以通过批处理每秒处理多达 90000 条消息。要在使用 <code>SendMessageBatch</code> 和 <code>DeleteMessageBatch</code> 时实现最大吞吐量 , 批处理请求中的所有消息都必须使用相同的消息组 ID。</p>

限额	描述
	<ul style="list-style-type: none"> 对于欧洲地区（伦敦）和南美洲（圣保罗）区域，您可以通过批处理每秒最多处理 45000 条消息。要在使用 <code>SendMessageBatch</code> 和 <code>DeleteMessageBatch</code> 时实现最大吞吐量，批处理请求中的所有消息都必须使用相同的消息组 ID。 在所有其他 AWS 区域，每个 API 操作的最大吞吐量为每秒 2,400 条（不含批处理）或 24,000 条（使用批处理）消息。 要申请超出区域限制的上限，请联系 AWS Support 中心 寻求帮助。 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。
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	<p>最小消息大小为 1 字节（1 个字符）。最大消息大小为 262144 字节（256 KiB）。</p> <p>要发送大于 256 KiB 的消息，您可以使用适用于 Java 的 亚马逊 SQS 扩展客户端库 和适用于 Python 的 亚马逊 SQS 扩展客户端库。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。</p> <div data-bbox="688 1318 1507 1493" style="border: 1px solid #add8e6; border-radius: 15px; padding: 10px; margin-top: 10px;"> <p> Note 此扩展库仅适用于同步客户端。</p> </div>
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。
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 亚马逊 SQS 政策配额 。

亚马逊 SQS 策略配额

下表列出了与策略相关的配额。

名称	最高
字节	8192
条件	10
委托人	50
语句	20
每个语句的操作	7

Amazon SQS 特征和功能

本主题提供了 Amazon SQS 中常用的功能，用于管理消息队列、优化性能、确保可靠的消息传送以及高效处理消息。

主题

- [在 Amazon SQS 中使用死信队列](#)
- [亚马逊 SQS 的消息元数据](#)
- [处理 Amazon SQS 消息所需的资源](#)
- [亚马逊 SQS 列表队列分页](#)
- [Amazon SQS 成本分配标签](#)
- [Amazon SQS 短轮询和长轮询](#)
- [Amazon SQS 可见性超时](#)
- [Amazon SQS 延迟队列](#)
- [Amazon SQS 临时队列](#)
- [Amazon SQS 消息计时器](#)
- [通过亚马逊 SQS EventBridge 控制台访问 Amazon Pipes](#)
- [使用扩展客户端库和亚马逊简单存储服务管理大型 Amazon SQS 消息](#)

在 Amazon SQS 中使用死信队列

Amazon SQS 支持死信队列 (DLQ)，对于未成功处理的消息，源队列可以将死信队列作为目标。DLQ 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处理失败的原因。为了获得最佳性能，最佳做法是将源队列和 DLQ 保持在相同 AWS 账户和区域内。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。
- 分析移至死信队列的消息的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](#)。

在将其配置为死信队列之前，必须先创建一个新队列。有关使用 Amazon SQS 控制台配置死信队列的信息，请参阅[了解如何使用 Amazon SQS 控制台配置死信队列](#)。有关死信队列的帮助，例如如何为移至死信队列的所有消息配置警报，请参阅[使用 Amazon 为死信队列创建警报 CloudWatch](#)

对死信队列使用策略

使用重新驱动策略指定 `maxReceiveCount` `maxReceiveCount`是指在消息移至死信队列之前，消费者可以从源队列接收消息的次数。例如，如果设置 `maxReceiveCount` 为较低的值（例如 1），则一次未能收到消息就会导致消息移至死信队列。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。

重新驱动允许策略指定哪些源队列可以访问死信队列。您可以选择是允许所有源队列、允许特定的源队列还是拒绝所有源队列使用死信队列。默认设置允许所有源队列使用死信队列。如果您选择使用 `byQueue` 选项允许特定队列，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。该 `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。

了解如何使用 Amazon SQS 控制台配置死信队列

死信队列是源队列可以针对未成功处理的消息的队列。有关更多信息，请参阅 [在 Amazon SQS 中使用死信队列](#)。

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。有关创建用作死信队列的队列的说明，请参阅 [使用 Amazon SQS 控制台创建队列](#)。

FIFO 队列的死信队列也必须是 FIFO 队列。同样，标准队列的死信队列也必须是标准队列。

[创建或编辑](#)队列时，您可以配置死信队列。

为现有队列配置死信队列（控制台）

1. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (队列)。

3. 选择队列并选择“编辑”。
4. 滚动到死信队列部分，然后选择启用。
5. 选择要与该源队列关联的现有死信队列的 Amazon 资源名称 (ARN)。
6. 要配置消息在发送到死信队列之前可接收的次数，请将最大接收次数设置为一个介于 1 和 1000 之间的值。
7. 配置完死信队列后，选择保存。

保存队列后，控制台会显示队列的详细信息页面。在详细信息页面，死信队列选项卡会显示死信队列的最大接收次数和死信队列 ARN。

了解如何在 Amazon 中配置死信队列重启 SQS

您可以使用死信队列重新驱动将未使用的消息移出现有的死信队列。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，如果死信队列是队FIFO列，则重新驱动目标队列也必须是FIFO队列。此外，您可以配置重新驱动速度以设置 Amazon SQS 传输消息的速率。

Note

当消息从队列移动到FIFO队列时 FIFODLQ，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，如果生产者正在向源FIFO队列发送消息，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来自生产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带有新消息的新消息，messageID并enqueueTime分配给重新驱动的消息。

主题

- [使用 Amazon 为现有标准队列配置死信队列重新驱动 SQS API](#)

- [使用 Amazon 控制台为现有标准队列配置死信队列重新驱动 SQS](#)
- [为死信队列重新驱动配置队列权限](#)

使用 Amazon 为现有标准队列配置死信队列重新驱动 SQS API

您可以使用SendMessageBatch、ReceiveMessage和操作配置死信队列重新驱动 : DeleteMessageBatchAPI

API行动	描述
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。只有当当前状态为时，才能取消邮件移动RUNNING。

使用 Amazon 控制台为现有标准队列配置死信队列重新驱动 SQS

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 Queues (队列)。
3. 选择已配置为[死信队列](#)的队列名称。
4. 选择“开始DLQ重新驾驶”。
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 个活跃的重装任务。

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API的方法
开始重新驱动消息	<ul style="list-style-type: none"> • 添加死信队列的 <code>sqs:StartMessageMoveTask</code>、<code>sqs:ReceiveMessage</code>、<code>sqs>DeleteMessage</code> 和 <code>sqs:GetQueueAttributes</code>。如果死信队列已加密（也称为 SSE 队列），则还需要 <code>kms:Decrypt</code> 获得用于加密消息的任何KMS密钥的权限。 • 添加目标队列的 <code>sqs:SendMessage</code>。如果目标队列已加密，则 <code>kms:GenerateDataKey</code> 也是必需的。
取消进行中的消息重新驱动	<ul style="list-style-type: none"> • 添加死信队列的 <code>sqs:CancelMessageMoveTask</code>、<code>sqs:ReceiveMessage</code>、<code>sqs>DeleteMessage</code> 和 <code>sqs:GetQueueAttributes</code>。如果死信队列已加密（也称为 SSE 队列），则 <code>kms:Decrypt</code> 也是必需的。
显示消息移动状态	<ul style="list-style-type: none"> • 添加死信队列的 <code>sqs:ListMessageMoveTasks</code> 和 <code>sqs:GetQueueAttributes</code>。

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重新驱动的最高权限：

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 <https://console.aws.amazon.com/iam/>。
2. 在导航窗格中，选择策略。
3. 创建具有以下权限的 [策略](#) 并将其附加到您的登录 IAM [用户](#) 或 [角色](#)：
 - sqs:StartMessageMoveTask
 - sqs:CancelMessageMoveTask
 - sqs:ListMessageMoveTasks
 - sqs:ListDeadLetterSourceQueues
 - sqs:ReceiveMessage
 - sqs>DeleteMessage
 - sqs:GetQueueAttributes
 - 死信队列中的 ResourceARN (例如，“arn: aws: sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>”)。
 - sqs:SendMessage
 - 目标队ResourceARN列的 (例如，“arn: aws: sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>”)。
 - kms:Decrypt - 允许解密操作。
 - kms:GenerateDataKey
 - 用于加密原始源队列中消息的任何KMS加密密钥的 (例如，“arn: aws: kms: Resource ARN<region>:<accountId>:key/<keyId_used to encrypt the message body>”)。
 - 用于重新驱动目标队列ARN的KMS加密密钥的资源 (例如，“arn: aws: kms:<region>:<accountId>:key/<keyId_used for the destination queue>”)。

访问策略应类似下文：

```
{
  "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:<DLQ_region>:<DLQ_accountId>:<DLQ_name>"
  },
  {
    "Effect": "Allow",
    "Action": "sqs:SendMessage",
    "Resource":
      "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>"
  },
  {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:<region>:<accountId>:key/<keyId_used to encrypt the
message body>"
  },
  {
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey"
    ],
    "Resource": "arn:aws:kms:<region>:<accountId>:key/<keyId_used for the
destination queue>"
  }
]
}

```

使用非加密队列对（带有死信队列的源队列）配置权限

使用以下步骤为未加密的标准死信队列配置最低权限：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. 登录 AWS Management Console 并打开 IAM 控制台，网址为 <https://console.aws.amazon.com/iam/>。
2. 在导航窗格中，选择策略。
3. 创建具有以下权限的 [策略](#) 并将其附加到您的登录 IAM [用户](#) 或 [角色](#)：
 - sqs:StartMessageMoveTask
 - sqs:CancelMessageMoveTask
 - sqs:ListMessageMoveTasks
 - sqs:ListDeadLetterSourceQueues
 - sqs:ReceiveMessage
 - sqs>DeleteMessage
 - sqs:GetQueueAttributes
 - 死信队列中的 ResourceARN (例如，“arn: aws: sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>”).
 - sqs:SendMessage
 - 目标队 ResourceARN 列的 (例如，“arn: aws: sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>”).

访问策略应类似下文：

```
{
  "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:<DLQ_region>:<DLQ_accountId>:<DLQ_name>"
    }
  ]
}
```

```

"Effect": "Allow",
  "Action": "sqs:SendMessage",
  "Resource":
    "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>"
  }
]
}

```

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 重命名死信队列重启的事件](#)
- [更新了死信队列重新驱动的权限](#)

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重新驱动 DLQ 中的消息。

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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:DeleteMessage",
        "sqs:GetQueueAttributes",
        "sqs:StartMessageMoveTask",
        "sqs:ListMessageMoveTasks",
        "sqs:CancelMessageMoveTask"
      ]
    }
  ]
}
```

```

    ],
    "Resource": "arn:aws:sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>"
  },
  {
    "Effect": "Allow",
    "Action": "sqs:SendMessage",
    "Resource":
      "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>"
  }
]
}

```

托管策略

以下托管策略包含所需的更新权限：

- AmazonSQS FullAccess — 包括以下死信队列重新驱动任务：开始、取消和列出。
- AmazonSQS ReadOnly 访问权限 — 提供只读访问权限，并包括列出死信队列重新驱动任务。

Step 1

Add permissions

Step 2

Review

Add permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

Add user to group

Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

Copy permissions

Copy all group memberships, attached managed policies, inline policies, and any existing permissions boundaries from an existing user.

Attach policies directly

Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1051)


[Create policy](#)



2 matches



1



<input type="checkbox"/>	Policy name	Type	Attached entities
<input checked="" type="checkbox"/>	AmazonSQSFullAccess	AWS managed	0
<input type="checkbox"/>	AmazonSQSReadOnly...	AWS managed	0

Cancel

Next

使用 sqs* 通配符的权限策略

```
{
  "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 (CMP)，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

您可以使用 Amazon CloudWatch 和指标为移至死信队列的任何消息配置警报。[ApproximateNumberOfMessagesVisible](#) 有关更多信息，请参阅 [为 Amazon SQS 指标创建 CloudWatch 警报](#)。收到消息已发送到死信队列的警报后，您可以使用 [轮询](#) 来查看消息以接收消息。

亚马逊 SQS 的消息元数据

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。您可以使用消息系统属性来存储其他 AWS 服务（如 AWS X-Ray）的元数据。

主题

- [Amazon SQS 消息属性](#)
- [Amazon SQS 消息系统属性](#)

Amazon SQS 消息属性

Amazon SQS 支持您使用消息属性，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）。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：尽管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，但[您可以使用消息系统属性](#)来存储 AWS 其他服务的元数据，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 256 KB 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 AWS SDK for Java，则可以跳过本节。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 或其中一个不支持 Amazon AWS SQS 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

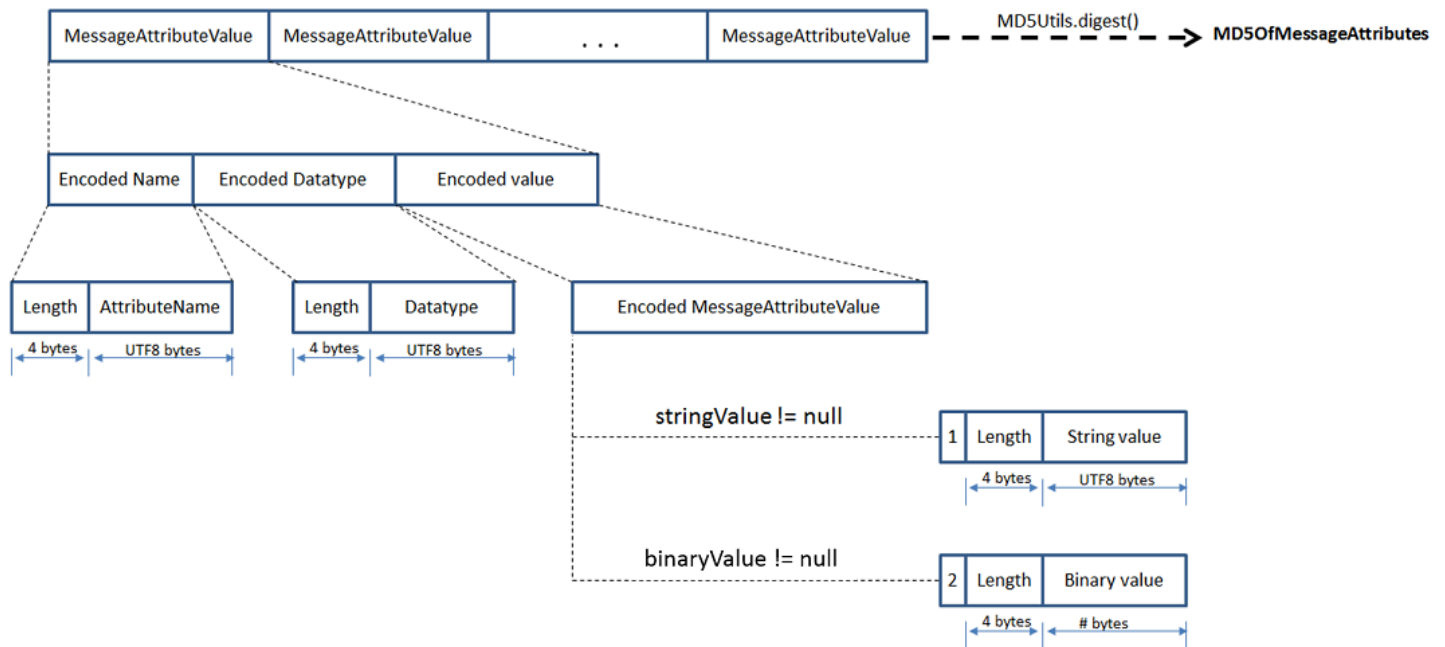
请始终在 MD5 消息摘要计算中包含自定义数据类型后缀。

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. 对名称编码：长度（4 字节）和名称的 UTF-8 字节。
2. 对数据类型编码：长度（4 字节）和数据类型的 UTF-8 字节。
3. 对值（1 个字节）的传输类型（String 或 Binary）编码。

i 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，对属性值编码：长度（4 字节）和值的 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`。其类型必须为 `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

为帮助您估计处理已排队消息所需的资源，Amazon SQS 可以确定队列中的已延迟、可见以及不可见消息的大致数量。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，因此某些指标的结果是近似值。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：

任务	属性名称
获取可从队列检索的大致消息数。	<code>ApproximateNumberOfMessagesVisible</code>
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。	<code>ApproximateNumberOfMessagesDelayed</code>

任务	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。	ApproximateNumberOfMessages NotVisible

亚马逊 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 [listQueues](#) 或 [listDeadLetterQueues](#) 请求中设置参数 `MaxResults`，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 `listQueues` 的下一个请求中将 `NextToken` 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[亚马逊 SQS 标准队列配额](#)。

Amazon SQS 短轮询和长轮询

Amazon SQS 提供了用于从队列接收消息的短轮询和长轮询选项。在这两个轮询选项之间进行选择时，请考虑您的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。
- 长轮询-在所有服务器上[ReceiveMessage](#)查询消息，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才会发送空响应。此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。

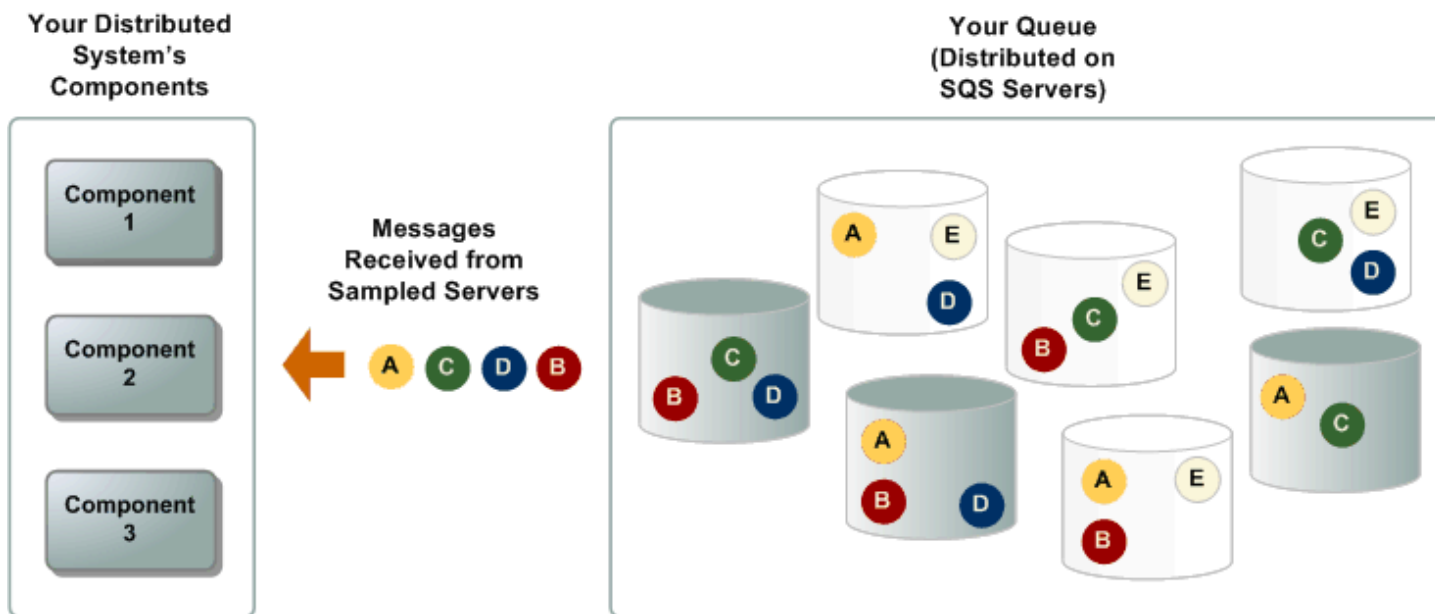
主题

- [通过短轮询来使用消息](#)
- [通过长轮询来使用消息](#)
- [长轮询和短轮询之间的区别](#)

通过短轮询来使用消息

当您使用短轮询使用来自队列（FIFO 或标准队列）的消息时，Amazon SQS 会对其服务器的子集（基于加权随机分布）进行采样，并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。长轮询通过消除空响应的数量（[ReceiveMessage](#) 请求时没有消息可用时）并消除假的空响应（消息可用但未包含在响应中），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 中设置长时间投票](#)。

长轮询具有以下好处：

- 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，即使队列中仍包含消息，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[确认亚马逊 SQS 队列为空](#)。

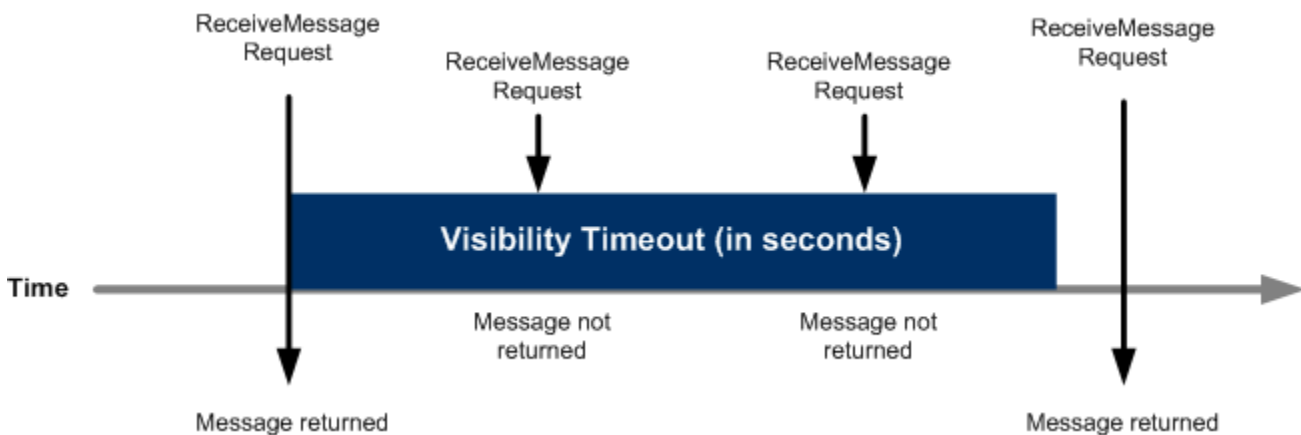
长轮询和短轮询之间的区别

如果以下列两种方式之一将 [WaitTimeSeconds](#) 请求的 [ReceiveMessage](#) 参数设置为 0，则会出现短轮询：

- 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

当使用者接收并处理来自队列的消息时，消息将保留在该队列中。Amazon SQS 不会自动删除消息。由于 Amazon SQS 是分布式系统，因此，无法保证使用者实际收到消息（例如，由于连接问题或由于使用者应用程序问题）。因此，使用者在接收和处理消息后必须从队列中删除该消息。



收到消息后，消息将立即保留在队列中。为防止其他使用者再次处理消息，Amazon SQS 设置了可见性超时，这是 Amazon SQS 防止所有使用者接收和处理消息的时间段。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。有关使用控制台配置队列可见性超时的信息，请参阅[使用 Amazon SQS 控制台配置队列参数](#)。

Note

对于标准队列，可见性超时无法保证不会接收消息两次。有关更多信息，请参阅 [亚马逊 SQS at-least-once 配送](#)。

FIFO 队列允许创建者或使用者尝试多次重试：

- 如果创建者检测到 SendMessage 操作失败，则可以根据需要使用相同的消息重复数据删除 ID 重试发送多次。假设创建者在重复数据删除间隔到期之前至少收到一次确认，则多次重试既不会影响消息的顺序，也不会引入重复消息。
- 如果使用者检测到 ReceiveMessage 操作失败，则可以根据需要使用相同的接收请求尝试 ID 重试多次。假设使用者在可见性超时到期之前至少收到一次确认，则多次重试不会影响消息的顺序。
- 当您收到带有消息组 ID 的消息时，除非您删除该消息或该消息变为可见，否则不会再返回具有同一消息组 ID 的消息。

主题

- [传输中消息](#)
- [设置可见性超时](#)
- [更改消息的可见性超时](#)
- [终止消息的可见性超时](#)

传输中消息

Amazon SQS 消息有三种基本状态：

1. 创建者发送到队列。
2. 使用者从队列中接收。
3. 从队列中删除。

在消息由创建者发送到队列但使用者尚未从队列中接收（即在状态 1 和 2 之间）之后，会将消息视为存储。存储的消息数量没有配额。在消息由使用者从队列中接收但尚未从队列中删除（即在状态 2 和 3 之间）之后，会将消息视为传输中。传输中消息数量没有配额。

⚠ Important

适用于传输中消息的配额与无限数量的存储消息无关。

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](#)。

对于 FIFO 队列，最多可以有 20000 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）。如果您达到此配额，Amazon SQS 将不会返回任何错误消息。

⚠ Important

使用 FIFO 队列时，如果在可见性超时窗口之外收到请求，则 `DeleteMessage` 操作将失败。如果可见性超时为 0 秒，则必须在消息发送的同一毫秒内将其删除，否则该消息将被视为放弃。这会在 `MaxNumberOfMessages` 参数大于 1 的情况下，导致 Amazon SQS 在对 `ReceiveMessage` 操作的同一响应中包含重复消息。有关更多详情，请参阅 [Amazon SQS FIFO API 的工作原理](#)。

设置可见性超时

可见性超时从 Amazon SQS 返回消息时开始。在这段时间里，使用者可以处理和删除消息。但是，如果使用者在删除消息之前失败，并且您的系统没有在可见性超时结束之前对该消息调用 [DeleteMessage](#) 操作，则其他使用者将可以看到该消息并且再次接收该消息。如果某条消息只能被接收一次，则您的使用者应在可见性超时期间内删除该消息。

每个 Amazon SQS 队列都具有 30 秒的默认可见性超时设置。您可以为整个队列更改此设置。通常，您应将可见性超时设置为您的应用程序处理消息并从队列中删除消息所需的最长时间。此外，在接收消息时，您还可以为返回的消息设置特殊的可见性超时，而无需更改整个队列的超时。有关更多信息，请参阅[在 Amazon SQS 中及时处理消息](#)部分中的最佳实践。

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`。

更改消息的可见性超时

当您收到来自队列的消息并开始处理该消息时，队列的可见性超时可能不够（例如，您可能需要处理和删除消息）。可通过使用 [ChangeMessageVisibility](#) 操作指定新的超时值来缩短或延长消息的可见性。

例如，如果队列的默认超时值为 60 秒，在您接收消息后 15 秒时，您发送了 `ChangeMessageVisibility` 调用并将 `VisibilityTimeout` 设为 10 秒，则这 10 秒从您进行 `ChangeMessageVisibility` 调用时开始计时。因此，自您最初更改可见性超时时 10 秒（共 25 秒）起，任何更改可见性超时或删除此消息的尝试都可能导致错误。

Note

新的超时期限自您调用 `ChangeMessageVisibility` 操作起生效。此外，新的超时期限仅应用于消息的特定接收。`ChangeMessageVisibility` 不会影响消息的后续接收或后续队列的超时。

终止消息的可见性超时

收到来自某个队列的某条消息时，您可能会发现，您实际上不想处理并删除该消息。Amazon SQS 允许您终止特定消息的可见性超时。这将使该消息对系统中的其他组件立即可见并且可用于处理。

要在调用 `ReceiveMessage` 后终止消息的可见性超时，请调用 [ChangeMessageVisibility](#) 并将 `VisibilityTimeout` 设置为 0 秒。

Amazon SQS 延迟队列

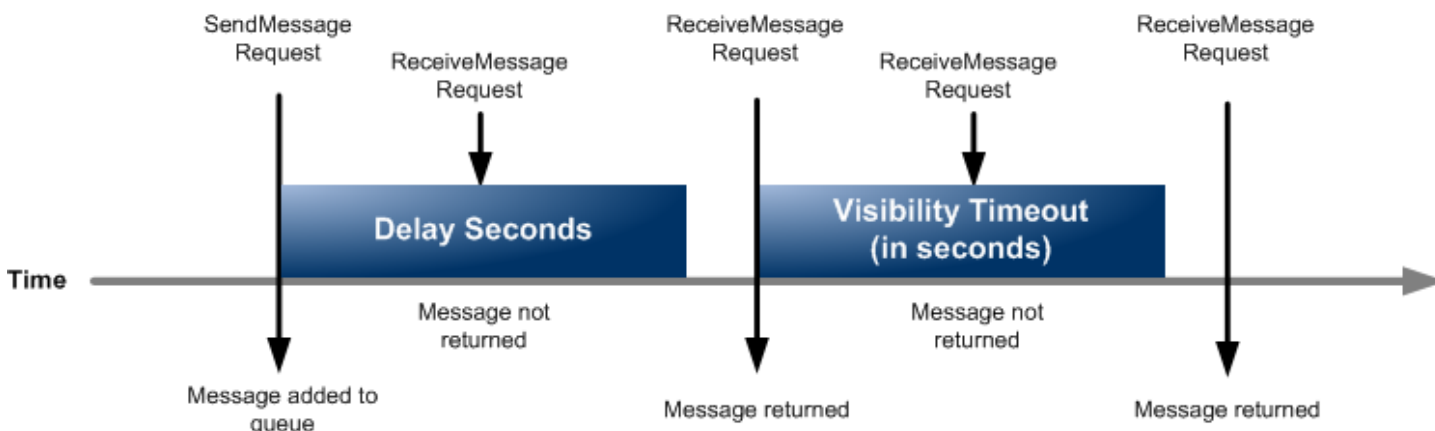
延迟队列允许您将向使用者发送新消息推迟几秒钟，例如，当您的使用者应用程序需要更多时间来处理消息时。如果您创建延迟队列，则发送到该队列的任何消息在延迟期间对使用者都不可见。队列的默认（最小）延迟为 0 秒。最大值为 15 分钟。有关使用控制台配置延迟队列的信息，请参阅[使用 Amazon SQS 控制台配置队列参数](#)。

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。



要为单条消息而不是整个队列设置延迟（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。

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。
- 它们与静态（普通）Amazon SQS 队列 API 兼容。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。

主题

- [虚拟队列](#)
- [请求-响应消息收发模式（虚拟队列）](#)
- [示例场景：处理登录请求](#)
 - [在客户端](#)
 - [在服务器端](#)
- [清除队列](#)

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[避免在虚拟队列中重复使用相同的消息组 ID](#)。

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[亚马逊 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。

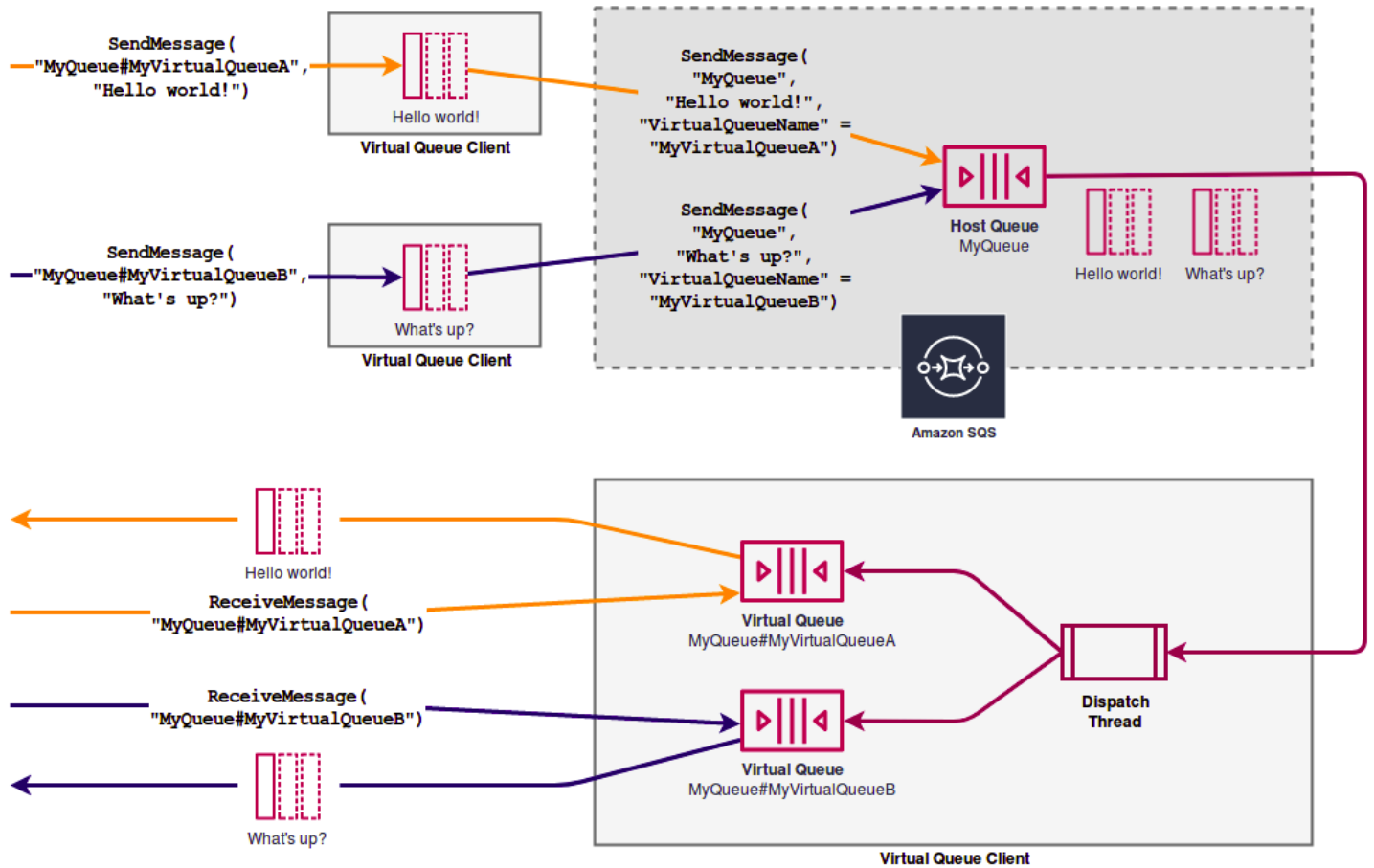
当使用者在虚拟队列 URL 上调用 `ReceiveMessage` API 操作时，临时队列客户端在本地阻止调用，直至后台线程将消息发送到虚拟队列中。（此过程类似于 [Buffered Asynchronous Client](#) 中的消息预取操作：单个 API 操作可以将消息提供到最多 10 个虚拟队列。）删除虚拟队列操作可以在不调用 Amazon SQS 本身的情况下删除任何客户端资源。

`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。

请求-响应消息收发模式（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，请参阅实现请求-响应系统。

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.sendMessage(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

消息计时器允许您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，则该消息在队列中的前 45 秒内对使用者不可见。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[发送消息](#)。

Note

FIFO 队列不支持单个消息的计时器。

要在整个队列（而不是单个消息）上设置延迟时段，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。

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

Amazon Pipes EventBridge 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
 - 上次修改管道的日期
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。

使用扩展客户端库和亚马逊简单存储服务管理大型 Amazon SQS 消息

您可以使用适用于 Java 的 Amazon SQS 扩展客户端库和适用于 Python 的亚马逊 SQS 扩展客户端库来发送大型消息。这对于消耗从 256 KB 到 2 GB 的大型消息负载特别有用。这两个库都将消息负载保存到亚马逊简单存储服务存储桶中，并将存储的 Amazon S3 对象的引用发送到 Amazon SQS 队列。

Note

Amazon SQS 扩展客户端库与标准队列和 FIFO 队列兼容。

主题

- [使用 Java 和 Amazon S3 管理大型亚马逊 SQS 消息](#)
- [使用 Python 和 Amazon S3 管理大型亚马逊 SQS 消息](#)

使用 Java 和 Amazon S3 管理大型亚马逊 SQS 消息

您可以使用[适用于 Java 的 Amazon SQS 扩展客户端库](#)和[亚马逊](#)简单存储服务 (Amazon S3) Service 来管理大型亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 消息。这对于消耗从 256 KB 到 2 GB 的大型消息负载特别有用。该库将消息负载保存到 Amazon S3 存储桶，并将包含存储的 Amazon S3 对象引用的消息发送到 Amazon SQS 队列。

您可以使用适用于 Java 的 Amazon SQS 扩展型客户端库执行以下操作：

- 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 存储桶中检索消息对象
- 从 Amazon S3 存储桶中删除消息对象

先决条件

以下示例使用 AWS Java 开发工具包。要安装和设置 SDK，请参阅《AWS SDK for Java 开发者指南》中的[“设置 AWS 适用于 Java 的 SDK”](#)。

在运行示例代码之前，请配置您的 AWS 证书。有关更多信息，请参阅《AWS SDK for Java 开发人员指南》中的设置开发 AWS [凭证和区域](#)。

[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 AWS SDK for Java 来管理 Amazon SQS 消息。使用亚马逊 SQS 控制台 AWS CLI、亚马逊 SQS HTTP API 或任何其他软件开发工具包都无法执行此操作。AWS

AWS 适用于 Java 的 SDK 2.x 示例：使用亚马逊 S3 管理大型亚马逊 SQS 消息

以下AWS 适用于 Java 的 SDK 2.x 示例创建了一个名称随机的 Amazon S3 存储桶，并添加了一条生命周期规则，用于在 14 天后永久删除对象。它还会创建一个名为 MyQueue 的队列并向该队列发送一条存储在 S3 存储桶中且大小超过 256 KB 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。

```
/*
 * 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 com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.*;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;

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

public class SQSExtendedClientExample {

    // Create an Amazon S3 bucket with a random name.
    private final static String S3_BUCKET_NAME = 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 AmazonS3 s3 = AmazonS3ClientBuilder.defaultClient();

    /*
     * 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 BucketLifecycleConfiguration.Rule expirationRule =
        new BucketLifecycleConfiguration.Rule();
    expirationRule.withExpirationInDays(14).withStatus("Enabled");
    final BucketLifecycleConfiguration lifecycleConfig =
        new BucketLifecycleConfiguration().withRules(expirationRule);

    // Create the bucket and allow message objects to be stored in the bucket.
    s3.createBucket(S3_BUCKET_NAME);
    s3.setBucketLifecycleConfiguration(S3_BUCKET_NAME, lifecycleConfig);
    System.out.println("Bucket created and configured.");

    /*
     * Set the Amazon SQS extended client configuration with large payload
     * support enabled.
     */
    final ExtendedClientConfiguration extendedClientConfig =
        new ExtendedClientConfiguration()
            .withLargePayloadSupportEnabled(s3, S3_BUCKET_NAME);

    final AmazonSQS sqsExtended =
        new AmazonSQSExtendedClient(AmazonSQSClientBuilder
            .defaultClient(), extendedClientConfig);

    /*
     * Create a long string of characters for the message object which will
     * be stored in the bucket.
     */
    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().toString();
final CreateQueueRequest createQueueRequest =
    new CreateQueueRequest(QueueName);
final String myQueueUrl = sqsExtended
    .createQueue(createQueueRequest).getQueueUrl();
System.out.println("Queue created.");

// Send the message.
final SendMessageRequest myMessageRequest =
    new SendMessageRequest(myQueueUrl, myLongString);
sqsExtended.sendMessage(myMessageRequest);
System.out.println("Sent the message.");

// Receive the message.
final ReceiveMessageRequest receiveMessageRequest =
    new ReceiveMessageRequest(myQueueUrl);
List<Message> messages = sqsExtended
    .receiveMessage(receiveMessageRequest).getMessages();

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

// Delete the message, the queue, and the bucket.
final String messageReceiptHandle = messages.get(0).getReceiptHandle();
sqsExtended.deleteMessage(new DeleteMessageRequest(myQueueUrl,
    messageReceiptHandle));
System.out.println("Deleted the message.");

sqsExtended.deleteQueue(new DeleteQueueRequest(myQueueUrl));
System.out.println("Deleted the queue.");

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

```
private static void deleteBucketAndAllContents(AmazonS3 client) {

    ObjectListing objectListing = client.listObjects(S3_BUCKET_NAME);

    while (true) {
        for (S3ObjectSummary objectSummary : objectListing
            .getObjectSummaries()) {
            client.deleteObject(S3_BUCKET_NAME, objectSummary.getKey());
        }

        if (objectListing.isTruncated()) {
            objectListing = client.listNextBatchOfObjects(objectListing);
        } else {
            break;
        }
    }

    final VersionListing list = client.listVersions(
        new ListVersionsRequest().withBucketName(S3_BUCKET_NAME));

    for (S3VersionSummary s : list.getVersionSummaries()) {
        client.deleteVersion(S3_BUCKET_NAME, s.getKey(), s.getVersionId());
    }

    client.deleteBucket(S3_BUCKET_NAME);
}
}
```

AWS 适用于 Java 的 SDK 2.x 示例：使用亚马逊 S3 管理大型亚马逊 SQS 消息

以下AWS 适用于 Java 的 SDK 2.x 示例创建了一个名称随机的 Amazon S3 存储桶，并添加了一条生命周期规则，用于在 14 天后永久删除对象。它还会创建一个名为 MyQueue 的队列并向该队列发送一条存储在 S3 存储桶中且大小超过 256 KB 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。

```
/*
 * 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 S3_BUCKET_NAME = 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(S3_BUCKET_NAME).build());

        s3.putBucketLifecycleConfiguration(PutBucketLifecycleConfigurationRequest.builder()
            .bucket(S3_BUCKET_NAME)
            .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, S3_BUCKET_NAME);
    }
}
```

```
    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(S3_BUCKET_NAME).build());

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

client.deleteObject(DeleteObjectRequest.builder().bucket(S3_BUCKET_NAME).key(object.key()).build());

        ListObjectVersionsResponse listVersionsResponse =
client.listObjectVersions(ListObjectVersionsRequest.builder().bucket(S3_BUCKET_NAME).build());

        listVersionsResponse.versions().forEach(version -> {

client.deleteObject(DeleteObjectRequest.builder().bucket(S3_BUCKET_NAME).key(version.key()).build());

        });

client.deleteBucket(DeleteBucketRequest.builder().bucket(S3_BUCKET_NAME).build());
    }
}
```

您可以[使用 Apache Maven](#) 为您的 Java 项目配置和构建 Amazon SQS 扩展客户端，也可以自己构建 SDK。从 SDK 中指定您在应用程序中使用的各个模块。

```
<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 管理大型亚马逊 SQS 消息

您可以使用[适用于 Python 的亚马逊简单队列服务扩展客户端库](#)和亚马逊简单存储服务来管理大型 Amazon SQS 消息。这对于消耗从 256 KB 到 2 GB 的大型消息负载特别有用。该库将消息负载保存到 Amazon S3 存储桶，并将包含存储的 Amazon S3 对象引用的消息发送到亚马逊 SQS 队列。

您可以使用适用于 Python 的扩展客户端库来执行以下操作：

- 指定有效负载是始终存储在 Amazon S3 中，还是仅在有效负载大小超过 256 KB 时存储在 S3 中
- 发送一条引用 Amazon S3 存储桶中存储的单个消息对象的消息
- 从 Amazon S3 存储桶中检索相应的负载对象
- 从 Amazon S3 存储桶中删除相应的负载对象

先决条件

以下是使用适用于 Python 的 Amazon SQS 扩展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。有关证书的信息，请参阅[凭证](#)。
- AWS 软件开发工具包：本页上的示例使用 AWS Python SDK Boto3。要安装和设置软件开发工具包，请参阅适用于 [Python 的开发AWS 工具包开发者指南中的适用于 Python 的开发AWS 工具包文档](#)
- Python 3.x (或更高版本) 和。pip
- [适用于 Python 的亚马逊 SQS 扩展客户端库，可从 PyPI 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。AWS 使用 CLI、亚马逊 SQS 控制台、亚马逊 SQS HTTP API 或任何其他软件开发工具包都无法执行此操作。AWS

配置消息存储

Amazon SQS 扩展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 True，则所有消息都存储在 Amazon S3 中。如果是 False，则小于 256 KB 的消息将不会序列化到 s3 存储桶。默认值为 False。
- `use_legacy_attribute`：如果是 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 的扩展客户端库管理大型 Amazon SQS 消息

以下示例使用随机名称创建一个 Amazon S3 存储桶。然后，它会创建一个名为的 Amazon SQS 队列，`MyQueue`并向该队列发送一条存储在 S3 存储桶中且超过 256 KB 的消息。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 = "S3_BUCKET_NAME"
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
```


使用亚马逊 SQS 控制台配置亚马逊 SQS 队列

使用 Amazon SQS 控制台配置和管理 Amazon Simple Queue Service (Amazon SQS) 队列和特征。您还可以使用控制台配置诸如服务器端加密之类的功能，将死信队列与您的队列关联起来，或者设置触发器来调用函数。AWS Lambda

主题

- [适用于 Amazon SQS 的基于属性的访问控制](#)
- [使用 Amazon SQS 控制台配置队列参数](#)
- [在 Amazon SQS 中配置访问策略](#)
- [使用 SQS 托管的加密密钥为队列配置服务器端加密](#)
- [使用 Amazon SQS 控制台为队列配置服务器端加密](#)
- [使用 Amazon SQS 控制台为队列配置成本分配标签](#)
- [使用亚马逊 SQS 控制台为队列订阅亚马逊 SNS 主题](#)
- [配置 Amazon SQS 队列以触发函数 AWS Lambda](#)
- [使用亚马逊自动将 AWS 服务发送到亚马逊 SQS 的通知 EventBridge](#)
- [使用 Amazon SQS 发送带有属性的消息](#)

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。要了解有关 ABAC 条件密钥的更多信息，请参阅《服务授权参考》中的[Amazon SQS 条件密钥](#)。

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

- [在 Amazon SQS 中为访问控制添加标签](#)
- [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](#)

在 Amazon SQS 中为访问控制添加标签

以下是如何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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessForProd",
      "Effect": "Deny",
      "Action": "sqs:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "prod"
        }
      }
    }
  ]
}
```

创建 IAM 用户和 Amazon SQS 队列

以下示例说明了如何使用 AWS Management Console 和创建 ABAC 策略来控制对 Amazon SQS 的访问。AWS CloudFormation

使用 AWS Management Console

创建 IAM 用户

1. 登录 AWS Management Console 并打开 IAM 控制台，[网址为 https://console.aws.amazon.com/iam/](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。

```
{
  "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"
            }
        }
    ]
}

```

4. 选择查看策略。
5. 选择 创建策略。

使用 AWS CloudFormation

使用以下示例 AWS 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[下载和配置](#) CLI。

```
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匹配。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，可以配置以下参数：

- 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，必须使用其中一个 AWS SDK。

- 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 可以根据消息正文自动创建重复数据删除 ID。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 中先进先出队列的高吞吐量](#) 和 [亚马逊 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. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。

2. 在导航窗格中，选择 Queues (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。范围为 1 KB 至 256 KB。默认值为 256 KB。
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 中先进先出队列的高吞吐量](#) 和 [亚马逊 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，可以配置其访问策略。

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。

为现有队列配置访问策略 (控制台)

1. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (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 中的身份和访问管理 SQS](#)。
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. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (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 中的静态加密 SQS](#)。

您分配给队列的 KMS 密钥必须具有密钥策略，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. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (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](#)。

在队列的详细信息页面上，标记选项卡显示队列的标签。

[创建](#)或[编辑](#)队列时，您可以为队列配置标签。

为现有队列配置标签 (控制台)

1. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。

2. 在导航窗格中，选择 Queues (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。

使用亚马逊 SQS 控制台为队列订阅亚马逊 SNS 主题

您可以为一个或多个 Amazon SQS 队列订阅 Amazon Simple Notification Service (Amazon SNS) 主题。当您发布消息到主题时，Amazon SNS 将向每个已订阅队列发送此消息。Amazon SNS 管理订阅和所有必要的权限。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 ?](#)

为 Amazon SQS 队列订阅 SNS 主题时，Amazon SNS 会使用 HTTPS 将消息转发到 Amazon SQS。有关将 Amazon SNS 用于加密的 Amazon SQS 队列的信息，请参阅[AWS 服务配置 KMS 权限](#)。

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 SNS 主题 (控制台)

1. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (队列)。
3. 从队列列表中，选择要订阅 SNS 主题的队列。
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，选择您队列的 SNS 主题。

如果该菜单中未列出 SNS 主题，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. 要验证订阅结果，请发布到主题，然后查看主题发送到队列的消息。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](#)。

如果您的 Amazon SQS 队列和 SNS 主题不同 AWS 账户，则主题所有者必须先确认订阅。有关更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的 [确认订阅](#)。

有关订阅跨区域 SNS 主题的信息，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 队列或 AWS Lambda 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 AWS Lambda 函数来处理 Amazon SQS 队列中的消息。Lambda 轮询队列并同步调用您的 Lambda 函数，其中有包含队列消息的事件。为使您的函数有时间处理每批记录，请将源队列的可见性超时至少设置为您在函数上[配置的超时的六倍](#)。这一额外的时间有利于 Lambda 在您的函数处理之前的批处理期间遇到限流时进行重试。

您可以指定另一个队列作为您的 Lambda 函数无法处理的消息的死信队列。

一个 Lambda 函数可以处理来自多个队列的项（为每个队列使用一个 Lambda 事件源）。您可以使用具有多个 Lambda 函数的同一队列。

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。

注意以下限制：

- 您的队列和 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

有关实现 Lambda 函数的信息，请参阅开发者指南中的[AWS Lambda 与 Amazon SQS 一起使用](#)。

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 [在 Amazon 中管理访问权限概述 SQS](#)。

配置队列以触发 Lambda 函数（控制台）

1. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (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 允许您实现 AWS 服务自动化，并对系统事件（例如应用程序可用性问题或资源更改）做出响应。来自 AWS 服务的事件 EventBridge 几乎是实时传送到的。您可以编写简单规则来指示您关注的事件，并指示要在事件匹配规则时执行的自动化操作。

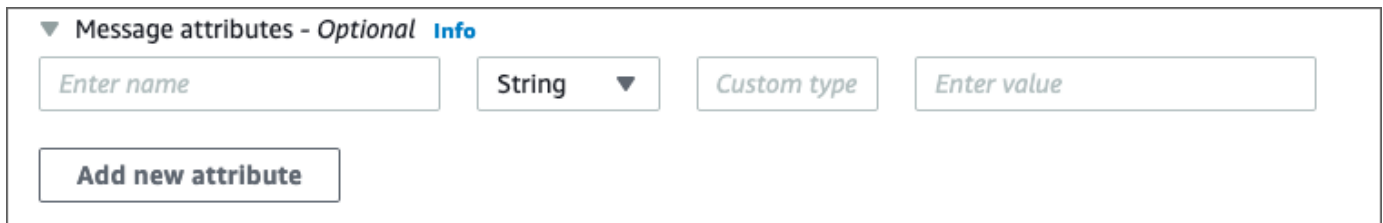
EventBridge 允许您设置各种目标，例如 Amazon SQS 标准队列和 FIFO 队列，这些目标接收 JSON 格式的事件。有关更多信息，请参阅《[亚马逊 EventBridge 用户指南](#)》中的[亚马逊 EventBridge 目标](#)。

使用 Amazon SQS 发送带有属性的消息

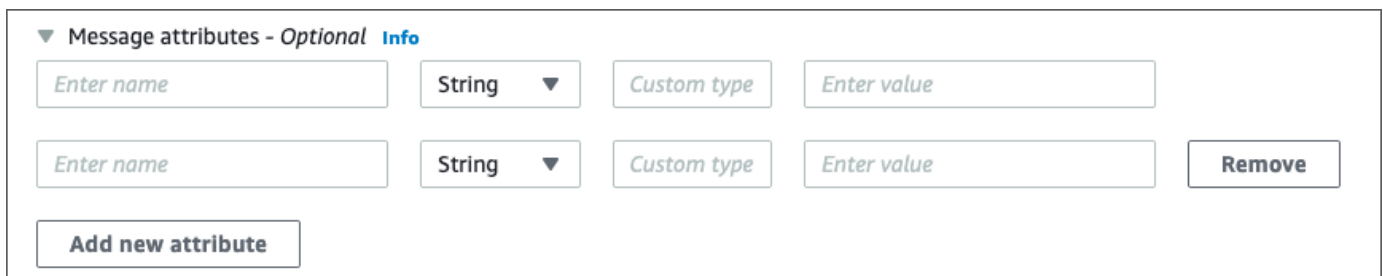
对于标准队列和 FIFO 队列，您可以在消息中包括结构化元数据（如时间戳、地理空间数据、签名和标识符）。有关更多信息，请参阅 [Amazon SQS 消息属性](#)。

使用 Amazon SQS 控制台向队列发送带有属性的消息

1. 通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (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。

亚马逊 SQS 最佳实践

Amazon SQS 管理和处理消息队列，促进应用程序不同部分之间的顺畅通信，并确保消息处理的可扩展性和可靠性。本指南涵盖了 Amazon SQS 的基本操作最佳实践，包括使用长轮询来最大限度地减少空响应、实施死信队列来处理错误，以及优化队列权限以确保安全。目标是提供一种具体且可操作的方法来操作和排查 Amazon SQS 消息。

主题

- [亚马逊 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](#)
- [亚马逊 SQS 消息处理和定时](#)

亚马逊 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以确保消息可靠性的技术。

主题

- [处理 Amazon SQS 中的请求错误](#)
- [在 Amazon SQS 中捕获有问题的消息](#)
- [在 Amazon SQS 中设置死信队列保留](#)

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，请在未收到来自 Amazon SQS 的消息、超时或错误消息后重试 [ReceiveMessage](#) 操作之前允许暂停（例如 200 毫秒）。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）。

在 Amazon SQS 中捕获有问题的消息

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

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

本主题提供了确保 Amazon SQS 中一致的消息处理的最佳实践，包括使用消息重复数据删除 ID 和消息组 ID 属性来防止重复消息并有效地管理消息组。

主题

- [避免 Amazon SQS 中的消息处理不一致](#)
- [使用 Amazon SQS 消息重复数据删除 ID](#)
- [使用 Amazon SQS 消息组 ID](#)
- [使用 Amazon SQS 接收请求尝试 ID](#)

避免 Amazon SQS 中的消息处理不一致

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。

使用 Amazon SQS 消息重复数据删除 ID

消息重复数据删除 ID 是用于对已发送消息进行重复数据删除的令牌。如果成功发送了带有特定消息重复数据删除 ID 的消息，则所有使用相同消息重复数据删除 ID 发送的消息都将被成功接受，但不会在 5 分钟的重复数据删除间隔内传输。

Note

Amazon SQS 继续跟踪消息重复数据删除 ID，即使在消息被接收和删除之后也是如此。

在 Amazon SQS 中提供消息重复数据删除 ID

创建者应在下列情况下为发送的每条消息提供消息重复数据删除 ID 值：

- Amazon SQS 必须视为唯一项的包含相同消息正文的已发送消息。
- Amazon SQS 必须视为唯一项且内容相同、但消息属性不同的已发送消息。
- Amazon SQS 必须视为重复项且内容不同（例如，包含在消息正文中的重试计数不同）的已发送消息。

在 Amazon SQS 中为单一生产者/消费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并且消息都是唯一的（因为消息正文中包含特定于应用程序的消息 ID），请遵循最佳实践：

- 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。
- 如果为 Amazon SQS FIFO 队列启用基于内容的重复数据删除，并且发送了带有重复数据删除 ID 的消息，则重复数据删除 ID 将覆盖生成的基于内容的 SendMessage 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。

在 Amazon SQS 中针对停机恢复场景进行设计

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，应确保创建者和使用者均可在客户端故障或网络中断的情况下进行恢复。

- 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 中处理可见性超时问题

为了获得最佳性能，请将可见性超时设置为大于 AWS SDK 读取超时。这适用于在短轮询或长轮询中使用 ReceiveMessage API 操作。

使用 Amazon SQS 消息组 ID

[MessageGroupId](#) 是指定消息属于特定消息组的标签。属于同一消息组的消息按相对于消息组的严格顺序逐个处理（但是，属于不同消息组的消息可能会不按顺序处理）。

在 Amazon SQS 中交错多个有序的消息组

要交错一个 FIFO 队列中的多个有序消息组，请使用消息组 ID 值（例如，多个用户的会话数据）。在此情况下，多个使用者可处理此队列，但每个用户的会话数据按 FIFO 方式处理。

Note

如果属于某个特殊消息组 ID 的消息不可见，则其他使用者可处理具有相同消息组 ID 的消息。

避免在 Amazon SQS 的多生产者/消费者系统中处理重复项

在吞吐量和延迟比顺序更重要的情况下，若要在具有多个创建者和使用者的系统中避免处理重复消息，创建者应为每条消息生成一个唯一的消息组 ID。

Note

在此情况下，将消除重复项。但是，无法保证消息的顺序。存在多个创建者和使用者的任何情况都会增加无意中传递重复消息的风险（如果工作线程在可见性超时内未处理消息，并且消息变得对其他工作线程可用）。

避免在 Amazon SQS 中积压大量具有相同消息组 ID 的消息

对于 FIFO 队列，最多可以有 20000 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）。如果您达到此配额，Amazon SQS 将不会返回任何错误消息。FIFO 队列会浏览前 2 万条消息以确定可用的消息组。这意味着，如果您在单个消息组中积压了消息，则在成功使用积压的消息之前，您无法使用其他消息组中稍后发送到该队列的消息。

Note

具有相同消息组 ID 的消息的积压可能是由于使用者无法成功处理一条消息造成的。消息处理问题可能是由于消息内容问题或使用者技术问题造成的。

要移走消息以免反复处理该消息，并取消阻止对具有相同消息组 ID 的其他消息的处理，请考虑设置[死信队列策略](#)。

避免在 Amazon SQS 中对虚拟队列重复使用相同的消息组 ID

要防止发送到具有相同主机队列的不同虚拟队列且具有相同消息组 ID 的消息互相阻塞，请避免在虚拟队列中重复使用相同的消息组 ID。

使用 Amazon SQS 接收请求尝试 ID

接收请求尝试 ID 是用于对 `ReceiveMessage` 调用进行重复数据删除的令牌。

在导致开发工具包和 Amazon SQS 之间产生连接问题的长时间网络中断期间，最佳实践是提供接收请求尝试 ID，并在开发工具包操作失败时使用相同的接收请求尝试 ID 进行重试。

亚马逊 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 中设置长时间投票](#)
- [在 Amazon SQS 中使用适当的轮询模式](#)

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 中设置长时间投票

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。长轮询通过消除空响应的数量（`ReceiveMessage` 请求时没有消息可用时）并消除假的空响应（消息可用但未包含在响应中），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 中使用适当的轮询模式

- 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

您可以使用 AWS SDK for Java 来构建与亚马逊简单队列服务 (Amazon SQS) 和其他服务交互的 Java 应用程序。AWS 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 Amazon SQS 队列操作的示例，例如如何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](#)。

本主题中的示例演示了其他 Amazon SQS 特征，例如服务器端加密 (SSE)、成本分配标签和消息属性。

主题

- [对 Amazon SQS 队列使用服务器端加密](#)
- [为 Amazon SQS 队列配置标签](#)
- [向 Amazon SQS 队列发送消息属性](#)

对 Amazon SQS 队列使用服务器端加密

您可以使用将服务器端加密 (SSE) 添加 AWS SDK for Java 到 Amazon SQS 队列。每个队列使用 AWS Key Management Service (AWS KMS) KMS 密钥生成数据加密密钥。此示例使用适用于 Amazon SQS 的 AWS 托管 KMS 密钥。有关使用 SSE 和 KMS 密钥角色的更多信息，请参阅[Amazon 中的静态加密 SQS](#)。

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](#)。

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

```
// Get the URL of your queue.
String myQueueName = "my queue";
GetQueueUrlResponse getQueueUrlResponse =

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

// 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");

// Create the SetQueueAttributesRequest.
SetQueueAttributesRequest set_attrs_request = SetQueueAttributesRequest.builder()
    .queueUrl(queueUrl)
    .attributes(attributes)
    .build();

sqsClient.setQueueAttributes(set_attrs_request);
```

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 AWS SDK for Java 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);
```

删除标签

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);
```

向 Amazon SQS 队列发送消息属性

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 AWS SDK for Java 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 230.000000000000000001。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccurateWeight", new MessageAttributeValue()
    .withDataType("Number")
    .withStringValue("230.000000000000000001"));
```

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

如果将消息发送到先进先出 (FIFO) 队列 , 请确保 sendMessage 方法在提供消息组 ID 之后执行。

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。

将 API 与亚马逊 SQS 配合使用

本节提供有关构建 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 application-to-application 通信。AWS 与查询协议相比, JSON 协议在请求和响应的序列化/反序列化方面效率更高。AWS 如果您仍然倾向于在 SQS API 中使用 AWS 查询协议, [Amazon 使用的 AWS JSON 协议支持哪些语言 SQS APIs ?](#) 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 格式的输出。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](#)。

主题

- [在 Amazon 中使用 AWS JSON 协议提出查询 API 请求 SQS](#)
- [在 Amazon SQS 中使用 AWS 查询协议发出查询 API 请求](#)
- [对 Amazon SQS 请求进行身份验证](#)
- [Amazon SQS 批处理操作](#)
- [将 Amazon SQS 与软件开发工具包配合使用 AWS](#)

在 Amazon 中使用 AWS JSON 协议提出查询 API 请求 SQS

在本节中，您将学习如何构建 Amazon SQS 终端节点、发出 POST 请求和解释响应。

Note

AWS JSON 大多数语言变体都支持协议。有关受支持语言变体的完整列表，请参阅 [Amazon 使用的 AWS JSON 协议支持哪些语言 SQS APIs ?](#)。

主题

- [构造端点](#)
- [提出 POST 请求](#)
- [解读亚马逊的 SQS JSON API 回应](#)
- [亚马逊 SQS AWS JSON 协议 FAQs](#)

构造端点

要使用 Amazon SQS 队列，您必须构建终端节点。有关 Amazon SQS 终端节点的信息，请参阅中的以下页面 Amazon Web Services 一般参考：

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

每个 Amazon SQS 终端节点都是独立的。例如，如果命名 MyQueue 了两个队列，一个队列有终端节点 `sqs.us-east-2.amazonaws.com` 而另一个队列有终端节点 `sqs.eu-west-2.amazonaws.com`，则这两个队列之间不共享任何数据。

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。

的结构 **AUTHPARAMS** 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POST请求

Amazon SQS POST 请求将查询参数作为表单发送到HTTP请求正文中。

以下是设置为的HTTP标题和X-Amz-Target设置为AmazonSQS.<operationName>的HTTP标题的Content-Type示例application/x-amz-json-1.0。

```
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HTTPPOST请求会向 Amazon SQS 队列发送一条消息。

Note

HTTP标题X-Amz-Target和Content-Type均为必填项。
根据HTTP客户的HTTP版本，您的客户可能会在HTTP请求中添加其他项目。

解读亚马逊的SQSJSONAPI回应

作为对操作请求的响应，Amazon 会SQS返回包含请求结果JSON的数据结构。有关更多信息，请参阅《[亚马逊简单队列服务API参考](#)》中的各个操作和[亚马逊SQS AWS JSON协议 FAQs](#)。

主题

- [成功的JSON响应结构](#)
- [JSON错误响应结构](#)

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，而该字段又包含已创建队列的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，Amazon SQS返回主响应，包括标HTTP题和正文。

在标HTTP头中，`x-amzn-RequestId`包含请求的。 `x-amzn-query-error`包含两条信息：错误的类型，以及错误是生产者错误还是消费者错误。

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

有关在 Amazon 上使用 AWS JSON协议的常见问题SQS。

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是用于异构系统之间通信的最广泛使用和接受的布线方法之一。亚马逊SQSJSON用作 AWS SDK客户端（例如 Java、Python、Golang JavaScript）和亚马逊SQS服务器之间进行通信的媒介。Amazon SQS API 操作HTTP请求接受以下格式的输入JSON。Amazon SQS 操作被执行，执行的响应将以的形式共享给SDK客户端JSON。与 AWS 查询相比，JSON在客户端和服务器之间传输数据方面效率更高。

- Amazon SQS AWS JSON 协议充当亚马逊SQS客户端和服务器的中介。
- 服务器不理解创建 Amazon SQS 操作时使用的编程语言，但它能理解 AWS JSON协议。
- Amazon SQS AWS JSON 协议使用亚马逊SQS客户端和服务器的序列化（将对象转换为JSONJSON格式）和反序列化（将格式转换为对象）。

如何开始使用 Amazon AWS JSON 协议SQS ?

要开始使用最新 AWS SDK版本以更快地向 Amazon 发送消息SQS，请 AWS SDK将您的版本升级到指定版本或任何后续版本。要了解有关SDK客户的更多信息，请参阅下表中的“指南”列。

以下是适用于 Amazon 的 AWS JSON协议的不同语言变体的SDK版本列表 SQSAPIs：

Language	SDK客户存储库	所需的SDK客户端版本	指南
C++	aws/ aws-sdk-cpp	1.11.98	AWS SDK对于 C++
Golang 1.x	aws/ aws-sdk-go	v1.47.7	AWS SDKfor Go
Golang 2.x	aws/ 2 aws-sdk-go-v	v1.28.0	AWS SDK适用于 Go V2
Java 1.x	aws/ aws-sdk-java	1.12.585	AWS SDK适用于 Java
Java 2.x	aws/ 2 aws-sdk-java-v	2.21.19	AWS SDK适用于 Java
JavaScript v2.x	aws/ aws-sdk-js	v2.1492.0	JavaScript on AWS
JavaScript v3.x	aws/ 3 aws-sdk-js-v	v3.447.0	JavaScript on AWS
.NET	aws/ aws-sdk-net	3.7.681.0	AWS SDK对于。 NET
PHP	aws/ aws-sdk-php	3.285.2	AWS SDK for PHP
Python-boto3	boto/boto3	1.28.82	AWS SDK适用于 Python (Boto3)

Language	SDK客户存储库	所需的SDK客户端版本	指南
Python-boto-core	boto/boto-core	1.31.82	AWS SDK适用于 Python (Boto3)
awscli	AWS CLI	1.29.82	AWS命令行界面
Ruby	aws/ aws-sdk-ruby	1.67.0	AWS SDK对于 Ruby

为我的 Amazon SQS 工作负载启用JSON协议会有什么风险？

如果您使用自定义客户端的自定义实现 AWS SDK或组合使用，并 AWS SDK与生成基于 AWS 查询（又名XML基于）SQS的响应的 Amazon 进行交互，则可能与 AWS JSON协议不兼容。如果您遇到任何问题，请联系 [Supp AWS ort](#)。

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

您必须将SDK版本更改为当前版本之前的版本。有关[如何开始使用 Amazon AWS JSON 协议SQS？](#)更多信息，请参阅。AWS SDK中列出的版本[如何开始使用 Amazon AWS JSON 协议SQS？](#)使用适用于 Amazon 的JSON有线协议SQS APIs。如果您将您的版本更改 AWS SDK为以前的版本，则您的 Amazon SQS APIs 将使用该 AWS 查询。

Amazon 使用的 AWS JSON协议支持哪些语言 SQS APIs？

Amazon SQS 支持所有通用 AWS SDKs语言变体 (GA)。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](#)。

Amazon 使用的 AWS JSON协议支持哪些区域 SQS APIs

亚马逊在所有提供亚马逊[AWS SQS服务的地区](#)都SQS支持 AWS JSON协议。

SQS使用该 AWS JSON协议升级到 Amazon 的指定 AWS SDK版本时，我可以期待哪些延迟改善？

AWS JSON与查询协议相比，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，Amazon JSON 协议可将 end-to-end 消息处理延迟SQS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。只要您的 AWS SDK 版本设置了除[如何开始使用 Amazon 协议中列出的版本之外的任何先前版本](#)，您就可以继续使用 [AWS 查询 AWS JSON 协议 SQS](#)。

在哪里可以找到有关 AWS JSON 协议的更多信息？

您可以在 Smithy 文档中找到有关 JSON [AWS JSON1.0 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 中使用 AWS JSON 协议提出查询 API 请求 SQS](#)。

在 Amazon SQS 中使用 AWS 查询协议发出查询 API 请求

在本节中，您将了解如何构造 Amazon SQS 端点，发出 GET 和 POST 请求以及解释响应。

主题

- [构造端点](#)
- [提出 GET 请求](#)
- [提出 POST 请求](#)
- [解释 Amazon SQS XML API 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，如果命名 MyQueue 了两个队列，一个队列有终端节点 `sqs.us-east-2.amazonaws.com` 而另一个队列有终端节点 `sqs.eu-west-2.amazonaws.com`，则这两个队列之间不共享任何数据。

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

队列名称和队列 URL 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

队列名称和队列 URL 区分大小写。

因为 GET 请求是 URL，因此您必须对所有参数值进行 URL 编码。由于 URL 中不允许使用空格，因此每个空格将在 URL 中编码为“%20”。示例的其余部分没有进行 URL 编码，以方便您阅读。

提出 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

在响应操作请求时，Amazon SQS 会返回包含请求结果的 XML 数据结构。有关更多信息，请参阅 [Amazon Simple Queue Service API 参考](#) 中的各操作。

主题

- [成功的 XML 响应结构](#)
- [XML 错误响应结构](#)

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，*ActionNameResponse*)。

此元素包含以下子元素：

- **ActionNameResult** – 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[步骤 1：创建 AWS 账户和 IAM 用户](#)）。接下来，请 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。

主题

- [使用 HMAC-SHA 的基本身份验证过程](#)
- [第 1 部分：来自用户的请求](#)
- [第 2 部分：来自的回应 AWS](#)

使用 HMAC-SHA 的基本身份验证过程

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。
- 使用您的秘密访问密钥计算得出的 HMAC-SHA 请求签名（秘密访问密钥是一个只有您和 AWS 知道的共享秘密；相关详情，请参阅[RFC2104](#)）。[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 Signature Version 4，该版本相比以前的版本可提供经过改进的基于 SHA256 的安全性和性能。创建使用 Amazon SQS 的新应用程序时，应使用 Signature Version 4。

2. 对请求签名必须采用 Base64 编码。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);
    }
}
```

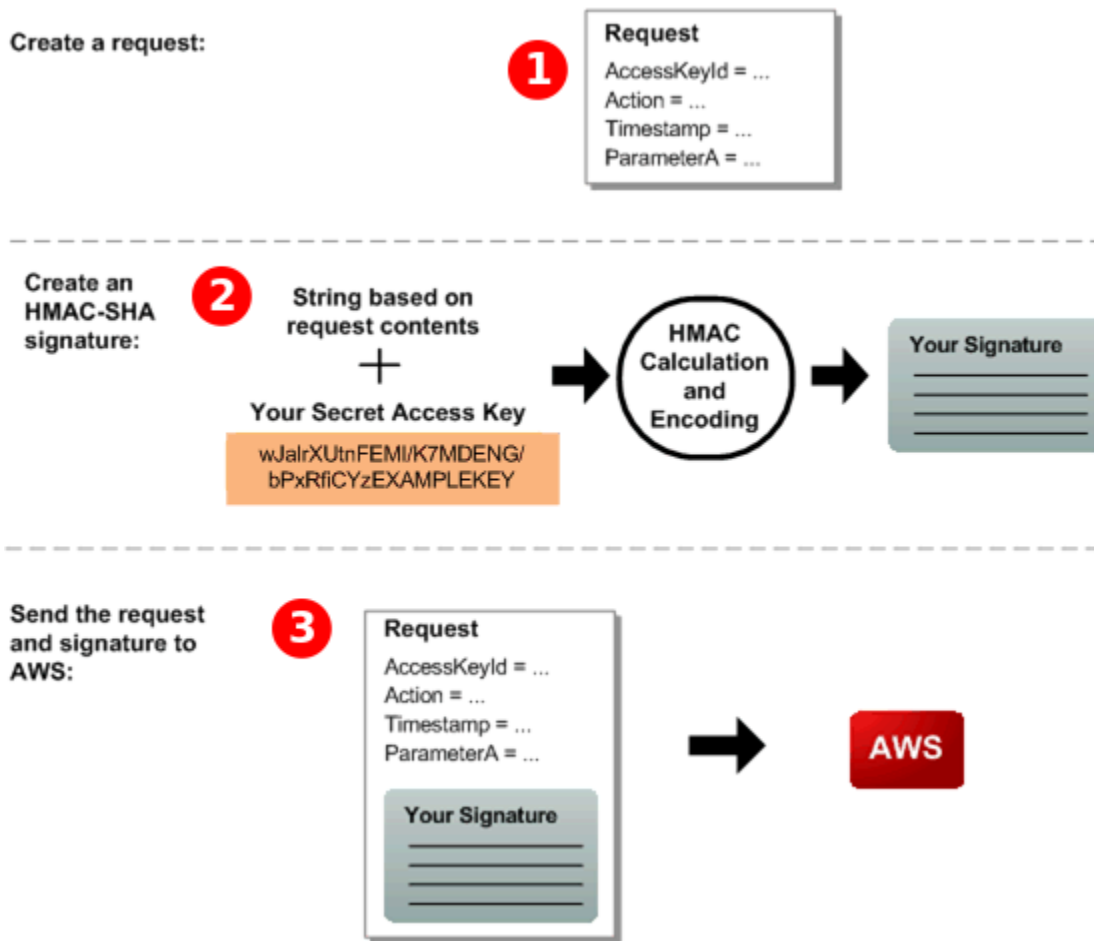
- 请求的 **时间戳**（或到期时间）。在请求中使用的时间戳必须是 `dateTime` 对象，并包含 [完整的日期以及小时、分钟和秒](#)。例如 `2007-01-31T23:59:59Z`。尽管没有强制要求，但还是建议您使用协调世界时（格林威治标准时间）时区提供该对象。

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），则请求将在指定时间后 15 分钟自动过期（AWS 服务器上 AWS 不会处理时间戳比当前时间早于 15 分钟的请求）。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

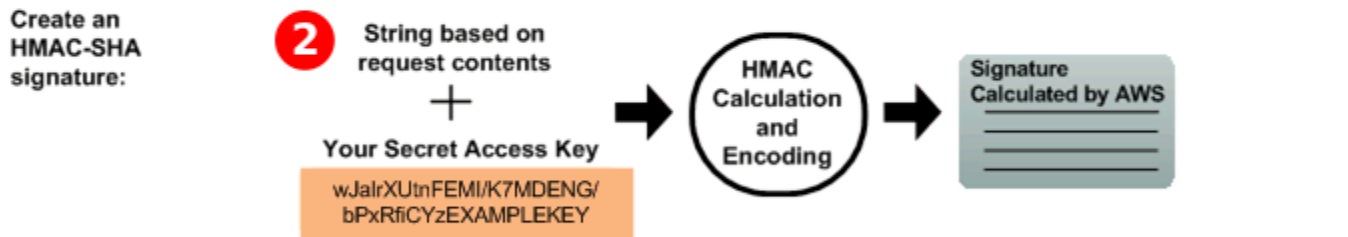
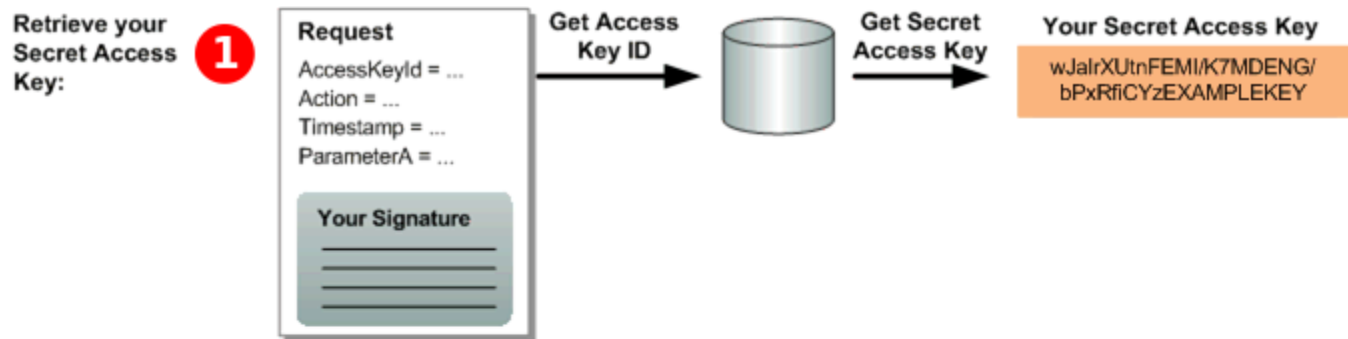
以下是使用 HMAC-SHA 请求签名对 AWS 请求进行身份验证时必须遵循的流程。



1. 构造对的请求 AWS。
2. 使用您的秘密访问密钥计算密钥哈希消息验证码 (HMAC-SHA) 签名。
3. 在请求中包含签名和您的访问密钥 ID，然后将请求发送到 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，则该请求将被丢弃，并 AWS 返回错误。

Amazon SQS 批处理操作

Amazon SQS 提供批量操作，可帮助您降低成本，并通过单个操作处理多达 10 条消息。这些批处理操作包括：

- [SendMessageBatch](#)
- [DeleteMessageBatch](#)
- [ChangeMessageVisibilityBatch](#)

使用批量操作，您可以在单个 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。

重要细节

- 消息大小限制：单次SendMessageBatch调用中发送的所有消息的总大小不能超过 262,144 字节（256 KiB）。
- 权限：您不能为SendMessageBatchDeleteMessageBatch、或显式设置权限ChangeMessageVisibilityBatch。相反，为SendMessageBatchDeleteMessageBatch、或ChangeMessageVisibilityBatch设置权限可以为相应批处理版本的操作设置权限。
- 控制台支持：Amazon SQS 控制台不支持批量操作。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，请考虑以下批量处理消息操作的最佳实践：

- 批量 API 操作：使用 [Amazon SQS 批量 API](#) 操作操作来发送、接收和删除消息，并通过单个操作更改多条消息的可见性超时。这减少了 API 调用的次数和相关成本。
- 客户端缓冲和长轮询：使用长轮询以及随附的缓冲异步客户端，将客户端缓冲与请求批处理相结合。AWS SDK for Java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。

使用 Amazon SQS 启用客户端缓冲和请求批处理

[AWS SDK for Java](#) 包括可访问 Amazon SQS 的 `AmazonSQSBufferedAsyncClient`。此客户端允许通过使用客户端缓冲来进行简单的请求批处理 - 将从客户端发出的调用先进行缓冲，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[通过 Amazon SQS 使用水平扩展和操作批处理来提高吞吐量](#)。

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。

主题

- [使用 AmazonSQS BufferedAsyncClient](#)
- [配置 AmazonSQS BufferedAsyncClient](#)

使用 AmazonSQS BufferedAsyncClient

在开始之前，请完成[设置 Amazon SQS](#) 中的步骤。

Important

目前与 AWS SDK for Java 2.x 不兼容 AmazonSQSBufferedAsyncClient。

可以基于 AmazonSQSAsyncClient 创建新的 AmazonSQSBufferedAsyncClient，例如：

```
// 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 AmazonSQS BufferedAsyncClient

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	描述
longPoll	true	如果 longPoll 设置为 true，AmazonSQS BufferedAsyncClient

参数	默认值	描述
		会在使用消息时尝试使用长轮询。
<code>longPollWaitTimeoutSeconds</code>	20 秒	<p>在返回空接收结果前，<code>ReceiveMessage</code> 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）。</p> <div data-bbox="1068 640 1510 861"><p> Note</p><p>如果禁用长轮询，则此设置不起作用。</p></div>
<code>maxBatchOpenMs</code>	200 毫秒	<p>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）。</p> <p>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）。</p> <p>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。</p>

参数	默认值	描述
maxBatchSize	每批 10 个请求	<p>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。</p> <div data-bbox="1068 478 1507 743"><p> Note</p><p>Amazon SQS 允许的最大值为每批 10 个请求。</p></div>
maxBatchSizeBytes	256 KiB	<p>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）。</p> <div data-bbox="1068 989 1507 1205"><p> Note</p><p>256 KiB 是亚马逊 SQS 允许的最大值。</p></div>

参数	默认值	描述
<code>maxDoneReceiveBatches</code>	10 个批处理	<p>AmazonSQSBufferedAsyncClient 在客户端预取和存储的接收批处理的最大数量。</p> <p>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)。</p> <div data-bbox="1068 800 1507 1066" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px;"><p> Note</p><p>0 表示已禁用所有消息预取，并且仅按需使用消息。</p></div>
<code>maxInflightOutboundBatches</code>	5 个批处理	<p>可以同时处理的最大活跃出站批处理数量。</p> <p>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。</p>

参数	默认值	描述
maxInflightReceive Batches	10 个批处理	<p>可以同时处理的最大活跃接收批处理数量。</p> <p>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。</p> <div style="border: 1px solid #00a0e3; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p> Note</p> <p>0 表示已禁用所有消息预取，并且仅按需使用消息。</p> </div>
visibilityTimeoutSeconds	-1	<p>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。</p> <div style="border: 1px solid #00a0e3; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p> Note</p> <p>-1 表示为队列选择默认设置。</p> <p>不能将可见性超时设置为 0。</p> </div>

通过 Amazon SQS 使用水平扩展和操作批处理来提高吞吐量

Amazon SQS 队列可以实现非常高的吞吐量。有关吞吐量配额的信息，请参阅[亚马逊 SQS 消息配额](#)。

要实现高吞吐量，您必须水平扩展消息创建者和使用者（添加更多创建者和使用者）。

主题

- [横向扩展](#)
- [操作批处理](#)
- [单一操作和批处理请求的有效 Java 示例](#)

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。

Note

进行水平扩展时，必须确保 Amazon SQS 客户端具有足够的连接或线程数量，以支持发送请求和接收响应的并发消息创建者和使用者的数量。例如，默认情况下，该 AWS SDK for Java [AmazonSQSClient](#) 类的实例最多保持 50 个与 Amazon SQS 的连接。要创建更多的并发创建者和使用者，则必须调整 [AmazonSQSClientBuilder](#) 对象上允许的创建者和使用者线程的最大数量，例如：

```
final AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()
    .withClientConfiguration(new ClientConfiguration()
        .withMaxConnections(producerCount + consumerCount))
    .build();
```

对于 [AmazonSQSAsyncClient](#)，还必须确保有足够的线程可用。

此示例仅适用于 Java v.1.x。

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。您可以通过队列的“监控”选项卡或在 [CloudWatch 控制台](#) 上访问这些指标和其他指标。

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 与软件开发工具包配合使用 AWS

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了解其首选语言构建应用程序。

SDK 文档	代码示例
AWS SDK for C++	AWS SDK for C++ 代码示例
AWS CLI	AWS CLI 代码示例
AWS SDK for Go	AWS SDK for Go 代码示例
AWS SDK for Java	AWS SDK for Java 代码示例
AWS SDK for JavaScript	AWS SDK for JavaScript 代码示例
AWS SDK for Kotlin	AWS SDK for Kotlin 代码示例
AWS SDK for .NET	AWS SDK for .NET 代码示例
AWS SDK for PHP	AWS SDK for PHP 代码示例

SDK 文档	代码示例
AWS Tools for PowerShell	PowerShell 代码示例工具
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 代码示例
AWS SDK for Ruby	AWS SDK for Ruby 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 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 配合使用

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 AWS SDK for Java 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。

该库支持根据 JMS [1.1](#) 规范向队列（JMS point-to-point 模型）发送和接收消息。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[亚马逊 SQS 支持 JMS 1.1 的实现](#)和[Amazon SQS 常见问题](#)。

主题

- [使用 JMS 和 Amazon SQS 的先决条件](#)
- [使用亚马逊 SQS Java 消息库](#)
- [将 Java 消息服务与其他 Amazon SQS 客户端一起使用](#)
- [在亚马逊 SQS 标准队列中使用 JMS 的 Java 工作示例](#)
- [亚马逊 SQS 支持 JMS 1.1 的实现](#)

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[在亚马逊 SQS 标准队列中使用 JMS 的 Java 工作示例](#)。

```
<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

使用亚马逊 SQS、CreateQueue API 或亚马逊 SQS Java 消息库中包含的封装亚马逊 SQS 客户端创建队列。AWS Management Console

- 有关使用 AWS Management Console 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[使用亚马逊 SQS Java 消息库](#)。

使用亚马逊 SQS Java 消息库

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，请参阅中的先决条件[使用 JMS 和 Amazon SQS 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[在 Amazon SQS 中精确处理一次](#)。

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[亚马逊 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 [亚马逊 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 [在亚马逊 SQS 标准队列中使用 JMS 的 Java 工作示例](#) 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 [使用亚马逊 SQS Java 消息库](#) 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 (JMSEException 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 [使用亚马逊 SQS Java 消息库](#) 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 [在亚马逊 SQS 标准队列中使用 JMS 的 Java 工作示例](#)。

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

[使用亚马逊 SQS Java 消息库](#) 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 [使用亚马逊 SQS Java 消息库](#) 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 [在亚马逊 SQS 标准队列中使用 JMS 的 Java 工作示例](#)。

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 消息服务与其他 Amazon SQS 客户端一起使用

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 管理大型亚马逊 SQS 消息](#)。

以下 Java 代码示例为扩展客户端库创建 JMS 提供程序。

在测试此示例[使用 JMS 和 Amazon SQS 的先决条件](#)之前，请参阅中的先决条件。

```
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();
```

在亚马逊 SQS 标准队列中使用 JMS 的 Java 工作示例

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[使用 JMS 和 Amazon SQS 的先决条件](#)之前，请先查看中的先决条件。

ExampleConfiguration.java

以下 Java SDK v 1.x 代码示例设置了用于其他 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 JMSEException {
        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 (JMSEException 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 JMSEException {
        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 (JMSEException 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 JMSEException, 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 (JMSEException 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 JMSEException, 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 JMSEException
 */
private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSEException {
    // 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 JMSEException
 */
```

```
private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSEException {
    // 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 JMSEException, 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 JMSEException
*/
private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSEException {
    // 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 JMSEException
 */
private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSEException {
    // 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 JMSEException {
        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 (JMSEException 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 JMSEException {
        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 JMSEException {
        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() );
    }
}
}
```

亚马逊 SQS 支持 JMS 1.1 的实现

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

您可以使用本节提供的教程探索 Amazon SQS 的特征和功能。

主题

- [使用创建 Amazon SQS 队列 AWS 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 AWS CloudFormation

您可以使用 AWS CloudFormation 控制台和 JSON (或 YAML) 模板来创建 Amazon SQS 队列。相关详情，请参阅《AWS CloudFormation 用户指南》中的[使用 AWS CloudFormation 模板和 AWS::SQS::Queue 资源](#)。

用于创建 AWS CloudFormation Amazon 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[在 Amazon SQS 中精确处理一次](#)。

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 [AWS 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。

AWS CloudFormation 开始创建 MyQueue 堆栈并显示 CREATE_IN_PROGRESS 状态。在此过程完成后，AWS CloudFormation 将显示 CREATE_COMPLETE 状态。

Stack Name	Created Time	Status	Description
<input checked="" type="checkbox"/> MyQueue	2017-02-20 11:39:47 UTC-0800	CREATE_COMPLETE	

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

在本教程中，您将了解如何通过安全的专用网络将消息发送到 Amazon SQS 队列。此网络包含一个包含 Amazon EC2 实例的 VPC。该实例通过接口 VPC 端点连接到 Amazon SQS，即使网络已从公共互联网断开连接，您也可以连接 Amazon EC2 实例并将消息发送到 Amazon SQS 队列。有关更多信息，请参阅 [适用于亚马逊的亚马逊 Virtual Private Cloud 终端节点 SQS](#)。

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 并按 `sqs.us-east-2.amazonaws.com` 格式指定端点。
- 私有 DNS 不支持传统端点，例如 `queue.amazonaws.com` 或 `us-east-2.queue.amazonaws.com`。

主题

- [步骤 1：创建 Amazon EC2 密钥对](#)
- [步骤 2：创建 AWS 资源](#)
- [步骤 3：确认您的 EC2 实例不可公开访问](#)
- [步骤 4：为 Amazon SQS 创建 Amazon VPC 端点](#)
- [步骤 5：向 Amazon SQS 队列发送消息](#)

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 and 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 AWS 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
- Amazon SQS 队列

1. 下载名为 [SQS-VPCE-Tutorial-CloudFormation.yaml](#) 的 AWS CloudFormation 模板 GitHub。
2. 登录 [AWS 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”。

AWS CloudFormation 开始创建堆栈并显示 CREATE_IN_PROGRESS 状态。在此过程完成后，AWS 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 AWS 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 (IPv4) 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 AWS CloudFormation 模板创建的队列，例如 vpc e-sqs-tutorial-stack-cfqueue-1abcdefgh2ijk。
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 Create Endpoint (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，如果您位于美国东部（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，并且将显示消息正文的 MD5 摘要和消息 ID，例如：

```
{
  "MD5ofMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
  "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
}
```

有关从您的 AWS 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-cfqueue-1abcdefgh2ijk），请参阅。[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](#)
- [删除亚马逊 SQS 队列](#)
- 在 Amazon EC2 用户指南中[@@ 终止您的实例](#)
- 《Amazon VPC 用户指南》中的[删除 VPC](#)
- AWS CloudFormation 用户指南中的[@@ 在 AWS CloudFormation 控制台上删除堆栈](#)
- 在 Amazon EC2 用户指南中[@@ 删除您的密钥对](#)

SQS使用 Amazon 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) 一起使用。

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和跨服务示例的上下文查看操作。

场景是展示如何通过同一服务中调用多个函数来完成特定任务任务的代码示例。

跨服务示例是指跨多个 AWS 服务工作的示例应用程序。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

开始使用

你好 Amazon SQS

以下代码示例展示了如何开始使用 Amazon SQS。

.NET

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for .NET API参考 [ListQueues](#)”中的。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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详细信息，请参阅“AWS SDK for C++ API参考 [ListQueues](#)”中的。

Go

SDK适用于 Go V2

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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() {  
    sdkConfig, err := config.LoadDefaultConfig(context.TODO())  
    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(context.TODO())  
        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\t%v\n", queueUrl)  
        }  
    }  
}
```

- 有关API详细信息，请参阅“AWS SDK for Go API参考 [ListQueues](#)”中的。

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Java 2.x API参考 [ListQueues](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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详细信息，请参阅“AWS SDK for JavaScript API参考 [ListQueues](#)”中的。

Kotlin

SDK对于 Kotlin 来说

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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 { 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详细信息，请参阅[ListQueues](#)中的 Kotlin AWS SDK API 参考。

代码示例

- [亚马逊SQS使用的操作 AWS SDKs](#)
 - [与 AWS SDK或AddPermission一起使用 CLI](#)
 - [与 AWS SDK或ChangeMessageVisibility一起使用 CLI](#)
 - [与 AWS SDK或ChangeMessageVisibilityBatch一起使用 CLI](#)
 - [与 AWS SDK或CreateQueue一起使用 CLI](#)
 - [与 AWS SDK或DeleteMessage一起使用 CLI](#)
 - [与 AWS SDK或DeleteMessageBatch一起使用 CLI](#)
 - [与 AWS SDK或DeleteQueue一起使用 CLI](#)
 - [与 AWS SDK或GetQueueAttributes一起使用 CLI](#)
 - [与 AWS SDK或GetQueueUrl一起使用 CLI](#)
 - [与 AWS SDK或ListDeadLetterSourceQueues一起使用 CLI](#)
 - [与 AWS SDK或ListQueues一起使用 CLI](#)
 - [与 AWS SDK或PurgeQueue一起使用 CLI](#)
 - [与 AWS SDK或ReceiveMessage一起使用 CLI](#)
 - [与 AWS SDK或RemovePermission一起使用 CLI](#)
 - [与 AWS SDK或SendMessage一起使用 CLI](#)
 - [与 AWS SDK或SendMessageBatch一起使用 CLI](#)
 - [与 AWS SDK或SetQueueAttributes一起使用 CLI](#)
- [亚马逊SQS使用的场景 AWS SDKs](#)
 - [使用创建并发布到FIFO亚马逊SNS主题 AWS SDK](#)
 - [使用接收和处理 Amazon S3 事件通知 AWS SDK。](#)
 - [使用将亚马逊SNS消息发布到亚马逊SQS队列 AWS SDK](#)
- [SQS使用 Amazon 发送和接收批量消息 AWS SDK](#)

- [使用 Amazon SQS 的无服务器示例 AWS SDKs](#)
 - [从亚马逊触发器调用 Lambda 函数 SQS](#)
 - [使用亚马逊触发器报告 Lambda 函数的批量项目失败 SQS](#)
- [使用 Amazon SQS 的跨服务示例 AWS SDKs](#)
 - [使用 Amazon 创建用于发送和检索消息的 Web 应用程序 SQS](#)
 - [使用 Step Functions 创建 Messenger 应用程序](#)
 - [创建 Amazon Textract 浏览器应用程序](#)
 - [使用 Amazon Rekognition 检测视频中的人物和物体 AWS SDK](#)
 - [将 AWS 消息处理框架用于 .NET 发布和接收 Amazon SQS 消息](#)

亚马逊SQS使用的操作 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。这些摘录称为 Amazon SQS API，是大型程序的代码摘录，这些程序必须在上下文中运行。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。

以下示例仅包括最常用的操作。有关完整列表，请参阅《[亚马逊简单队列服务 \(AmazonSQS\) API 参考](#)》。

示例

- [与 AWS SDK或AddPermission一起使用 CLI](#)
- [与 AWS SDK或ChangeMessageVisibility一起使用 CLI](#)
- [与 AWS SDK或ChangeMessageVisibilityBatch一起使用 CLI](#)
- [与 AWS SDK或CreateQueue一起使用 CLI](#)
- [与 AWS SDK或DeleteMessage一起使用 CLI](#)
- [与 AWS SDK或DeleteMessageBatch一起使用 CLI](#)
- [与 AWS SDK或DeleteQueue一起使用 CLI](#)
- [与 AWS SDK或GetQueueAttributes一起使用 CLI](#)
- [与 AWS SDK或GetQueueUrl一起使用 CLI](#)
- [与 AWS SDK或ListDeadLetterSourceQueues一起使用 CLI](#)
- [与 AWS SDK或ListQueues一起使用 CLI](#)
- [与 AWS SDK或PurgeQueue一起使用 CLI](#)
- [与 AWS SDK或ReceiveMessage一起使用 CLI](#)

- [与 AWS SDK或RemovePermission一起使用 CLI](#)
- [与 AWS SDK或SendMessage一起使用 CLI](#)
- [与 AWS SDK或SendMessageBatch一起使用 CLI](#)
- [与 AWS SDK或SetQueueAttributes一起使用 CLI](#)

与 AWS SDK或AddPermission一起使用 CLI

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 SendMessageFromMyQueue --aws-account-ids 12345EXAMPLE --actions SendMessage
```

输出 :

```
None.
```

- 有关API详细信息，请参阅“[AddPermission AWS CLI命令参考](#)”。

PowerShell

用于 PowerShell

示例 1：此示例 AWS 账户 允许指定用户从指定队列发送消息。

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE  
-Label SendMessageFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```


- 有关API详细信息，请参阅 [AWS Tools for PowerShell Cmdlet 参考AddPermission](#)中的。

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

与 AWS SDK或ChangeMessageVisibility一起使用 CLI

以下代码示例演示如何使用 ChangeMessageVisibility。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#)中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for C++ 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详细信息，请参阅“[ChangeMessageVisibility AWS CLI命令参考](#)”。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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详细信息，请参阅“AWS SDK for JavaScript API参考 [ChangeMessageVisibility](#)”中的。

SDK对于 JavaScript (v2)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [ChangeMessageVisibility](#)”中的。

PowerShell

用于 PowerShell

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 x 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详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 [ChangeMessageVisibility](#) 中的。

Ruby

SDK对于 Ruby

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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_message_result_before = sqs.receive_message({
    queue_url: queue_url,
    max_number_of_messages: 10 # Receive up to 10 messages, if there are that
    many.
  })

  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
  })
end
```

```
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 '#{receive_queue_name}', as it
  does not exist."
end
```

- 有关API详细信息，请参阅“AWS SDK for Ruby API参考 [ChangeMessageVisibility](#)”中的。

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

与 AWS SDK或ChangeMessageVisibilityBatch一起使用 CLI

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超时可见性

此示例将 2 条指定消息的超时可见性更改为 10 小时 (10 小时 x 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详细信息，请参阅 [“ChangeMessageVisibilityBatch AWS CLI命令参考”](#)。

PowerShell

用于 PowerShell

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。第一条消息的可见性超时更改为 10 小时（10 小时 * 60 分钟 * 60 秒 = 36000 秒）。第二条消息的可见性超时更改为 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详细信息，请参阅 [AWS Tools for PowerShell Cmdlet 参考ChangeMessageVisibilityBatch](#)中的。

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

与 AWS SDK或CreateQueue一起使用 CLI

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](#)

.NET

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#)中进行设置和运行。

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详细信息，请参阅“AWS SDK for .NET API参考 [CreateQueue](#)”中的。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for C++ 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详细信息，请参阅“[CreateQueue AWS CLI命令参考](#)”。

Go

SDK适用于 Go V2

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
// 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(queueName string, isFifoQueue bool) (string,
error) {
  var queueUrl string
  queueAttributes := map[string]string{}
  if isFifoQueue {
```

```
    queueAttributes["FifoQueue"] = "true"
  }
  queue, err := actor.SqsClient.CreateQueue(context.TODO(), &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详细信息，请参阅“AWS SDK for Go API参考 [CreateQueue](#)”中的。

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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
```

```
.getUrl(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)
            .numberOfMessages(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详细信息，请参阅“AWS SDK for Java 2.x API参考 [CreateQueue](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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;
};
```

使用长轮询创建一个 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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [CreateQueue](#)”中的。

SDK对于 JavaScript (v2)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [CreateQueue](#)”中的。

Kotlin

SDK对于 Kotlin 来说

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
suspend fun createQueue(queueNameVal: String): String {
    println("Create Queue")
    val createQueueRequest =
        CreateQueueRequest {
            queueName = queueNameVal
        }

    SqsClient { 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详细信息，请参阅[CreateQueue](#)中的 Kotlin AWS SDK API 参考。

PowerShell

用于 PowerShell

示例 1：此示例创建了一个具有指定名称的队列。

```
New-SQSQueue -QueueName MyQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关API详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考[CreateQueue](#)中的。

Python

SDK适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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
```

- 有关API详细信息，请参阅[CreateQueue](#)中的 AWS SDKPython (Boto3) API 参考。

Ruby

SDK对于 Ruby

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
# 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详细信息，请参阅“AWS SDK for Ruby API参考 [CreateQueue](#)”中的。

SAP ABAP

SDK对于 SAP ABAP

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

创建 Amazon SQS 标准队列。

```

TRY.
    oo_result = lo_sqs->createqueue( iv_queue_name = iv_queue_name ).      "
oo_result is returned for testing purposes. "
    MESSAGE 'SQS queue created.' TYPE 'I'.
    CATCH /aws1/cx_sqsqueuedeletedrecently.
        MESSAGE 'After deleting a queue, wait 60 seconds before creating another
queue with the same name.' TYPE 'E'.
    CATCH /aws1/cx_sqsqueueexists.
        MESSAGE 'A queue with this name already exists.' TYPE 'E'.
ENDTRY.

```

创建等待消息到达的 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_queue_name = iv_queue_name
        it_attributes = lt_attributes
    ).
    MESSAGE 'SQS queue created.' TYPE 'I'.
    CATCH /aws1/cx_sqsqueuedeletedrecently.
        MESSAGE 'After deleting a queue, wait 60 seconds before creating another
queue with the same name.' TYPE 'E'.
    CATCH /aws1/cx_sqsqueueexists.
        MESSAGE 'A queue with this name already exists.' TYPE 'E'.
ENDTRY.

```

- 有关API详细信息，请参阅[CreateQueue](#)中的AWS SDK以供SAPABAPI参考。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

与 AWS SDK 或 `DeleteMessage` 一起使用 CLI

以下代码示例演示如何使用 `DeleteMessage`。

.NET

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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详细信息，请参阅“AWS SDK for .NET API参考 [DeleteMessage](#)”中的。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for C++ 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详细信息，请参阅“[DeleteMessage AWS CLI命令参考](#)”。

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Java 2.x API参考 [DeleteMessage](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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详细信息，请参阅“AWS SDK for JavaScript API参考 [DeleteMessage](#)”中的。

SDK对于 JavaScript (v2)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [DeleteMessage](#)”中的。

Kotlin

SDK对于 Kotlin 来说

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient { 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 { region = "us-east-1" }.use { sqsClient ->
    sqsClient.deleteQueue(request)
    println("$queueUrlVal was deleted!")
}
}
```

- 有关API详细信息，请参阅[DeleteMessage](#)中的 Kotlin AWS SDK API 参考。

PowerShell

用于 PowerShell

示例 1：此示例从指定队列中删除具有指定收据句柄的消息。

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- 有关API详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考[DeleteMessage](#)中的。

Python

SDK适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅[DeleteMessage](#)中的 AWS SDKPython (Boto3) API 参考。

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

与 AWS SDK或DeleteMessageBatch一起使用 CLI

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

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
/// <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详细信息，请参阅“AWS SDK for .NET API参考 [DeleteMessageBatch](#)”中的。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for C++ 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": "AQEB1mg1...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
    "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```


输出 :

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- 有关API详细信息，请参阅 [“DeleteMessageBatch AWS CLI命令参考”](#)。

Go

SDK适用于 Go V2

 Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
// 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(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(context.TODO(),
    &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详细信息，请参阅“AWS SDK for Go API参考 [DeleteMessageBatch](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for JavaScript API参考 [DeleteMessageBatch](#)”中的。

PowerShell

用于 PowerShell

示例 1：此示例从指定队列中删除 2 封具有指定收据句柄的消息。

```
$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 = "AQEBq0VY...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详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 [DeleteMessageBatch](#) 中的。

Python

SDK适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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
```

- 有关API详细信息，请参阅[DeleteMessageBatch](#)中的 AWS SDKPython (Boto3) API 参考。

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

与 AWS SDK或DeleteQueue一起使用 CLI

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](#)

.NET

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

使用队列删除队列URL。

```

/// <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详细信息，请参阅“AWS SDK for .NET API参考 [DeleteQueue](#)”中的。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```

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详细信息，请参阅“AWS SDK for C++ 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详细信息，请参阅“[DeleteQueue AWS CLI命令参考](#)”。

Go

SDK适用于 Go V2

 Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
// 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(queueUrl string) error {
    _, err := actor.SqsClient.DeleteQueue(context.TODO(), &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详细信息，请参阅“AWS SDK for Go API参考 [DeleteQueue](#)”中的。

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Java 2.x API参考 [DeleteQueue](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

删除亚马逊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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [DeleteQueue](#)”中的。

SDK对于 JavaScript (v2)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

删除亚马逊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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [DeleteQueue](#)”中的。

Kotlin

SDK对于 Kotlin 来说

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient { 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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
    }
}
```

```
        println("$queueUrlVal was deleted!")
    }
}
```

- 有关API详细信息，请参阅[DeleteQueue](#)中的 Kotlin AWS SDK API 参考。

PowerShell

用于 PowerShell

示例 1：此示例删除指定的队列。

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关API详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考[DeleteQueue](#)中的。

Python

SDK适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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
```

- 有关API详细信息，请参阅[DeleteQueue](#)中的 AWS SDKPython (Boto3) API 参考。

Ruby

SDK对于 Ruby

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Ruby API参考 [DeleteQueue](#)”中的。

SAP ABAP

SDK对于 SAP ABAP

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
TRY.
    lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).
```

```
MESSAGE 'SQS queue deleted' TYPE 'I'.
ENDTRY.
```

- 有关API详细信息，请参阅[DeleteQueue](#)中的AWS SDK以供SAPABAPI参考。

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

与 AWS SDK或GetQueueAttributes一起使用 CLI

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

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
/// <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详细信息，请参阅“AWS SDK for .NET API参考 [GetQueueAttributes](#)”中的。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for C++ 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详细信息，请参阅 [“GetQueueAttributes AWS CLI命令参考”](#)。

Go

SDK适用于 Go V2

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
// 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(queueUrl string) (string, error) {
    var queueArn string
    arnAttributeName := types.QueueAttributeNameQueueArn
    attribute, err := actor.SqsClient.GetQueueAttributes(context.TODO(),
        &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详细信息，请参阅“AWS SDK for Go API参考 [GetQueueAttributes](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for JavaScript API参考 [GetQueueAttributes](#)”中的。

PowerShell

用于 PowerShell

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":"2008-10-17","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":
  "SendMessageFromMyQueue","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":"2008-10-17","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":

  "SendMessageFromMyQueue","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详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 [GetQueueAttributes](#) 中的。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅 [将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

与 AWS SDK 或 `GetQueueUrl` 一起使用 CLI

以下代码示例演示如何使用 `GetQueueUrl`。

.NET

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for .NET API参考 [GetQueueUrl](#)”中的。

C++

SDK对于 C++

 Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for C++ API参考 [GetQueueUrl](#)”中的。

CLI

AWS CLI

要排队 URL

此示例获取指定队列的URL。

命令:

```
aws sqs get-queue-url --queue-name MyQueue
```

输出:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- 有关API详细信息，请参阅“[GetQueueUrl AWS CLI命令参考](#)”。

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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

- 有关API详细信息，请参阅“AWS SDK for Java 2.x API参考 [GetQueueUrl](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

获取 URL Amazon SQS 队列。

```
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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [GetQueueUrl](#)”中的。

SDK对于 JavaScript (v2)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

获取 URL 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 = {
  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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [GetQueueUrl](#)”中的。

PowerShell

用于 PowerShell

示例 1：此示例列URL出了具有指定名称的队列。

```
Get-SQSQueueUrl -QueueName MyQueue
```

输出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 有关API详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 [GetQueueUrl](#) 中的。

Python

SDK适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅[GetQueueUrl](#)中的 AWS SDKPython (Boto3) API 参考。

SAP ABAP

SDK对于 SAP ABAP

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```

    TRY.
        oo_result = lo_sqs->getqueueurl( iv_queue_name = iv_queue_name ).
    oo_result is returned for testing purposes. "
        MESSAGE 'Queue URL retrieved.' TYPE 'I'.
    CATCH /aws1/cx_sqsqueue_does_not_exist.
        MESSAGE 'The requested queue does not exist.' TYPE 'E'.
    ENDTRY.

```

- 有关API详细信息，请参阅[GetQueueUrl](#)中的AWS SDK以供SAP ABAP API参考。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

与 AWS SDK 或 `ListDeadLetterSourceQueues` 一起使用 CLI

以下代码示例演示如何使用 `ListDeadLetterSourceQueues`。

CLI

AWS CLI

列出死信来源队列

此示例列出了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详细信息，请参阅[“ListDeadLetterSourceQueues AWS CLI 命令参考”](#)。

PowerShell

用于 PowerShell

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详细信息，请参阅 [AWS Tools for PowerShell Cmdlet 参考ListDeadLetterSourceQueues](#)中的。

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

与 AWS SDK或ListQueues 一起使用 CLI

以下代码示例演示如何使用 ListQueues。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#)中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for C++ 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详细信息，请参阅“[ListQueues AWS CLI命令参考](#)”。

Go

SDK适用于 Go V2

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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() {  
    sdkConfig, err := config.LoadDefaultConfig(context.TODO())  
    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(context.TODO())
    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详细信息，请参阅“AWS SDK for Go API参考 [ListQueues](#)”中的。

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```

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详细信息，请参阅“AWS SDK for Java 2.x API参考 [ListQueues](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

列出您的 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);
        urls.forEach((url) => console.log(url));
    }

    return urls;
};
```

- 有关更多信息，请参阅 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [ListQueues](#)”中的。

SDK对于 JavaScript (v2)

 Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

列出您的 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 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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [ListQueues](#)”中的。

Kotlin

SDK对于 Kotlin 来说

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
suspend fun listQueues() {
    println("\nList Queues")

    val prefix = "que"
    val listQueuesRequest =
        ListQueuesRequest {
            queueNamePrefix = prefix
        }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.listQueues(listQueuesRequest)
        response.queueUrls?.forEach { url ->
            println(url)
        }
    }
}
```

- 有关API详细信息，请参阅[ListQueues](#)中的 Kotlin AWS SDK API 参考。

PowerShell

用于 PowerShell

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：此示例列出了所有以指定名称开头的队列。

```
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详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 [ListQueues](#) 中的。

Python

SDK适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅[ListQueues](#)中的 AWS SDKPython (Boto3) API 参考。

Ruby

SDK对于 Ruby

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Ruby API参考 [ListQueues](#)”中的。

Rust

SDK对于 Rust

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

检索该地区列出的第一个 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详细信息，请参见[ListQueues](#)中的 Rust AWS SDK API 参考。

SAP ABAP

SDK对于 SAP ABAP

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
TRY.
    oo_result = lo_sqs->listqueues( ).      " oo_result is returned for
testing purposes. "
    MESSAGE 'Retrieved list of queues.' TYPE 'I'.
ENDTRY.
```

- 有关API详细信息，请参阅[ListQueues](#)中的AWS SDK以供SAPABAPI参考。

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

与 AWS SDK或PurgeQueue一起使用 CLI

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详细信息，请参阅“[PurgeQueue AWS CLI命令参考](#)”。

PowerShell

用于 PowerShell

示例 1：此示例删除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详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考[PurgeQueue](#)中的。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

与 AWS SDK 或 `ReceiveMessage` 一起使用 CLI

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：

- [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

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

使用队列接收来自队列的消息 URL。

```
/// <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 queue 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详细信息，请参阅“AWS SDK for .NET API参考 [ReceiveMessage](#)”中的。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for C++ 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": "AIDAIKMSNQ7TEXAMPLE",
    "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 `mPostalCode` 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": "AIDAIKMSNQ7TEXAMPLE",
  "SentTimestamp": "1442428276921"
},
"MessageAttributes": {
  "PostalCode": {
    "DataType": "String",
    "StringValue": "ABC123"
  }
}
]
```

- 有关API详细信息，请参阅“[ReceiveMessage AWS CLI命令参考](#)”。

Go

SDK适用于 Go V2

 Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
  SqsClient *sqs.Client
}

// GetMessage uses the ReceiveMessage action to get messages from an Amazon SQS
// queue.
func (actor SqsActions) GetMessage(queueUrl string, maxMessages int32, waitTime
int32) ([]types.Message, error) {
  var messages []types.Message
```

```
result, err := actor.SqsClient.ReceiveMessage(context.TODO(),
&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详细信息，请参阅“AWS SDK for Go API参考 [ReceiveMessage](#)”中的。

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Java 2.x API参考 [ReceiveMessage](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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,
        })),
      }),
    );
  }
};
```

使用长轮询支持从 Amazon SQS 队列接收消息。

```
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详细信息，请参阅“AWS SDK for JavaScript API参考 [ReceiveMessage](#)”中的 SDK对于 JavaScript (v2)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

使用长轮询支持从 Amazon SQS 队列接收消息。

```
// 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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [ReceiveMessage](#)”中的。

Kotlin

SDK对于 Kotlin 来说

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
suspend fun receiveMessages(queueUrlVal: String?) {
    println("Retrieving messages from $queueUrlVal")

    val receiveMessageRequest =
        ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.receiveMessage(receiveMessageRequest)
        response.messages?.forEach { message ->
            println(message.body)
        }
    }
}
```

- 有关API详细信息，请参阅[ReceiveMessage](#)中的 Kotlin AWS SDK API 参考。

PowerShell

用于 PowerShell

示例 1：此示例列出了指定队列最多接下来要接收的 10 条消息的信息。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 详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 [ReceiveMessage](#) 中的。

Python

SDK 适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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
```

- 有关API详细信息，请参阅[ReceiveMessage](#)中的 AWS SDKPython (Boto3) API 参考。

Ruby

SDK对于 Ruby

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Ruby API参考 [ReceiveMessage](#)”中的。

Rust

SDK对于 Rust

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```

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详细信息，请参见[ReceiveMessage](#)中的 Rust AWS SDK API 参考。

SAP ABAP

SDK对于 SAP ABAP

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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.

```

使用长轮询支持从 Amazon SQS 队列接收消息。

```

TRY.
    oo_result = lo_sqs->receivemessage(           " oo_result is returned for
testing purposes. "
        iv_queueurl = iv_queue_url

```

```
        iv_waitemptyseconds = 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详细信息，请参阅[ReceiveMessage](#)中的AWS SDK以供SAPABAPAPI参考。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

与 AWS SDK 或 `RemovePermission` 一起使用 CLI

以下代码示例演示如何使用 `RemovePermission`。

CLI

AWS CLI

移除权限

此示例从指定队列中移除带有指定标签的权限。

命令:

```
aws sqs remove-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessageFromMyQueue
```

输出:

```
None.
```

- 有关API详细信息，请参阅“[RemovePermission AWS CLI 命令参考](#)”。

PowerShell

用于 PowerShell

示例 1：此示例从指定队列中删除带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详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考[RemovePermission](#)中的。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

与 AWS SDK 或 `SendMessage` 一起使用 CLI

以下代码示例演示如何使用 `SendMessage`。

.NET

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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详细信息，请参阅“AWS SDK for .NET API参考 [SendMessage](#)”中的。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
    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详细信息，请参阅“AWS SDK for C++ 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：

```
{
  "MD50fMessageBody": "51b0a325...39163aa0",
  "MD50fMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- 有关API详细信息，请参阅“[SendMessage AWS CLI命令参考](#)”。

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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);
    }
}
}
```

- 有关API详细信息，请参阅“AWS SDK for Java 2.x API参考 [SendMessage](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [SendMessage](#)”中的。

SDK对于 JavaScript (v2)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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 [AWS SDK for JavaScript 开发人员指南](#)。
- 有关API详细信息，请参阅“AWS SDK for JavaScript API参考 [SendMessage](#)”中的。

Kotlin

SDK对于 Kotlin 来说

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
suspend fun sendMessages(
    queueUrlVal: String,
    message: String,
) {
    println("Sending multiple messages")
    println("\nSend message")
    val sendRequest =
        SendMessageRequest {
            queueUrl = queueUrlVal
            messageBody = message
            delaySeconds = 10
        }

    SqsClient { 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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient.sendMessageBatch(sendMessageBatchRequest)
        println("Batch message were successfully sent.")
    }
}
```



```
    }
}
```

- 有关API详细信息，请参阅[SendMessage](#)中的 Kotlin AWS SDK API 参考。

PowerShell

用于 PowerShell

示例 1：此示例将具有指定属性和消息正文的消息发送到指定队列，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 c739-4d0c-818b-1820eEXAMPLE	51b0a3256d59467f973009b73EXAMPLE	c35fed8f-

- 有关API详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考[SendMessage](#)中的。

Python

SDK适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅[SendMessage](#)中的 AWS SDKPython (Boto3) API 参考。

Ruby

SDK对于 Ruby

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Ruby API参考 [SendMessage](#)”中的。

Rust

SDK对于 Rust

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参见[SendMessage](#)中的 Rust AWS SDK API 参考。

SAP ABAP

SDK对于 SAP ABAP

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```

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详细信息，请参阅[SendMessage](#)中的AWS SDK以供SAPABAPAPI参考。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

与 AWS SDK 或 `SendMessageBatch` 一起使用 CLI

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详细信息，请参阅 [“SendMessageBatch AWS CLI命令参考”](#)。

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Java 2.x API参考 [SendMessageBatch](#)”中的。

PowerShell

用于 PowerShell

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。第一条消息的传送延迟 15 秒，第二条消息的传送延迟 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详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考 [SendMessageBatch](#) 中的。

Python

SDK适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```

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详细信息，请参阅[SendMessageBatch](#)中的 AWS SDKPython (Boto3) API 参考。

Ruby

SDK对于 Ruby

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for Ruby API参考 [SendMessageBatch](#)”中的。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅 [将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

与 AWS SDK 或 `SetQueueAttributes` 一起使用 CLI

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

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

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详细信息，请参阅“AWS SDK for .NET API参考 [SetQueueAttributes](#)”中的。

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```

    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详细信息，请参阅“AWS SDK for C++ 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详细信息，请参阅“[SetQueueAttributes AWS CLI命令参考](#)”。

Go

SDK适用于 Go V2

 Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
// 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(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(context.TODO(),
&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详细信息，请参阅“AWS SDK for Go API参考 [SetQueueAttributes](#)”中的。

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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详细信息，请参阅“AWS SDK for JavaScript API参考 [SetQueueAttributes](#)”中的。

PowerShell

用于 PowerShell

示例 1：此示例说明如何设置让队列订阅SNS主题的策略。向主题发布消息时，会向订阅队列发送一条消息。

```
# 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
$policy = @"
{
  "Version": "2008-10-17",
  "Id": "$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详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考[SetQueueAttributes](#)中的。

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

亚马逊SQS使用的场景 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。这些场景向您展示了如何通过调用 Amazon 中的多个函数来完成特定任务SQS。每个场景都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行代码的说明。

示例

- [使用创建并发布到FIFO亚马逊SNS主题 AWS SDK](#)
- [使用接收和处理 Amazon S3 事件通知 AWS SDK。](#)
- [使用将亚马逊SNS消息发布到亚马逊SQS队列 AWS SDK](#)
- [SQS使用 Amazon 发送和接收批量消息 AWS SDK](#)

使用创建并发布到FIFO亚马逊SNS主题 AWS SDK

以下代码示例展示了如何创建和发布到 A FIFO mazon SNS 主题。

Java

SDK适用于 Java 2.x

 Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

此示例

- 创建一个 Amazon SNS FIFO 主题、两个亚马逊SQSFIFO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");

        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“参AWS SDK for Java 2.x API考”中的以下主题。

- [CreateTopic](#)
- [Publish](#)
- [订阅](#)

Python

SDK适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

创建亚马逊SNSFIFO主题，为该主题订阅亚马逊SQSFIFO和标准队列，然后向该主题发布消息。

```
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),
                },
            )
            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 Python (Boto3) API 参考中的AWS SDK以下主题。
 - [CreateTopic](#)
 - [Publish](#)
 - [订阅](#)

SAP ABAP

SDK对于 SAP ABAP

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

创建FIFO主题，在 Amazon SQS FIFO 队列中订阅该主题，然后向亚马逊SNS主题发布消息。

```

    " Creates a FIFO topic. "
    DATA lt_tpc_attributes TYPE /aws1/
cl_snstopicattrsm_w=>tt_topicattributesmap.
    DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsm_w=>ts_topicattributesmap_maprow.
    ls_tpc_attributes-key = 'FifoTopic'.
    ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsm_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_snstopiclimitexcexdex.
    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_snssubscriptionlmt00.

```

```

        MESSAGE 'Unable to create subscriptions. You have reached the maximum
number of subscriptions allowed.' TYPE 'E'.
    ENENTRY.

    " 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'.
    ENENTRY.

```

- 有关API详细信息，请参阅中的以下主题AWS SDK以供SAPABAPI参考。
 - [CreateTopic](#)
 - [Publish](#)
 - [订阅](#)

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

使用接收和处理 Amazon S3 事件通知 AWS SDK。

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

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

此示例说明如何使用 Amazon 处理 S3 通知事件SQS。

```
/**
 * 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“参AWS SDK for Java 2.x API考”中的以下主题。
 - [DeleteMessageBatch](#)
 - [GetQueueAttributes](#)
 - [PutBucketNotificationConfiguration](#)
 - [ReceiveMessage](#)

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

使用将亚马逊SNS消息发布到亚马逊SQS队列 AWS SDK

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

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

在命令提示符中运行交互式场景。

```
///  
/// <summary>  
/// Console application to run a workflow 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 workflow 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 workflow.
    /// </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 workflow, 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\n"}and queue URL {queueUrl}" +
                $"{"\r\n"}has 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 messageId = 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($"{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:" +
            $"{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;
}
}
```

创建一个封装 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;
}
}

```

创建一个封装 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“参AWS SDK for .NET API考”中的以下主题。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)

- [订阅](#)
- [Unsubscribe](#)

C++

SDK对于 C++

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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 ostream;
        ostream << "Enter a name for " << (first ? "an" : "the next")
                << " SQS queue. ";
        queueName = askQuestion(ostream.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“参AWS SDK for C++ API考”中的以下主题。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [订阅](#)
 - [Unsubscribe](#)

Go

SDK适用于 Go V2

 Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

在命令提示符中运行交互式场景。

```
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() (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(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(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(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(
    queueName string, queueUrl string, topicName string, topicArn string, ordinal
    string,
    isFifoTopic bool) (string, bool) {

    queueArn, err := runner.sqsActor.GetQueueArn(queueUrl)
    if err != nil {
        panic(err)
    }
    log.Printf("The ARN of your queue is: %v.\n", queueArn)

    err = runner.sqsActor.AttachSendMessagePolicy(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(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(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(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
message? (y/n) ", "y")
}
}

func (runner ScenarioRunner) PollForMessages(queueUrls []string) {
    log.Println("Polling queues for messages...")
    for _, queueUrl := range queueUrls {
        var messages []types.Message
        for {
            currentMsgs, err := runner.sqsActor.GetMessages(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(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(
    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()
        }
    }()
    queueCount := 2

    log.Println(strings.Repeat("-", 88))
    log.Printf("Welcome to messaging with topics and queues.\n\n"+
        "In this workflow, 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()
    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(ordinal, isFifoTopic)
    resources.queueUrls = append(resources.queueUrls, queueUrl)

    _, filtering := runner.SubscribeQueueToTopic(queueName, queueUrl, topicName,
topicArn, ordinal, isFifoTopic)
    usingFilters = usingFilters || filtering
}

log.Println(strings.Repeat("-", 88))
runner.PublishMessages(topicArn, isFifoTopic, contentBasedDeduplication,
usingFilters)
log.Println(strings.Repeat("-", 88))
runner.PollForMessages(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()
}

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}

```

定义一个封装本示例中使用的 Amazon SNS 操作的结构。

```

// 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(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(context.TODO(), &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(topicArn string) error {
    _, err := actor.SnsClient.DeleteTopic(context.TODO(), &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(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(context.TODO(), &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 subscribe 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(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(context.TODO(), &publishInput)
    if err != nil {
        log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
        err)
    }
    return err
}
```

定义一个封装本示例中使用的 Amazon SQS 操作的结构。

```
// 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(queueName string, isFifoQueue bool) (string,
error) {
    var queueUrl string
    queueAttributes := map[string]string{}
    if isFifoQueue {
        queueAttributes["FifoQueue"] = "true"
    }
    queue, err := actor.SqsClient.CreateQueue(context.TODO(), &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(queueUrl string) (string, error) {
    var queueArn string
    arnAttributeName := types.QueueAttributeNameQueueArn
    attribute, err := actor.SqsClient.GetQueueAttributes(context.TODO(),
    &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(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(context.TODO(),
&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(queueUrl string, maxMessages int32, waitTime
int32) ([]types.Message, error) {
    var messages []types.Message
    result, err := actor.SqsClient.ReceiveMessage(context.TODO(),
&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(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(context.TODO(),
&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(queueUrl string) error {
    _, err := actor.SqsClient.DeleteQueue(context.TODO(), &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详细信息，请参阅“参AWS SDK for Go API考”中的以下主题。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)

- [订阅](#)
- [Unsubscribe](#)

Java

SDK适用于 Java 2.x

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * This Java example performs these 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.
 */
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 = "814548047983";
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 workflow, 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 = "{\n" +
    "    \"Statement\": [\n" +
    "        {\n" +
    "            \"Effect\": \"Allow\", \n" +
    "            \"Principal\": {\n" +
    "                \"Service\": \"sns.amazonaws.com\" \n" +
    "            }, \n" +
    "            \"Action\": \"sqs:SendMessage\", \n" +
    "            \"Resource\": \"arn:aws:sqs:us-east-1:\" +
accountId + \":\" + sqsQueueName + "\", \n" +
    "            \"Condition\": {\n" +
    "                \"ArnEquals\": {\n" +
    "                    \"aws:SourceArn\": \"arn:aws:sns:us-east-1:\" +
accountId + \":\" + topicName + "\" \n" +
    "                } \n" +
    "            } \n" +
    "        ] \n" +
    "    }";

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> queueAttrs = response.attributesAsStrings();
    for (Map.Entry<String, String> queueAtt : queueAttrs.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“参AWS SDK for Java 2.x API考”中的以下主题。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [订阅](#)
 - [Unsubscribe](#)

JavaScript

SDK对于 JavaScript (v3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

这是此工作流程的入口点。

```
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";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  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工作流程。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.
  let 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, deduplicationId, 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“参AWS SDK for JavaScript API考”中的以下主题。

- [CreateQueue](#)
- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [Publish](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [订阅](#)
- [Unsubscribe](#)

Kotlin

SDK对于 Kotlin 来说

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
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 workflow, 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> = mutableListOfOf()
    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，请参阅中的以下主题以获取 Kotlin AWS SDK API 参考。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)

- [订阅](#)
- [Unsubscribe](#)

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

SQS 使用 Amazon 发送和接收批量消息 AWS SDK

以下代码示例展示了如何：

- 创建亚马逊 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。

Python

SDK 适用于 Python (Boto3)

Note

还有更多相关信息 [GitHub](#)。查找完整示例，学习如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

创建用于封装 Amazon SQS 消息函数的函数。

```
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(
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 Python (Boto3) API 参考中的AWS SDK以下主题。
 - [CreateQueue](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [ReceiveMessage](#)
 - [SendMessageBatch](#)

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前SDK版本的详细信息。

使用 Amazon SQS 的无服务器示例 AWS SDKs

以下代码示例展示了如何将 Amazon SQS 与配合使用 AWS SDKs。

示例

- [从亚马逊触发器调用 Lambda 函数 SQS](#)
- [使用亚马逊触发器报告 Lambda 函数的批量项目失败 SQS](#)

从亚马逊触发器调用 Lambda 函数 SQS

以下代码示例说明如何实现一个 Lambda 函数，该函数接收通过从队列接收消息而触发的事件。SQS 该函数从事件参数检索消息并记录每条消息的内容。

.NET

AWS SDK for .NET

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。

使用使用 Lambda 使用一个SQS事件。 .NET。

```
// 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.DefaultLambdaJsonSerializer))]

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

SDK适用于 Go 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 在 Lambda 上使用SQS事件。

```
// 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

SDK适用于 Java 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。

使用 Java 在 Lambda 上使用SQS事件。

```
// 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

SDK对于 JavaScript (v3)

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。

使用使用 Lambda 使用一个SQS事件。 JavaScript

```
// 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;
  }
}
```

使用使用 Lambda 使用一个SQS事件。 TypeScript


```
// 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

SDK for PHP

 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。

使用使用 Lambda 使用一个SQS事件。PHP

```
// 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

SDK适用于 Python (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 在 Lambda 中使用SQS事件。

```
# 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

SDK对于 Ruby

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 在 Lambda 上使用SQS事件。

```
# 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

SDK对于 Rust

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 Rust 在 Lambda 上使用 SQS 事件。

```
// 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 AWS SDK 开发者指南和代码示例的完整列表，请参阅 [将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

使用亚马逊触发器报告 Lambda 函数的批量项目失败 SQS

以下代码示例说明如何为从队列接收事件的 Lambda 函数实现部分批量响应。SQS 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

AWS SDK for .NET

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 Lambda 报告SQS批量项目失败。NET。

```
// 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.DefaultLambdaJsonSerializer))]
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

SDK适用于 Go 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 G SQS o 报告使用 Lambda 的批处理项目失败。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "encoding/json"
    "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 /* Your message processing condition here */ {
            batchItemFailures = append(batchItemFailures, map[string]interface{}{
                "itemIdentifier": message.MessageId})
        }
    }

    sqsBatchResponse := map[string]interface{}{
        "batchItemFailures": batchItemFailures,
    }
    return sqsBatchResponse, nil
}

func main() {
    lambda.Start(handler)
}
```

Java

SDK适用于 Java 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。

使用 Java 使用 Lambda 报告 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>();
        String messageId = "";
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
                messageId = message.getMessageId();
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.add(new
                SQSBatchResponse.BatchItemFailure(messageId));
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

JavaScript

SDK对于 JavaScript (v3)

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 Lambda 报告SQS批量项目失败。 JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
// 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 Lambda 报告SQS批量项目失败。 TypeScript

```
// 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

SDK for PHP

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 Lambda 报告SQS批量项目失败。PHP

```
// 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

SDK适用于 Python (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 使用 Lambda 报告 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:
            # process message
        except Exception as e:
            batch_item_failures.append({"itemIdentifier":
record['messageId']})

    sqs_batch_response["batchItemFailures"] = batch_item_failures
    return sqs_batch_response
```

Ruby

SDK对于 Ruby

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 使用 Lambda 报告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
```

```
end
end
```

Rust

SDK对于 Rust

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 Rust 使用 Lambda 报告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 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

使用 Amazon SQS 的跨服务示例 AWS SDKs

以下示例应用程序用于 AWS SDKs 将 Amazon SQS 与其他应用程序合并 AWS 服务。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如何设置和运行应用程序的说明。

示例

- [使用 Amazon 创建用于发送和检索消息的 Web 应用程序 SQS](#)
- [使用 Step Functions 创建 Messenger 应用程序](#)
- [创建 Amazon Textract 浏览器应用程序](#)
- [使用 Amazon Rekognition 检测视频中的人物和物体 AWS SDK](#)
- [将 AWS 消息处理框架用于 .NET 发布和接收 Amazon SQS 消息](#)

使用 Amazon 创建用于发送和检索消息的 Web 应用程序 SQS

以下代码示例展示了如何使用 Amazon 创建消息应用程序 SQS。

Java

SDK 适用于 Java 2.x

演示如何使用 Amazon 开发 SQS API 用于发送和检索消息 REST API 的 Spring。

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

- 亚马逊 SQS

Kotlin

SDK对于 Kotlin 来说

演示如何使用 Amazon 开发SQSAPI用于发送和检索消息RESTAPI的 Spring。

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
- 亚马逊 SQS

有关 AWS SDK开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前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

SDK适用于 Python (Boto3)

演示如何使用 with 创建信使应用程序，该 AWS SDK for Python (Boto3) 应用程序从 Amazon DynamoDB 表中检索消息记录并通过亚马逊简单队列服务 (Amazon) 将其发送。AWS Step Functions SQS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。
- 更新状态机定义以同时向亚马逊简单队列服务 (AmazonSQS) 发送消息。
- 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SQS亚马逊。

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
- 亚马逊 SQS
- Step Functions

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 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

SDK 对于 JavaScript (v3)

演示如何使用 AWS SDK for JavaScript 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。它使用亚马逊简单存储服务 (Amazon S3) Service 进行存储，对于通知，它会轮询订阅亚马逊简单通知服务 (Amazon SQS) 主题的亚马逊简单队列服务 (Amazon SNS) 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
- 亚马逊 SNS
- 亚马逊 SQS
- Amazon Textract

Python

SDK 适用于 Python (Boto3)

演示如何 AWS SDK for Python (Boto3) 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) 主题发布通知。
- 轮询亚马逊简单队列服务 (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 S3
- 亚马逊 SNS
- 亚马逊 SQS
- Amazon Textract

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

使用 Amazon Rekognition 检测视频中的人物和物体 AWS SDK

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。

Python

SDK 适用于 Python (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 配置为在任务完成并订阅亚马逊简单队列服务 SNS (Amazon) 队列时通知亚马逊简单通知服务 SQS (Amazon) 主题。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
- 亚马逊 SNS

- 亚马逊 SQS

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

将 AWS 消息处理框架用于 .NET 发布和接收 Amazon SQS 消息

以下代码示例说明如何使用 SQS 消息处理框架创建用于发布和接收 Amazon AWS 消息的应用程序。
NET。

.NET

AWS SDK for .NET

提供 AWS 消息处理框架的教程。NET。本教程创建了一个允许用户发布 Amazon SQS 消息的 Web 应用程序和一个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《AWS SDK for .NET 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

- 亚马逊 SQS

有关 AWS SDK 开发者指南和代码示例的完整列表，请参阅[将 Amazon SQS 与软件开发工具包配合使用 AWS](#)。本主题还包括有关入门的信息以及有关先前 SDK 版本的详细信息。

对 Amazon SQS 中的问题进行故障排除

以下主题针对您在使用亚马逊 SQS 控制台、亚马逊 SQS API 或其他工具和 Amazon SQS 时可能遇到的常见错误和问题提供了疑难解答建议。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

- [对 Amazon 中的拒绝访问错误进行故障排除 SQS](#)
- [对亚马逊 SQS API 错误进行故障排除](#)
- [对 Amazon SQS 死信队列和DLQ重新驱动问题进行故障排除](#)
- [解决 Amazon SQS 中的 FIFO 限制问题](#)
- [对 Amazon SQS ReceiveMessage API 调用未返回的消息进行故障排除](#)
- [对 Amazon SQS 网络错误进行故障排除](#)
- [使用 AWS X-Ray 排查 Amazon Simple Queue Service 队列问题](#)

对 Amazon 中的拒绝访问错误进行故障排除 SQS

以下主题涵盖了 Amazon SQS API 通话中最常见的原因 AccessDenied 或 AccessDeniedException 错误。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 电话中的“AccessDeniedAccessDeniedException”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-

```
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.
```

主题

- [Amazon SQS 队列策略和IAM政策](#)
- [AWS Key Management Service 权限](#)
- [VPC端点策略](#)
- [组织服务控制政策](#)

Amazon SQS 队列策略和IAM政策

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SQSAPI拨打 Amazon 电话的IAM委托人。如果IAM委托人来自同一个账户，则亚马逊SQS队列策略或 Ident AWS ity and Access Management (IAM) 策略必须包含明确允许操作访问的权限。
- 如果委托人是一个IAM实体：
 - 您可以通过检查的右上角或使用命令来识别您的IAM用户或角色。AWS Management Console[aws sts get-caller-identity](#)
 - 检查与IAM用户或角色相关的IAM策略。您可以使用以下方法之一：
 - 使用IAM策略[模拟器测试IAM策略](#)。
 - 查看不同的[IAM策略类型](#)。
 - 如果需要，请编辑您的[IAM用户政策](#)。
 - 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。
- 如果委托人是跨账户委托人，则 Amazon SQS 队列策略和IAM策略都必须明确允许访问。
- 如果策略使用条件元素，请检查该条件是否限制了访问权限。

Important

任一策略中的明确拒绝都会优先于明确的允许。以下是[亚马逊SQS政策](#)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 `KmsMasterKeyId` 属性，也可以从队列控制台的“加密”下使用属性。

- [制作者所需的权限](#) :

```
{
  "Effect": "Allow",
  "Action": [
    "kms:GenerateDataKey"
  ],
  "Resource": "<Key ARN>"
}
```

- [消费者所需的权限](#) :

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "<Key ARN>"
}
```

- [跨账户访问所需的权限](#) :

```
{
  "Effect": "Allow",
  "Action": [
    "kms:DescribeKey",
    "kms:Decrypt",
    "kms:ReEncrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "<Key ARN>"
}
```

您可以使用以下任一方法为 Amazon SQS 队列启用加密 :

- [SSE-Amazon SQS](#) (加密密钥由亚马逊SQS服务创建和管理。)
- [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 托管密 [KMS 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，请使用客户托管密钥。这样做可以让你编辑密钥策略。

VPC 端点策略

如果您 [SQS 通过亚马逊虚拟私有云 \(Amazon VPC\) 终端节点访问亚马逊](#)，则亚马逊 SQS VPC 终端节点策略必须允许访问。您可以为亚马逊的亚马逊 VPC 终端节点创建策略 SQS，在其中可以指定以下内容：

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 IAM 用户 *MyUser* 允许向 Amazon SQS 队列发送消息 *MyQueue*。其他操作、IAM 用户和 Amazon SQS 资源被拒绝通过 VPC 终端节点进行访问。

```
{
  "Statement": [{
    "Action": ["sqs:SendMessage"],
    "Effect": "Allow",
    "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
    "Principal": {
      "AWS": "arn:aws:iam:123456789012:user/MyUser"
    }
  }]
}
```

组织服务控制政策

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 策略不会阻止向 Amazon 发出的任何请求 SQS。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](#)。

对亚马逊 SQS API 错误进行故障排除

以下主题涵盖调用 Amazon SQS API 时返回的最常见错误以及如何对其进行故障排除。

主题

- [QueueDoesNotExist 错误](#)
- [InvalidAttributeValue 错误](#)
- [ReceiptHandle 错误](#)

QueueDoesNotExist 错误

当亚马逊 SQS 服务找不到上述亚马逊 SQS 操作队列时，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的区域是否正确。当你没有在客户端上配置区域时，SDK 或从[配置文件](#)或环境变量 AWS CLI 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 会将该区域设置为 us-east-1。
- 队列最近可能被删除：如果队列在 API 调用之前被删除，那么 API 调用将返回此错误。在错误发生之前检查是否 CloudTrail 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.
```

检查权限，然后使用正确的权限进行 API 调用。

有关排除 QueueDoesNotExist 错误的更多详细信息，请参阅在[我的 Amazon SQS 队列发出 API 调用时如何解决 QueueDoesNotExist 错误？](#) 在 AWS 知识中心指南中。

InvalidAttributeValue 错误

更新 Amazon SQS 队列资源策略或使用错误策略或委托人的属性时，将返回此错误。

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[验证 IAM 策略](#)。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。
- 委托人无效：确保该 Principal 元素存在于资源策略中并且该值有效。如果您的 Amazon SQS 资源策略 Principal 元素包含 IAM 实体，则在使用该策略之前，请确保该实体存在。Amazon SQS

会验证资源策略并检查 IAM 实体。如果 IAM 实体不存在，您将收到一条错误消息。要确认 IAM 实体，请使用[GetRole](#)和 [GetUserAPI](#)。

有关如何解决InvalidAttributeValue错误的更多信息，请参阅在[我的 Amazon SQS 队列发出 API 调用时如何解决 QueueDoesNotExist 错误？](#) 在AWS 知识中心指南中。

ReceiptHandle 错误

发出 [DeleteMessage](#)API 调用时，如果收据句柄不正确ReceiptHandleIsInvalid或已过期，则InvalidParameterValue可能会返回错误或。

- ReceiptHandleIsInvalid 错误：如果收据账号不正确，您将收到与以下示例类似的错误：

```
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：如果收据账号已过期，您将收到与以下示例类似的错误：

```
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：

接收句柄是为每条收到的消息创建的，并且仅在可见性超时时间内有效。当可见性超时时间到期时，消息将在队列中对消费者可见。当你再次收到消费者发来的消息时，你会收到一个新的收据账号。为防止出现错误或过期的收据处理错误，请使用正确的收据句柄在 Amazon SQS 队列可见性超时时间内删除该消息。

有关如何解决ReceiptHandle错误的更多信息，请参阅在使用 [Amazon SQS DeleteMessage API 调用时如何解决“ReceiptHandleIsInvalid”和“InvalidParameter值”错误？](#) 在AWS 知识中心指南中。

对 Amazon SQS 死信队列和DLQ重新驱动问题进行故障排除

以下主题涵盖了 Amazon SQS DLQ 和DLQ重新驱动问题的最常见原因以及如何对其进行故障排除。有关更多信息，请参阅[如何解决 Amazon SQS DLQ 重新驱动问题？](#) 在AWS 知识中心指南中。

主题

- [DLQ问题](#)
- [DLQ-重新驱动问题](#)

DLQ问题

了解常见DLQ问题以及如何解决这些问题。

主题

- [使用控制台查看消息可能会导致消息移至死信队列](#)
- [死信队列的 `NumberOfMessagesSent` 和 `NumberOfMessagesReceived` 不匹配](#)
- [创建和配置死信队列重新驱动器](#)
- [标准消息和FIFO队列消息故障处理](#)

使用控制台查看消息可能会导致消息移至死信队列

A SQS mazon 会根据相应队列的重新驱动策略在控制台中查看消息。因此，如果您在控制台中查看相应队列的重新驱动策略中指定的次数的消息，则该消息将被移至相应队列的死信队列。

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，它将由 `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`和的值可能会 `NumberOfMessagesReceived`有所不同。

创建和配置死信队列重新驱动器

[死信队列重新驱动需要您设置适当的权限，让 Amazon 从死信队列接收消息，并将消息发送SQS到目标队列。](#) 如果您没有正确的权限，则死信队列重新驱动任务可能会失败。您可以查看邮件重新驱动任务的状态以修复问题，然后重试。

标准消息和FIFO队列消息故障处理

[标准队列](#)会一直处理消息，直到[保留期到期](#)。这种持续处理可最大限度地减少队列被未消耗的消息阻塞的机会。如果有大量消息但消费者反复无法删除，则会增加成本，并给硬件带来额外的负担。为了降低成本，请将失败的消息移至死信队列。

标准队列还允许发送大量动态消息。如果您的大部分消息无法使用，也没有发送到死信队列，则您的消息处理速度可能会减慢。要保持队列的效率，请确保您的应用程序正确处理消息。

[FIFO队列](#)通过按顺序使用来自消息组的消息来提供一次性处理。因此，尽管使用者可以继续从另一个消息组检索已排序的消息，但在成功处理阻塞该队列的消息或将其移至死信队列之前，第一个消息组仍不可用。

此外，FIFO队列允许减少传输中消息的数量。要防止FIFO队列被消息阻止，请确保您的应用程序正确处理消息。

有关更多信息，请参阅[亚马逊 SQS 消息配额](#) 和[亚马逊 SQS 最佳实践](#)。

DLQ-重新驱动问题

了解常见的 DLQ-redrive 问题以及如何解决这些问题。

主题

- [AccessDenied 权限问题](#)
- [NonExistentQueue 错误](#)
- [CouldNotDetermineMessageSource 错误](#)

AccessDenied 权限问题

当DLQ重新驱动失败时，就会发生AccessDenied错误，因为 AWS Identity and Access Management (IAM) 实体没有所需的权限。

错误消息示例：

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

发出DLQ重新驱动请求需要以下API权限：

要开始重新驱动消息，请执行以下操作：

- 死信队列权限：
 - `sqs:StartMessageMoveTask`
 - `sqs:ReceiveMessage`
 - `sqs>DeleteMessage`
 - `sqs:GetQueueAttributes`
- 目标队列权限：
 - `sqs:SendMessage`
- KMS加密死信队列时的密钥：
 - `kms:Decrypt`
- KMS目标队列加密时的密钥：
 - `kms:GenerateDataKey`

要取消正在进行的留言重新存档，请执行以下操作：

- 死信队列权限：
 - `sqs:CancelMessageMoveTask`
 - `sqs:ReceiveMessage`
 - `sqs>DeleteMessage`
 - `sqs:GetQueueAttributes`

要显示邮件移动状态，请执行以下操作：

- 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。检查并重新驱动到存在的 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开始DLQ重新驱动时会发生CouldNotDetermineMessageSource错误：

- 直接发送给 SQS with 的 Amazon 消息 [SendMessage](#) API。DLQ
- 来自亚马逊简单通知服务 (Amazon SNS) 主题或 AWS Lambda 函数的消息，其中DLQ已配置。

要解决此错误，请在开始重新驱动时选择“重新驱动到自定义目的地”。然后，进入 Amazon SQS 队列，将所有消息从移ARN DLQ至目标队列。

错误消息示例：

```
Failed: CouldNotDetermineMessageSource
```

解决 Amazon SQS 中的 FIFO 限制问题

默认情况下，FIFO 队列支持、和 [DeleteMessage](#) 的每个 API 操作每秒 300 个 [SendMessage](#) 事务。[ReceiveMessage](#) 即使队列中有可用消息，超过 300 TPS 的请求也会 [ThrottlingException](#) 出现错误。要缓解这种情况，您可以使用以下方法：

- [在 Amazon SQS 中为 FIFO 队列启用高吞吐量](#)。
- 使用 Amazon SQS API 批量操作和 [SendMessageBatch](#) [DeleteMessageBatch](#)，[ChangeMessageVisibilityBatch](#) 将每个 API 操作每秒最多 3,000 条消息的 TPS 限制提高到每秒 3,000 条，从而降低成本。对于 [ReceiveMessage](#) API，将 [MaxNumberOfMessages](#) 参数设置为每笔交易最多接收十条消息。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 ID 的消息。使用来自相同 [ReceiveMessage](#) API 请求的收据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。这允许在 FIFO 队列分区之间均匀分配。有关更多信息，请参阅 [使用 Amazon SQS 消息组 ID](#)。

有关更多信息，请参阅 [为什么我的 Amazon SQS FIFO 队列没有返回其他消息组中的所有消息或其他消息组中的消息？](#) 在 AWS 知识中心指南中。

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消费者的最常见原因以及如何对其进行故障排除。有关更多信息，请参阅[为什么我无法接收来自我的 Amazon SQS 队列的消息？](#) 在AWS 知识中心指南中。

主题

- [空队列](#)
- [已达到飞行限制](#)
- [消息延迟](#)
- [消息正在传送中](#)
- [轮询方法](#)

空队列

要确定队列是否为空，请使用长轮询来调用 [ReceiveMessageAPI](#)。您也可以使用 `ApproximateNumberOfMessagesVisible`、`ApproximateNumberOfMessagesNotVisible`、和 `ApproximateNumberOfMessagesDelayed` CloudWatch 指标。如果在几分钟内将所有指标值都设置为 0，则队列将被视为空。

已达到飞行限制

[如果您使用长轮询并且队列的飞行限制（FIFO 为20000，标准为默认值为120000），Amazon SQS 将不会返回超过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](#)，或者消息是使用[消息计时器](#)发送的，则在延迟时间结束之前消息才可见。要验证队列是否配置为延迟队列，请使用 [GetQueueAttributesAPI](#) `DelaySeconds` 属性，或者从队列控制台的“传送延迟”下方。检查[ApproximateNumberOfMessagesDelayed](#) CloudWatch 指标以了解是否有任何消息延迟。

消息正在传送中

如果其他消费者对消息进行了轮询，则该消息将在[可见性超时](#)时间内处于传输状态或不可见。额外的民意调查可能会返回空的接收结果。检查[ApproximateNumberOfMessagesVisible](#) CloudWatch指标以了

解可接收的消息数量。对于 FIFO 队列，如果带有消息组 ID 的消息正在传输中，则除非您删除该消息或该消息变为可见，否则不会再返回任何消息。这是因为[消息排序](#)是在 FIFO 队列中的消息组级别维护的。

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，即使消息可供接收，也可能无法收到。随后的民意调查请求将返回消息。

如果您使用的是[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且不超过指定的最大数量后发送响应。如果的值 `ReceiveMessageWaitTimeSeconds` 太低，则可能无法收到所有可用消息。

对 Amazon SQS 网络错误进行故障排除

以下主题涵盖了 Amazon SQS 中网络问题的最常见原因以及如何对其进行故障排除。

主题

- [ETIMEOUT error](#)
- [UnknownHostException error](#)

ETIMEOUT error

当客户端无法建立与 Amazon SQS 终端节点的 TCP 连接时，就会发生ETIMEOUT错误。

故障排除：

- 检查网络连接

通过运行类似的命令来测试您与 Amazon SQS 的网络连接。telnet

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) 规则必须允许流向端口 80 或 443 的流量。
 - 网络 ACL 出站 (出口) 规则必须允许流向 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亚马逊弹性计算云 (Amazon EC2) 实例必须具有互联网连接。
- 亚马逊虚拟私有云 (亚马逊 VPC) 终端节点

如果您通过 Amazon VPC 终端节点访问 Amazon SQS，则终端节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 终端节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，亚马逊 SQS VPC 终端节点 AWS Identity and Access Management (IAM) 策略必须允许访问。以下示例 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* 其他操作、IAM 用户和 Amazon SQS 资源被拒绝通过 VPC 终端节点进行访问。

```
{
  "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：

使用该 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 传统端点的 SDK：

```
nslookup <region>.queue.amazonaws.com
```

如果您收到不成功的输出，请按照 [DNS 的工作原理以及如何解决部分或间歇性的 DNS 故障中的说明进行操作](#) 在AWS 知识中心指南中。

如果您收到了有效的输出，则很可能是应用程序级别的问题。要解决应用程序级别的问题，请尝试以下方法：

- 重新启动您的应用程序。
- 确认您的 Java 应用程序没有错误的 DNS 缓存。如果可能，请将您的应用程序配置为符合 DNS TTL。有关更多信息，请参阅 [为 DNS 名称查询设置 JVM TTL](#)。

有关如何解决网络错误的更多信息，请参阅 [如何解决 Amazon SQS 的“ETIMEOUT”和“UnknownHost 异常”连接错误？](#) 在AWS 知识中心指南中。

使用 AWS X-Ray 排查 Amazon Simple Queue Service 队列问题

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、响应以及您的应用程序对下游 AWS 资源、微服务、数据库和 HTTP Web API 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

- [亚马逊的数据保护 SQS](#)
- [Amazon 中的身份和访问管理 SQS](#)
- [Amazon SQS 中的日志记录和监控](#)
- [Amazon 合规性验证 SQS](#)
- [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](#)

亚马逊的数据保护 SQS

分 AWS [担责任模型](#)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架构上的内容的控制。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FAQ](#)。有关欧洲数据保护的信息，请参阅[责任AWS 共担模型和AWS安全GDPR](#)博客上的博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭据并使用 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。我们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)。
- 使用SSL/TLS与 AWS 资源通信。我们需要 TLS 1.2，建议使用 TLS 1.3。
- 使用API进行设置和用户活动记录 AWS CloudTrail。
- 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 通过命令行界面或访问时需要 FIPS 140-3 经过验证的加密模块API，请使用端点。FIPS有关可用FIPS端点的更多信息，请参阅[联邦信息处理标准 \(FIPS\) 140-3](#)。

我们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 AWS 服务使用控制台、API、SQS AWS CLI或与 Amazon 或其他人合作的情况 AWS SDKs。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URL向外部服务器提供，我们强烈建议您不要在中包含凭据信息，URL以验证您对该服务器的请求。

以下各节提供有关Amazon数据保护的信息SQS。

主题

- [Amazon 中的数据加密 SQS](#)
- [Amazon 中的互联网流量隐私 SQS](#)

Amazon 中的数据加密 SQS

数据保护是指保护传输中（往返亚马逊时SQS）和静态数据（存储在亚马逊SQS数据中心的磁盘上时）的数据。您可以使用安全套接字层 (SSL) 或客户端加密来保护传输中的数据。默认情况下，Amazon 使用磁盘加密SQS存储消息和文件。您可以通过请求 Amazon 在将消息保存SQS到其数据中心的加密文件系统之前对其进行加密来保护静态数据。亚马逊SQS建议使用SSE来优化数据加密。

主题

- [Amazon 中的静态加密 SQS](#)
- [亚马逊SQS密钥管理](#)

Amazon 中的静态加密 SQS

服务器端加密 (SSE) 允许您在加密队列中传输敏感数据。SSE使用SQS管理的加密密钥 (SSE-SQS) 或在 (-) 中管理的密钥保护队列中的 AWS Key Management Service 邮件内容。SSE KMS有关SSE使用进行管理的信息 AWS Management Console，请参阅以下内容：

- [配置 SSE-SQS 用于队列 \(控制台\)](#)
- [配置 SSE-KMS 用于队列 \(控制台\)](#)

有关SSE使用 AWS SDK for Java（以及[CreateQueueSetQueueAttributes](#)、[GetQueueAttributes](#)操作）进行管理的信息，请参阅以下示例：

- [对 Amazon SQS 队列使用服务器端加密](#)

- [为配置KMS权限 AWS 服务](#)

SSEAmazon SQS 收到消息后立即对其进行加密。消息以加密形式存储，Amazon 仅在消息发送给授权消费者时才会对其进行SQS解密。

⚠ Important

所有向SSE启用了签名版本的队列发出的请求都必须使用HTTPS[签名版本 4](#)。

使用默认密钥（Amazon 的AWS 托管KMS密钥SQS）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 服务的某些功能与标准队列兼容，SSE但仅适用于标准队列：

- [Auto Scaling 生命周期挂钩](#)
- [AWS Lambda 死信队列](#)

有关其他服务与加密队列的兼容性的信息，请参阅 [为 AWS 服务配置KMS权限](#) 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 Amazon SQS 与一起 AWS KMS使用时，加密您的消息数据的数据[密钥](#)也会被加密并与其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 Amazon 的 AWS 托管KMS密钥SQS，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 ?](#)

主题

- [加密范围](#)
- [关键术语](#)

加密范围

SSE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)
- 每队列指标

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。SSE启用后，匿名SendMessage和对加密队列的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，请务必禁用SSE。这不会影响 Amazon 的正常运作SQS：

- 仅在启用队列加密后发送消息时对其进行加密。Amazon SQS 不对积压的消息进行加密。
- 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：

- 当 Amazon SQS 将消息从加密的源队列移动到未加密的死信队列时，该消息将保持加密状态。
- 当 Amazon SQS 将消息从未加密的源队列移到加密的死信队列时，该消息将保持未加密状态。

关键术语

以下关键术语可以帮助您更好地理解功能SSE。有关详细说明，请参阅 [《Amazon 简单队列服务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 AWS KMS 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。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KMS中的 AWS 托管密钥或自定义密KMS钥ARN的别名ARN、别名、密钥 ID 或密钥。虽然 Amazon AWS 托管KMS密钥的别名始终SQS为alias/aws/sqs，但例如，自定义KMS密钥的别名可以是alias/MyAlias。您可以使用这些KMS密钥来保护 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，亚马逊将SQS使用亚马逊的 AWS 托管KMS密钥SQS。
- 首次使用为队列指定亚马逊的 AWS 托管KMS密钥时，AWS KMS 会SQS为亚马逊创建 AWS 托管KMS密钥SQS。AWS Management Console
- 或者，首次对SSE已启用的队列使用SendMessage或SendMessageBatch操作时，AWS KMS 会为 Amazon 创建 AWS 托管KMS密钥SQS。

您可以创建KMS密钥，定义控制密KMS钥使用方式的策略，并使用控制 AWS KMS 台的“客户管理的密钥”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 开发者指南》中的[密KMS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参考文献[KeyId](#)中的。有关查找KMS密钥标识符的信息，请参阅[查找密钥 ID 和ARN](#) 《AWS Key Management Service 开发者指南》。

Important

使用需要支付额外费用 AWS KMS。有关更多信息，请参阅 [估算成本 AWS KMS](#) 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](#)。

亚马逊SQS密钥管理

Amazon SQS 与 AWS Key Management Service (KMS) 集成以管理服务器端加密的[KMS密钥](#) (SSE)。 [Amazon 中的静态加密 SQS](#)有关SSE信息和密钥管理定义，请参阅。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。以下各节提供有关在 Amazon SQS 服务中使用密KMS键和数据密钥的信息。

主题

- [配置 AWS KMS 权限](#)
- [了解数据密钥重用周期](#)
- [估算成本 AWS KMS](#)
- [AWS KMS 错误](#)

配置 AWS KMS 权限

每个KMS密钥都必须有密钥策略。请注意，您无法修改 Amazon AWS 托管KMS密钥的密钥策略 SQS。此KMS密钥的策略包括账户中所有委托人（有权使用 Amazon 的SQS）使用加密队列的权限。

对于客户托管KMS密钥，您必须配置密钥策略以为每个队列创建者和使用者添加权限。为此，您需要在KMS密钥策略中将生产者和消费者指定为用户。有关 AWS KMS 权限的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的[AWS KMS 资源和操作](#)或[AWS KMS API权限参考](#)。

或者，您可以在分配给生成和使用加密消息的委托人的IAM策略中指定所需的权限。有关更多信息，请参阅《AWS Key Management Service 开发者指南》AWS KMS中的将[IAM策略与一起使用](#)。

Note

虽然您可以配置向亚马逊发送和接收的全ARN局权限SQS，但 AWS KMS 需要在IAM策略Resource部分明确命名特定区域的全部KMS密钥。

为 AWS 服务配置KMS权限

有几项 AWS 服务充当可以向 Amazon SQS 队列发送事件的事件源。要允许这些事件源使用加密队列，您必须创建客户托管KMS密钥，并在密钥策略中添加权限，以便服务使用所需 AWS KMS API方法。执行以下步骤来配置权限。

Warning

更改用于加密您的 Amazon SQS 消息的KMS密钥时，请注意，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KMS密钥并确保其密钥策略授予亚马逊SQS的权限。kms:GenerateDataKey更新到用于加密新消息的新KMS密钥后，请确保在删除或禁用旧KMS密钥之前，处理所有使用旧密KMS钥加密的现有消息并将其从队列中删除。

1. 创建客户管理的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 服务事件源使用该kms:GenerateDataKeyAPI方法，请在KMS密钥策略中添加以下语句。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "service.amazonaws.com"
    },
    "Action": [
      "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

事件源	服务名称
Amazon S3 事件通知	s3.amazonaws.com
亚马逊SNS主题订阅	sns.amazonaws.com

3. 使用您的KMS密钥@@ [配置现有SSE队列](#)。ARN
4. 向ARN事件源提供加密队列的。

为制作者配置 AWS KMS 权限

当[数据密钥重用期](#)到期时，生产者下次调用SendMessage或SendMessageBatch也会触发对的调用kms:GenerateDataKey。生产者必须拥有KMS密钥的kms:GenerateDataKey权限。

在生产者的IAM政策中添加以下声明。请记住为密钥资源和队列资源使用正确的ARN值。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey"
    ],
    "Resource": "arn:aws:kms:us-east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  }, {
    "Effect": "Allow",
    "Action": [
      "sqs:SendMessage"
    ],
    "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
  }]
}
```

为消费者配置 AWS KMS 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，使用者必须拥有用于加密指定队列中消息的任何密KMS钥的kms:Decrypt权限。如果队列充当[死信队列](#)，则使用者还必须拥有用于加密源队列中消息的任何密KMS钥的kms:Decrypt权限。在消费者的IAM政策中添加以下声明。请记住为密钥资源和队列资源使用正确的ARN值。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:us-east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  }, {
    "Effect": "Allow",
    "Action": [
      "sqs:ReceiveMessage"
    ],
    "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
  }]
}
```

在混淆副手保护的情况下配置 AWS KMS 权限

当密钥策略语句中的主体为[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](#)。要使用这些条件密钥，请将值设置为正在加密的资源的 Amazon 资源名称 (ARN)。如果您不知道该 ARN 资源的类型，请 `aws:SourceAccount` 改用。

在此 KMS 关键政策中，允许账户拥有的服务中的特定资源 111122223333 在使用 Amazon 期间 KMS 进行调 SSE 用 Decrypt 和 GenerateDataKey 操作 SQS。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "<replaceable>service</replaceable>.amazonaws.com"
    },
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "*",
    "Condition": {
      "ArnEquals": {
```

```
    "aws:SourceArn": [  
      "arn:aws:service::111122223333:resource"  
    ]  
  }  
}  
]]  
}
```

使用SSE启用的 Amazon SQS 队列时，以下服务支持aws:SourceArn：

- 亚马逊 SNS
- Amazon S3
- CloudWatch 活动
- AWS Lambda
- CodeBuild
- Amazon Connect Customer Profiles
- AWS Auto Scaling
- Amazon Chime

了解数据密钥重用周期

[数据密钥重复使用期限](#)定义了 Amazon SQS 重复使用相同数据密钥的最长持续时间。当数据密钥重复使用期结束时，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。
- 目前，在SQS:SendMessage和SQS:SendMessageBatch操作期间仅使用该KMS:GenerateDataKey权限。该kms:Decrypt权限仅在SQS:ReceiveMessage操作期间使用。
- [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。

估算成本 AWS KMS

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

尽管以下公式可以让你很好地了解预期成本，但由于 Amazon 的分布式特性，实际成本可能会更高SQS。

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[委托人的](#)数量。

C是从 Amazon SQS 队列中接收的消费委托人的数量。

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 2 个主体和 1 个队列的 AWS KMS API呼叫数

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：计算多个生产者和使用者以及 2 个队列的 AWS KMS API 呼叫次数

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

当您与 Amazon SQS 合作时 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 中的互联网流量隐私 SQS

亚马逊虚拟私有云 (AmazonVPC) 终端节点 SQS 是其中的一个逻辑实体 VPC，它只允许连接到亚马逊 SQS。它们会将请求 VPC 路由到 Amazon SQS 并将响应路由回 VPC。以下各节提供有关使用 VPC 终端节点和创建 VPC 终端节点策略的信息。

主题

- [适用于亚马逊的亚马逊 Virtual Private Cloud 终端节点 SQS](#)
- [为亚马逊创建亚马逊 VPC 终端节点策略 SQS](#)

适用于亚马逊的亚马逊 Virtual Private Cloud 终端节点 SQS

如果您使用亚马逊 VPC 托管您的 AWS 资源，则可以在您 VPC 和亚马逊之间建立连接 SQS。您可以使用此连接将消息发送到您的 Amazon SQS 队列，而无需通过公共互联网。

Amazon VPC 允许您在自定义虚拟网络中启动 AWS 资源。您可以使用VPC来控制您的网络设置，例如 IP 地址范围、子网、路由表和网络网关。有关的更多信息VPCs，请参阅 [Amazon VPC 用户指南](#)。

要将您连接VPC到 AmazonSQS，您必须先定义一个接口VPC终端节点，这样您就可以将您的服务VPC连接到其他 AWS 服务。该终端节点SQS无需互联网网关、网络地址转换 (NAT) 实例或连接，即可提供与 Amazon 的可靠、可扩展的VPN连接。有关更多信息，请参阅本指南[示例 5：如果访问不是来自VPC终端节点，则拒绝访问](#)中的[教程：从 Amazon Virtual Private Cloud 将消息发送到 Amazon SQS 队列](#)和，以及亚马逊VPC用户指南中的[接口VPC终端节点 \(AWS PrivateLink\)](#)。

Important

- 您只能在亚马逊SQS终端节点上使用亚马逊 Virtual HTTPS Private Cloud。
- 当您将 Amazon 配置SQS为从亚马逊发送消息时VPC，必须启用私有功能DNS并按照格式指定终端节点`sqs.us-east-2.amazonaws.com`。
- Pri DNS vate 不支持旧版端点，例如`queue.amazonaws.com`或`us-east-2.queue.amazonaws.com`。

为亚马逊创建亚马逊VPC终端节点策略 SQS

您可以为亚马逊VPC终端节点创建策略，SQS在其中指定以下内容：

- 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终端节点控制对服务的访问](#)

以下示例VPC终端节点策略指定允许用户MyUser向 Amazon SQS 队列发送消息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`。
- 尝试使用此 VPC 端点的其他用户和规则。
- MyUser 向其他 Amazon SQS 队列发送消息。

Note

用户仍然可以在外部使用其他 Amazon SQS API 操作 VPC。有关更多信息，请参阅 [示例 5：如果访问不是来自 VPC 终端节点，则拒绝访问](#)。

Amazon 中的身份和访问管理 SQS

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）和授权（拥有权限）使用 Amazon SQS 资源。IAM 无需支付额外费用即可使用。AWS 服务

受众

您使用 AWS Identity and Access Management (IAM) 的方式会有所不同，具体取决于您在 Amazon 上所做的工作 SQS。

服务用户 — 如果您使用 Amazon SQS 服务完成工作，则您的管理员会为您提供所需的凭证和权限。当您使用更多的 Amazon SQS 功能来完成工作时，您可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Amazon 中的某项功能 SQS，请参阅 [Amazon Simple Queue Service 身份和访问权限故障排查](#)。

服务管理员 — 如果您负责公司的亚马逊 SQS 资源，则可能拥有对亚马逊的完全访问权限 SQS。您的工作是确定您的服务用户应该访问哪些亚马逊 SQS 功能和资源。然后，您必须向 IAM 管理员提交更改服务用户权限的请求。查看此页面上的信息以了解的基本概念 IAM。要详细了解贵公司如何 IAM 与 Amazon 搭配使用 SQS，请参阅 [Amazon 简单队列服务的工作原理 IAM](#)。

IAM 管理员 — 如果您是 IAM 管理员，则可能需要详细了解如何编写策略来管理对 Amazon 的访问权限 SQS。要查看您可以在中使用的 SQS 基于身份的 Amazon 策略示例 IAM，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 用户身份或通过担任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。AWS IAM Identity Center（IAM 身份中心）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员之前使用 IAM 角色设置了联合身份。当您使用联合访问 AWS 时，您就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》[中的如何登录到您 AWS 账户的](#)。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的[签署 AWS API 请求](#)。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如，AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅用户指南中的[多重身份验证](#)和 AWS IAM Identity Center 用户指南 AWS [中的使用多因素身份验证 \(MFA\)](#)。IAM

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以 root 用户身份登录的任务的完整列表，请参阅用户指南中的[需要根用户凭证的 IAM 任务](#)。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity Center 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。AWS Directory Service 当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户

和应用程序中使用。有关IAM身份中心的信息，请参阅[什么是IAM身份中心？](#)在《AWS IAM Identity Center 用户指南》中。

IAM 用户和组

[IAM用户](#)是您内部 AWS 账户 对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时证书，而不是创建拥有密码和访问密钥等长期凭证的IAM用户。但是，如果您有需要IAM用户长期凭证的特定用例，我们建议您轮换访问密钥。有关更多信息，请参阅《IAM用户指南》中的[针对需要长期凭证的用例定期轮换访问密钥](#)。

[IAM群组](#)是指定IAM用户集合的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可以拥有一个名为的群组，IAMAdmins并授予该群组管理IAM资源的权限。

用户与角色不同。用户唯一地与某个人或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《[IAM用户指南](#)》中的[何时创建IAM用户（而不是角色）](#)。

IAM角色

[IAM角色](#)是您内部具有特定权限 AWS 账户 的身份。它与IAM用户类似，但与特定人员无关。您可以AWS Management Console 通过[切换IAM角色在中临时扮演角色](#)。您可以通过调用 AWS CLI 或 AWS API操作或使用自定义操作来代入角色URL。有关使用角色的方法的更多信息，请参阅《IAM用户指南》中的[使用IAM角色](#)。

IAM具有临时证书的角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM用户指南》中的[为第三方身份提供商创建角色](#)。如果您使用 IAM Identity Center，则需要配置权限集。为了控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 会将权限集关联到中的IAM角色。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的[权限集](#)。
- 临时IAM用户权限-IAM 用户或角色可以代入一个IAM角色，为特定任务临时获得不同的权限。
- 跨账户访问-您可以使用IAM角色允许其他账户中的某人（受信任的委托人）访问您账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解角色和基于资源的跨账户访问策略之间的区别，请参阅IAM用户指南[IAM中的跨账户资源访问权限](#)。

- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序 EC2 或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
- 转发访问会话 (FAS)-当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限 AWS 服务以及 AWS 服务 向下游服务发出请求的请求。FAS 只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出请求。在这种情况下，您必须具有执行这两个操作的权限。有关提出 FAS 请求时的政策详情，请参阅[转发访问会话](#)。
- 服务角色-服务 [IAM 角色](#) 是服务代替您执行操作的角色。IAM 管理员可以在内部创建、修改和删除服务角色 IAM。有关更多信息，请参阅《IAM 用户指南》AWS 服务中的[创建角色以向委派权限](#)。
- 服务相关角色-服务相关角色是一种链接到的服务角色。AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 Amazon 上运行的应用程序 EC2 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这比在 EC2 实例中存储访问密钥更可取。要为 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含该角色，并允许在 EC2 实例上运行的程序获得临时证书。有关更多信息，请参阅 IAM 用户指南中的[使用 IAM 角色向在 Amazon EC2 实例上运行的应用程序授予权限](#)。

要了解是使用 IAM 角色还是使用 IAM 用户，请参阅 [《用户指南》中的何时创建 IAM 角色（而不是 IAM 用户）](#)。

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。策略是其中的一个对象 AWS，当与身份或资源关联时，它会定义其权限。AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档结构和内容的更多信息，请参阅 [《IAM 用户指南》中的 JSON 策略概述](#)。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。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。

默认情况下，用户和角色没有权限。要授予用户对其所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。然后，管理员可以将 IAM 策略添加到角色中，用户可以代入这些角色。

IAM无论您使用何种方法执行操作，策略都会定义该操作的权限。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或获取角色信息 AWS API。

基于身份的策略

基于身份的策略是可以附加到身份（例如IAM用户、用户组或角色）的JSON权限策略文档。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[创建IAM策略](#)。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管策略或内联策略之间进行选择，请参阅《IAM用户指南》中的在[托管策略和内联策略之间进行选择](#)。

基于资源的策略

基于资源的JSON策略是您附加到资源的策略文档。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。

基于资源的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IAM中使用 AWS 托管策略。

访问控制列表 (ACLs)

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。ACLs与基于资源的策略类似，尽管它们不使用JSON策略文档格式。

Amazon S3 和 Amazon VPC 就是支持的服务示例ACLs。AWS WAF要了解更多信息ACLs，请参阅《亚马逊简单存储服务开发者指南》中的[访问控制列表 \(ACL\) 概述](#)。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界-权限边界是一项高级功能，您可以在其中设置基于身份的策略可以向IAM实体（IAM用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 Principal 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中

的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM用户指南》中的[IAM实体的权限边界](#)。

- 服务控制策略 (SCPs)-SCPs 是为中的组织或组织单位 (OU) 指定最大权限的JSON策略 AWS Organizations。AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有账户。对成员账户中的实体（包括每个实体）的权限进行了SCP限制 AWS 账户根用户。有关 Organization SCPs s 和的更多信息，请参阅《AWS Organizations 用户指南》中的[服务控制策略](#)。
- 会话策略 – 会话策略是当您以编程方式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。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。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。要了解在涉及多种策略类型时如何 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](#)。

在 Amazon 中管理访问权限概述 SQS

每个 AWS 资源都归人所有 AWS 账户，创建或访问资源的权限受权限策略的约束。账户管理员可以将权限策略附加到IAM身份（用户、群组和角色），某些服务（例如 AmazonSQS）也支持将权限策略附加到资源。

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最佳实践](#)。

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。

主题

- [Amazon Simple Queue Service 资源和操作](#)
- [了解资源所有权](#)
- [管理对资源的访问](#)
- [指定策略元素：操作、效果、资源和主体](#)

Amazon Simple Queue Service 资源和操作

在 Amazon 中 SQS，唯一的资源是队列。在策略中，使用 Amazon 资源名称 (ARN) 来标识该政策适用的资源。以下资源具有与之 ARN 关联的唯一资源：

资源类型	ARN 格式
队列	<code>arn:aws:sqs: <i>region</i>:<i>account_id</i>:<i>queue_name</i></code>

以下是队列 ARN 格式的示例：

- ARN 对于在美国东部 (俄亥俄州) my_queue 地区命名的队列，属于 AWS 账户 123456789012：

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- ARN 对于 my_queue 在 Amazon SQS 支持的每个不同区域中命名的队列：

```
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 匹配所有前缀为 : 的 my_prefix_ 队列

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

您可以通过调用 [GetQueueAttributes](#) 操作来获取现有队列的 ARN 值。该 QueueArn 属性的值是队列 ARN 的。有关的更多信息 ARNs，请参阅《IAM 用户指南》[IAM ARNs](#) 中的。

Amazon SQS 提供了一组与队列资源配合使用的操作。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

AWS 账户 拥有在账户中创建的资源，无论谁创建了这些资源。具体而言，资源所有者是 AWS 账户对资源创建请求进行身份验证的委托人实体 (即根账户、用户或 IAM 角色)。以下示例说明了它的工作原理：

- 如果您使用您的 AWS 账户 根账户证书创建亚马逊 SQS 队列，则您的 AWS 账户 就是该资源的所有者 (在亚马逊中 SQS，资源是亚马逊队 SQS 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。
- 如果您在中创建 AWS 账户 具有创建 Amazon SQS 队列权限的IAM角色，则任何能够担任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IAM在 Amazon 环境中使用SQS。它不提供有关IAM服务的详细信息。有关完整IAM文档，请参阅[什么是IAM？](#)在《IAM用户指南》中。有关IAM策略语法和描述的信息，请参阅IAM用户指南中的[AWS IAM策略参考](#)。

附加到IAM身份的策略称为基于身份的策略（IAM策略），附加到资源的策略称为基于资源的策略。

基于身份的策略

向您的用户授予访问您的 Amazon SQS 队列权限的方法有两种：使用亚马逊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@@ 权限策略附加到您账户中的用户或群组-要向用户授予创建 Amazon SQS 队列的权限，请将权限策略附加到该用户所属的用户或群组。
- 将@@ 权限策略附加到其他用户中 AWS 账户-要向用户授予创建亚马逊SQS队列的权限，请将亚马逊SQS权限策略附加到其他用户中 AWS 账户。

跨账户权限不能应用于以下操作：

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)

- [ListQueueTags](#)
 - [RemovePermission](#)
 - [SetQueueAttributes](#)
 - [StartMessageMoveTask](#)
 - [TagQueue](#)
 - [UntagQueue](#)
- 向角色附加权限策略 (授予跨账户权限) -要授予跨账户权限，请将基于身份的权限策略附加到角色。IAM例如，AWS 账户 A 管理员可以创建一个角色来向 AWS 账户 B (或 AWS 服务) 授予跨账户权限，如下所示：
 - 账户 A 管理员创建一个IAM角色并向该角色附加权限策略 (该策略授予账户 A 中资源的权限)。
 - 账户 A 管理员向将账户 B 标识为能够代入该角色的主体的角色附加信任策略。
 - 账户 B 管理员向账户 B 中的任何用户委派代入该角色的权限。这将允许账户 B 中的用户创建或访问账户 A 中的队列。

Note

如果要向 AWS 服务授予担任该角色的权限，则信任策略中的委托人也可以是 AWS 服务委托人。

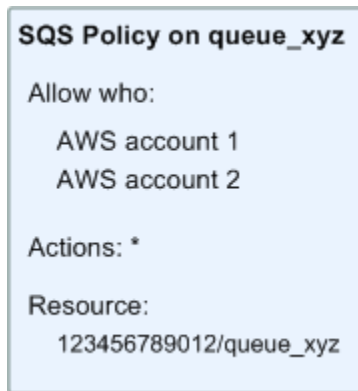
有关使用委派权限IAM的更多信息，请参阅《IAM用户指南》中的[访问管理](#)。

虽然 Amazon SQS 使用IAM策略，但它有自己的策略基础架构。您可以使用带有队列的 Amazon SQS 策略来指定哪些 AWS 账户有权访问队列。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)。您可以授予权限的具体操作是 Amazon SQS 操作总列表的子集。当您编写亚马逊SQS政策并指定“*允许所有亚马逊SQS操作”时，这意味着用户可以执行此子集中的所有操作。

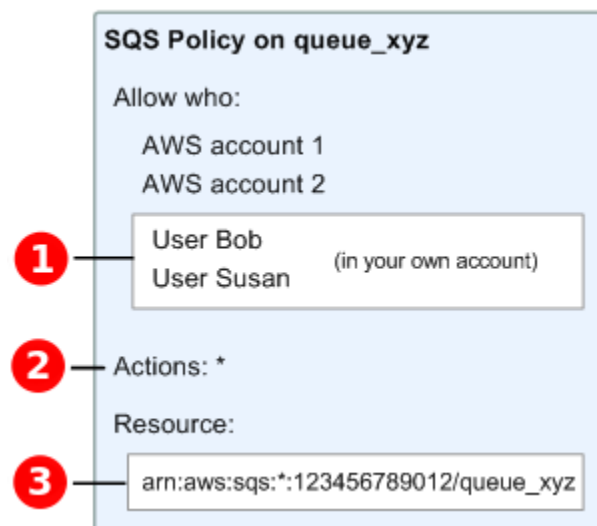
下图说明了其中一项基本的Amazon SQS 政策的概念，该政策涵盖了操作的子集。该策略适用于queue_xyz，并授予 AWS 账户 1 和 AWS 账户 2 在指定队列中使用任何允许的操作的权限。

Note

策略中的资源指定为123456789012/queue_xyz，其中123456789012是拥有队列的账户的账户 ID。AWS



随着用户IAM和 Amazon 资源名称 (ARNs) 的引入和概念，SQS政策发生了一些变化。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角色委派跨 AWS 账户访问权限](#)。

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) 来指定资源，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 策略，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。

在引入之前 IAM，Amazon SQS 会自动让队列的创建者完全控制队列（即访问该队列上所有可能的亚马逊 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。任何有权创建队列的用户还必须有权使用其他 Amazon SQS 操作才能对创建的队列执行任何操作。

以下示例策略允许用户使用所有 Amazon SQS 操作，但仅限于名称以文字字符串为前缀的队列。bob_queue_

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
  }]
}
```

有关更多信息在 [Amazon 上使用策略 SQS](#)，请参阅用户指南中的和 [身份（用户、组和角色）](#)。IAM

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](#)。为了授予这些操作的权限，Amazon SQS 定义了一组您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：

- 资源-在策略中，您可以使用 Amazon 资源名称 (ARN) 来标识该政策适用的资源。
- 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。
- 委托人 — 在基于身份的策略（IAM策略）中，策略所关联的用户是隐式委托人。对于基于资源的策略，您可以指定要接收权限的用户、帐户、服务或其他实体（仅适用于基于资源的策略）。

要了解有关 Amazon SQS 政策语法和描述的更多信息，请参阅IAM用户指南中的[AWS IAM政策参考](#)。

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 简单队列服务的工作原理 IAM

在使用IAM管理亚马逊访问权限之前SQS，请先了解亚马逊有哪些IAM功能可供使用SQS。

IAM您可以在 Amazon 简单队列服务中使用的功能

IAM功能	亚马逊SQS支持
基于身份的策略	是
基于资源的策略	是
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACLs	不支持
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
服务相关角色	否

要全面了解 Amazon SQS 和其他 AWS 服务如何使用大多数 IAM 功能，请参阅《IAM 用户指南》IAM 中 [与之配合使用的 AWS 服务](#)。

访问控制

访问控制列表 (ACLs) 控制哪些委托人 (账户成员、用户或角色) 有权访问资源。ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持的服务示例 ACLs。AWS WAF 要了解更多信息 ACLs，请参阅《亚马逊简单存储服务开发者指南》中的 [访问控制列表 \(ACL\) 概述](#)。

Note

重要的是要明白，所有人 AWS 账户 都可以将其权限委托给其账户下的用户。跨账户访问允许您共享对 AWS 资源的访问权限，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 [启用跨账户访问权限](#)。

[Amazon SQS 自定义政策的局限性](#) 有关 Amazon SQS 自定义政策中跨内容权限和条件密钥的更多详细信息，请参阅。

亚马逊基于身份的政策 SQS

支持基于身份的策略：是

基于身份的策略是可以附加到身份 (例如 IAM 用户、用户组或角色) 的 JSON 权限策略文档。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 [创建 IAM 策略](#)。

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。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可以在 JSON 策略中使用的所有元素，请参阅 IAM 用户指南中的 [IAM JSON 策略元素参考](#)。

Amazon 基于身份的政策示例 SQS

要查看 Amazon SQS 基于身份的政策示例，请参阅 [策略最佳实践](#)

Amazon 内部基于资源的政策 SQS

支持基于资源的策略：是

基于资源的JSON策略是您附加到资源的策略文档。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。

要启用跨账户访问权限，您可以将整个账户或另一个账户中的IAM实体指定为基于资源的策略中的委托人。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时AWS账户，可信账户中的IAM管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM用户指南》IAM中的[跨账户资源访问权限](#)。

针对亚马逊的政策行动 SQS

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什么。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Action元素描述了可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的AWS API操作同名。也有一些例外，例如没有匹配API操作的仅限权限的操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看亚马逊SQS操作列表，请参阅《服务授权参考》中的“亚马逊简单队列服务[定义的资源](#)”。

Amazon 中的策略操作在操作前SQS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 基于身份的政策示例，请参阅 [策略最佳实践](#)

Amazon 的政策资源 SQS

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什么。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。

ResourceJSON策略元素指定要应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。最佳做法是，使用资源的 [Amazon 资源名称 \(ARN\)](#) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

要查看亚马逊SQS资源类型及其列表ARNs，请参阅《服务授权参考》中的“[亚马逊简单队列服务定义的操作](#)”。要了解您可以使用哪些操作来指定每ARN种资源，请参阅 [Amazon 简单队列服务定义的资源](#)。

要查看 Amazon SQS 基于身份的政策示例，请参阅 [策略最佳实践](#)

Amazon 的政策条件密钥 SQS

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什么。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。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。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。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在资源上标有IAM用户的用户名时，您才能向IAM用户授予访问该资源的权限。有关更多信息，请参阅《IAM用户指南》中的[IAM策略元素：变量和标签](#)。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅《IAM用户指南》中的[AWS 全局条件上下文密钥](#)。

要查看亚马逊SQS条件密钥列表，请参阅服务授权参考中的亚马逊简单队列服务的[条件密钥](#)。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 基于身份的政策示例，请参阅。[策略最佳实践](#)

ACLs在亚马逊 SQS

支持ACLs：否

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。ACLs与基于资源的策略类似，尽管它们不使用JSON策略文档格式。

ABAC与亚马逊合作 SQS

支持ABAC（策略中的标签）：部分

基于属性的访问控制 (ABAC) 是一种基于属性定义权限的授权策略。在中 AWS，这些属性称为标签。您可以将标签附加到IAM实体（用户或角色）和许多 AWS 资源。为实体和资源添加标签是的第一步。ABAC然后，您可以设计ABAC策略，允许在委托人的标签与他们尝试访问的资源上的标签匹配时进行操作。

ABAC在快速增长的环境中很有用，也有助于解决策略管理变得繁琐的情况。

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。

有关的更多信息ABAC，请参阅[什么是ABAC？](#) 在《IAM用户指南》中。要查看包含设置步骤的教程ABAC，请参阅IAM用户指南中的[使用基于属性的访问控制 \(ABAC\)](#)。

在 Amazon 上使用临时证书 SQS

支持临时凭证：是

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关其他信息，包括哪些 AWS 服务 适用于临时证书 [AWS 服务](#)，请参阅《IAM用户指南》IAM中的“适用于临时证书”。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM用户指南》中的[切换到角色 \(控制台\)](#)。

您可以使用 AWS CLI 或手动创建临时证书 AWS API。然后，您可以使用这些临时证书进行访问 AWS。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[中的临时安全证书IAM](#)。

Amazon 的转发访问会话 SQS

支持转发访问会话 (FAS)：是

当您使用IAM用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS使用调用委托人的权限 AWS 服务以及 AWS 服务 向下游服务发出请求的请求。FAS只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出请求。在这种情况下，您必须具有执行这两个操作的权限。有关提出FAS请求时的政策详情，请参阅[转发访问会话](#)。

Amazon 的服务角色 SQS

支持服务角色：是

服务IAM角色是服务代替您执行操作的角色。IAM管理员可以在内部创建、修改和删除服务角色IAM。有关更多信息，请参阅《IAM用户指南》AWS 服务中的[创建角色以向委派权限](#)。

Warning

更改服务角色的权限可能会中断 Amazon 的SQS功能。仅当 Amazon SQS 提供相关指导时，才可编辑服务角色。

Amazon 的服务相关角色 SQS

支持服务相关角色：否

服务相关角色是一种与服务相关联的 AWS 服务服务角色。服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户，并且归服务所有。IAM管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅与之[配合IAM使用的AWS 服务](#)。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

亚马逊SQS更新了托管 AWS 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 托管策略更简单。[创建IAM客户托管策略](#)以仅向您的团队提供他们所需的权限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 托管策略添加其他权限以支持新功能。此类更新会影响附加策略的所有身份（用户、组和角色）。当推出新功能或有新操作可用时，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，还 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。当服务启动一项新功能时，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[工作职能AWS 托管策略](#)。

AWS 托管策略：AmazonSQSFull 访问权限

您可以将该AmazonSQSFullAccess政策附加到您的 Amazon SQS 身份。此政策授予允许对 Amazon 进行完全访问的权限SQS。

要查看此策略的权限，请参阅《AWS 托管策略参考》中的 [AmazonSQSFull 策略](#)。

AWS 托管策略：AmazonSQSRead OnlyAccess

您可以将该AmazonSQSReadOnlyAccess政策附加到您的 Amazon SQS 身份。此政策授予允许对 Amazon 进行只读访问的权限SQS。

要查看此策略的权限，请参阅《AWS 托管策略参考》AmazonSQSReadOnlyAccess中的 [策略](#)。

亚马逊SQS更新了托管 AWS 策略

查看SQS自该服务开始跟踪这些变更以来亚马逊 AWS 托管策略更新的详细信息。要获取有关此页面更新的自动提醒，请在 Amazon SQS [文档历史记录](#)页面上订阅 RSS Feed。

更改	描述	日期
AmazonSQSReadOnlyAccess	亚马逊SQS添加了该ListQueueTags 操作，该操作将检索与指定亚马逊SQS队列关联的所有标签。它允许您查看出于组织或元数据目的分配给队列的键值对。此操作与该ListQueueTags API操作关联。	2024 年 6 月 20 日
AmazonSQSReadOnlyAccess	Amazon SQS 添加了一项新操作，允许您在特定源队列下列出最新的消息移动任务（最多 10 个）。此操作与该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

使用以下信息来帮助您诊断和修复与Amazon合作时可能遇到的常见问题，SQS以及IAM。

主题

- [我无权在 Amazon 上执行任何操作 SQS](#)
- [我无权执行 iam : PassRole](#)
- [我想允许我以外的人访问我的 AWS 账户 Amazon SQS 资源](#)

我无权在 Amazon 上执行任何操作 SQS

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。

如果您需要帮助，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

如果您收到错误消息，说您无权执行该 `iam:PassRole` 操作，则必须更新您的政策，以允许您将角色传递给亚马逊 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。

当名为的 IAM 用户 `marymajor` 尝试使用控制台在 Amazon 中执行操作时，会出现以下示例错误 SQS。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。

如果您需要帮助，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访问我的 AWS 账户 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，可以担任角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon 是否 SQS 支持这些功能，请参阅 [Amazon 简单队列服务的工作原理 IAM](#)。
- 要了解如何提供对您拥有的资源的 [访问权限](#)，请参阅《IAM 用户指南》中的 [AWS 账户 向其他 IAM 用户提供访问权限](#)。AWS 账户
- 要了解如何向第三方提供对您的资源的 [访问权限 AWS 账户](#)，请参阅 IAM 用户指南中的 [向第三方提供访问权限](#)。AWS 账户

- 要了解如何通过联合身份验证提供访问权限，请参阅《用户指南》中的[向经过外部身份验证的用户提供访问权限（联合身份验证）](#)。IAM
- 要了解使用角色和基于资源的策略进行跨账户访问的区别，请参阅IAM用户指南[IAM中的跨账户资源访问权限](#)。

在 Amazon 上使用政策 SQS

本主题提供了基于身份的策略示例，在这些策略中，账户管理员可以将权限策略附加到IAM身份（用户、群组和角色）。

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 [在 Amazon 中管理访问权限概述 SQS](#)。

除外ListQueues，所有 Amazon SQS 操作都支持资源级权限。有关更多信息，请参阅 [Amazon SQS API 权限：操作和资源参考](#)。

主题

- [使用亚马逊SQS和IAM政策](#)
- [使用 Amazon SQS 控制台所需的权限](#)
- [Amazon 基于身份的政策示例 SQS](#)
- [亚马逊SQS政策的基本示例](#)
- [使用带有 Amazon SQS 访问策略语言的自定义策略](#)

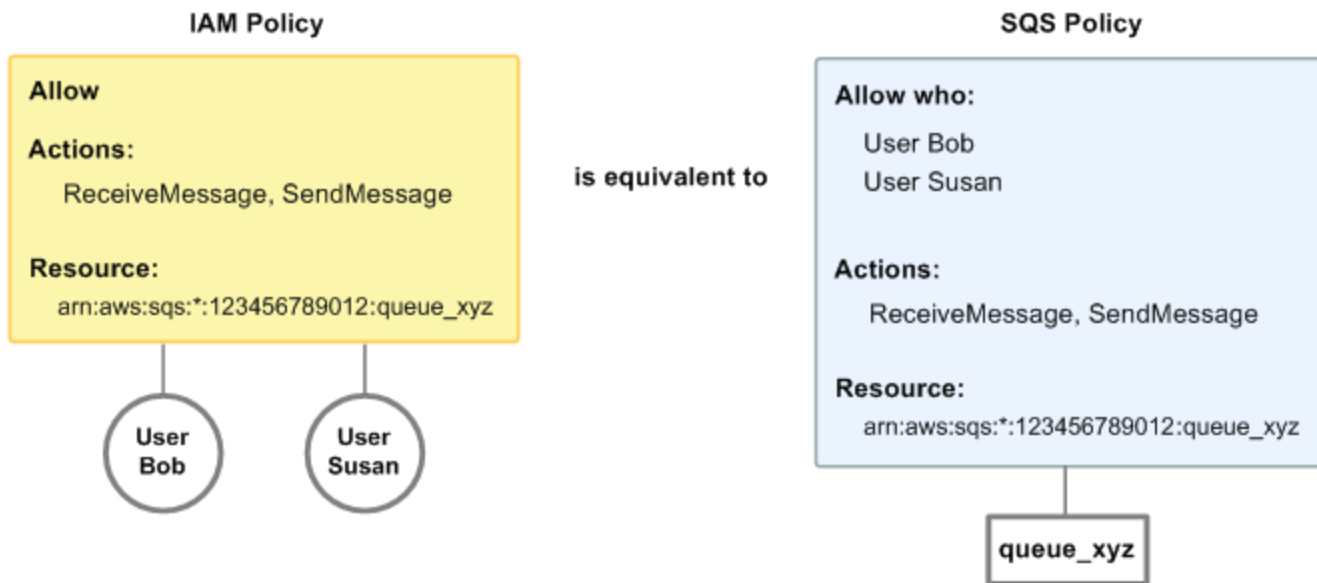
使用亚马逊SQS和IAM政策

您可以通过两种方式向您的用户授予对您的亚马逊SQS资源的访问权限：使用亚马逊SQS策略系统和使用IAM策略系统。您可以使用其中任意一套或两套系统。在绝大部分情况下，无论采用上述哪种方式，都可以得到同样的结果。

例如，下图显示了一项IAM策略和与之相当的 Amazon SQS 政策。该IAM政策向亚马逊授予权限SQSReceiveMessage和对您的 AWS 账户queue_xyz中调用的队列进行SendMessage操作，并且该政策附加到名为 Bob 和 Susan 的用户（Bob 和 Susan 拥有政策中规定的权限）。亚马逊的这项SQS政策还赋予了 Bob 和 Susan 对同一个队列SendMessage执行ReceiveMessage和操作的权利。

Note

以下示例显示了没有条件的简单策略。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与亚马逊政策之间有一个主要区别：亚马逊SQSSQS政策系统允许您向其他账户授予权限，但IAM不允许您向其他AWS账户授予权限。

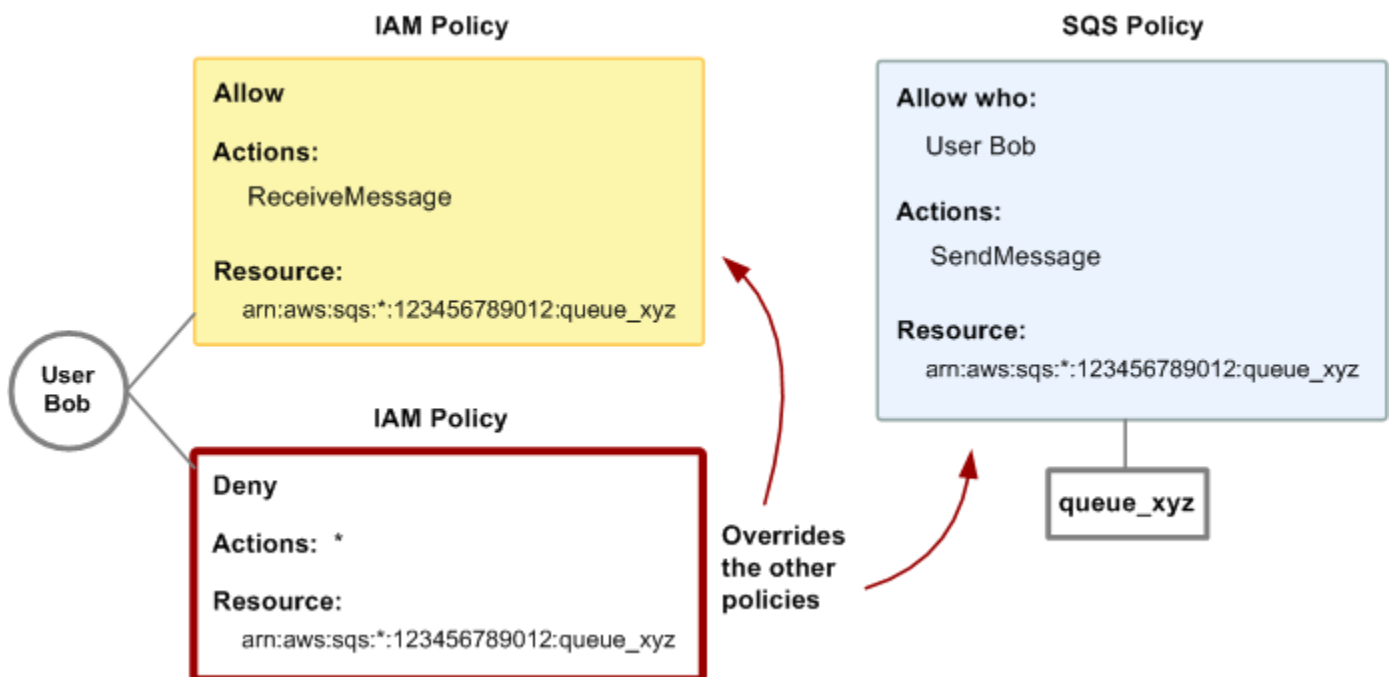
您可以自行决定如何综合使用上述两种系统来管理您的权限。以下示例展示这两种策略系统是如何共同运行的。

- 在第一个示例中，Bob 拥有适用于其账户的IAMSQS政策和 Amazon 政策。该IAM政策向他的账户授予ReceiveMessage操作权限queue_xyz，而亚马逊SQS政策则允许他的账户在同一队列上SendMessage执行操作。下图阐明了这一概念。



如果 Bob 向发送 `ReceiveMessage` 请求 `queue_xyz`，则 IAM 策略允许该操作。如果 Bob 向发送 `SendMessage` 请求 `queue_xyz`，则 Amazon SQS 政策允许该操作。

- 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 [使用带有 Amazon SQS 访问策略语言的自定义策略](#)。下图阐明了这一概念。



您还可以在 Amazon SQS 政策中添加其他声明，拒绝 Bob 对队列进行任何类型的访问。它与添加拒绝 Bob 访问队列的 IAM 策略具有相同的效果。有关涵盖 Amazon SQS 操作和资源的政策示例，请参

阅[亚马逊SQS政策的基本示例](#)。有关编写 Amazon SQS 政策的更多信息，请参阅[使用带有 Amazon SQS 访问策略语言的自定义策略](#)。

使用 Amazon SQS 控制台所需的权限

想要使用亚马逊SQS控制台的用户必须拥有使用该用户中的亚马逊SQS队列的最低权限 AWS 账户。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。除了亚马逊SQS权限外，要为亚马逊SQS队列订阅亚马逊SNS主题，控制台还需要亚马逊SNS操作的权限。

如果您创建的IAM策略比所需的最低权限更严格，则控制台可能无法按预期运行，供拥有该IAM策略的用户使用。

对于仅调用 AWS CLI 或 Amazon SQS 操作的用户，您无需为其设置最低控制台权限。

Amazon 基于身份的政策示例 SQS

默认情况下，用户和角色无权创建或修改 Amazon SQS 资源。他们也无法使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或来执行任务 AWS API。要授予用户对其所需资源执行操作的权限，IAM管理员可以创建IAM策略。然后，管理员可以将IAM策略添加到角色中，用户可以代入这些角色。

要了解如何使用这些示例策略文档创建IAM基于身份的JSON策略，请参阅IAM用户指南中的[创建IAM策略](#)。

有关亚马逊定义的操作和资源类型（包括每种资源类型的格式）的详细信息SQS，请参阅《服务授权参考》中的[Amazon Simple Queue 服务的操作、资源和条件密钥](#)。ARNs

Note

当您为 Amazon A EC2 uto Scaling 配置生命周期挂钩时，您无需编写策略即可向亚马逊SQS 队列发送消息。有关更多信息，请参阅《[亚马逊EC2用户指南](#)》中的 [Amazon A EC2 uto Scaling 生命周期挂钩](#)。

主题

- [策略最佳实践](#)
- [使用亚马逊SQS控制台](#)
- [允许用户查看他们自己的权限](#)

- [允许用户创建队列](#)
- [允许开发者向共享队列写消息](#)
- [允许管理员获取队列的大致大小](#)
- [允许合作伙伴向特定队列发送消息](#)

策略最佳实践

基于身份的策略决定了是否有人可以在您的账户中创建、访问或删除亚马逊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予权限，请使用为许多常见用例授予权限的AWS托管策略。它们在你的版本中可用AWS账户。我们建议您通过定义针对您的用例的AWS客户托管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托管策略](#)。
- 应用最低权限权限-使用IAM策略设置权限时，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。有关使用应用权限IAM的更多信息，请参阅IAM用户指南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-您可以在策略中添加条件以限制对操作和资源的访问权限。例如，您可以编写一个策略条件来指定所有请求都必须使用发送SSL。如果服务操作是通过特定的方式使用的，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AWS CloudFormation。有关更多信息，请参阅《IAM用户指南》中的[IAMJSON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 会验证新的和现有的策略，以便策略符合IAM策略语言 (JSON) 和IAM最佳实践。IAM Access Analyzer 提供了 100 多项策略检查和可行的建议，可帮助您制定安全和实用的策略。有关更多信息，请参阅《IAM用户指南》中的 [IAM Access Analyzer 策略验证](#)。
- 需要多重身份验证 (MFA)-如果您的场景需要IAM用户或 root 用户 AWS 账户，请打开MFA以提高安全性。要要求MFA何时调用API操作，请在策略中添加MFA条件。有关更多信息，请参阅《IAM用户指南》中的[配置MFA受保护的API访问权限](#)。

有关最佳做法的更多信息IAM，请参阅《IAM用户指南》IAM中的[安全最佳实践](#)。

使用亚马逊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列出和查看有关您的 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。

您无需为仅拨打 AWS CLI 或的用户设置最低控制台权限 AWS API。相反，只允许访问与他们尝试执行的API操作相匹配的操作。

为确保用户和角色仍然可以使用亚马逊SQS控制台，还需要将亚马逊SQSAmazonSQSReadOnlyAccess AWS 托管策略附加到这些实体。有关更多信息，请参阅《[用户指南](#)》中的[向IAM用户添加权限](#)。

允许用户查看他们自己的权限

此示例说明如何创建允许IAM用户查看附加到其用户身份的内联和托管策略的策略。此策略包括在控制台上或使用或以编程方式完成此操作的 AWS CLI 权限。AWS API

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupForUser",
        "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 创建了一个策略，允许他访问所有 Amazon SQS 操作，但只能访问名称以文字字符串为前缀的队列。alice_queue_

Amazon SQS 不会自动授予队列创建者使用队列的权限。因此，除了 IAM 策略中的操作外，我们还必须明确授予 Bob 使用所有 Amazon SQS CreateQueue 操作的权限。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
  }]
}
```

允许开发者向共享队列写消息

在以下示例中，我们为开发者创建了一个群组，并附加了一个策略，允许该群组使用 Amazon SQS SendMessage 操作，但只能使用属于指定 AWS 账户 且已命名的队列 MyCompanyQueue。

```
{
  "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、ReceiveMessage 和 SendMessage。

Note

尽管 * 包括其他权限类型提供的访问权限，但 Amazon 会单独 SQS 考虑权限。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。

允许管理员获取队列的大致大小

在以下示例中，我们为经理创建了一个群组，并附加了一个策略，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。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:GetQueueAttributes",
    "Resource": "*"
  }]
}
```

允许合作伙伴向特定队列发送消息

您可以使用 Amazon SQS 策略或政策来完成此任务。IAM 如果您的合作伙伴有 AWS 账户，则使用 Amazon SQS 策略可能会更容易。但是，合作伙伴公司中任何拥有 AWS 安全凭证的用户都可以向队列发送消息。如果您想限制特定用户或应用程序的访问权限，则必须像对待自己公司的用户一样对待合作伙伴，并使用 IAM 政策代替 Amazon SQS 政策。

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小组。
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。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
  }]
}
```

亚马逊SQS政策的基本示例

本节显示了常见的 Amazon SQS 用例的策略示例。

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 Management Console 使用用户的证书登录以验证权限。

示例 1：向一个人授予一个权限 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）444455556666/queue1地区命名的队列的SendMessage权限。

```
{
  "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 编号111122223333SendMessage和ReceiveMessage权限444455556666/queue1。

```
{
  "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个不同的 AWS 账户 数字 (111122223333和444455556666) 使用亚马逊SQS 允许共享访问美国东部 (俄亥俄州) 123456789012/queue1地区名为队列的所有操作的权限。

```
{
  "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 AWS 账户 账户使用亚马逊SQS允许在美国东部 (俄亥俄州) 地区共享访问的所有操作的权限。username1 123456789012/queue1

跨账户权限不能应用于以下操作：

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

```

{
  "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。

```
{
  "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。

```
{
  "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

以下示例策略授予所有用户（匿名用户）使用所有可能的 Amazon SQS 操作的权限，这些操作可以共享给名为的队列111122223333/queue1，但前提是请求来自该192.0.2.0/24CIDR范围。

```

{
  "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：不同范围内的用户的白名单和黑名单权限 CIDR

以下策略示例具有两项陈述：

- 第一条语句授予192.0.2.0/24CIDR范围内的所有用户（匿名用户）（除外192.0.2.188）使用名为 111122223333 /queue1 的队列的SendMessage操作的权限。
- 第二条语句阻止12.148.72.0/23CIDR范围内的所有用户（匿名用户）使用队列。

```

{
  "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"
    }
  }
}
}]
}
```

使用带有 Amazon SQS 访问策略语言的自定义策略

如果您只想允许基于 AWS 账户身份和基本权限（例如 for [SendMessage](#) 或 [ReceiveMessage](#)）的 Amazon SQS 访问权限，则无需编写自己的政策。您可以直接使用 Amazon SQS [AddPermission](#) 操作。

如果您想根据更具体的条件（例如请求收到的时间或请求者的 IP 地址）明确拒绝或允许访问，则需要编写自己的亚马逊SQS政策，并使用亚马逊SQS [SetQueueAttributes](#) 操作将其上传到 AWS 系统。

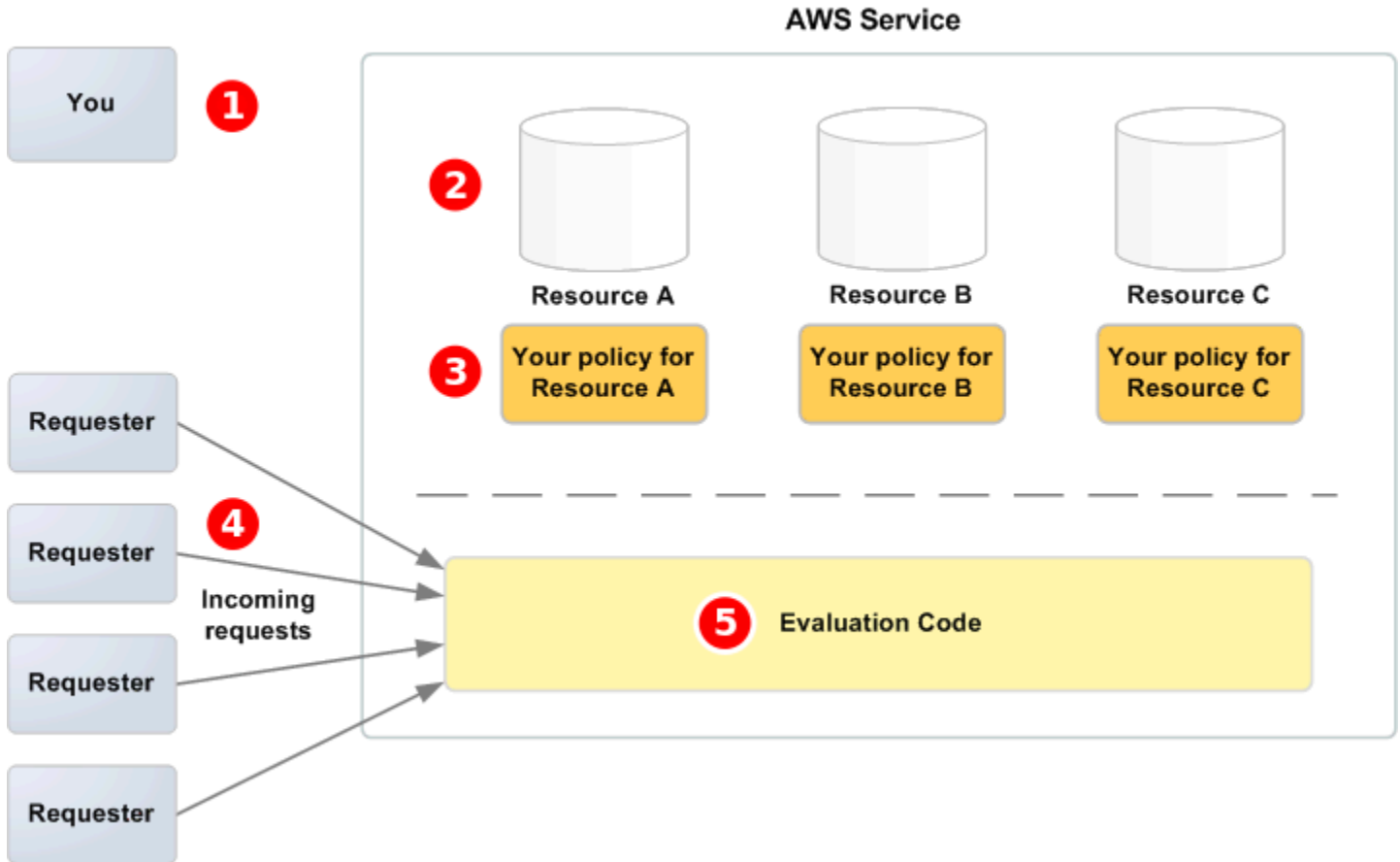
主题

- [亚马逊SQS访问控制架构](#)
- [Amazon SQS 访问控制流程工作流程](#)
- [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 访问策略语言中明确拒绝和默认拒绝之间的关系](#)
- [Amazon SQS 自定义政策的局限性](#)

- [自定义 Amazon SQS 访问策略语言示例](#)

亚马逊SQS访问控制架构

下图描述了您的 Amazon SQS 资源的访问控制。



1
您，资源所有者。

2
的资源包含在 AWS 服务中（例如，Amazon 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 服务提供给API您用来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。

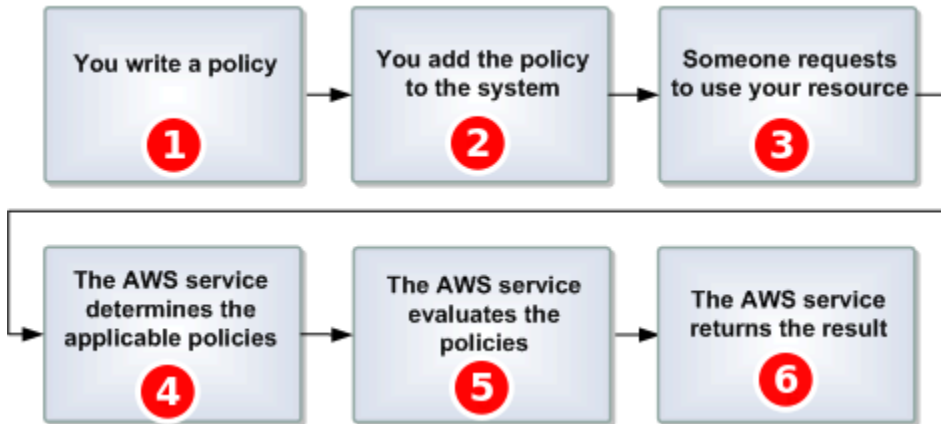
您

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 访问控制流程工作流程

下图描述了使用 Amazon 访问策略语言进行 SQS 访问控制的一般工作流程。

**1**

为队列编写了 Amazon SQS 政策。

您

2

将保单上传到 AWS。该 AWS 服务提供了一个 API，供您用来上传您的政策。例如，您可以使用 Amazon SQS `SetQueueAttributes` 操作上传针对特定亚马逊 SQS 队列的策略。

您

3

有人发送了使用您的 Amazon SQS 队列的请求。

有

4

亚马逊 SQS 会检查所有可用的亚马逊 SQS 政策，并确定哪些政策适用。

亚

5

SQS 会评估政策并确定是否允许请求者使用您的队列。

Amazo

6

据政策评估结果，Amazon SQS 要么向申请者返回 `Access denied` 错误，要么继续处理请求。

根

Amazon SQS 访问策略语言关键概念

要编写自己的策略，您必须熟悉[JSON](#)并了解一些关键概念。

允许

将[Statement](#)设为 [效果](#) 时allow的结果。

操作

[主体](#) 拥有权限可以执行的活动，通常是对 AWS的请求。

默认拒绝

没有 [允许](#) 或 [显式拒绝](#) 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[Policy](#)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：[默认拒绝](#)、[允许](#)和[显式拒绝](#)。

显式拒绝

将[Statement](#)设为 [效果](#) 时deny的结果。

评估

Amazon SQS 用于根据以下条件确定是否应拒绝或允许传入请求的流程[Policy](#)。

发布者

编写[Policy](#)以对资源授予权限的用户。顾名思义，发行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。

密钥

设置访问限制指定的特性。

许可

使用[Condition](#)和 [密钥](#)允许或拒绝对资源的访问的概念。

Policy

充当一条或多条[语句](#)容器的文档。



Amazon SQS 使用该政策来确定是否向用户授予资源访问权限。

主体

收到[许可](#)中[Policy](#)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 [Policy](#) 文档的一部分。

请求者

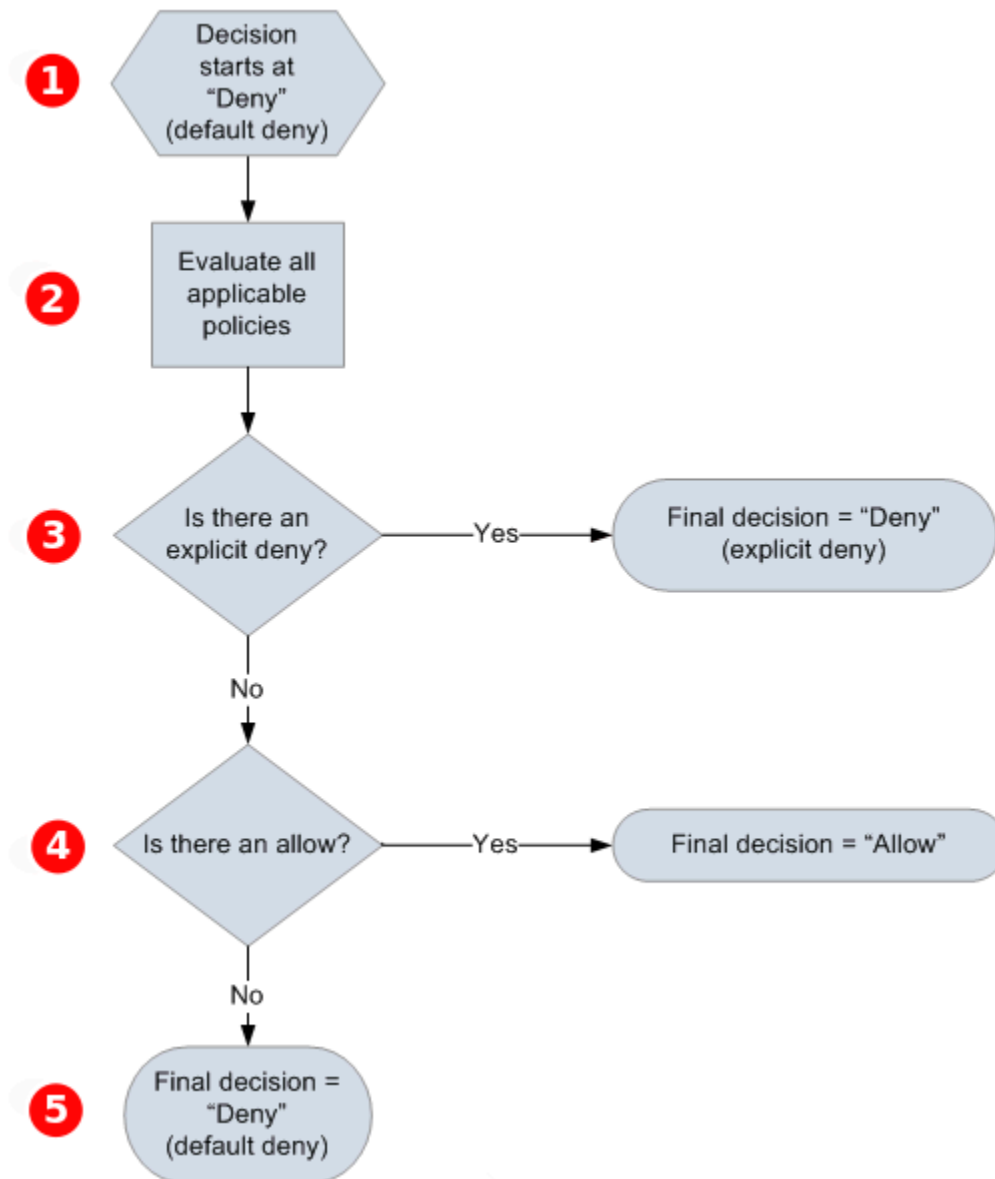
发送访问[资源](#)请求的用户。

Amazon SQS 访问策略语言评估逻辑

在评估时，Amazon 会SQS决定是否应允许或拒绝资源所有者以外的其他人的请求。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[默认拒绝](#)。
- [显式拒绝](#) 将覆盖任意允许。
- 策略评估的顺序并不重要。

下图详细描述了 Amazon 如何SQS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）。

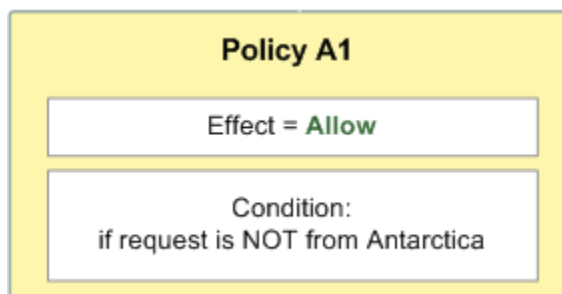
5

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）。

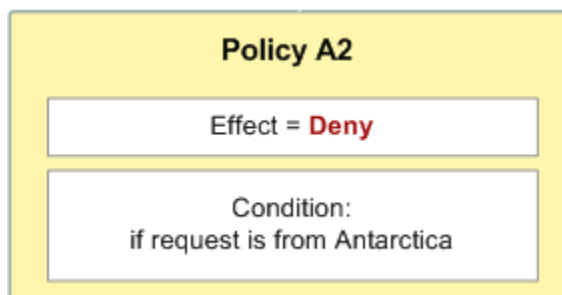
Amazon SQS 访问策略语言中明确拒绝和默认拒绝之间的关系

如果 Amazon SQS 政策不直接适用于请求，则该请求会导致[默认拒绝](#)。例如，如果用户请求使用亚马逊的权限，SQS但适用于该用户的唯一策略可以使用 DynamoDB，则该请求会导致默认拒绝。

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，那么根据策略中元素的值，该请求的结果可能为[允许](#)或[显式拒绝](#)。[效果](#)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 的 SQS 政策。

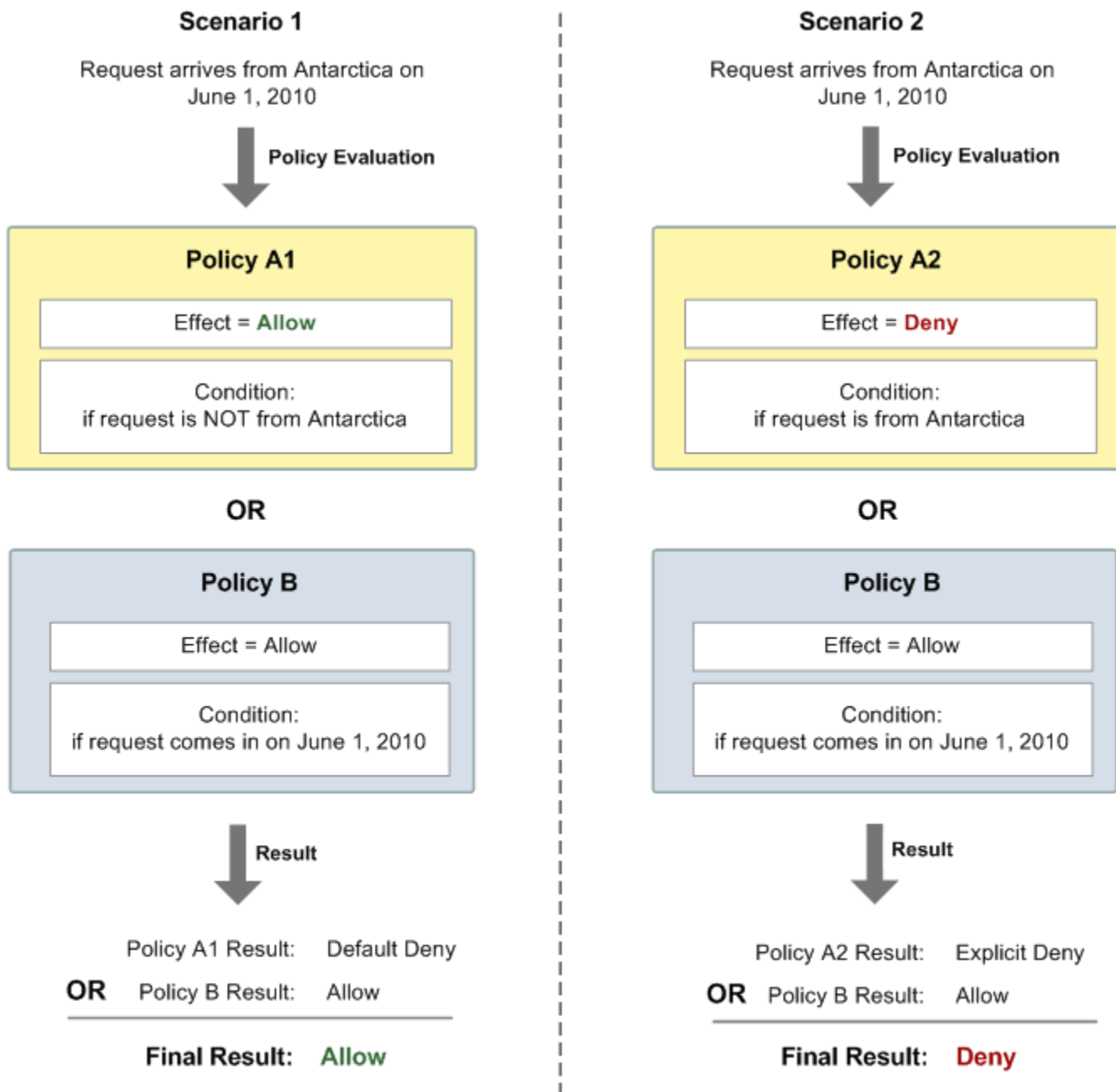


如果用户从美国发出请求，那么条件已经满足（该请求不是来自南极洲），则该请求的结果是允许。但是，如果用户从南极洲发出请求，那么条件未满足，该请求的默认结果为默认拒绝。您可以编写策略 A2，显式拒绝来自南极洲的请求，这样结果就变为显式拒绝。下列示意图说明了该策略。



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 自定义政策的局限性

跨账户存取

跨账户权限不能应用于以下操作：

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

条件键

目前，Amazon 仅SQS支持[中可用的条件键](#)的有限子集IAM。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

以下是典型的 Amazon SQS 访问策略示例。

示例 1：为一个账户授予权限

以下示例亚马逊SQS政策授予 AWS 账户 111122223333 向4445555666所有者发送和接收的权限。queue2 AWS 账户

```
{
  "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 政策允许在特定时间段内对您的账户拥有的队列进行一个或多个 AWS 账户访问权限。有必要编写此政策并 SQS 使用 [SetQueueAttributes](#) 操作将其上传到亚马逊，因为该 [AddPermission](#) 操作不允许在授予队列访问权限时指定时间限制。

```
{
  "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：向来自 Amazon EC2 实例的请求授予权限

以下示例亚马逊SQS策略允许访问来自亚马逊EC2实例的请求。此示例以“[示例 2：对一个或多个账户授予权限](#)”示例为基础：它将访问限制在 2009 年 6 月 30 日中午 12 点之前 (UTC)，它限制了对 IP 范围的访问。203.0.113.0/24有必要编写此政策并SQS使用[SetQueueAttributes](#)操作将其上传到 Amazon，因为该[AddPermission](#)操作不允许在授予队列访问权限时指定 IP 地址限制。

```
{
  "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 政策拒绝特定 AWS 账户 访问您的队列。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SQS使用[SetQueueAttributes](#)操作将其上传到 Amazon，因为该[AddPermission](#)操作不允许拒绝访问队列（它只允许向队列授予访问权限）。

```
{
  "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：如果访问不是来自VPC终端节点，则拒绝访问

以下示例亚马逊SQS政策限制访问权限queue1：111122223333 只能通过VPC终端节点 IDvpce-1a2b3c4d（使用条件指定）执行[SendMessage](#)和[ReceiveMessage](#)操作。aws:sourceVpce有关更多信息，请参阅 [适用于亚马逊的亚马逊 Virtual Private Cloud 终端节点 SQS](#)。

Note

- 该aws:sourceVpce条件不需要VPC终端节点资源，只需要VPC终端节点 ID。ARN
- 您可以修改以下示例，通过拒绝第二条语句中的所有 Amazon SQS 操作 (sqs:*)，将所有操作限制在特定的VPC终端节点上。但是，这样的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）都必须通过策略中定义的特定VPC端点进行，这可能会阻止用户将来修改队列权限。

```
{
  "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"
      }
    }
  }
]
```

在 Amazon 上使用临时安全证书 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”。

您可以使用这些临时安全证书向 Amazon 提出请求 SQS。这些 API 库使用这些凭据计算必要的签名值来验证您的请求。如果您使用过期的凭证发送请求，Amazon 会 SQS 拒绝该请求。

Note

您不能基于临时凭证设置策略。

先决条件

1. IAM用于创建临时安全证书：
 - 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. 使用临时私有访问密钥而不是您自己的私有访问密钥来签署您的查询API请求。

Note

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，并添加安全令牌。有关IAM支持临时安全证书的更多信息，请参阅IAM用户指南中的[授予对 AWS 资源的临时访问权限](#)。

使用临时安全证书调用 Amazon SQS Query API 操作

1. 使用申请临时安全令牌 AWS Identity and Access Management。有关更多信息，请参阅《[IAM用户指南](#)》中的[创建临时安全证书以允许IAM用户访问](#)。

IAM返回安全令牌、访问密钥 ID 和私有访问密钥。

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。

以下示例演示如何使用临时安全证书对 Amazon SQS 请求进行身份验证。的结构 **AUTHPARAMS** 取决于API请求的签名。有关更多信息，请参阅《[Amazon Web Services 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 通过使用与 AWS Key Management Service \(SSE\) 集成的服务器端加密 \(\) 在应用程序之间交换敏感数据。](#) KMS 通过与 Amazon SQS 的集成 AWS KMS，您可以集中管理保护亚马逊 SQS 的密钥以及保护您的其他 AWS 资源的密钥。

多个 AWS 服务可以充当向 Amazon 发送事件的事件源 SQS。要使事件源能够访问加密的 Amazon SQS 队列，您需要使用 [客户管理](#) 的 AWS KMS 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。您可以通过使用 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政策和密 AWS KMS 钥 SQS 策略控制对您的加密亚马逊 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](#)。

本指南还描述了基于资源的策略如何使用

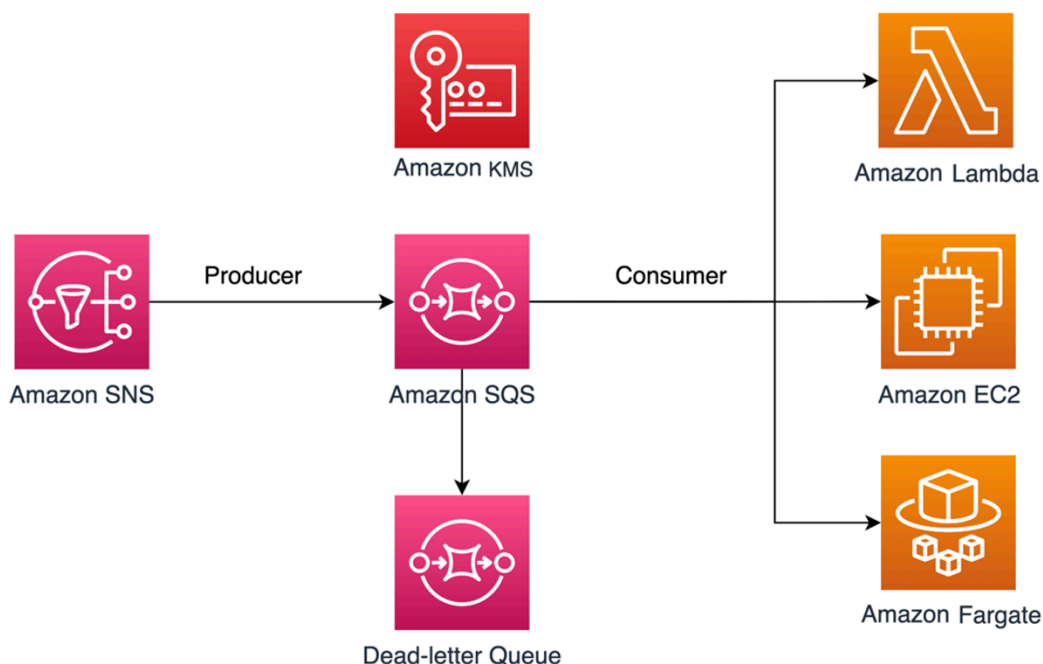
[aws:SourceArn](#)、[aws:SourceAccount](#) 和 [aws:PrincipalOrgID](#) 全局 IAM 条件上下文键来解决 [混淆副手问题](#)。

主题

- [概述](#)
- [Amazon 的最低权限密钥策略 SQS](#)
- [Amazon 关于死信队列的SQS政策声明](#)
- [防范跨服务混淆代理问题](#)
- [使用IAM访问分析器查看跨账户访问权限](#)

概述

在本主题中，我们将引导您了解一个常见的用例，以说明如何构建密钥策略和 Amazon SQS 队列策略。此使用案例如下图所示。



在此示例中，消息创建者是一个[亚马逊简单通知服务 \(SNS\)](#) 主题，该主题配置为将消息分发到您的加密亚马逊SQS队列。消息使用者是一种计算服务，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中 (VPC)，则本指南中包含的[DenyReceivingIfNotThroughVPC](#)政策声明允许您将消息接收限制为特定私有云VPC。

Note

本指南仅以政策声明的形式包含所需的IAM权限。要制定政策，您需要将声明添加到您的 Amazon SQS 策略或 AWS KMS 密钥策略中。本指南未提供有关如何创建 Amazon SQS 队列或 AWS KMS 密钥的说明。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](#)。

本指南中定义的亚马逊SQS政策不支持将消息直接重新发送到相同或不同的亚马逊SQS队列。

Amazon 的最低权限密钥政策 SQS

在本节中，我们将介绍 AWS KMS 用于加密亚马逊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](#)
- [向亚马逊SNSKMS授予向队列发布消息的权限 SNS](#)
- [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 AWS KMS 密钥的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。将此声明添加到 AWS KMS 密钥策略时，请务必替换 *<admin-role ARN>* 使用用于部署 AWS KMS 密钥、管理密钥或两者兼而有之的IAM角色的 Amazon 资源名称 (ARN)。AWS KMS 这可以是您的部署管道的IAM角色，也可以是[您的组织在 Organizations 中的管理员角色](#)。AWS

```
{
  "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 角色授予对您的密钥元数据的只读访问权限，请将 `AllowReadAccessToKeyMetadata` 声明添加到您的密钥策略中。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 委托人只要其基于身份的策略具有以下声明中列出的权限，就可以访问密钥元数据：`kms:Describe*`、`kms:Get*`、和 `kms:List*`。一定要更换 `<account-ID>` 用你自己的信息。

```

{
  "Sid": "AllowReadAccessToKeyMetadata",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
    ]
  },
  "Action": [
    "kms:Describe*",

```

```

    "kms:Get*",
    "kms:List*"
  ],
  "Resource": "*"
}

```

向亚马逊SNSKMS授予向队列发布消息的权限 SNS

要允许您的亚马逊SNS主题向加密的亚马逊SQS队列发布消息，请将AllowSNSToSendToSQS政策声明添加到您的密钥策略中。此声明授予亚马逊使用该 AWS KMS 密钥发布到您的亚马逊SQS队列的SNS权限。一定要更换 *<account-ID>* 用你自己的信息。

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: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语句向亚马逊SQS消息使用者授予解密从加密的 Amazon SQS 队列收到的消息所需的权限。附上政策声明时，请替换 *<consumer's runtime role ARN>* 使用消息使用ARN者的IAM运行时角色。

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

亚马逊最低权限SQS政策

本节将引导您了解本指南所涵盖的用例（例如，Amazon to AmazonSQS）的最低权限亚马逊SQSSNS队列策略。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。这些Deny声明可防止其他非预期实体访问亚马逊队SQS列，同时将目标实体排除在政策条件之内。

Amazon SQS 政策包括以下声明，我们在下面详细描述了这些声明：

- [限制 Amazon 的SQS管理权限](#)
- [限制指定组织的 Amazon SQS 队列操作](#)
- [向消费者授SQS予亚马逊权限](#)
- [实施传输中加密](#)
- [将消息传输限制为特定的 Amazon SNS 主题](#)
- [（可选）将消息接收限制到特定VPC端点](#)

限制 Amazon 的SQS管理权限

以下RestrictAdminQueueActions政策声明将 Amazon SQS 管理权限仅限于您用于部署队列、管理队列或两者兼而有之的一个或多个角色。IAM请务必更换 *<placeholder values>* 用你自己的信息。指定用于部署 Amazon SQS 队列的IAM角色以及应具有 Amazon SQS 管理权限的所有管理员角色。ARN ARNs

```
{
```

```

"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

为了帮助保护您的 Amazon SQS 资源免受外部访问（由[AWS 组织](#)外部实体访问），请使用以下声明。此声明将 Amazon SQS 队列访问权限限制为您在中指定的组织Condition。一定要更换 **<SQS queue ARN>** 以及用于部署 Amazon SQS 队列的IAM角色的；以及 ARN **<org-id>**，并附上您的组织 ID。

```

{
  "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>"
    ]
  }
}
}
}

```

向消费者授SQS予亚马逊权限

要从 Amazon SQS 队列接收消息，您需要向消息使用者提供必要的权限。以下策略声明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。在您的 Amazon SQS 策略中添加声明时，请务必替换 *<consumer's IAM runtime role ARN>* 与使用ARN者使用的IAM运行时角色相同；以及 *<SQS queue ARN>*，其中包含用于部署 Amazon SQS 队列的IAM角色。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亚马逊SQS队列的消息，请将该DenyOtherConsumersFromReceiving声明添加到亚马逊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。一定要更换 *<SQS queue ARN>* 以及 *<consumer's runtime role ARN>* 用你自己的信息。

```

{
  "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政策声明强制消费者和制作者使用安全通道 (TLS连接) 来发送和接收来自 Amazon SQS 队列的消息。一定要更换 *<SQS queue ARN>* 使用用于部署 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 策略声明允许指定的 Amazon SNS 主题向亚马逊 SQS 队列发送消息。一定要更换 *<SQS queue ARN>* 与用于部署 Amazon SQS 队列的 IAM 角色相同；以及 ARN *<SNS topic ARN>*，以亚马逊为 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。Replace (替换) *<SQS queue ARN>* 以及 *<SNS topic ARN>* 用你自己的信息。

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

(可选) 将消息接收限制到特定VPC端点

要将消息仅限于特定的[VPC终端节点](#)，请将以下政策声明添加到您的 Amazon SQS 队列策略中。此语句可防止消息使用者从队列接收消息，除非消息来自所需的VPC端点。Replace (替换) *<SQS queue ARN>* 与用于部署 Amazon SQS 队列的IAM角色相同；以及 ARN *<vpce_id>* 使用终端VPC端节点的ID。

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

Amazon 关于死信队列的SQS政策声明

将以下策略声明 (由其声明 ID 标识) 添加到您的DLQ访问策略中：

- RestrictAdminQueueActions
- DenyQueueActionsOutsideOrg
- AllowConsumersToReceiveFromTheQueue
- DenyOtherConsumersFromReceiving
- DenyUnsecureTransport

除了在DLQ访问策略中添加上述策略声明外，您还应添加一条声明，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

要限制仅访问同一账户的 Amazon SQS 队列，请在DLQ队列DenyAnyProducersExceptSQS策略中添加以下政策声明。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DLQ之前

进行部署，因此在创建SQS ARN时您不会知道 Amazon DLQ。如果您只需要将访问权限限制为一个 Amazon SQS 队列，请在知道时Condition使用您的亚马逊SQS源队列进行修改。aws:SourceArn ARN

```
{
  "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或多个角色上。该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 队列以删除可能的错误消息。测试应用程序时，您可以通过 Amazon SQS 队列驱动大量消息，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（所谓的被调用服务）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条件上下文密钥。这会将服务拥有的权限限制在 Organizations 中的特定资源、特定账户或特定 AWS 组织。

使用IAM访问分析器查看跨账户访问权限

您可以使用 A [AWS IAMccess Analy](#) zer 查看您的亚马逊SQS队列策略和 AWS KMS 密钥策略，并在亚马逊SQS队列或 AWS KMS 密钥向外部实体授予访问权限时提醒您。IAMAccess Analyzer 可帮助识别组织中的[资源](#)以及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 A IAM ccess Analyzer 时指定的 AWS 组织中的 AWS 帐户或组织。

IAMAccess 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 Access Analyzer 如何识别对您的 AWS 资源的意外 AWS IAM访问的更多信息，请参阅此[博客文章](#)。

有关 A AWS IAM ccess Analyzer 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](#)和编写可以附加到IAM身份的权限策略时，可以使用下表作为参考。包括每个 Amazon Simple Queue Service 操作、您可以授予执行该操作权限的相应操作，以及您可以为其授予权限的 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 支持中[提供的全局条件上下文密钥IAM](#)。

Amazon 简单队列API服务和操作所需的权限

[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

本节提供有关 Amazon SQS 的日志和监控选项的信息，包括 CloudTrail 如何使用捕获 API 调用，以及用于深入了解队列活动和性能的 CloudWatch 指标。

主题

- [使用 AWS CloudTrail 记录 Amazon SQS API 调用](#)
- [使用监控亚马逊 SQS 队列 CloudWatch](#)

使用 AWS CloudTrail 记录 Amazon SQS API 调用

与 Amazon SQS 集成 AWS CloudTrail，用于记录来自用户、角色或服务的亚马逊 SQS 调用。AWS CloudTrail 捕获与亚马逊 SQS 标准队列和 FIFO 队列相关的 API 调用作为事件，包括通过亚马逊 SQS 控制台发起的交互以及通过调用亚马逊 SQS API 以编程方式发起的交互。

主题

- [亚马逊 SQS 信息位于 CloudTrail](#)
- [中的管理活动 CloudTrail](#)
- [中的数据事件 CloudTrail](#)
- [示例：亚马逊 SQS 的 CloudTrail 管理事件](#)
- [示例：亚马逊 SQS CloudTrail 的数据事件](#)

亚马逊 SQS 信息位于 CloudTrail

CloudTrail 在您创建 AWS 账户时默认处于开启状态。当支持的 Amazon SQS 事件活动发生时，会将其与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以查看、搜索和下载您 AWS 账户的近期活动。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的[使用 CloudTrail 事件历史记录查看事件](#)。

调用队列管理操作（例如）的 Amazon SQS API `AddPermission` 被归类为管理事件，并且 CloudTrail 默认处于登录状态。Amazon SQS API 是在亚马逊 SQS 队列上执行的大量操作，`SendMessage` 例如被归类为数据事件，在您选择使用后记录下来。CloudTrail

使用 CloudTrail 收集到的信息，您可以识别向 Amazon SQS API 提出的特定请求、请求者的 IP 地址或身份以及请求的日期和时间。如果您配置 CloudTrail 跟踪，则可以将 CloudTrail 事件持续传输到 Amazon S3 存储桶，并可选择传送到 Amazon CloudWatch Logs 和 AWS EventBridge。如果未配置

跟踪，则只能在 CloudTrail 控制台的事件中查看管理事件的事件历史记录。有关更多信息，请参阅《AWS CloudTrail 用户指南》<https://docs.aws.amazon.com/awsccloudtrail/latest/userguide/>中的[创建跟踪概述](#)。

中的管理活动 CloudTrail

Amazon SQS 将以下 API 操作记录为管理事件：

- [AddPermission](#)
- [CreateQueue](#)
- [CancelMessageMoveTask](#)
- [DeleteQueue](#)
- [ListMessageMoveTasks](#)
- [PurgeQueue](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

不支持以下 Amazon SQS API 进行 CloudTrail 日志记录：

- [GetQueueAttributes](#)
- [GetQueueUrl](#)
- [ListDeadLetterSourceQueues](#)
- [ListQueueTags](#)
- [ListQueues](#)

中的数据事件 CloudTrail

[数据事件](#)提供有关在资源上或资源中执行的资源操作的信息，例如向 Amazon SQS 队列发送或从其接收 Amazon SQS 消息。数据事件是指默认情况下 CloudTrail 不记录的大容量活动。您可以使用 API 为 SQS 队列启用数据事件 CloudTrail API 操作日志记录。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的[记录数据事件](#)。

借 CloudTrail 助，您可以使用高级事件选择器来决定记录和记录哪些 Amazon SQS API 活动。要记录 Amazon SQS 数据事件，您必须包括资源类型 `AWS::SQS::Queue`。设置完成后，您可以通过选择要记录的特定数据事件（例如使用 `eventName` 过滤器来跟踪 `SendMessage` 事件）来进一步调整日志记录首选项。有关更多信息，请参阅《AWS CloudTrail API 参考》中的 [AdvancedEventSelector](#)。

Amazon SQS 数据事件：

- [SendMessage](#)
- [SendMessageBatch](#)
- [ReceiveMessage](#)
- [DeleteMessage](#)
- [DeleteMessageBatch](#)
- [ChangeMessageVisibility](#)
- [ChangeMessageVisibilityBatch](#)

记录数据事件将收取额外费用。有关更多信息，请参阅 [AWS CloudTrail 定价](#)。

示例：亚马逊 SQS 的 CloudTrail 管理事件

以下示例显示了支持的 API 的 CloudTrail 日志条目：

AddPermission

以下示例显示了 `AddPermission` API 调用的 CloudTrail 日志条目。

```
{
  "Records": [
    {
      "eventVersion": "1.06",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Alice"
      },
      "eventTime": "2018-06-28T22:23:46Z",
```

```

    "eventSource": "sqs.amazonaws.com",
    "eventName": "AddPermission",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
    "requestParameters": {
      "actions": [
        "SendMessage"
      ],
      "AWSAccountIds": [
        "123456789012"
      ],
      "label": "MyLabel",
      "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue"
    },
    "responseElements": null,
    "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
    "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
  }
]
}

```

CreateQueue

以下示例显示了 CreateQueue API 调用的 CloudTrail 日志条目。

```

{
  "Records": [
    {
      "eventVersion": "1.06",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Alejandro",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Alejandro"
      },
      "eventTime": "2018-06-28T22:23:46Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "CreateQueue",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "203.0.113.1",

```

```
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
    "requestParameters": {
      "queueName": "MyQueue"
    },
    "responseElements": {
      "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
    },
    "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
    "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
  }
]
}
```

DeleteQueue

以下示例显示了 DeleteQueue API 调用的 CloudTrail 日志条目。

```
{
  "Records": [
    {
      "eventVersion": "1.06",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Carlos",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Carlos"
      },
      "eventTime": "2018-06-28T22:23:46Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "DeleteQueue",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "203.0.113.2",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
      "requestParameters": {
        "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
      },
      "responseElements": null,
      "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
      "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
    }
  ]
}
```

```

]
}

```

RemovePermission

以下示例显示了 RemovePermission API 调用的 CloudTrail 日志条目。

```

{
  "Records": [
    {
      "eventVersion": "1.06",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Jane",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Jane"
      },
      "eventTime": "2018-06-28T22:23:46Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "RemovePermission",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "203.0.113.3",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
      "requestParameters": {
        "label": "label",
        "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
      },
      "responseElements": null,
      "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
      "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
    }
  ]
}

```

SetQueueAttributes

以下示例显示了以下内容的 CloudTrail 日志条目 SetQueueAttributes :

```

{
  "Records": [

```

```
{
  "eventVersion": "1.06",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAI44QH8DHBEXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Maria",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Maria"
  },
  "eventTime": "2018-06-28T22:23:46Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "SetQueueAttributes",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "203.0.113.4",
  "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
  "requestParameters": {
    "attributes": {
      "VisibilityTimeout": "100"
    }
  },
  "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue"
},
"responseElements": null,
"requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
"eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
}
]
```

示例：亚马逊 SQS CloudTrail 的数据事件

以下是特定于 Amazon SQS 数据 CloudTrail 事件 API 的事件示例：

SendMessage

以下示例显示了 CloudTrail 的数据事件SendMessage。

```
{
  "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": {
  "mD50fMessageBody": "9a4e3f7a614d9dd9f8722092dbda17a2",
  "mD50fMessageSystemAttributes": "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"
}
}

```

ReceiveMessage

以下示例显示了 CloudTrail 的数据事件 ReceiveMessage。

```

{
  "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": "2023-11-07T23:59:24Z",
"eventSource": "sqs.amazonaws.com",

```

```

    "eventName": "ReceiveMessage",
    "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",
        "maxNumberOfMessages": 10
    },
    "responseElements": null,
    "requestID": "8b4d4643-8f49-52cd-a6e8-1b875ed54b99",
    "eventID": "f3f23ab7-b0a4-4b71-afc0-141209c49206",
    "readOnly": true,
    "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"
    }
}

```

DeleteMessageBatch

以下示例显示了 CloudTrail 的数据事件 DeleteMessageBatch。

```

{
  "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": "2023-11-07T23:59:24Z",
"eventSource": "sqs.amazonaws.com",
"eventName": "DeleteMessageBatch",
"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",
  "entries": [
    {
      "id": "0",
      "receiptHandle": "AQEBefxM104zyZGF87DehbRbmri91w2W7mMdD0GrBjQa8e/
hpb4RbXHPZ9tLBV1eECbChQIE5NtaDuoZhZP0kTy0eN46EyRR4jXDzE3A1kbP1X1mA9f2fUuTrXx8aeCoCA3I3woNg3f
h1LS94tjAZqV2krc4BaC2pYggyHWcW019HwIV8T/bjNMIeZoQwOM5V
+o9vHPfewz5QGr5SKpDo7uE7Umyk5n5CJZvcn1efp/
mrwtaCIb9M7cCQUYcZm2ZmZDnI09XpGTai3m2dQ0M83pnNh0nvDfPkHpoa+hX1TrUmxCupCWHJwA8HFJ10/
CCJsodMNFthLBA9S57dkBZCsw41G8jAmgQ0MkvZ0UL5mg00FQQd1Yrw0zvthjCgiwdzn0yXoMzxIZMBxkY14E4nVVZ7M
h8oRk2C7gByzg2kYJ0LnUvLJFT8DQE28JZppEC9klvrdR/BWiPT7asc="
    }
  ]
},
"responseElements": {
  "successful": [
    {
      "id": "0"
    }
  ]
}

```

```

    ],
    "failed": []
  },
  "requestID": "fe423091-5642-5ba5-9256-6d5587de52f1",
  "eventID": "88c8020d-d769-4985-8ecb-ee0b59acc418",
  "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"
  }
}

```

ChangeMessageVisibilityBatch

以下示例显示了 CloudTrail 的数据事件 ChangeMessageVisibilityBatch。

```

{
  "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": "2023-11-07T23:59:01Z",
"eventSource": "sqs.amazonaws.com",
"eventName": "ChangeMessageVisibilityBatch",
"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": {
  "visibilityTimeout": 0,
  "entries": [
    {
      "id": "0",
      "receiptHandle":
"AQEB2M5cVYg5gslhWME6537hdjcaPn0YPA5M0W460TTb0DzP1e631yPwm8qxd401hDj/
B4ntTMnsgBTa95t14tNx7Vn96jKJ5rIoZ7iI8TRmkT1caKodKIPs8w9yndZq50c2FPQxtyH+2L3UHF/
abV3szqVWX0LZR4PwX8zZkWWQGNCNnY2q21GCG586F8Qwvr0FYoXNwB8ymd1t77e1PDPknq1Io3JFuzkEsndkkETy4fv
15PHX17nXxaC+DURV1MPX0uSFACGmWqAoyk50HKwG0jLQgpySL/
TcnQXC1vFq8kNXGwyVzJsbwHp0HxI7oce69vaD6DaWFP75d3hx+PJeG9pauQCKzVP3skt3Hw/
zDC7YfKcALD3aCwMmeNDwT3w0BUG6XZdG51YhtFtTQYV7YuS3i/
Jh3HShGbtm07JK0EFiPkxv2+XNaAX3gFEpbng6zamTanfyMXCJIigIAEqiyWHQ=",
      "visibilityTimeout": 2271
    }
  ],
  "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue"
},
"responseElements": {
  "successful": [
    {
      "id": "0"
    }
  ]
},
"requestID": "d49ab65f-9dc7-54b8-875c-eb9b4c42988b",
"eventID": "ca16c8c2-c4ba-4eb5-a54c-e650a10266d4",
"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"
}
}
```

使用监控亚马逊 SQS 队列 CloudWatch

亚马逊 SQS 和亚马逊 CloudWatch 集成在一起，因此您可以使用 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、控制台、使用 CloudWatch 或 AP CloudWatch I 查看和分析队列 AWS CLI 的指标。](#)您还可以为 [Amazon 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，并每隔一分钟推送到该 CloudWatch 队列。这些指标是在所有符合活跃状态 CloudWatch 准则的队列上收集的。CloudWatch 如果队列包含任何消息，或者有任何操作可以访问该队列，则认为该队列最长处于活动状态六个小时。

当 Amazon SQS 队列处于非活动状态超过六小时时，Amazon SQS 服务将被视为处于休眠状态，并停止向该服务提供指标。CloudWatch 在您的亚马逊 SQS 队列处于非活动状态期间，无法在 Amazon SQS 的 CloudWatch 指标中显示缺失的数据或表示零的数据。

Note

- 当 Amazon SQS 队列调用 API 的用户未获得授权且请求失败时，可以激活 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 指标中最多会延迟 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 队列都支持指标。

主题

- [访问 Amazon SQS 的 CloudWatch 指标](#)
- [为 Amazon SQS 指标创建 CloudWatch 警报](#)
- [亚马逊 SQS 的可用 CloudWatch 指标](#)

访问 Amazon SQS 的 CloudWatch 指标

亚马逊 SQS 和亚马逊 CloudWatch 集成在一起，因此您可以使用 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、控制台、使用 CloudWatch 或 AP CloudWatch I 查看和分析队列 AWS CLI 的指标。](#)您还可以为 [Amazon SQS 指标设置 CloudWatch 警报](#)。

Amazon SQS 控制台

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。

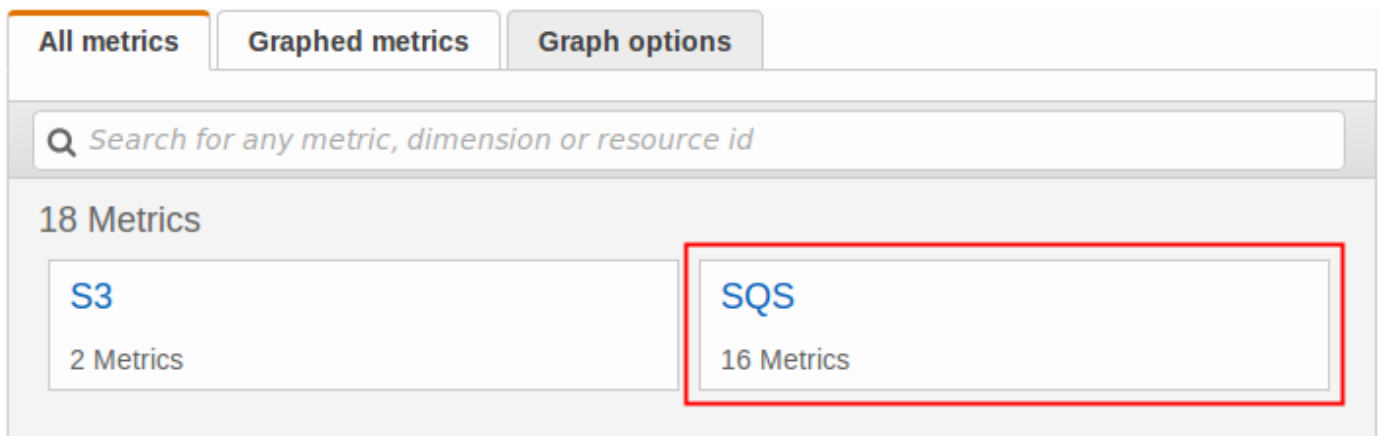
4. 要了解特定图形表示的内容，请将光标悬停在所需图形旁的  上，或参阅 [亚马逊 SQS 的可用 CloudWatch 指标](#)。
5. 要同时更改所有图形的时间范围，请针对 Time Range 选择所需时间范围 (例如，Last Hour)。
6. 要查看单个图形的其他统计数据，请选择该图形。
7. 在 CloudWatch 监控详细信息对话框中，选择一个统计数据 (例如，总和)。有关受支持的统计数据的列表，请参阅 [亚马逊 SQS 的可用 CloudWatch 指标](#)。
8. 要更改单个图形显示的时间范围和时间间隔 (例如，显示最近 24 小时的时间范围而不是前 5 分钟，或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。

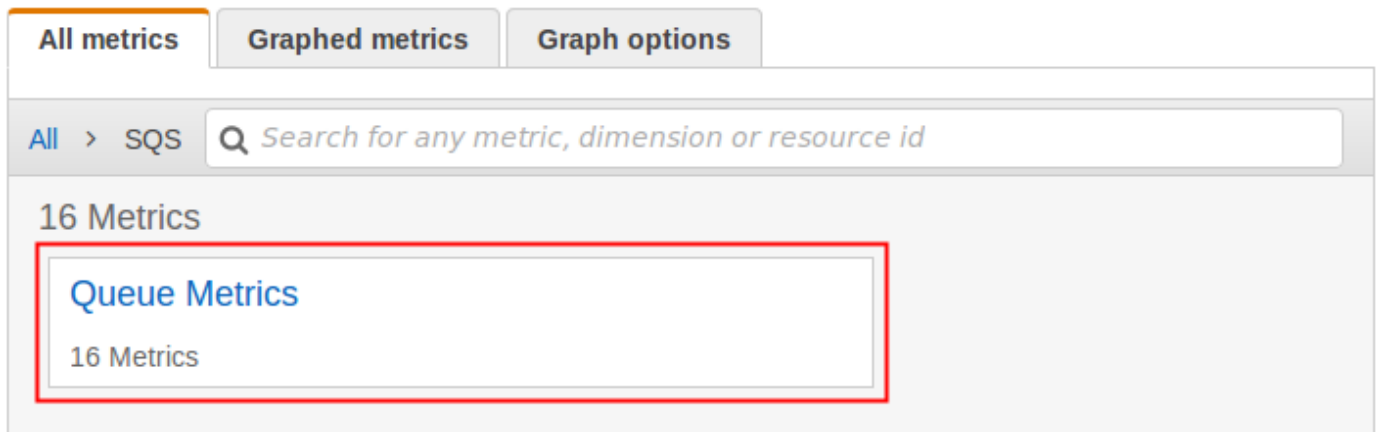
- （可选）要使用其他 CloudWatch 功能，请在监控选项卡上选择查看所有 CloudWatch 指标，然后按照[亚马逊 CloudWatch 控制台](#)过程中的说明进行操作。

亚马逊 CloudWatch 控制台

- 登录 [CloudWatch 控制台](#)。
- 在导航面板上，选择 Metrics。
- 选择 SQS 指标命名空间。



- 选择 Queue Metrics 指标维度。



- 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。

All metrics		
Graphed metrics		
Graph options		
All > SQS > Queue Metrics		
Search for any metric, dimension or resource id		
<input type="checkbox"/>	QueueName (16)	Metric Name
<input type="checkbox"/>	MyQueue	ApproximateAgeOfOldestMessage
<input type="checkbox"/>	MyQueue	NumberOfMessagesDelayed
<input type="checkbox"/>	MyQueue	NumberOfMessagesNotVisible
<input type="checkbox"/>	MyQueue	NumberOfMessagesVisible
<input type="checkbox"/>	MyQueue	
<input type="checkbox"/>	MyQueue	
<input type="checkbox"/>	MyQueue	
<input type="checkbox"/>	MyQueue	
<input type="checkbox"/>	MyQueue	NumberOfMessagesSent

有关更多信息和其他选项，请参阅亚马逊 CloudWatch 用户指南中的[图表指标](#)和使用亚马逊 CloudWatch[控制面板](#)。

AWS Command Line Interface

要使用访问 Amazon SQS 指标 AWS CLI，请运行命令。[get-metric-statistics](#)

有关更多信息，请参阅 Amazon CloudWatch 用户指南中的[获取指标的统计数据](#)。

CloudWatch API

要使用 CloudWatch API 访问亚马逊 SQS 指标，请使用操作。[GetMetricStatistics](#)

有关更多信息，请参阅 Amazon CloudWatch 用户指南中的[获取指标的统计数据](#)。

为 Amazon SQS 指标创建 CloudWatch 警报

CloudWatch 允许您根据指标阈值触发警报。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 Management Console 并打开 CloudWatch 控制台，[网址为 https://console.aws.amazon.com/cloudwatch/](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. 对于 SQS > 队列指标，选择要为其设置警报的 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. 选择创建警报。

将创建警报。

亚马逊 SQS 的可用 CloudWatch 指标

Amazon SQS 将以下指标发送至。 CloudWatch

Note

由于Amazon SQS采用分布式架构，因此某些指标的结果是近似值。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

AWS/SQS 命名空间包括以下指标。

指标	描述
ApproximateAgeOfOldestMessage	<p>队列中最旧的未删除消息的大约存在时间。</p> <div data-bbox="938 800 1511 1885" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p>Note</p> <ul style="list-style-type: none"> 在接收消息三次（或以上）且未处理时，该消息将会移至队列的后面，而 ApproximateAgeOfOldestMessage 指标会指示尚未接收超过三次的第二旧的消息。即使队列具有重新驱动策略，也会发生此操作。 由于单个毒丸消息（多次接收但从未删除）会扭曲此指标，直到成功使用毒丸消息之前，指标中都不会包含毒丸消息的使用期限。 如果队列有重新驱动策略，当达到配置的最大接收数目后，消息将会移至死信队列。当消息移至死信队列，死信队列的 ApproximateAgeOfOldestMessage 指标表示该消 </div>

指标	描述
	<p>息移至死信队列的时间（而不是该消息发送的原始时间）。</p> <ul style="list-style-type: none">• 对于 FIFO 队列，消息不会移到队列的后面，因为这将违反 FIFO 顺序保证。如果已配置 DLQ，则该消息将转至 DLQ。否则，它将阻止该消息组，直到成功删除或消息组过期。 <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：秒</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p>

指标	描述
<code>ApproximateNumberOfGroupsWithInflightMessages</code>	<p>包含传输中消息的大致消息组数量，其中消息在消费者从队列中接收到但尚未从队列中删除后被视为正在传输。该指标可以通过增加 FIFO 消息组或扩展使用者来帮助您排查和优化 FIFO 队列吞吐量。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：计数</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p> <p>有关当前的 FIFO 吞吐量和飞行限制，请参阅亚马逊 SQS 消息配额。</p>
<code>ApproximateNumberOfMessagesDelayed</code>	<p>队列中延迟且无法立即读取的消息数量。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。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：计数</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p>

指标	描述
ApproximateNumberOfMessagesNotVisible	<p>处于空中状态的消息的数量。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。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：计数</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p>
ApproximateNumberOfMessagesVisible	<p>要处理的消息数。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：计数</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p> <p>发送给流程的消息数量没有限制，但是您可以将此积压限制在保留期内。</p>

指标	描述
NumberOfEmptyReceives ¹	<p>未返回消息的 ReceiveMessage API 调用数量。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：计数</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p>
NumberOfDeduplicatedSentMessages	<p>发送到队列的经过重复数据删除的消息数量。该指标可以帮助确定创建者是否向 Amazon SQS FIFO 队列发送重复的消息。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：计数</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p>

指标	描述
NumberOfMessagesDeleted ¹	<p>从队列删除的消息数量。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：计数</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p> <p>Amazon SQS 会针对使用有效接收句柄的每一次成功删除操作（包括删除重复项）发布 NumberOfMessagesDeleted 指标。以下情形可能会使 NumberOfMessagesDeleted 指标值高于预期：</p> <ul style="list-style-type: none">• 在属于相同消息的不同接收句柄上调用 DeleteMessage 操作：如果该消息未在可见性超时过期之前处理，则该消息将对其他可对其执行处理和再次删除操作的使用者可用，从而使 NumberOfMessagesDeleted 指标值增大。• 在相同接收句柄上调用 DeleteMessage 操作：如果该信息已被处理和删除，但是您若使用相同的接收句柄再次调用 DeleteMessage 操作，将返回一个成功状态，使 NumberOfMessagesDeleted 指标值增大。

指标	描述
NumberOfMessagesReceived ¹	<p>调用 <code>ReceiveMessage</code> 操作返回的消息数量。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：计数</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p>
NumberOfMessagesSent ¹	<p>添加到队列的消息数量。</p> <p>如果您手动向死信队列发送消息，它将由 <code>NumberOfMessagesSent</code> 指标捕获。但是，如果由于处理尝试失败而将消息发送到死信队列，则该指标不会捕获该消息。因此，<code>NumberOfMessagesSent</code> 和 <code>NumberOfMessagesReceived</code> 的值可能不同。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：计数</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p>

指标	描述
SentMessageSize ¹	<p>添加到队列的消息大小。</p> <p>报告标准：如果队列处于活动状态，则报告非负值。</p> <p>单位：字节</p> <p>有效统计数据：平均值、最小值、最大值、总和、数据样本（在 Amazon SQS 控制台中显示为样本数）</p> <div style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p> Note</p> <p>SentMessageSize 在向相应的队列发送至少一条消息之前，不会在 CloudWatch 控制台中显示为可用指标。</p> </div>

¹ 这些指标是从服务角度计算的，可能包括重试。不要依赖这些指标的绝对值，也不要使用它们来估算当前队列状态。

Amazon SQS 指标的维度

Amazon SQS 发送到 CloudWatch 的唯一维度是。QueueName 这表示所有可用统计信息会通过 QueueName 进行筛选。


Amazon 合规性验证 SQS

要了解是否属于特定合规计划的范围，请参阅 AWS 服务“[按合规计划划分的范围](#)”，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AWS](#)。

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您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。AWS 提供了以下资源来帮助实现合规性：

- [安全与合规性快速入门指南](#) — 这些部署指南讨论了架构注意事项，并提供了部署以安全性和合规性为重点 AWS 的基准环境的步骤。
- [在 Amazon Web Services 上进行HIPAA安全与合规架构](#) — 本白皮书描述了各公司如何使用 AWS 来创建HIPAA符合条件的应用程序。

 Note

并非所有 AWS 服务 人都有HIPAA资格。有关更多信息，请参阅 [《HIPAA合格服务参考》](#)。

- [AWS 合AWS 规资源](#) — 此工作簿和指南集可能适用于您的行业和所在地区。
- [AWS 客户合规指南](#) — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践，AWS 服务 并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)) 的安全控制。
- [使用AWS Config 开发人员指南中的规则评估资源](#) — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- [AWS Security Hub](#)— 这 AWS 服务 可以全面了解您的安全状态 AWS。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控件的列表，请参阅 [Security Hub 控件参考](#)。
- [Amazon GuardDuty](#) — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务 检测您的工作负载、容器和数据面临的潜在威胁。GuardDuty 可以帮助您满足各种合规性要求 PCIDSS，例如满足某些合规性框架规定的入侵检测要求。
- [AWS Audit Manager](#)— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

Amazon SQS 中的恢复功能

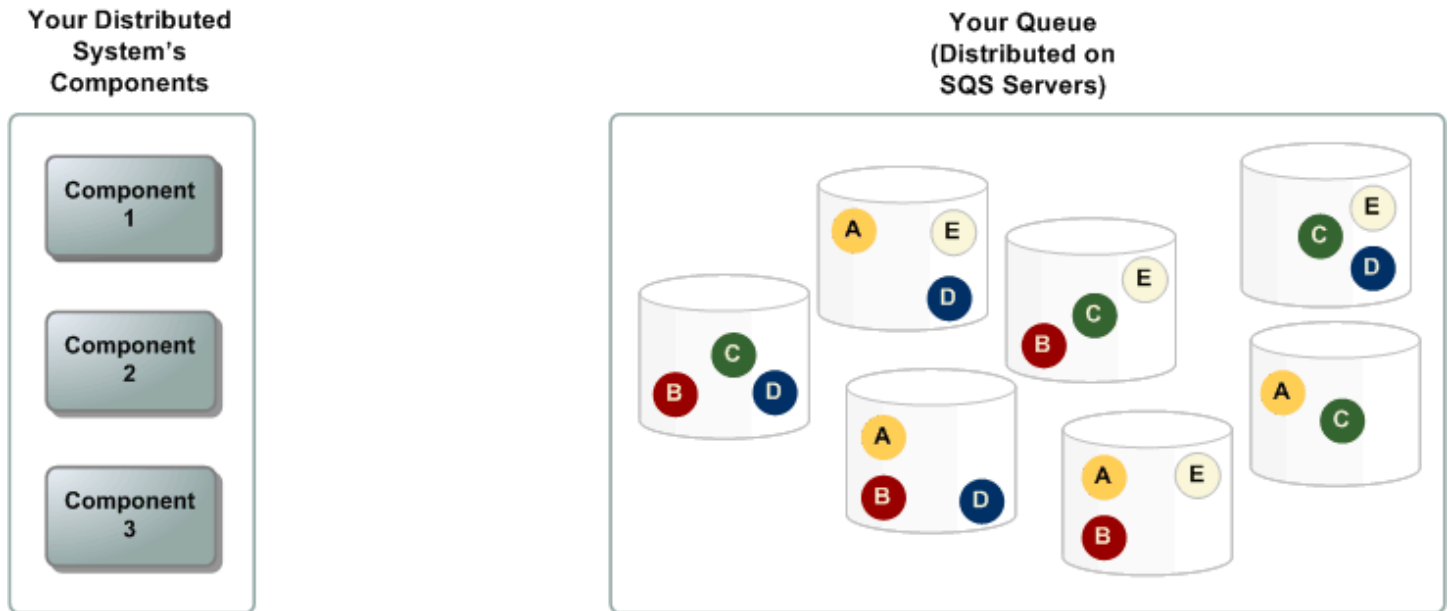
AWS 全球基础设施是围绕 AWS 区域和可用区构建的。AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。有关 AWS 区域和可用区的更多信息，请参阅[AWS 全球基础设施](#)。

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

分布式消息传送系统有三个主要组成部分：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 Amazon SQS。客户端必须支持传输层安全性 (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 Amazon VPC 端点或特定 VPC 的访问。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：如果访问不是来自 VPC 终端节点，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，您应该在自己的安全策略中查看这些功能。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。

主题

- [确保队列不可公开访问](#)
- [实施最低权限访问](#)
- [对需要 Amazon SQS 访问权限的应用程序和 AWS 服务使用 IAM 角色](#)
- [实施服务器端加密](#)
- [实施传输中数据加密](#)
- [考虑使用 VPC 端点来访问 Amazon SQS](#)

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 中的身份和访问管理 SQS](#)
- [Amazon SQS API 权限：操作和资源参考](#)
- [使用带有 Amazon SQS 访问策略语言的自定义策略](#)

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，您不必向 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 中的静态加密 SQS](#) 和 [亚马逊 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控制从特定 Amazon VPC 端点或特定 VPC 对队列的访问。

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 VPC 或 VPC 端点有权访问您的队列。

有关更多信息，请参阅 [适用于亚马逊的亚马逊 Virtual Private Cloud 终端节点 SQS](#) 和 [为亚马逊创建亚马逊 VPC 终端节点策略 SQS](#)。

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	可用于处理队列的 AWS CLI 命令的描述。
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 高级支持信息	有关 AWS Premium Support 信息的主要网页，Premium Support 是一个一对一的快速响应支持渠道，可帮助您在 AWS 基础架构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。要获取有关本文档更新的通知，请订阅 [RSSFeed](#)。

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。有关服务功能的区域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
AWS kms:Decrypt	Amazon SQS 不再需要获得kms:Decrypt 许可SendMessage API。客户现在只需要kms:GenerateDataKey 获得用于加密队列的KMS密钥的权限，但仍需要呼叫kms:Decrypt 权限ReceiveMessage 。	2024 年 7 月 24 日
FIFO指标更新	对亚马逊指标的 <code>NumberOfDuplicatedSentMessages</code> 并 <code>ApproximateNumberOfGroupsWithInflightMessages</code> 已添加到亚马逊SQSFIFO指标中。	2024 年 7 月 3 日
ListQueueTags AmazonSQSReadOnlyAccess 托管策略支持的操作	AmazonSQSReadOnlyAccess 托管策略支持检索ListQueueTags 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 的 AWS 托管政策SQS以及这些政策的更新	Amazon SQS 添加了一项新操作，允许您在特定源队列下列出最新的消息移动任务（最多 10 个）。此操作与该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 配置死信队列重新驱动器。SQS APIs	2023 年 6 月 7 日
ABAC适用于亚马逊 SQS	基于属性的访问控制 (ABAC) 使用队列标签实现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高吞吐量的文档优化。	2022 年 10 月 20 日
默认服务器端加密 (SSE) 可用	默认情况下，服务器端加密 (SSE) 使用SQS自有加密 (SSE-SQS)。	2022 年 9 月 26 日
Amazon SQS 感到困惑的副手保护支持可用	混乱的代理保护允许您在他们的请求中指定新的标题，在使用 Amazon managed 时，这些标头会根据KMS政策中的条件进行SQS检查SSE。	2021 年 12 月 29 日
SSE“托管”可用	Amazon m SQS anaged SSE (SSE-SQS) 是一种托管服务器端加密，它使用亚马逊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

FIFO队列中的消息具有高吞吐量	Amazon SQS FIFO 队列的高吞吐量为FIFO队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
预览版为FIFO队列中的消息提供了高吞吐量	Amazon SQS FIFO 队列的高吞吐量处于预览版，可能会发生变化。此功能为FIFO队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 。	2020 年 12 月 17 日
全新 Amazon SQS 主机设计	为了简化开发和生产工作流程，Amazon SQS 控制台提供了 全新的用户体验 。	2020 年 7 月 8 日
Amazon SQS 支持和的 listQueues 分页 listDeadLetterSourceQueues	您可以指定要从 listQueue 或 listDeadLetterSourceQueues 请求返回的最大结果数。	2020 年 6 月 22 日
除了 AWS GovCloud (美国) 地区外，Amazon 在所有 AWS 地区都SQS支持 1 分钟的亚马逊 CloudWatch 指标	Amazon SQS 的“一分钟” Amazon CloudWatch 指标适用于除区域之外的所有 AWS GovCloud (US) 区域。	2020 年 1 月 9 日
亚马逊SQS支持 1 分钟指标 CloudWatch	亚马逊的一分钟 CloudWatch 指标目前SQS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	您可以将到达FIFO队列的消息配置为 Lambda 函数触发器。	2019 年 11 月 25 日

适用于 Amazon SQS 的服务器端加密 (SSE) 在中国地区可用	SSEfor SQS Amazon 已在中国地区推出。	2019 年 11 月 13 日
FIFO中东 (巴林) 区域提供队列	FIFO中东 (巴林) 地区提供队列。	2019 年 10 月 10 日
适用于亚马逊的 Amazon Virtual Private Cloud (亚马逊 VPCSQS) 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 AWS GovCloud (美国西部) VPC地区的亚马逊向您的亚马逊SQS队列发送消息。	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	您可以使用 X-Ray 对通过亚马逊SQS队列的消息进行故障排除。此版本为SendMessage 和SendMessageBatch API操作添加了MessageSystemAttribute 请求参数 (允许您通过 Amazon 发送 X-Ray 跟踪标头SQS)、 ReceiveMessage API操作的AWSTraceHeader 属性和MessageSystemAttributeValue 数据类型。	2019 年 8 月 28 日
您可以在创建时为 Amazon SQS 队列添加标签	您可以使用单个 Amazon SQS API 调用、AWS SDK函数或 AWS Command Line Interface (AWS CLI) 命令同时创建队列并指定其标签。此外, Amazon 还SQS支持aws:TagKeys 和 aws:RequestTag AWS Identity and Access Management (IAM) 键。	2019 年 8 月 22 日

Amazon 的临时队列客户端SQS现已上市	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
SSEfor SQS Amazon 已在 AWS GovCloud (美国东部) 地区推出	亚马逊SQS服务器端加密 (SSE) 在 AWS GovCloud (美国东部) 地区可用。	2019 年 6 月 20 日
FIFO亚太地区 (香港)、中国 (北京)、(美国东部) 和 AWS GovCloud AWS GovCloud (美国西部) 地区提供队列	FIFO在亚太地区 (香港)、中国 (北京)、(美国东部) 和 AWS GovCloud AWS GovCloud (美国西部) 地区提供队列。	2019 年 5 月 15 日
亚马逊VPC终端节点策略适用于亚马逊 SQS	您可以为亚马逊创建亚马逊 VPC终端节点策略SQS。	2019 年 4 月 4 日
FIFO欧洲 (斯德哥尔摩) 和中国 (宁夏) 区域提供队列	FIFO欧洲 (斯德哥尔摩) 和中国 (宁夏) 地区提供队列。	2019 年 3 月 14 日
FIFO队列在所有可用 Amazon 的地区SQS都可用	FIFO在美国东部 (俄亥俄州)、美国东部 (弗吉尼亚北部)、美国西部 (加利福尼亚北部)、美国西部 (俄勒冈)、亚太地区 (孟买)、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)、加拿大 (中部)、欧洲 (法兰克福)、欧洲 (爱尔兰)、欧洲 (伦敦)、欧洲 (巴黎) 和南美洲 (圣保罗) 地区提供队列。	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。