
Amazon Simple Queue Service

开发人员指南



Amazon Simple Queue Service: 开发人员指南

Copyright © 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
Amazon SQS 的优势	1
Amazon SQS、Amazon MQ 和 Amazon SNS 之间的区别	1
队列类型	2
Amazon SQS 入门的常见任务	2
Amazon SQS 的定价	3
设置	4
第 1 步 : 创建 AWS 账户	4
第 2 步 : 创建 IAM 用户	4
第 3 步 : 获取访问密钥 ID 和秘密访问密钥	5
第 4 步 : 为使用示例代码做好准备	6
后续步骤	6
入门	7
Prerequisites	7
步骤 1: 创建队列	7
步骤 2: 发送消息	7
步骤 3: 接收和删除消息	8
步骤 4: 删 除队列	8
后续步骤	9
配置 Amazon SQS	10
了解 Amazon SQS 控制台	10
创建队列	11
编辑队列	12
配置队列参数	12
配置访问策略	13
为队列配置 SSE	13
配置死信队列	14
为队列配置标签	15
为队列订阅主题	15
配置 Lambda 触发器	16
Prerequisites	16
管理队列	18
发送消息	18
消息属性	19
接收和删除消息	19
清除队列	20
删除队列	20
确认队列为空	21
Amazon SQS 的工作原理	23
基本 架构	23
分布式队列	23
消息生命周期	24
标准队列	25
消息排序	25
至少一次传递	25
FIFO 队列	25
消息排序	26
关键术语	26
FIFO 传送逻辑	27
确切一次处理	27
从标准队列移至 FIFO 队列	28
FIFO 队列的高吞吐量	28
Compatibility	33
队列和消息标识符	33

标准队列和 FIFO 队列的标识符	33
FIFO 队列的其他标识符	34
消息元数据	34
消息属性	35
消息系统属性	37
处理 消息所需的资源	37
列出队列分页	38
成本分配标签	38
短轮询和长轮询	38
通过短轮询来使用消息	39
通过长轮询来使用消息	39
长轮询和短轮询之间的区别	40
死信队列	40
死信队列的工作方式	40
死信队列有哪些好处？	41
不同的队列类型如何处理消息失败？	41
何时应使用死信队列？	41
排查死信队列的问题	42
可见性超时	42
传输中的消息	43
设置可见性超时	43
更改消息的可见性超时	44
终止消息的可见性超时	44
延迟队列	44
临时队列	45
虚拟队列	45
请求-响应消息收发模式（虚拟队列）	46
示例方案：处理登录请求	46
清除队列	48
消息定时器	48
最佳实践	49
标准队列和 FIFO 队列的建议	49
使用 消息	49
降低 成本	51
从标准队列移至 FIFO 队列	51
针对 FIFO 队列的其他建议	52
使用 消息重复数据删除 ID	52
使用 消息组 ID	53
使用接收请求尝试 ID	53
Java 软件开发工具包示例	54
使用服务器端加密	54
将 SSE 添加到现有队列	54
为队列禁用 SSE	55
使用 SSE 创建队列	55
检索 SSE 属性	55
配置标签	55
列出标签	56
添加或更新标签	56
删除标签	56
发送消息属性	57
定义属性	57
发送带有属性的消息	58
管理大消息	58
Prerequisites	59
示例：使用 Amazon S3 管理大型 Amazon SQS 消息	59
使用 JMS	62
Prerequisites	62

Java 消息传递库入门	63
创建 JMS 连接	63
创建 Amazon SQS 队列	63
同步发送消息	64
同步接收消息	65
异步接收消息	66
使用客户端确认模式	67
使用无序确认模式	67
将 JMS 客户端与其他 Amazon SQS 客户端一起使用	68
实际可用的 Java 示例：结合使用 JMS 与 Amazon SQS 标准队列	69
ExampleConfiguration.java	69
TextMessageSender.java	71
SyncMessageReceiver.java	72
AsyncMessageReceiver.java	73
SyncMessageReceiverClientAcknowledge.java	74
SyncMessageReceiverUnorderedAcknowledge.java	77
SpringExampleConfiguration.xml	79
SpringExample.java	80
ExampleCommon.java	82
支持的 JMS 1.1 实施	83
支持的常用接口	83
支持的消息类型	83
支持的消息确认模式	83
JMS 定义的标头和预留属性	83
教程	85
创建 Amazon SQS 队列 (AWS CloudFormation)	85
从 VPC 发送消息	86
第 1 步：创建 Amazon EC2 key pair	86
第 2 步：CreateAWSresources	87
第 3 步：确认您的 EC2 实例不可公开访问	87
第 4 步：为 Amazon SQS 创建一个亚马逊 VPC 终端节点	88
第 5 步：向您的 Amazon SQS 队列发送消息	89
配额	90
与队列相关的配额	90
与消息相关的配额	91
与策略相关的配额	92
自动化和问题排查	94
使用 EventBridge 自动处理通知	94
使用 X-Ray 问题排查队列	94
安全性	95
数据保护	95
数据加密	95
互联网络流量隐私	101
Identity and Access Management	102
Authentication	103
访问控制	103
概述	104
使用基于身份的策略	108
将自定义策略与访问策略语言配合使用	116
使用临时安全凭证	125
API 权限参考	126
日志记录和监控	128
使用 CloudTrail 记录 API 调用	129
使用 CloudWatch 监控队列	132
合规性验证	139
故障恢复能力	140
分布式队列	140

基础设施安全性	140
最佳实践	140
预防性最佳实践	141
使用 API	143
提出查询 API 请求	143
构建终端节点	143
提出 GET 请求	144
提出 POST 请求	144
对请求进行身份验证	145
解释响应	147
批处理操作	148
启用客户端缓冲和请求批处理	149
利用水平扩展和操作批处理来提高吞吐量	152
相关资源	161
文档历史记录	162
AWS术语表	164
	clxv

什么是 Amazon Simple Queue Service ?

Amazon Simple Queue Service (Amazon SQS) 提供了安全、持久且可用的托管队列，可让您集成和分离分布式软件系统和组件。Amazon SQS 提供了常见的结构，例如[死信队列 \(p. 40\)](#)和[成本分配标签 \(p. 38\)](#)。它提供了一个通用 Web 服务 API，您可以使用任何编程语言（AWSSDK 支持）。

Amazon SQS 同时支持[Standard \(p. 25\)](#)和[FIFO 队列 \(p. 25\)](#)。有关更多信息，请参阅[队列类型 \(p. 2\)](#)。

主题

- [Amazon SQS 的优势 \(p. 1\)](#)
- [Amazon SQS、Amazon MQ 和 Amazon SNS 之间的区别 \(p. 1\)](#)
- [队列类型 \(p. 2\)](#)
- [Amazon SQS 入门的常见任务 \(p. 2\)](#)
- [Amazon SQS 的定价 \(p. 3\)](#)

Amazon SQS 的优势

- 安全—[您控制 \(p. 102\)](#)谁能向 Amazon SQS 队列发送消息并从队列接收消息。
[服务器端加密 \(SSE\) \(p. 96\)](#)可让您通过使用 AWS Key Management Service(AWS KMS)。
- 持久性—为确保您消息的安全，Amazon SQS 将消息存储在多个服务器上。标准队列支持[至少一次消息传递 \(p. 25\)](#)和 FIFO 队列支持[完全一次的消息处理 \(p. 27\)](#)。
- 可用性—Amazon SQS 使用[冗余基础设施 \(p. 23\)](#)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 可以处理每个[缓冲的请求 \(p. 149\)](#)独立扩展，无需任何预配置指令即可处理任何负载增加或峰值。
- 可靠性—Amazon SQS 在处理期间锁定消息，以便多个生成者可同时发送消息，多个使用者可同时接收消息。
- 自定义—您的队列不必完全相同，例如，您可以在[队列上设置默认延迟 \(p. 44\)](#)。可以存储大于 256 KB 的消息内容[Amazon Simple Storage Service \(Amazon S3\) 使用 \(p. 58\)](#)或 Amazon DynamoDB，Amazon SQS 持有指向 Amazon S3 对象的指针，您也可以将一个大消息拆分为几个小消息。

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

Amazon SQS 和[Amazon SNS](#)是高度可扩展、易于使用且不需要您设置消息代理的队列和主题服务。我们建议对新的应用程序使用这些服务，这样可以实现几乎不受限制的可扩展性和简单 API。

Amazon MQ是一项托管消息代理服务，可兼容许多常见消息代理。我们建议使用 Amazon MQ 从依赖与 API（如 JMS）或协议（如 AMQP、MQTT、OpenWire 和 STOMP）兼容的现有消息代理迁移应用程序。

队列类型

下表介绍了标准队列和 FIFO 队列的功能。

标准队列	FIFO 队列
<p>无限吞吐量— 标准队列的每个 API 操作每秒支持接近无限的 API 调用 (<code>SendMessage</code>、<code>ReceiveMessage</code>，或者<code>DeleteMessage</code>)。</p> <p>至少一次传递— 至少传递一次消息，但偶尔会传递消息的多个副本。</p> <p>尽力订购— 有时，消息传递的顺序与其发送顺序不同。</p>	<p>高吞吐量— 如果您使用 批处理 (p. 148)，则 FIFO 队列的每个 API 方法每秒最多支持 3000 条消息 (<code>SendMessageBatch</code>、<code>ReceiveMessage</code>，或者<code>DeleteMessageBatch</code>)。每秒 3000 条消息代表 300 条 API 调用，每个调用包含 10 条消息的批处理。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 (<code>SendMessage</code>、<code>ReceiveMessage</code> 或 <code>DeleteMessage</code>) 每秒最多支持 300 个 API 调用。</p> <p>确切一次处理— 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>先进先出交付— 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>当事件的顺序重要时，请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。

Amazon SQS 入门的常见任务

- 要使用 Amazon SQS 创建您的第一个队列并发送、接收和删除消息，请参阅。[Amazon SQS 入门 \(p. 7\)](#)。
- 要触发 Lambda 函数，请参阅[配置队列以触发队列 AWS Lambda 函数 \(控制台\) \(p. 16\)](#)。
- 要探索 Amazon SQS 的功能和架构，请参阅[Amazon SQS 的工作原理 \(p. 23\)](#)。
- 要找出有助于您充分利用 Amazon SQS 的准则和注意事项，请参阅。[亚马逊 SQS 的最佳实践 \(p. 49\)](#)。
- 浏览 Amazon SQS 示例，了解 AWS 软件开发工具包，例如[AWS SDK for Java 2.x 开发人员指南](#)。
- 要了解 Amazon SQS 操作的信息，请参阅[Amazon Simple Queue Service API 参考](#)。

- 了解有关 Amazon SQS 的信息 AWS CLI 命令更多讨论，请参阅 [AWS CLI 命令参考](#)。

Amazon SQS 的定价

Amazon SQS 没有前期费用。前一百万个月的请求是免费的。之后，您将根据请求的数量和内容以及与 Amazon S3 和 AWS Key Management Service。

想要了解有关信息，请参阅 [Amazon SQS 定价](#)。

设置 Amazon SQS

主题

- 第 1 步：创建 AWS 账户 (p. 4)
- 第 2 步：创建 IAM 用户 (p. 4)
- 第 3 步：获取访问密钥 ID 和秘密访问密钥 (p. 5)
- 第 4 步：为使用示例代码做好准备 (p. 6)
- 后续步骤 (p. 6)

您必须完成以下步骤，然后才能首次使用 Amazon SQS。

第 1 步：创建 AWS 账户

要访问任何AWS服务，您首先需要创建一个 **AWS 账户**，这是一个可以使用的 Amazon.com 账户 AWS 产品。您可以使用AWS 账户以查看您的活动和使用量报告并管理身份验证和访问。

为了避免使用AWS 账户根用户执行 Amazon SQS 操作，最佳实践是为需要管理权限的每个人创建一个 IAM 用户。

如需设置新账户

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，您将接到一通电话，要求您使用电话键盘输入一个验证码。

第 2 步：创建 IAM 用户

自行创建管理员用户并将该用户添加到管理员组（控制台）

1. 登录到**IAM 控制台**作为帐户所有者，方法是选择根用户并输入您的AWS账户电子邮件地址。在下一页上，输入您的密码。

Note

强烈建议您遵守使用**Administrator** IAM 用户遵守并安全锁定根用户凭证。只在执行少数**账户和服务管理任务**时才作为根用户登录。

2. 在导航窗格中，选择用户，然后选择添加用户。
3. 对于 User name (用户名)，输入 **Administrator**。
4. 选中 旁边的复选框 AWS Management Console 访问。然后选择自定义密码，并在文本框中输入新密码。
5. (可选) 默认情况下，AWS 要求新用户在首次登录时创建新密码。您可以清除 User must create a new password at next sign-in (用户必须在下次登录时创建新密码) 旁边的复选框以允许新用户在登录后重置其密码。
6. 选择 Next: Permissions (下一步：权限)。
7. 在设置权限下，选择将用户添加到组。
8. 选择创建组。
9. 在 Create group (创建组) 对话框中，对于 Group name (组名称)，输入 **Administrators**。

10. 选择筛选策略，然后选择 AWS 托管 - 作业功能来过滤表格内容。
11. 在策略列表中，选中 AdministratorAccess 的复选框。然后选择 Create group (创建组)。

Note

您必须先激活 IAM 用户和角色对账单的访问权限，然后才能使用 AdministratorAccess 权限访问 AWS Billing and Cost Management 控制台。为此，请按照“[向账单控制台委派访问权限”教程第 1 步](#)中的说明进行操作。

12. 返回到组列表中，选中您的新组所对应的复选框。如有必要，选择 Refresh 以在列表中查看该组。
13. 选择 Next:。标签。
14. (可选) 通过以键值对的形式附加标签来向用户添加元数据。有关在 IAM 中使用标签的更多信息，请参阅 [标记 IAM 实体](#) 中的 IAM 用户指南。
15. 选择 Next:。审核以查看要添加到新用户的组成员资格的列表。如果您已准备好继续，请选择 Create user。

您可使用此相同的流程创建更多的组和用户，并允许您的用户访问 AWS 账户资源。要了解有关使用限制用户对特定 AWS 资源，请参阅 [访问管理和示例策略](#)。

第 3 步：获取访问密钥 ID 和秘密访问密钥

要使用 Amazon SQS 操作（例如，使用 Java 或通过 AWS Command Line Interface），您需要访问密钥 ID 和秘密访问密钥。

Note

访问密钥 ID 和秘密访问密钥特定于 AWS Identity and Access Management。请勿与其他凭证相混淆 AWS 服务，如 Amazon EC2 密钥对。

访问密钥包含访问密钥 ID 和秘密访问密钥，用于签署对 AWS。如果没有访问密钥，您可以从 AWS Management Console。作为最佳实践，请勿使用 AWS 账户根用户访问密钥，用于不必要时的任务。相反，[创建新的管理员 IAM 用户](#)，为您自己提供访问密钥。

您只能在创建密钥时查看或下载私有访问密钥。以后您无法恢复这些属性。但是，您可以随时创建新的访问密钥。您还必须拥有执行所需 IAM 操作的权限。有关更多信息，请参阅 [访问 IAM 资源所需的权限](#) 中的 IAM 用户指南。

为 IAM 用户创建访问密钥

1. 登录 AWS Management Console，单击 <https://console.aws.amazon.com/iam/> 打开 IAM 控制台。
 2. 在导航窗格中，选择 Users。
 3. 选择要为其创建访问密钥的用户的名称，然后选择安全凭证选项卡。
 4. 在访问密钥部分，选择创建访问密钥。
 5. 要查看新的访问 key pair，请选择 Show (显式)。关闭此对话框后，您将无法再次访问该秘密访问密钥。您的凭证类似如下：
 - 访问密钥 ID：秋季花园 7 个例子
 - 秘密访问密钥：wJalrrxUtnFEMI/K7MDENG/bPxRcrFEMILEKEY
 6. 要下载密钥对，请选择下载 .csv 文件。将密钥存储在安全位置。关闭此对话框后，您将无法再次访问该秘密访问密钥。
- 请对密钥保密以保护您的 AWS 账户，切勿通过邮件发送密钥。请勿对组织外部共享密钥，即使有来自 AWS 或 Amazon.com 的询问。合法代表 Amazon 的任何人永远都不会要求您提供秘密密钥。
7. 在您下载 .csv 文件中，选择 Close。在创建访问密钥时，默认情况下，密钥对处于活动状态，并且您可以立即使用此密钥对。

相关主题

- [什么是 IAM？中的 IAM 用户指南](#)
- [AWS 安全凭证在 AWS 一般参考](#)

第 4 步：为使用示例代码做好准备

本指南包括使用 AWSSDK for Java。要运行示例代码，请按照[入门 AWS 适配 SDK for Java 2.0 开发工具包](#)。

您可以开发 AWS 应用程序，如 Go，JavaScript，Python 和 Ruby。有关更多信息，请参阅[用于开发和管理应用程序的工具 AWS](#)。

Note

Amazon SQS 无需使用诸如 AWS Command Line Interface(AWS CLI) 或 Windows PowerShell。您可以在[上找到 AWS CLI 示例 Amazon SQS 部分](#)的 AWS CLI 命令参考。您可以在 [Amazon Shell 示例](#) 中查找 Windows PowerShell 示例。[AWS Tools for PowerShell Cmdlet 参考](#)。

后续步骤

您现在已准备好[入门 \(p. 7\)](#)管理 Amazon SQS 队列和消息，使用 AWS Management Console。

Amazon SQS 入门

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。

主题

- [Prerequisites \(p. 7\)](#)
- [步骤 1: 创建队列 \(p. 7\)](#)
- [步骤 2: 发送消息 \(p. 7\)](#)
- [步骤 3: 接收和删除消息 \(p. 8\)](#)
- [步骤 4: 删除队列 \(p. 8\)](#)
- [后续步骤 \(p. 9\)](#)

Prerequisites

在开始之前，请完成 [设置 Amazon SQS \(p. 4\)](#) 中的步骤。

步骤 1: 创建队列

Amazon SQS 任务是创建队列。此过程演示如何创建和配置 FIFO 队列。

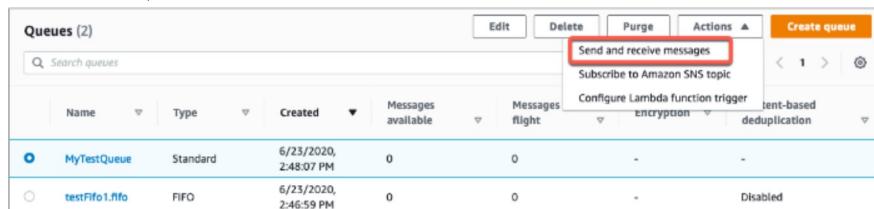
1. 打开 Amazon SQS 控制台[https://console.aws.amazon.com/sqs/。](https://console.aws.amazon.com/sqs/)
2. 选择 Create queue (创建队列)。
3. 在存储库的创建队列页面上，指定正确的区域。
4. 这些区域有：标准默认处于选中状态。选择FIFO。
创建队列后，您将无法更改队列类型。
5. 输入一个名称为您的队列。FIFO 队列的名称必须以.fifo后缀。
6. 要使用默认参数创建队列，请滚动到底部，然后选择创建队列。Amazon SQS 创建队列并显示队列的详细信息页。

Amazon SQS 在系统中传播有关新队列的信息。由于 Amazon SQS 是一个分布式系统，因此您可能会在队列显示在Queues页。

步骤 2: 发送消息

创建队列后，您可以向队列发送消息。

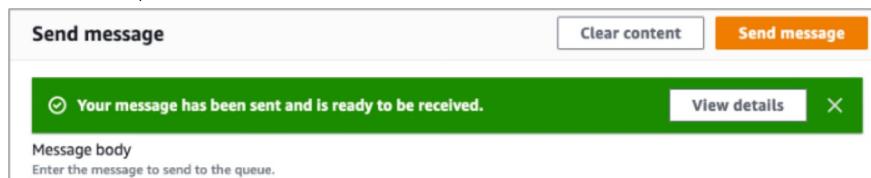
1. 从左侧导航窗格中选择Queues。从队列列表中选择您创建的队列。
2. From操作中，选择发送和接收消息。



控制台将显示发送和接收消息页。

3. 输入文本消息正文
4. 输入一个消息组 ID。有关更多信息，请参阅 [FIFO 传递逻辑 \(p. 27\)](#)。
5. (可选) 输入消息重复数据删除 ID。如果启用基于内容的重复数据消除，则不需要消息重复数据消除 ID。有关更多信息，请参阅 [FIFO 传递逻辑 \(p. 27\)](#)。
6. 选择 Send Message (发送消息)。

将发送消息，控制台会显示成功消息。选择查看详细信息以显示有关已发送消息的信息。



步骤 3: 接收和删除消息

向队列发送消息后，您可以从队列中检索消息。从队列请求消息时，您无法指定要检索的消息。而应指定要检索的最大消息数量（最多 10 条）。

1. 从Queues页面上，选择一个队列。
2. From队列操作中，选择发送和接收消息。

Queues (2)							Actions ▾	Create queue
Name	Type	Created	Messages available	Messages in flight	Encryption	Deduplication		
MyTestQueue	Standard	6/23/2020, 2:48:07 PM	0	0	-	-		
testFifo1.fifo	FIFO	6/23/2020, 2:46:59 PM	0	0	-	Disabled		

控制台将显示发送和接收消息页。

3. 选择轮询消息。

Amazon SQS 开始轮询服务器以查找队列中的消息。右侧的进度条接收消息部分显示轮询持续时间。

这些区域有：消息部分显示已接收消息的列表。对于每条消息，列表将显示邮件 ID、发送日期、大小和接收计数。

4. 要删除消息，请选择您要删除的消息，然后选择Delete。
5. 在删除消息对话框中，选择Delete。

步骤 4: 删除队列

1. 从队列列表中选择您创建的队列。
2. 从Queues页面上，选择要删除的队列。
3. 选择删除队列。

控制台将显示删除队列对话框。

4. 在删除队列对话框中，输入**delete**。

5. 选择 Delete。

后续步骤

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：

- 配置队列，包括 SSE 和其他功能 ([p. 10](#))。
- 发送带有属性的消息。([p. 19](#))
- 从 VPC 发送消息。([p. 86](#))
- 了解有关 Amazon SQS 工作流和流程的更多信息：[Read队列的工作原理 \(p. 23\)](#)、[最佳实践 \(p. 49\)](#)，[和配额 \(p. 90\)](#)。您还可以探索[Amazon SQS 文章和教程](#)。如果您有任何疑问，请浏览[Amazon SQS 常见问题](#)或参与[Amazon SQS 开发人员论坛](#)。
- 了解如何以编程方式与 Amazon SQS 交互：[Read使用 API \(p. 143\)](#)并探索[示例代码和库](#)和开发人员中心：
 - [Java](#)
 - [JavaScript](#)
 - [PHP](#)
 - [Python](#)
 - [Ruby](#)
 - [Windows 和 .NET](#)
- 在 [Amazon SQS 队列自动化和问题排查 \(p. 94\)](#)部分中了解如何监控成本和资源。
- 在 [安全性 \(p. 95\)](#)部分中了解如何保护和访问您的数据。

配置 Amazon SQS 队列 (控制台)

使用 Amazon SQS 控制台可配置和管理 Amazon Simple Queue Service (Amazon SQS) 队列和功能。您还可以使用控制台配置服务器端加密等功能、将死信队列与队列关联起来，或者设置触发器以调用 AWS Lambda function。

主题

- 了解 Amazon SQS 控制台 (p. 10)
- 创建 Amazon SQS 队列 (控制台) (p. 11)
- 编辑 Amazon SQS 队列 (p. 12)
- 配置队列参数 (控制台) (p. 12)
- 配置访问策略 (控制台) (p. 13)
- 为队列 (控制台) 配置服务器端加密 (SSE) (p. 13)
- 配置死信队列 (控制台) (p. 14)
- 为 Amazon SQS 队列配置成本分配标签 (控制台) (p. 15)
- 为 Amazon SQS 队列订阅 Amazon SNS 主题 (控制台) (p. 15)
- 配置队列以触发队列AWS Lambda函数 (控制台) (p. 16)

了解 Amazon SQS 控制台

当您打开控制台时，选择队列以显示队列页。这些区域有：队列页面提供有关活动区域中所有队列的信息。

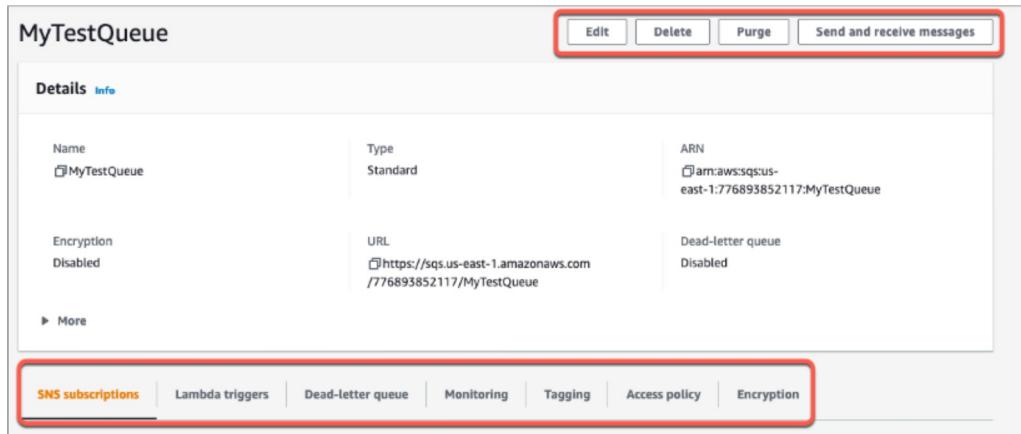
Queues (2)							
<input type="text"/> Search queues							
Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication	
<input type="radio"/> MyTestQueue	Standard	6/23/2020, 2:48:07 PM	0	0	-	-	
<input type="radio"/> testFifo1 fifo	FIFO	6/23/2020, 2:46:59 PM	0	0	-	Disabled	

每个队列的条目显示队列类型和有关队列的其他信息。这些区域有：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。

从队列页面上，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，然后选择您要在队列上执行的操作。

Queues (2)							
<input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Purge"/> <input type="button" value="Actions ▾"/> <input type="button" value="Create queue"/>							
Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication	
<input checked="" type="radio"/> MyTestQueue	Standard	6/23/2020, 2:48:07 PM	0	0	-	-	
<input type="radio"/> testFifo1 fifo	FIFO	6/23/2020, 2:46:59 PM	0	0	-	Disabled	

您还可以选择队列名称，该名称将打开详细信息页面。这些区域有：详细信息页面包含的操作与队列页。此外，您可以选择详细信息部分查看其他配置详细信息和操作。



创建 Amazon SQS 队列（控制台）

您可以使用 Amazon SQS 控制台创建[标准队列 \(p. 25\)](#)和[FIFO 队列 \(p. 25\)](#)。控制台为除队列名称之外的所有设置提供默认值。

创建 Amazon SQS 队列（控制台）

1. 打开 Amazon SQS 控制台<https://console.aws.amazon.com/sqs/>。
2. 选择 Create queue (创建队列)。
3. 适用于类型，标准队列类型是默认设置的。要创建 FIFO 队列，选择[FIFO](#)。

Note

创建队列之后，无法更改队列类型。

4. 输入名称对于队列。FIFO 队列的名称必须以.fifo后缀。
5. (可选) 控制台为队列设置默认值[配置参数 \(p. 12\)](#)。在配置，您可以为以下参数设置新值：
 - a. 适用于可见性超时中，输入持续时间和单位。范围为 0 秒到 12 小时。默认值为 30 秒。
 - b. 适用于消息保留期中，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. 适用于传输延迟中，输入持续时间和单位。范围为 0 秒到 15 分钟。默认值为 0 秒。
 - d. 适用于最大消息大小中，输入一个值。范围从 1 KB 到 256 KB。默认值为 256 KB。
 - e. 适用于接收消息等待时间中，输入一个值。范围是从 0 到 20 秒。默认值为 0 秒，设置[短轮询 \(p. 38\)](#)。任何非零值都设置长轮询。
 - f. 对于 FIFO 队列，选择启用内容重复数据删除以启用基于内容的重复数据删除。默认设置为禁用。
 - g. (可选) 对于 FIFO 队列，要为队列中的发送和接收消息启用更高的吞吐量，请选择启用高吞吐量 FIFO。

选择此选项会更改相关选项（重复数据删除范围和 FIFO 吞吐量限制）设置为启用 FIFO 队列的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，则正常吞吐量对队列有效，并按指定的方式执行重复数据消除。有关更多信息，请参阅[FIFO 队列的高吞吐量 \(p. 28\)](#)和[与消息相关的配额 \(p. 91\)](#)。

6. (可选) 定义访问策略。这些区域有：[访问策略 \(p. 122\)](#)定义帐户、用户以及能访问队列的角色。访问策略还定义了操作（例如[SendMessage](#)、[ReceiveMessage](#)，或者[DeleteMessage](#)），用户可以访问。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：

- 选择基本配置谁能向队列发送消息以及谁能从队列接收消息。控制台会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 Advanced 以直接修改 JSON 访问策略。这允许您指定每个委托人（帐户、用户或角色）可以执行的自定义操作集。
7. (可选) 配置[加密 \(p. 13\)](#)对于队列，展开加密。
 8. (可选) 要配置[死信队列 \(p. 14\)](#)以接收无法投递的邮件，请展开死信队列。
 9. (可选) 要添加[标签 \(p. 15\)](#)添加到队列中，展开标签。
 10. 选择 Create queue (创建队列)。Amazon SQS 创建队列并显示队列的详细信息页。

Amazon SQS 在系统中传播有关新队列的信息。由于 Amazon SQS 是一个分布式系统，因此您可能会在控制台在队列页。

创建队列后，您可以[发送消息 \(p. 18\)](#)添加到它，并[接收和删除消息 \(p. 19\)](#)。您还可以[编辑 \(p. 12\)](#)队列类型除外的任何队列配置设置。

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功能。

编辑 Amazon SQS 队列（控制台）

1. 打开“[队列](#)”页。
2. 选择一个队列，然后选择编辑。
3. (可选) 在配置，请更新队列的[配置参数 \(p. 12\)](#)。
4. (可选) 要更新[访问策略 \(p. 13\)](#)，在访问策略，请修改 JSON 策略。
5. (可选) 添加、更新或删除[加密 \(p. 13\)](#)，展开加密。
6. (可选) 要添加、更新或删除[死信队列 \(p. 14\)](#)（允许您接收无法投递的消息），请展开死信队列。
7. (可选) 要添加、更新或删除[标签 \(p. 15\)](#)对于队列，展开标签。
8. 选择 Save。

控制台将显示详细信息页面。

配置队列参数（控制台）

当您时[create \(p. 11\)](#)或者[编辑 \(p. 12\)](#)队列，您可以配置以下参数：

- 可见性超时—从队列（由一个使用者）接收的消息对其他消息使用者不可见的时间长度。有关更多信息，请参阅。[可见性超时 \(p. 42\)](#)。

Note

使用控制台配置可见性超时可以配置队列中所有消息的超时值。要配置单个或多个消息的超时，必须使用 AWS 开发工具包。

- 消息保留期—Amazon SQS 保留队列中保留消息的时间量。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。有关更多信息，请参阅。[消息保留期](#)。
- 配送延迟—Amazon SQS 在传递添加队列中的消息之前将延迟的时长。有关更多信息，请参阅。[配送延迟 \(p. 44\)](#)。
- 最大消息大小—此队列的最大消息大小。有关更多信息，请参阅。[最大消息大小 \(p. 58\)](#)。
- 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 \(p. 38\)](#)。

- 启用基于内容的重复数据删—Amazon SQS 可以根据消息正文自动创建重复数据消除 ID。有关更多信息，请参阅[Amazon SQS FIFO \(先进先出\) 队列 \(p. 25\)](#)。
- 启用高吞吐量 FIFO—用于为队列中的消息启用高吞吐量。选择此选项会更改相关选项（重复数据删除范围和FIFO 吞吐量限制）设置为启用 FIFO 队列的高吞吐量所需的设置。有关更多信息，请参阅[FIFO 队列的高吞吐量 \(p. 28\)](#)和[与消息相关的配额 \(p. 91\)](#)。

为现有队列配置队列参数（控制台）

- 从打开 Amazon SQS 控制台<https://console.aws.amazon.com/sqs/>。
- 在导航窗格中，选择 Queues (队列)。选择一个队列，然后选择编辑。
- 滚动到配置部分。
- 适用于可见性超时中，输入持续时间和单位。范围为 0 秒至 12 小时。默认值为 30 秒。
- 适用于消息保留期中，输入持续时间和单位。范围为 1 分钟至 14 天。默认值为 4 天。
- 适用于配送延迟中，输入持续时间和单位。范围为 0 秒至 15 分钟。默认值为 0 秒。
- 适用于最大消息大小中，输入一个值。范围为 1 KB 到 256 KB。默认值为 256 KB。
- 对于标准队列，输入接收消息等待时间。范围为 0 至 20 秒。默认值为 0 秒，该值将[短轮询 \(p. 38\)](#)。任何非零值都设置长轮询。
- 对于 FIFO 队列，请选择启用基于内容的重复数据删启用基于内容的重复数据删除。默认设置是禁用。
- （可选）对于 FIFO 队列，要为队列中的发送和接收消息启用更高的吞吐量，请选择启用高吞吐量 FIFO。

选择此选项会更改相关选项（重复数据删除范围和FIFO 吞吐量限制）设置为启用 FIFO 队列的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，则正常吞吐量对队列有效，并按指定的方式执行重复数据消除。有关更多信息，请参阅[FIFO 队列的高吞吐量 \(p. 28\)](#)和[与消息相关的配额 \(p. 91\)](#)。

- 完成配置队列参数后，选择Save。

配置访问策略（控制台）

当您时[编辑 \(p. 12\)](#)一个队列，您可以配置其访问策略。

访问策略定义了可以访问队列的帐户、用户和角色。访问策略还定义了操作（例如Send Message、Receive Message，或者Delete Message），用户可以访问。默认策略仅允许队列所有者发送和接收消息。

配置现有队列（控制台）的访问策略

- 打开 Amazon SQS 控制台<https://console.aws.amazon.com/sqs/>。
- 在导航窗格中，选择 Queues (队列)。
- 选择一个队列，然后选择编辑。
- 滚动到访问策略部分。
- 编辑输入框中的访问策略语句。
- 配置完访问策略后，选择Save。

为队列（控制台）配置服务器端加密 (SSE)

要保护队列消息中的数据，您可以为队列启用服务器端加密 (SSE)。Amazon SQS 与 AWS Key Management Service(AWS KMS) 来管理[客户主密钥](#)(CMK) 用于服务器端加密 (SSE)。有关使用 SSE 的信息，请参阅[静态加密 \(p. 96\)](#)。

您分配给队列的 CMK 必须具有一个密钥策略，其中包括有权使用该队列的所有委托人的权限。想要了解有关信息，请参阅[密钥管理 \(p. 98\)](#)。

如果您不是 CMK 的拥有者，或者您没有使用拥有[kms>ListAliases](#)和[kms>DescribeKey](#)权限，则无法查看有关 Amazon SQS 控制台上的 CMK 的信息。要求 CMK 拥有者授予您这些权限。有关更多信息，请参阅[密钥管理 \(p. 98\)](#)。

当您时[create \(p. 11\)](#)或者[编辑 \(p. 12\)](#)一个队列，您可以配置 SSE。

为现有队列配置 SSE (控制台)

1. 从打开 Amazon SQS 控制台<https://console.aws.amazon.com/sqs/>。
 2. 在导航窗格中，选择 Queues (队列)。
 3. 选择一个队列，然后选择编辑。
 4. 滚动到加密部分。
 5. 在加密部分，选择Enabled (已启用)以启用 SSE。
- 此控制台将显示说明，账户，以及CMK ARN CMK 的。
6. 指定队列的 CMK ID。有关更多信息，请参阅[关键术语 \(p. 97\)](#)。
 - a. 选择选择 CMK 别名选项。
 - b. 默认密钥为AWSAmazon SQS 的托管 CMK。要使用此键，请从客户主密钥列出。
 - c. 要使用来自的自定义 CMK AWS 账户，请从客户主密钥列出。有关如何创建自定义 CMK 的说明，请参阅[创建密钥](#)中的AWS Key Management Service开发人员指南。
 - d. 使用列表中不包含的自定义 CMK，或者使用来自其他的自定义 CMK AWS 账户中，选择输入 CMK 别名并输入 CMK Amazon 资源名称 (ARN)。
 7. (可选) 数据密钥重用周期，请指定介于 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 \(p. 100\)](#)。
 8. 完成 SSE 配置后，选择Save。

配置死信队列 (控制台)

A死信队列是一个队列，一个或多个源队列可用于未成功使用的消息。有关更多信息，请参阅[Amazon SQS 死信队列 \(p. 40\)](#)。

Amazon SQS不是自动创建死信队列。必须先创建队列，然后才能将其用作死信队列。

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。

当您时[create \(p. 11\)](#)或者[编辑 \(p. 12\)](#)队列，您可以配置死信队列。

为现有队列配置死信队列 (控制台)

1. 打开 Amazon SQS 控制台<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (队列)。
3. 选择一个队列，然后选择编辑。
4. Scroll II Scroll II死信队列部分并选择启用。
5. 选择某个现有的 Amazon 资源名称 (ARN)死信队列您希望与此源队列关联的内容。
6. 要配置消息在发送到死信队列之前可接收的次数，请将最大接收数量设置为一个介于 1 和 1000 之间的值。
7. 配置死信队列后，请选择Save。

保存队列后，控制台将显示详细信息页面。在存储库的详细信息页面上，死信队列选项卡显示最大接收数量和死信队列中的 ARN死信队列。

为 Amazon SQS 队列配置成本分配标签（控制台）

为帮助组织和标识 Amazon SQS 队列，您可以向这些队列添加标签。有关更多信息，请参阅 [Amazon SQS 成本分配标签 \(p. 38\)](#)。

在存储库的详细信息页面中，标记选项卡显示队列的标签。

当您时[create \(p. 11\)](#)或者[编辑 \(p. 12\)](#)一个队列，您可以为其配置标签。

为现有队列配置标签（控制台）

1. 从打开 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. 滚动到Tags部分。
5. 添加、修改或删除队列标签：
 - a. 要添加标签，请选择添加新标签中，输入密钥和值，然后选择添加新标签。
 - b. 要更新标签，请更改其密钥和值。
 - c. 要删除标签，请选择Remove旁边的键值对。
6. 完成配置标签后，选择Save。

为 Amazon SQS 队列订阅 Amazon SNS 主题（控制台）

可以为一个或多个 Amazon SQS 队列订阅 Amazon Simple Notification Service (Amazon SNS) 主题。当您将一条消息发布到某个主题时，Amazon SNS 会向每个已订阅队列发送此消息。Amazon SQS 管理订阅和任何必要的权限。有关 Amazon SNS 的更多信息，请参阅[什么是 Amazon Simple Notification Service？](#)中的Amazon Simple Notification Service 开发人员指南。

当您为 Amazon SQS 队列订阅 SNS 主题时，Amazon SNS 使用 HTTPS 将消息转发到 Amazon SQS。有关将 Amazon SNS 与加密 Amazon SQS 队列结合使用的信息，请参阅[配置 AWS 服务的 KMS 权限 \(p. 98\)](#)。

为队列订阅 SNS 主题（控制台）

1. 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. 从操作中，选择订阅 Amazon SNS 主题。
5. 从为此队列指定可用的 Amazon SNS 主题菜单中，为队列选择 SNS 主题。

如果菜单中未列出 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 Save。

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 [向 Amazon SNS 主题发布消息](#) 中的 Amazon Simple Notification Service 开发人员指南。

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

有关订阅跨区域 SNS 主题的信息，请参阅[将 Amazon SNS 消息发送到 Amazon SQS 队列或 AWS Lambda 函数在不同的区域](#)中的 Amazon Simple Notification Service 开发人员指南

配置队列以触发队列AWS Lambda函数 (控制台)

您可以使用 AWS Lambda 函数来处理 Amazon SQS 队列中的消息。Lambda 轮询该队列，并通过一个包含队列消息的事件来同步调用 Lambda 函数。您可以指定另一个队列作为死信队列来获取您的 Lambda 函数无法处理的消息。

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。

如果将加密队列与 Lambda 函数相关联，但 Lambda 不轮询消息，请添加 `kms:Decrypt` 权限设置为您的 Lambda 执行角色。

注意以下限制：

- 您的队列和 Lambda 函数必须位于同一 AWS 区域。
- 一个[加密队列 \(p. 96\)](#)，它使用默认密钥 (AWS 托管 CMK) 无 Lambda 不同的 AWS 账户。

有关实施 Lambda 函数的信息，请参阅[使用 AWS Lambda 通过 Amazon SQS 中的 AWS Lambda](#) 开发人员指南。

Prerequisites

要配置 Lambda 函数触发器，必须满足以下要求：

- 如果您使用 IAM 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`
- 如果将加密队列与 Lambda 函数相关联，请将 `kms:Decrypt` 权限设置为 Lambda 执行角色。

有关更多信息，请参阅[Amazon SQS 中的访问管理概述 \(p. 104\)](#)。

配置队列以触发 Lambda 函数 (控制台)

1. 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. 在存储库的队列页面上，选择要配置的队列。
4. 在队列的页面上，选择 Lambda 触发器选项卡。

5. 在存储库的Lambda 触发器页面上，选择一个 Lambda 触发器。

如果列表中不包含您需要的 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 Save (保存)。

6. 选择 Save。控制台将保存配置并显示详细信息页面。

在存储库的详细信息页面上，Lambda 触发器选项卡显示 Lambda 函数及其状态。Lambda 函数大约需要 1 分钟时间与您的队列关联。

7. 要验证配置的结果，[向队列发送消息 \(p. 18\)](#)，然后在 Lambda 控制台查看触发 Lambda 函数。

管理 Amazon SQS 队列（控制台）

创建和配置队列后，您可以使用 Amazon SQS 控制台向队列发送消息，然后从队列中进行检索。

主题

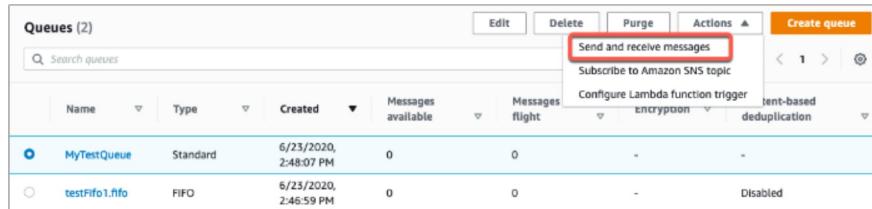
- 向队列发送消息（控制台）(p. 18)
- 发送带有属性的消息（控制台）(p. 19)
- 接收和删除消息（控制台）(p. 19)
- 从 Amazon SQS 队列中清除消息（控制台）(p. 20)
- 删除 Amazon SQS 队列 (p. 20)
- 确认队列为空 (p. 21)

向队列发送消息（控制台）

创建队列后，您可以向其发送消息。

发送消息（控制台）

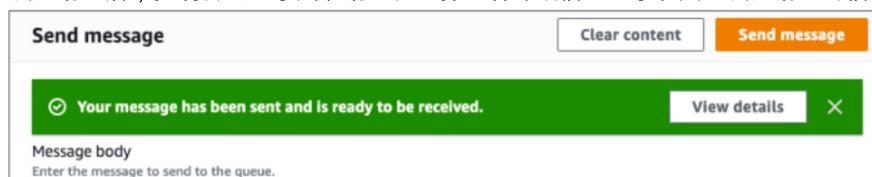
1. 从打开 Amazon SQS 控制台<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (队列)。
3. 在存储库的Queues页面上，选择队列。
4. From操作中，选择发送和接收消息。



控制台将显示发送和接收消息页。

5. 在消息正文中，输入消息文本。
6. 对于先进先出 (FIFO) 队列，输入消息组 ID。有关更多信息，请参阅 [FIFO 传递逻辑 \(p. 27\)](#)。
7. (可选) 对于 FIFO 队列，您可以输入消息重复数据删除 ID。如果为队列启用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 [FIFO 传递逻辑 \(p. 27\)](#)。
8. (可选) 对于标准队列，您可以为配送延迟，然后选择单位。例如，输入60并选择秒。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 消息定时器 \(p. 48\)](#)。
9. 选择 Send Message (发送消息)。

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



发送带有属性的消息 (控制台)

对于标准队列和 FIFO 队列，您可以在消息中包含结构化元数据（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 \(p. 35\)](#)。

将带有属性的消息发送到队列 (控制台)

1. 打开 Amazon SQS 控制台，网址为 <https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (队列)。
3. 在存储库的Queues页面上，选择队列。
4. From操作中，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，请选择字符串、数字，或者二进制。
 - c. (可选) 输入自定义数据类型。例如，您可以将 `byte`、`int`，或者 `float` 作为自定义数据类型数
字。
 - d. 在值文本框中，输入消息属性值。

▼ Message attributes - Optional [Info](#)

Enter name String Custom type Enter value

Add new attribute

6. 要添加其他消息属性。, 请选择Add Attributes。

▼ Message attributes - Optional [Info](#)

Enter name String Custom type Enter value

Enter name String Custom type Enter value Remove

Add new attribute

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 Remove。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中之后，选择发送消息。将发送消息并控制台显示成功消息。要查看有关已发送邮件的邮件属性的信息，请选择查看详细信息。选择 Done 关闭消息详细信息对话框。

接收和删除消息 (控制台)

将消息发送到队列后，您可以接收和删除它们。当您从队列请求消息时，您不能指定要检索的消息。而应指定要检索的最大消息数量（最多 10 条）。

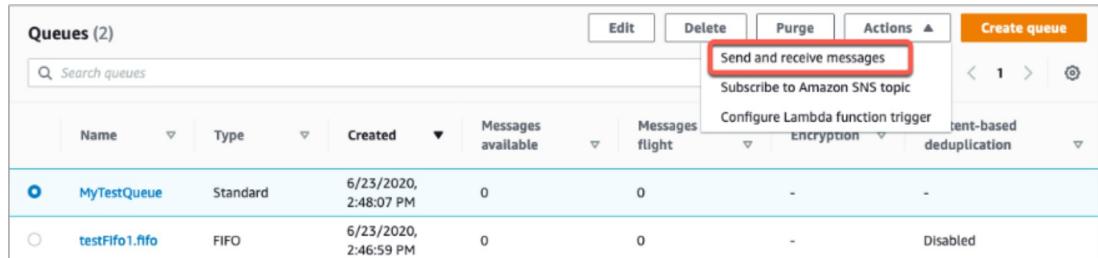
Note

由于 Amazon SQS 是分布式系统，因此消息极少的队列可能会显示对接收请求的空响应。在这种情况下，请重新运行请求以获取您的消息。根据您的应用程序的需求，您可能必须使用 [短轮询或长轮询 \(p. 38\)](#) 接收消息。

Amazon SQS 不会在为您检索消息之后自动删除它，以防您未成功地接收消息（例如，如果使用者出现故障或者您断开连接）。要删除某个消息，您必须发送确认您已成功接收并处理它的单独请求。请注意，您必须收到一条消息，然后才能将其删除。

接收消息和删除消息 (控制台)

1. 打开 Amazon SQS 控制台[https://console.aws.amazon.com/sqs/。](https://console.aws.amazon.com/sqs/)
2. 在导航窗格中，选择 Queues (队列)。
3. 在存储库的Queues页面上，选择队列。
4. From操作中，选择发送和接收消息。



控制台将显示发送和接收消息页。

5. 选择轮询消息。

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

这些区域有：消息部分显示已接收消息的列表。对于每封邮件，列表将显示邮件 ID、发送日期、大小和接收计数。

6. 要删除消息，请选择要删除的消息，然后选择Delete。
7. 在删除消息对话框中，选择Delete。

从 Amazon SQS 队列中清除消息 (控制台)

如果您不想删除 Amazon SQS 队列，但需要删除队列中的所有消息，请使用清除队列。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

当您清除队列时，您不能检索任何已删除的消息。

清除队列 (控制台)

1. 从打开 Amazon SQS 控制台[https://console.aws.amazon.com/sqs/。](https://console.aws.amazon.com/sqs/)
2. 在导航窗格中，选择 Queues (队列)。
3. 在存储库的Queues页面上，选择要清除的队列。
4. 选择清除。
5. 在清除队列对话框中，通过输入并选择清除。

所有消息将从队列中清除。控制台显示确认横幅。

删除 Amazon SQS 队列

如果您不再使用 Amazon SQS 队列，并且预计在不久的将来不会使用该队列，我们建议将其删除。

Tip

如果要在删除队列之前验证队列是否为空，请参阅[确认队列为空 \(p. 21\)](#)。

您可以删除队列，即使它不为空。删除队列中的消息，但不删除队列本身，[清除队列 \(p. 20\)](#)。

删除队列 (控制台)

1. 从打开 Amazon SQS 控制台<https://console.aws.amazon.com/sqs/>。
2. 在导航窗格中，选择 Queues (队列)。
3. 在存储库的队列页面上，选择要删除的队列。
4. 选择 Delete (删除)。
5. 在删除队列对话框中，输入**delete**。
6. 选择 Delete (删除)。

删除队列 (AWS CLI AWSAPI)

您可以使用以下命令之一删除队列：

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

确认队列为空

在大部分情况下，您可以使用[长轮询 \(p. 39\)](#)来确定队列是否为空。在极少数情况下，即使队列仍包含消息，您也可能会收到空响应，特别是如果接收消息等待时间创建队列时。此部分介绍如何确认队列为空。

确认队列为空 (控制台)

1. 阻止所有生产者发送消息。
2. 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. 在存储库的队列页面上，选择队列。
5. 选择 Monitoring 选项卡。
6. 在“监视”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。离开刷新间隔at1分钟。
7. 观察以下仪表板：
 - 延迟的消息大致数
 - 不可见的消息大致数
 - 可见消息的大致数

当他们都显示0值，说明队列为空。

要确认队列为空 (AWS CLI、AWSAPI)

1. 阻止所有生产者发送消息。
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

当他们都是0，说明队列为空。

如果您依赖 Amazon CloudWatch 指标，请确保在将队列视为空之前看到多个连续零数据点。有关 CloudWatch 指标的更多信息，请参阅[适用于 Amazon SQS 的可用 CloudWatch 指标 \(p. 135\)](#)。

Amazon SQS 的工作原理

本节描述 Amazon SQS 队列的类型及其基本属性。它还描述了队列和消息的标识符以及各种队列和消息管理的工作流程。

主题

- [Amazon SQS 基本架构 \(p. 23\)](#)
- [Amazon SQS 标准队列 \(p. 25\)](#)
- [Amazon SQS FIFO \(先进先出\) 队列 \(p. 25\)](#)
- [Amazon SQS 队列和消息标识符 \(p. 33\)](#)
- [消息元数据 \(p. 34\)](#)
- [处理 Amazon SQS 消息所需的资源 \(p. 37\)](#)
- [列出队列分页 \(p. 38\)](#)
- [Amazon SQS 成本分配标签 \(p. 38\)](#)
- [Amazon SQS 短轮询和长轮询 \(p. 38\)](#)
- [Amazon SQS 死信队列 \(p. 40\)](#)
- [Amazon SQS 可见性超时 \(p. 42\)](#)
- [Amazon SQS 延迟队列 \(p. 44\)](#)
- [Amazon SQS 临时队列 \(p. 45\)](#)
- [Amazon SQS 消息定时器 \(p. 48\)](#)

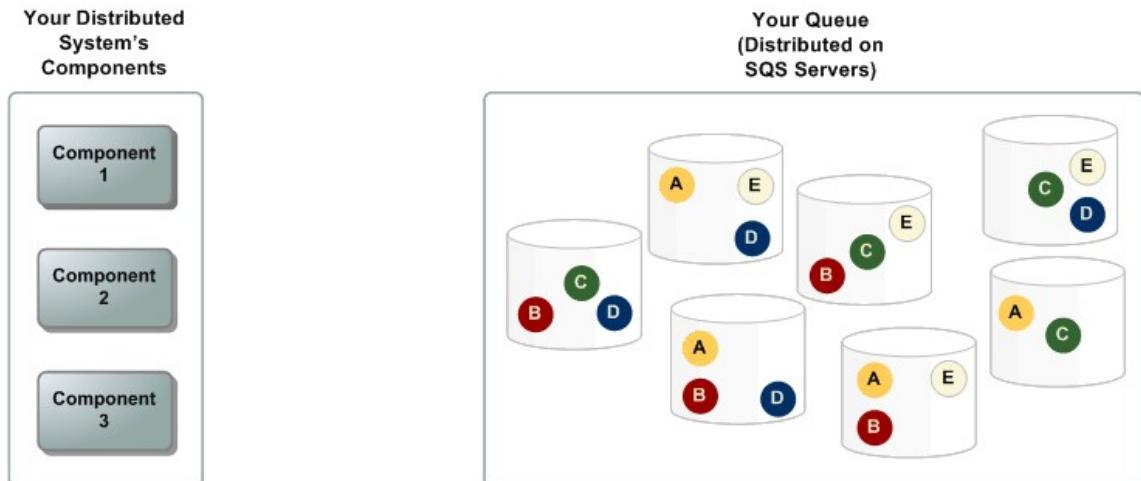
Amazon SQS 基本架构

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

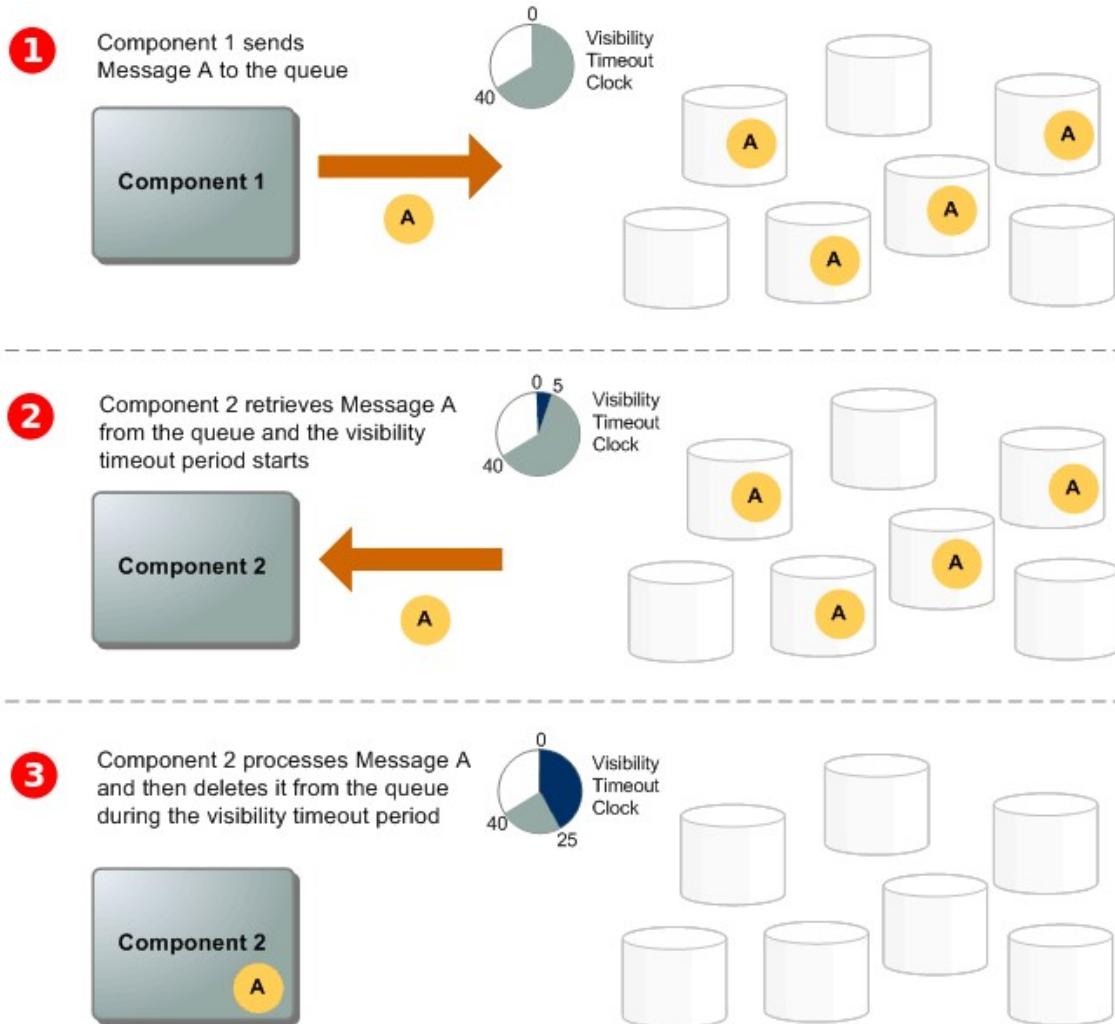
分布式消息传送系统有三个主要组成部分：分布式系统的组件、队列（分布在 Amazon SQS 服务器上）以及队列中的消息。

在下面的情况下，您的系统具有多个创建器（向队列发送消息的组件）以及使用者（从队列接收消息的组件）。队列（保存从 A 到 E 的消息）在多个 Amazon SQS 服务器上冗余存储消息。



消息生命周期

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



① 创建者 (Component 1) 将 Message A 发送到一个队列，该消息在 Amazon SQS 服务器间冗余分布。

② 使用者 (Component 2) 准备好处理消息时，就从队列使用消息，然后返回 Message A。在处理 Message A 期间，它仍保留在队列中，并且在[可见性超时 \(p. 42\)](#)期间不返回至后续接收请求。

③ 使用者 (Component 2) 从队列中删除 Message 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 标准队列

Amazon SQS 报价标准作为默认队列类型。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。标准队列支持至少一次消息传递。但是，由于存在允许近乎无限吞吐量的高度分布式架构，偶尔会有一条消息的多个副本不按顺序传送。标准队列会尽最大努力进行排序，保证了消息大致按其发送的顺序进行传送。

有关如何使用 Amazon SQS 控制台创建和配置队列的信息，请参阅[创建 Amazon SQS 队列（控制台）\(p. 11\)](#)。有关 Java 示例，请参阅[Amazon SQS Java 软件开发工具包示例 \(p. 54\)](#)。

您可以在很多情况下使用标准消息队列（只要应用程序能够处理多次到达和不按顺序到达的消息），例如：

- 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。
- 将任务分配到多个工作人员节点—处理大量信用卡验证请求。
- Batch 消息以供将来处理—计划要添加到数据库的多个条目。

有关与标准队列相关的配额，请参阅[与队列相关的配额 \(p. 90\)](#)。

有关使用标准队列的最佳实践，请参阅[针对 Amazon SQS 标准队列和 FIFO 队列的建议 \(p. 49\)](#)。

消息排序

标准队列会尽量保持消息顺序，但可能有一条消息的多个副本可能不按顺序传送。如果您的系统需要保留订单，我们建议您使用[FIFO（先进先出）队列 \(p. 25\)](#)，或者在每条消息中添加排序信息，以便在收到消息后对其进行重新排序。

至少一次传递

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

如果出现这种情况，则该不可用服务器上的消息副本将不会被删除，并且您在接收消息时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 FIFO（先进先出）队列

FIFO 队列具有[标准队列 \(p. 25\)](#)。

FIFO（先进先出）队列能在操作和事件的顺序很重要或者不能容忍重复消息的情况下改善应用程序之间的消息收发。您可能使用 FIFO 队列的情况示例如下：

- 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。

FIFO 队列还提供“只执行一次”的处理，但每秒事务数 (TPS) 有限。有关吞吐量配额的信息，请参阅[与消息相关的配额 \(p. 91\)](#)。

Note

FIFO 队列的名称必须以`.fifo`后缀。后缀计入 80 个字符的队列名称配额。若要确定队列是否[FIFO \(p. 25\)](#)，您可以检查队列名称是否以后缀结尾。

Amazon SQS FIFO 队列在所有提供的区域 Amazon SQS 可用

有关如何使用 Amazon SQS 控制台创建和配置队列的信息，请参阅[创建 Amazon SQS 队列（控制台）\(p. 11\)](#)。有关 Java 示例，请参阅[Amazon SQS Java 软件开发工具包示例 \(p. 54\)](#)。

有关使用 FIFO 队列的最佳实践，请参阅[针对 Amazon SQS FIFO 队列的其他建议 \(p. 52\)](#)和[针对 Amazon SQS 标准队列和 FIFO 队列的建议 \(p. 49\)](#)。

有关客户端和服务与 FIFO 队列的兼容性的信息，请参阅[Compatibility \(p. 33\)](#)。

主题

- [消息排序 \(p. 26\)](#)
- [关键术语 \(p. 26\)](#)
- [FIFO 传递逻辑 \(p. 27\)](#)
- [确切一次处理 \(p. 27\)](#)
- [从标准队列移至 FIFO 队列 \(p. 28\)](#)
- [FIFO 队列的高吞吐量 \(p. 28\)](#)
- [Compatibility \(p. 33\)](#)

消息排序

FIFO 队列改进并补充了标准队列 ([p. 25](#))。此队列类型最重要的特性是[FIFO \(先进先出\) 配送 \(p. 27\)](#)和[确切一次处理 \(p. 27\)](#)：

- 发送和接收消息的顺序严格保持一致；一条消息传送一次后就保持可用，直到使用者处理并删除它为止。
- 不会将重复项引入到队列中。

此外，FIFO 队列支持消息组，该组允许一个队列中存在多个有序的消息组。FIFO 队列中的消息组数量没有限额。

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](#)。

消息重复数据删除 ID

用于消除已发送消息的重复数据的令牌。如果成功发送具有特定消息重复数据消除 ID 的邮件，则会成功接受具有相同消息重复数据消除 ID 的任何邮件，但不会在 5 分钟的重复数据消除间隔内传递。

Note

消息重复数据消除适用于整个队列，而不适用于单个邮件组。

即使在收到和删除消息之后，Amazon SQS 也会继续跟踪消息重复数据消除 ID。

消息组 ID

指定消息属于特定消息组的标记。属于同一消息组的邮件始终按照相对于消息组的严格顺序逐个处理（但是，属于不同消息组的邮件可能会按顺序处理）。

接收请求尝试 ID

用于重复数据消除的令牌[ReceiveMessage](#)调用。

序列号

Amazon SQS 为每条消息分配的大型非连续数字。

FIFO 传送逻辑

以下概念可帮助您更好地了解 FIFO 的消息收发。

发送消息

如果多条消息连续发送到 FIFO 队列（每条消息具有不同的消息重复数据删除 ID），Amazon SQS 将存储消息并确认传输。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，Amazon SQS 将按消息到达以供处理的顺序存储消息。要确保 Amazon SQS 保留发送和接收消息的顺序，每位创建者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 代表 Amazon SQS 队列中不同的有序的消息组。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 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 会首先尝试尽可能多地返回具有相同消息组 ID 的消息。这时其他使用者能处理具有不同消息组 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，则您可接收来自属于 10 条消息的同一批中其他消息组 ID 的消息，但仍按 FIFO 顺序。

多次重试

FIFO 队列允许生产者或使用者尝试多次重试：

- 如果生产者检测到失败 `SendMessage` 操作时，它可以使用相同的消息重复数据删除 ID，根据需要重试发送多次。假设创建者在重复数据删除间隔过期之前至少收到一个确认，多次重试既不会影响邮件的排序，也不会引入重复项。
- 如果使用者检测到失败 `ReceiveMessage` 操作，它可以根据需要重试次数，使用相同的接收请求尝试 ID。假设使用者在可见性超时到期之前至少收到一个确认，多次重试不会影响消息的排序。
- 当您收到具有消息组 ID 的邮件时，除非您删除该邮件或其变为可见，否则不会返回同一消息组 ID 的其他消息。

确切一次处理

与标准队列不同，FIFO 队列不会引入重复消息。FIFO 队列可帮助您避免向队列发送重复消息。如果您重试 `SendMessage` 操作时，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 [CreateQueue](#)、[GetQueueAttributes](#) 和 [SetQueueAttributes](#) 中的操作 Amazon Simple Queue Service API 参考。
- 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 [Send Message](#)、[SendMessageBatch](#) 和 [ReceiveMessage](#) 中的操作 Amazon Simple Queue Service API 参考。

从标准队列移至 FIFO 队列

如果有使用标准队列的现有应用程序并想要利用 FIFO 队列的排序或确切一次处理功能，则需要正确地配置队列和应用程序。

Note

您无法将现有标准队列转换为 FIFO 队列。要实现转移，必须为应用程序创建新的 FIFO 队列，或者删除现有标准队列并重新将其创建为 FIFO 队列。

要确保您的应用程序能够正确地与 FIFO 队列配合使用，请使用以下核对表：

- 如果您使用[批处理 \(p. 148\)](#)，则 FIFO 队列的每个 API 方法每秒最多支持 3000 条消息 (`SendMessageBatch`、`ReceiveMessage`，或者`DeleteMessageBatch`)。每秒 3000 条消息代表 300 条 API 调用，每个调用带有包含 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。
- FIFO 队列不支持每消息延迟，仅支持每队列延迟。如果您的应用程序在每条消息上设置相同的 `DelaySeconds` 参数值，您必须将应用程序修改为删除每消息延迟并改为在整个队列上设置 `DelaySeconds`。
- 发送到 FIFO 队列的每条消息都需要消息组 ID。如果您不需要多个有序的消息组，请为您的所有消息指定相同的消息组 ID。
- 在将消息发送到 FIFO 队列之前，请确认以下内容：
 - 如果您的应用程序可发送具有相同的消息正文的消息，您可以将应用程序修改为针对每条已发送消息提供唯一的消息重复数据删除 ID。
 - 如果您的应用程序发送具有独特的消息正文的消息，您可以启用基于内容的重复数据删除。
- 您不必对使用者进行任何代码更改。但是，如果处理消息需要较长时间且您的可见性超时已设置为一个较高值，请考虑向每个 `ReceiveMessage` 操作添加接收请求尝试 ID。这允许您在网络发生故障时重试接收尝试，并防止队列由于接收尝试失败而导致的暂停。

有关更多信息，请参阅 [Amazon Simple Queue Service API 参考](#)。

FIFO 队列的高吞吐量

高吞吐量[FIFO 队列 \(p. 25\)](#)支持每个 API 每秒更多的请求数。若要增加 FIFO 队列的高吞吐量请求数，可以增加所使用的消息组数。每个消息组支持每秒 300 个请求。有关 FIFO 配额具有高吞吐量的每队列配额的信息，请参阅[与消息相关的配额 \(p. 91\)](#)和[为 SQS FIFO 队列提供高吞吐量的分区和数据分配 \(p. 29\)](#)。

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，该功能包括三个新选项：

- 启用高吞吐量 FIFO—为当前 FIFO 队列中的消息提供更高的吞吐量。
- 重复消除范围—指定是在队列级别还是消息组级别进行重复数据消除。
- 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。

启用 FIFO 队列（控制台）的高吞吐量

- 启动[creating \(p. 11\)](#)或者[编辑 \(p. 12\)](#)一个先进先出队列。
- 为队列指定选项时，选择启用高吞吐量 FIFO。

启用 FIFO 队列的高吞吐量可设置相关选项，如下所示：

- 重复消除范围设置为消息组，这是对 FIFO 队列使用高吞吐量所需的设置。
- FIFO 吞吐量限制设置为每消息组 ID，这是对 FIFO 队列使用高吞吐量所需的设置。

如果更改了为 FIFO 队列使用高吞吐量所需的任何设置，则正常吞吐量对队列有效，并按指定的方式执行重复数据消除。

3. 继续指定队列的所有选项。完成后，选择创建队列或者Save。

创建或编辑 FIFO 队列后，您可以发送消息 ([p. 18](#)) 到它和接收和删除消息 ([p. 19](#))，所有这些都在一个更高的 TPS。

为 SQS FIFO 队列提供高吞吐量的分区和数据分配

Amazon SQS 将 FIFO 队列数据存储在分区。A 分区是为队列分配的存储，这些分配在队列上可用区中自动复制 AWS 区域。您不管理分区。相反，Amazon SQS 处理分区管理。

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

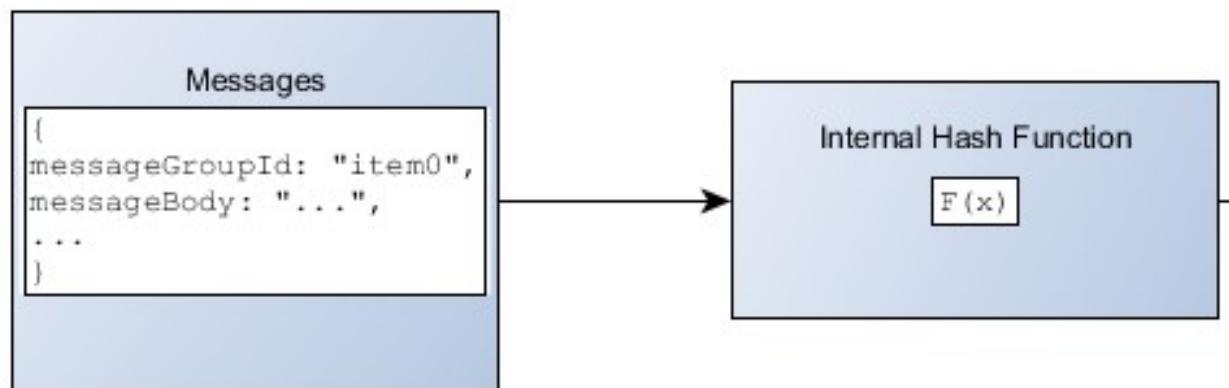
- 如果当前请求数率接近或超过现有分区可以支持的速率，则会分配额外的分区，直到队列达到区域配额。
有关配额的信息，请参阅[与消息相关的配额 \(p. 91\)](#)。
- 如果当前分区的利用率较低，则分区的数量可能会减少。

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

要将消息添加到 FIFO 队列，Amazon SQS 会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存储消息的分区。

下图显示了跨多个分区的队列。队列的消息组 ID 基于项目编号。Amazon SQS 使用其哈希函数决定新项目的存储位置；在这种情况下，它将基于字符串的哈希值 item0。请注意，项目的存储顺序与添加到队列中的顺序相同。每个项目的位置由其消息组 ID 的哈希值决定。



Note

Amazon SQS 经过优化，不论 FIFO 队列的分区数如何，都可在这些分区上统一分配项目。AWS 建议您使用具有较多非重复值的消息组 ID。

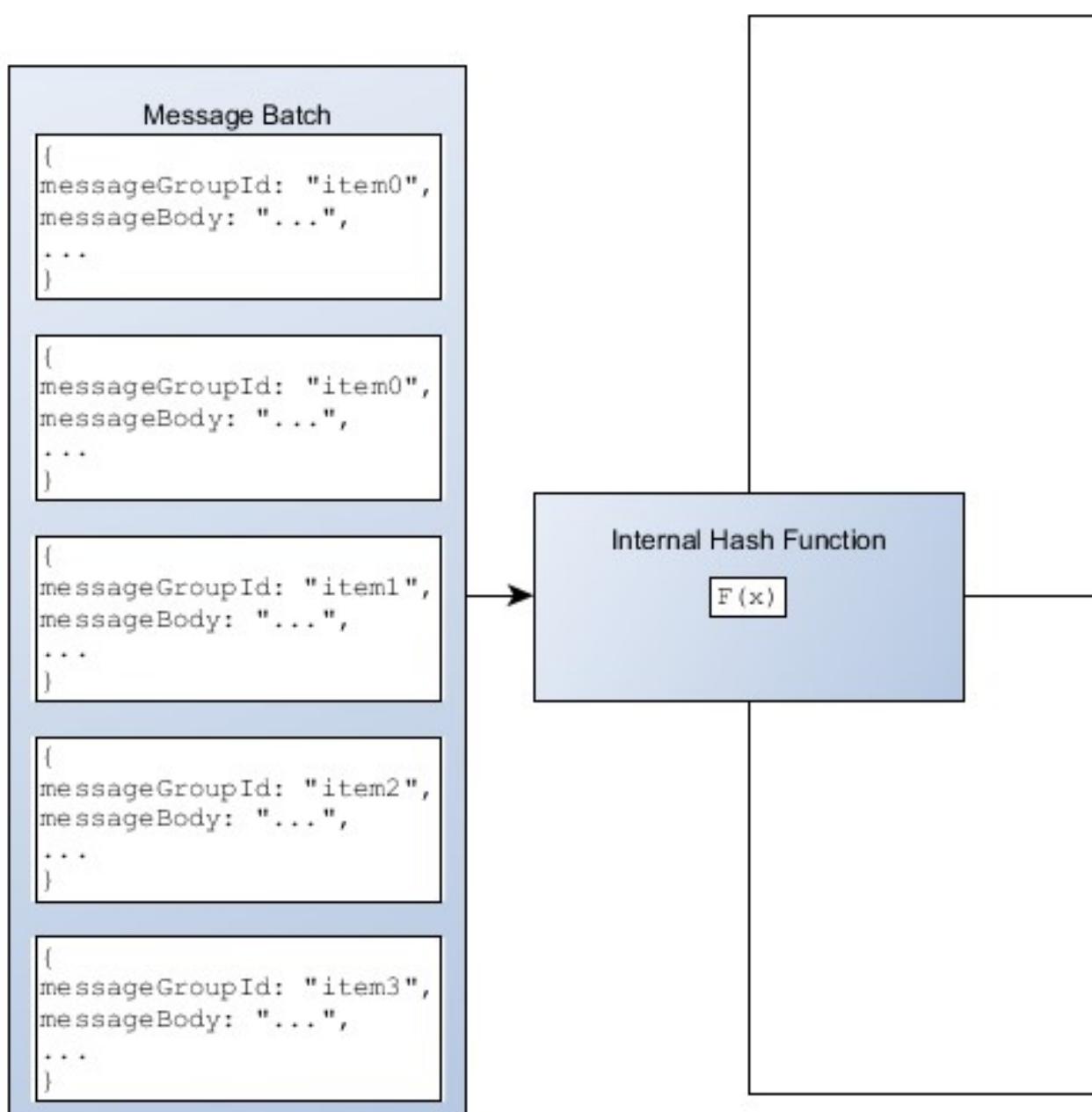
优化分区使用率

每个分区支持带有批处理的消息每秒最多 3000 条，或者每秒支持 300 条消息，用于发送、接收和删除操作。

使用批处理 API 时，每条消息都会根据[按消息组 ID 分发数据 \(p. 29\)](#)。路由到同一分区的邮件在单个事务中进行分组和处理。

要优化分区利用率，AWS 建议尽可能使用相同消息组 ID 批处理消息。

在以下示例中，将发送一批具有各种消息组 ID 的消息。批处理被拆分为三个组，每个组都根据分区的配额进行计数。



Note

Amazon SQS 仅保证具有相同消息组 ID 的消息在批处理请求中分组。根据内部哈希函数的输出和分区的数量，可能会对具有不同消息组 ID 的邮件进行分组。由于哈希函数或分区数量可以随时更改，因此在某一点分组的消息以后可能不会对其进行分组。

Compatibility

客户

Amazon SQS 缓冲异步客户端当前不支持 FIFO 队列。

服务

如果您的应用程序使用多个AWS服务，或者AWS和外部服务，请务必了解哪些服务功能不支持 FIFO 队列。

一段时间AWS或向 Amazon SQS 发送通知的外部服务可能与 FIFO 队列不兼容，尽管允许您将 FIFO 队列设置为目标。

以下功能AWS服务当前与 FIFO 队列不兼容：

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

有关其他服务与 FIFO 队列的兼容性的信息，请参阅服务文档。

Amazon SQS 队列和消息标识符

本部分主要介绍标准队列和 FIFO 队列的标识符。这些标识符可帮助您查找并操作特定队列和消息。

主题

- [Amazon SQS 标准队列和 FIFO 队列的标识符 \(p. 33\)](#)
- [Amazon SQS FIFO 队列的其他标识符 \(p. 34\)](#)

Amazon SQS 标准队列和 FIFO 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，其中包括队列名称和其他 Amazon SQS 组件。每当您要对队列执行操作时，都需要提供其队列 URL。

FIFO 队列的名称必须以`.fifo`后缀。后缀计入 80 个字符的队列名称配额。确定队列是否为 [FIFO \(p. 25\)](#)，您可以检查队列名称是否以后缀结尾。

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

每条消息都接收一个系统分配的消息 IDAmazon SQS 在[SendMessage](#)响应。此标识符用于识别消息。(但，要删除消息，您需要消息的接收句柄。) 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/WeR4mq2OKpEGYWbnLmpRCJVAYeMjeU5ZBdtcQ+QE
aUMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Amazon SQS FIFO 队列的其他标识符

有关以下标识符的更多信息，请参阅[确切一次处理 \(p. 27\)](#)和[Amazon Simple Queue Service API 参考](#)。

消息重复数据删除 ID

用于消除已发送消息的重复数据的令牌。如果成功发送具有特定消息重复数据消除 ID 的邮件，则会成功接受具有相同消息重复数据消除 ID 的任何邮件，但不会在 5 分钟的重复数据消除间隔内传递。

消息组 ID

指定消息属于特定消息组的标记。属于同一消息组的邮件始终按照相对于消息组的严格顺序逐个处理(但是，属于不同消息组的邮件可能会失序)。

序列号

Amazon SQS 为每条消息分配的大型非连续数字。

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 \(p. 35\)](#)
- [Amazon SQS 消息系统属性 \(p. 37\)](#)

Amazon SQS 消息属性

Amazon SQS 允许您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）。消息属性。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消息的信息，请参阅[发送带有属性的消息（控制台）\(p. 19\)](#)。

Note

不要将消息属性与消息系统属性：虽然您可以使用消息属性将自定义元数据附加到应用程序的 Amazon SQS 消息，但您可以使用。[消息系统属性 \(p. 37\)](#) 来存储其他 AWS 服务，例如 AWS X-Ray。

主题

- [消息属性组件 \(p. 35\)](#)
- [消息属性数据类型 \(p. 35\)](#)
- [计算消息属性的 MD5 消息摘要 \(p. 36\)](#)

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 和消息正文也不应为空或 null。

每个消息属性包含以下组件：

- 名称— 消息属性名称可包含以下字符：A-Z、a-z、0-9，下划线（_），连字符（-）和期间（.）。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[sendMessage](#) 中的 Amazon Simple Queue Service API 参考）。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，如果类型是 Number，Amazon SQS 将验证数字值。

Amazon SQS 支持逻辑数据类型 string、Number，和 Binary 与格式的可选自定义数据类型标签 `.custom-data-type`

- 字符串–String 属性可以存储使用任意有效 XML 字符的 Unicode 文本。

- 数字—Number 属性可以存储正数或负数值。数字最多可精确到 38 位，并且介于 10^{-128} 和 10^{+126} 之间。

Note

Amazon SQS 删除开头和结尾的零。

- 二进制—二进制属性可以存储任意二进制数据，例如压缩数据、加密数据或图像。
- Custom (自定义)—要创建自定义数据类型，请将自定义类型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

如果使用 AWS SDK for Java，可以跳过本节。这些区域有：MessageMD5ChecksumHandlerSDK for Java 支持 Amazon SQS 消息属性的 MD5 消息摘要。

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

Note

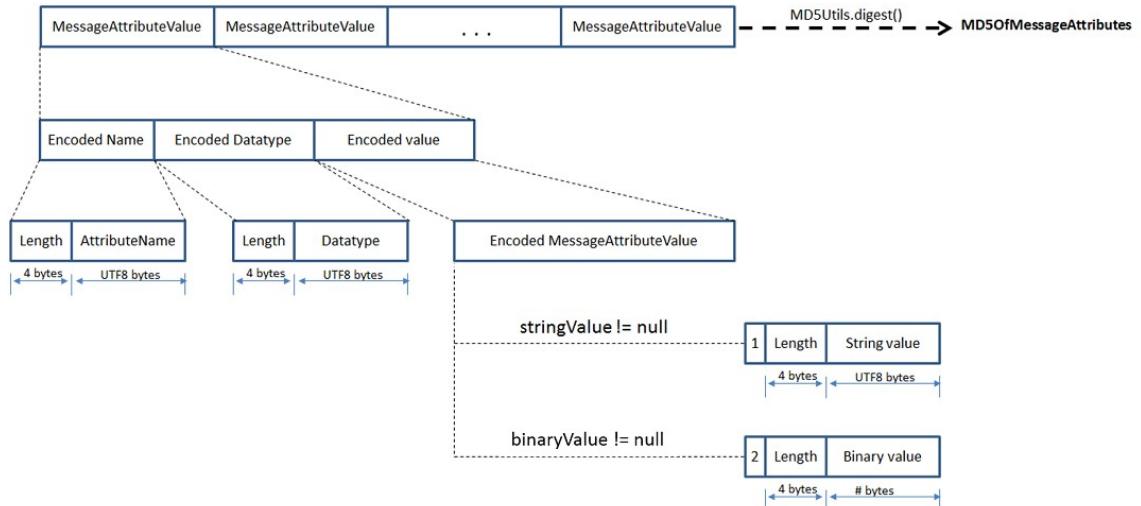
在 MD5 消息摘要计算中始终包含自定义数据类型后缀。

Overview

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。

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

而您可以使用。消息属性 (p. 35) 将自定义元数据附加到 Amazon SQS 消息，您可以使用消息系统属性来存储其他 AWS 服务，例如 AWS X-Ray。有关更多信息，请参阅 MessageSystemAttribute 请求参数 SendMessage 和 SendMessageBatch API 操作，AWSTraceHeader 属性 ReceiveMessage API 操作和 MessageSystemAttributeValue 中的数据类型 Amazon Simple Queue Service API 参考。

消息系统属性的结构与消息属性完全一样，除了以下例外：

- 当前唯一受支持的消息系统属性是 AWSTraceHeader。它的类型必须是 string 而且值必须为格式正确的 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、可见以及不可见的消息的大致数量。有关可见性的更多信息，请参阅“Amazon SQS 可见性超时 (p. 42)”。

Note

对于标准队列，由于 Amazon SQS 的分布式体系结构，结果是近似的。在大多数情况下，计数应接近队列中的实际消息数。

对于 FIFO 队列，结果是精确的。

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

列出队列分页

这些区域有：`listQueues`和`listDeadLetterQueues`API方法支持可选的分页控件。默认情况下，这些API方法在响应消息中返回多达1000个队列。您可以设置`MaxResults`参数在每个响应中返回较少的结果。

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。如果未设置`MaxResults`，则响应最多包含1000个结果，并且`NextToken`值为空。

如果您设置`MaxResults`，则响应将包含的值`NextToken`如果要显示更多结果。在对`listQueues`的下一个请求中将`NextToken`作为参数，以接收下一页结果。如果要显示更多结果，则`NextToken`值为空。

Amazon SQS 成本分配标签

要组织并标识您的Amazon SQS队列以进行成本分配，您可以添加元数据标签标识队列的用途、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” \(p. 15\)](#)

您可以使用成本分配标签组织AWS账单，以反映您自己的成本结构。要执行此操作，请注册以获取AWS账户账单来包含标签键和值。有关更多信息，请参阅[设置月度成本分配报告](#)中的AWS Billing and Cost Management用户指南。

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

Queue	密钥	值
MyQueueA	QueueType	Production
MyQueueB	QueueType	Testing

Note

使用队列标签时，请牢记以下准则：

- 我们建议不要向队列添加超过50个标签。
- 标签没有任何语义含义。Amazon SQS将标签解释为字符串。
- 标签区分大小写。
- 具有与现有标签相同的键的新标签将覆盖现有标签。
- 标记操作仅限于每个TPAWS账户。如果您的应用程序需要更高的吞吐量，[提交请求](#)。

有关标签限制的完整列表，请参阅[与队列相关的配额 \(p. 90\)](#)。

Amazon SQS 短轮询和长轮询

Amazon SQS提供短轮询和长轮询以接收来自队列的消息。默认情况下，队列使用短轮询。

与短轮询，`ReceiveMessage`请求仅查询服务器的一个子集（基于加权随机分布），以查找可包含在响应中的消息。即使查询没有找到任何消息，Amazon SQS也会立即发送响应。

与长轮询，`ReceiveMessage`请求查询所有服务器的消息。Amazon SQS在收集至少一条可用消息（最多为可在请求中指定的最大消息数）后发送响应。仅当轮询等待时间过期时，Amazon SQS才会发送空响应。

以下部分解释短轮询和长轮询的详细信息。

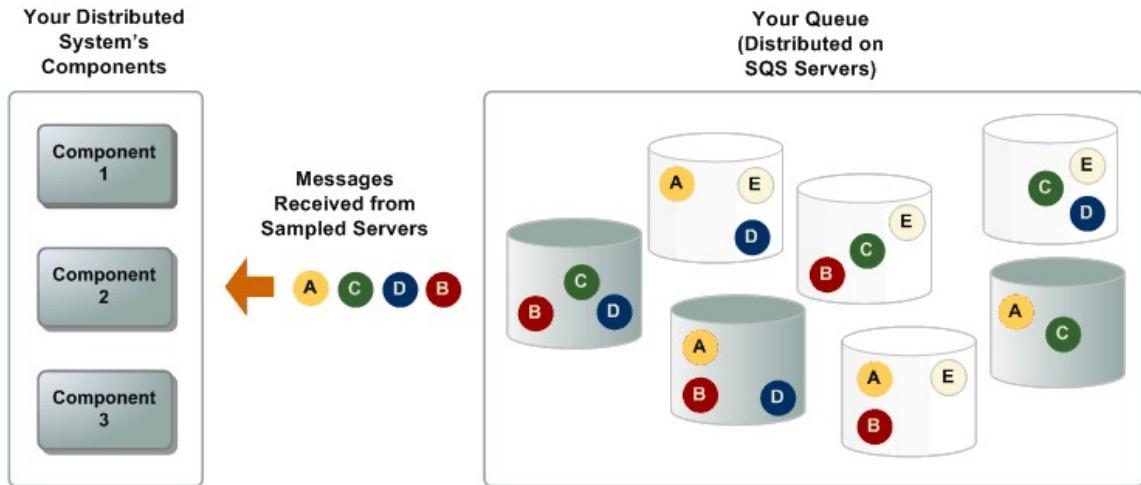
主题

- [通过短轮询来使用消息 \(p. 39\)](#)
- [通过长轮询来使用消息 \(p. 39\)](#)
- [长轮询和短轮询之间的区别 \(p. 40\)](#)

通过短轮询来使用消息

当通过短轮询从队列中使用消息时，Amazon SQS 会对其一部分服务器（基于加权随机分布）进行采样，并且仅从这些服务器返回消息。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 1,000 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提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

当等待时间 `ReceiveMessageAPI` 操作大于 0，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（没有消息可用于 `ReceiveMessage` 请求）和假的空响应（消息可用但未包含在响应中）。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[配置队列参数（控制台）\(p. 12\)](#)。有关最佳实践，请参阅[设置长轮询 \(p. 50\)](#)。

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，从而减少空响应。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，即使队列仍包含消息，您也可能会收到空响应，特别是如果您为 `ReceiveMessageWaitTimeSeconds` 参数。
- 通过查询所有（而不是子集）Amazon SQS 服务器，减少假的空响应。
- 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[确认队列为空 \(p. 21\)](#)。

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：

- 这些区域有：`ReceiveMessage` 呼叫集 `WaitTimeSeconds` 到 0。
- `ReceiveMessage` 调用不会设置 `WaitTimeSeconds`，但队列属性 `ReceiveMessageWaitTimeSeconds` 将设置为 0。

Amazon SQS 死信队列

Amazon SQS 支持死信队列，其他队列（源队列）可将目标为无法成功处理（使用）的消息。死信队列有助于调试您的应用程序或消息传递系统，因为它们可让您隔离有问题的消息以确定其处理失败的原因。有关使用 Amazon SQS 控制台创建队列并为其配置死信队列的信息，请参阅[配置死信队列（控制台）\(p. 14\)](#)。

Important

Amazon SQS 确实不是自动创建死信队列。必须先创建队列，然后才能将其用作死信队列。

主题

- [死信队列的工作方式 \(p. 40\)](#)
- [死信队列有哪些好处？\(p. 41\)](#)
- [不同的队列类型如何处理消息失败？\(p. 41\)](#)
- [何时应使用死信队列？\(p. 41\)](#)
- [排查死信队列的问题 \(p. 42\)](#)

死信队列的工作方式

有时会因各种可能的问题（例如，创建者应用程序或使用者应用程序内的错误条件或导致您的应用程序代码出现问题的意外状态更改）而导致无法处理消息。例如，如果用户使用某特定产品 ID 下达 Web 订单，但产品 ID 已被删除，则 Web 商店的代码会失败并显示错误，而且包含订单请求的消息将发送到死信队列。

有时，创建器和使用器可能无法解释其用于通信的协议的各个方面，从而导致消息中断或丢失。此外，使用器的硬件错误可能会中断消息负载。

这些区域有：重新驱动策略指定源队列，死信队列以及 Amazon SQS 将消息从前者移至后者的条件（如果源队列的使用者无法处理消息指定次数）。当 `ReceiveCount` 对于消息超过 `maxReceiveCount`，Amazon SQS 会将消息移至死信队列（带有其原始消息 ID）。例如，如果源队列具有重新驱动策略，并且 `maxReceiveCount` 设置为 5，并且源队列的使用器收到一条消息 6 次而从未删除它，则 Amazon SQS 会将该消息移至死信队列。

要指定死信队列，您可以使用控制台或 AWS SDK for Java。您必须为将消息发送到死信队列的每个队列执行此操作。同一类型的多个队列可将一个死信队列作为目标。有关更多信息，请参阅[配置死信队列（控制台）\(p. 14\)](#)以及 `RedrivePolicy`[CreateQueue](#) 或 `SetQueueAttributes` 操作的 属性。

Important

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。

您必须使用相同的 AWS 账户创建死信队列以及向死信队列发送消息的其他队列。此外，死信队列必须驻留在使用死信队列的其他队列所在的区域中。例如，如果在美国东部（俄亥俄）区域中创建一个队列，并且要对该队列使用死信队列，则第二个队列也必须位于美国东部（俄亥俄）区域中。

消息的过期始终基于其原始入队时间戳。当消息移至死信队列时，入队时间戳将不会改变。这些区域有：`ApproximateAgeOfOldestMessage` 度量指示消息何时移动到死信队列，而不是当最初发送消息时。例如，假定邮件在原始队列中花费 1 天时间才移动到死信队列。如果死信队列的保留期

为 4 天，则会在 3 天后从死信队列中删除该邮件，并且 ApproximateAgeOfOldestMessage 为 3 天。因此，最佳做法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。

死信队列有哪些好处？

死信队列的主要任务是处理消息失败。利用死信队列，您可以留出和隔离无法正确处理的消息以确定其处理失败的原因。设置死信队列可让您执行以下操作：

- 为传输到死信队列的任何消息配置警报。
- 查看日志以了解可能导致消息传输到死信队列的异常。
- 分析传输到死信队列的消息的内容，以诊断软件问题或创建器/使用器的硬件问题。
- 确定是否为使用者提供了充足的时间来处理消息。

不同的队列类型如何处理消息失败？

标准队列

标准队列 (p. 25) 会在保留期结束前继续处理消息。这种连续处理消息可最大程度地减小队列由无法处理的消息阻止的几率。连续消息处理还可以为队列提供更快的恢复。

在一个处理数千条消息的系统中，拥有使用器反复无法确认和删除的大量消息可能会增加成本并给硬件带来额外负载。最好是在几次处理尝试之后将失败的消息移至死信队列，而不是在这些消息到期前一直尝试处理它们。

Note

标准队列允许大量正在传输的消息。如果您的大多数消息无法使用且无法发送到死信队列，则处理有效消息的速率将下降。因此，要保持队列的效率，请确保应用程序正确处理消息。

FIFO 队列

FIFO 队列 (p. 25) 通过按顺序使用消息组中的消息，提供精确一次处理。因此，尽管使用者可继续检索另一个消息组中的有序消息，但在阻止队列的消息得到成功处理之前，第一个消息组将保持不可用状态。

Note

FIFO 队列允许少量正在传输的消息。因此，为了防止 FIFO 队列被消息阻止，请确保您的应用程序正确处理消息处理。

何时应使用死信队列？



请将死信队列用于标准队列。当您的应用程序不依赖排序时，您应始终利用死信队列。死信队列可帮助您排查不正确的消息传输操作的问题。

Note

即使您使用死信队列，也应继续监控您的队列并重试发送因临时原因而失败的消息。



请不要使用死信队列来减少消息数和降低将系统公开给毒丸消息（可接收但无法处理的消息）的几率。



如果需要无限地重试传输消息，请不要对标准队列使用死信队列。例如，如果您的程序必须等待某个依赖过程变得有效或可用，请不要使用死信队列。



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

排查死信队列的问题

在某些情况下，Amazon SQS 死信队列的行为可能并不总是符合预期。此部分概述了常见问题并说明如何解决这些问题。

使用控制台查看消息可能会导致消息移至死信队列

Amazon SQS 会根据相应队列的重新驱动策略在控制台中查看消息时进行计数。因此，如果在控制台中查看相应队列的重新驱动策略中指定的次数，则该消息将移至相应队列的死信队列中。

要调整此行为，您可以执行下列操作之一：

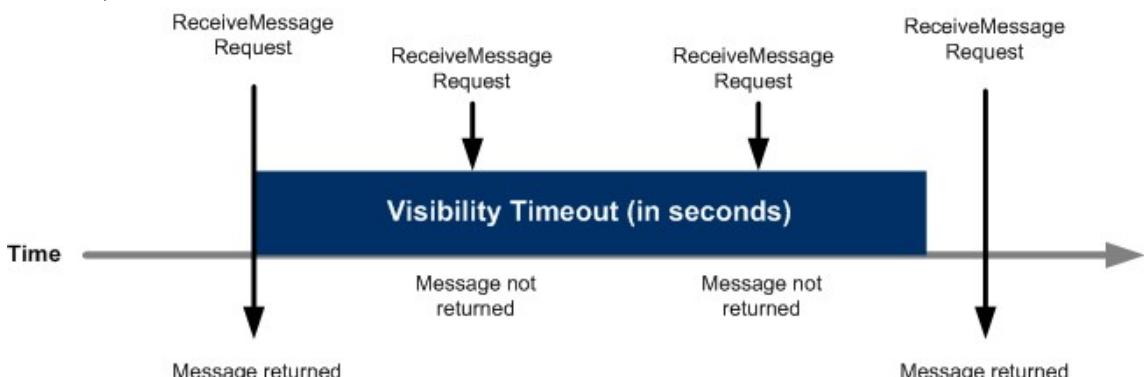
- 针对相应队列的重新驱动策略增大 Maximum Receives 设置。
- 避免在控制台中查看相应队列的消息。

死信队列的 NumberOfMessagesSent 和 NumberOfMessagesReceived 不匹配

如果您手动向死信队列发送消息，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 NumberOfMessagesReceived 的值可能不同。

Amazon SQS 可见性超时

当使用者接收并处理来自某个队列的消息时，消息将保留在该队列中。Amazon SQS 不会自动删除消息。因为 Amazon SQS 是分布式系统，所以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。为防止其他用户再次处理消息，Amazon SQS 会将可见性超时，Amazon SQS 阻止其他用户接收并处理消息的一段时间。消息的默认可见性超时为 30 秒。最短为 0 秒。最长时间为 12 小时。有关使用控制台为队列配置可见性超时的信息，请参阅[配置队列参数（控制台）\(p. 12\)](#)。

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[至少一次传递 \(p. 25\)](#)。

FIFO 队列允许生产者或使用者尝试多次重试：

- 如果生产者检测到失败SendMessage操作时，它可以使用相同的消息重复数据消除 ID，根据需要重试发送多次。假设创建者在重复数据消除间隔过期之前至少收到一个确认，多次重试既不会影响邮件的排序，也不会引入重复项。
- 如果使用者检测到失败ReceiveMessage操作，它可以根据需要重试次数，使用相同的接收请求尝试 ID。假设使用者在可见性超时到期之前至少收到一个确认，多次重试不会影响消息的排序。
- 当您收到具有消息组 ID 的邮件时，除非您删除该邮件或其变为可见，否则不会返回同一消息组 ID 的其他消息。

主题

- [传输中的消息 \(p. 43\)](#)
- [设置可见性超时 \(p. 43\)](#)
- [更改消息的可见性超时 \(p. 44\)](#)
- [终止消息的可见性超时 \(p. 44\)](#)

传输中的消息

Amazon SQS 消息有三种基本状态：

1. 由生产者发送到队列。
2. 消费者从队列中接收。
3. 已从队列中删除。

消息被视为存储的在生产者将其发送到队列中，但尚未被使用者从队列中接收（即状态 1 和 2 之间）之后。已存储邮件的数量没有配额。消息被视为在 Flight，但尚未从队列中删除（即状态 2 和 3 之间）。存在传输中的消息数量配额。

Important

应用于正在进行的邮件的配额与unlimited已存储消息的数量。

对于大多数标准队列（取决于队列流量和消息积压），最多可能有 120000 个传输中的消息（由使用者从队列接收，但尚未从队列中删除）。如果您在使用[短轮询 \(p. 39\)](#)，亚马逊 SQS 将返回OverLimit错误消息。如果您使用[长轮询 \(p. 39\)](#)，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，最多可能有 20000 个传输中的消息（使用者从队列接收，但尚未从队列中删除）。如果您达到此配额，Amazon SQS 不返回任何错误消息。

设置可见性超时

Amazon SQS 返回消息时，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[及时处理消息 \(p. 49\)](#)部分中的最佳实践。

如果您不知道处理消息需要多长时间，请在检测信号适用于您的消费者流程：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消息），将可见性超时延长每分钟 2 分钟。

Important

最大可见性超时为从 Amazon SQS 收到 `ReceiveMessage` 请求。延长可见性超时不会重置 12 小时的最大值。如果您的使用者需要超过 12 小时，请考虑使用步进功能。

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后 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

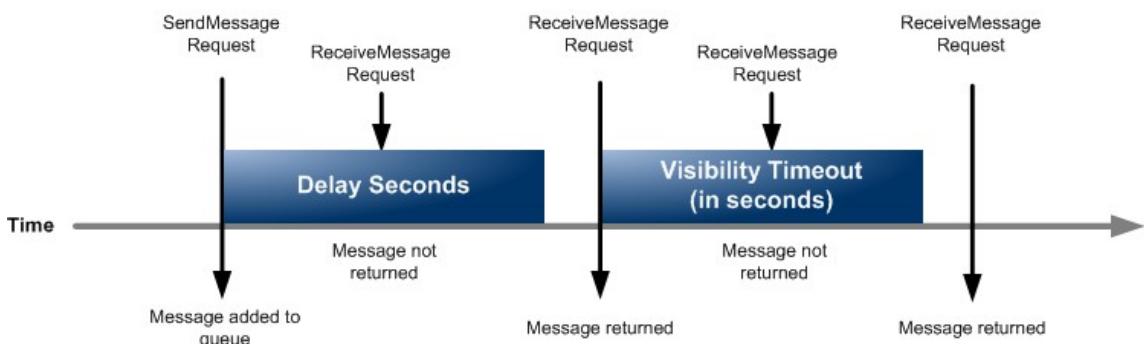
延迟队列可让您将新消息的传递操作推迟若干秒数，例如，当您的使用者应用程序需要更多的时间来处理消息。如果您创建延迟队列，则发送到队列的任何消息在延迟期间对使用者都不可见。队列的默认（最小）延迟为 0 秒。最短值为 15 分钟。有关使用控制台配置延迟队列的信息，请参阅 [配置队列参数（控制台）\(p. 12\)](#)。

Note

对于标准队列，每队列延迟设置为不追溯-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，每队列延迟设置为追溯-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 \(p. 42\)](#)，因为这两种功能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。



若要将延迟秒数设置为单个消息，而不是在整个队列上，请使用[消息定时器 \(p. 48\)](#)允许 Amazon SQS 使用消息计时器的DelaySeconds值而不是延迟队列的DelaySeconds值。

Amazon SQS 临时队列

临时队列可帮助您节省开发时间和部署成本，使用常见消息模式（如请求-响应。可以使用[临时队列客户端](#)创建高吞吐量、经济高效、由应用程序管理的临时队列。

客户端映射多个临时队列（根据需要为特定进程创建的应用程序管理队列）自动到单个 Amazon SQS 队列中。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 \(p. 45\)](#)
- [请求-响应消息收发模式（虚拟队列）\(p. 46\)](#)
- [示例方案：处理登录请求 \(p. 46\)](#)
 - [在客户端 \(p. 46\)](#)
 - [在服务器端 \(p. 47\)](#)
- [清除队列 \(p. 48\)](#)

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 \(p. 53\)](#)。

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[与消息相关的配额 \(p. 91\)](#)。

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。

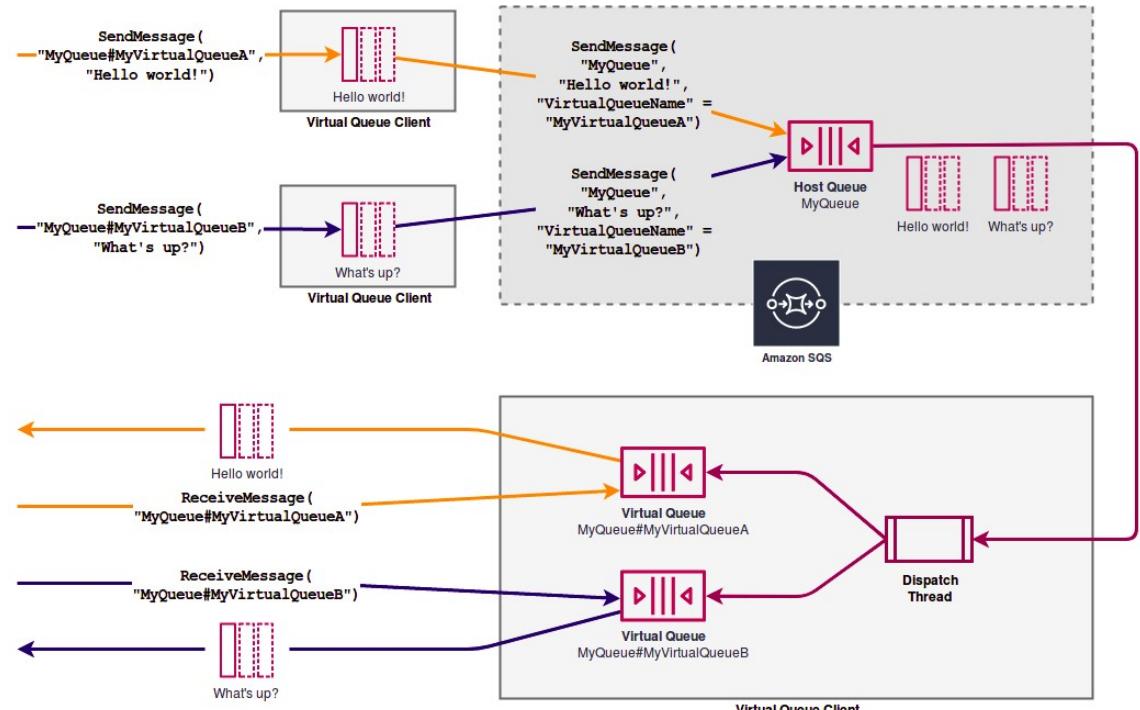
当使用者在虚拟队列 URL 上调用 `ReceiveMessage` API 操作时，临时队列客户端在本地阻止调用，直至后台线程将消息发送到虚拟队列中。（此过程类似于 [Buffered Asynchronous Client \(p. 149\)](#) 中的消息预取操作：单个 API 操作可以将消息提供到最多 10 个虚拟队列。）删除虚拟队列操作可以在不调用 Amazon SQS 本身的情况下删除任何客户端资源。

`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 `_RequesterClientQueueName_`），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 [实施请求-响应系统 \(p. 51\)](#)。

下图显示了使用此模式时的常见配置。



示例方案：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();  
  
private final 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();  
  
    private final AmazonSQS(String requestQueueUrl) {  
        this.requestQueueUrl = requestQueueUrl;  
    }  
  
    // Process login requests from the client.  
    public void handleLoginRequest(Message message) {  
  
        // Process the login and return a serialized result.  
    }  
}
```

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

清除队列

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

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

Note

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

后台线程使用 `ListQueues` 和 `ListTags` API 操作，以检查具有已配置前缀的所有队列，并删除至少五分钟内没有被标记的所有队列。这样，如果一个客户端未完全关闭，则其他活动客户端在之后会进行清除。为了减少重复工作，具有相同前缀的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 消息定时器

使用消息计时器可为添加到队列的消息指定初始不可见时段。例如，如果您发送带有 45 秒计时器的消息，则消息在队列中的前 45 秒内对消费者不可见。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计时器的信息，请参阅。[向队列发送消息（控制台）\(p. 18\)](#)。

Note

FIFO 队列不支持单个消息的计时器。

要在整个队列（而不是各条消息）上设置延迟期间，请使用[延迟队列 \(p. 44\)](#)。各条消息的消息计时器设置将覆盖任何 `DelaySeconds` 值在 Amazon SQS 延迟队列上。

亚马逊 SQS 的最佳实践

这些最佳实践可帮助您充分利用 Amazon SQS。

主题

- 针对 Amazon SQS 标准队列和 FIFO 队列的建议 (p. 49)
- 针对 Amazon SQS FIFO 队列的其他建议 (p. 52)

针对 Amazon SQS 标准队列和 FIFO 队列的建议

下列最佳实践可帮助您使用 Amazon SQS 降低成本并高效地处理消息。

主题

- 使用 Amazon SQS 消息 (p. 49)
- 降低 Amazon SQS 成本 (p. 51)
- 从 Amazon SQS 标准队列移动到 FIFO 队列 (p. 51)

使用 Amazon SQS 消息

下列准则可帮助您使用 Amazon SQS 高效地处理消息。

主题

- 及时处理消息 (p. 49)
- 处理请求错误 (p. 50)
- 设置长轮询 (p. 50)
- 捕获有问题的消息 (p. 50)
- 设置死信队列保留期 (p. 50)
- 避免不一致的消息处理 (p. 51)
- 实施请求-响应系统 (p. 51)

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 \(p. 43\)](#)。
- 如果您不知道处理消息需要多长时间，请创建检测信号适用于您的消费者流程：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消息），将可见性超时延长每分钟 2 分钟。

Important

最大可见性超时为从 Amazon SQS 收到 [ReceiveMessage](#) 请求。延长可见性超时不会重置 12 小时的最大值。如果您的使用者需要超过 12 小时，请考虑使用 Step Functions。

处理请求错误

要处理请求错误，请使用下列策略之一：

- 如果您使用 AWS 开发工具包，则您已有自动重试和回退逻辑在您的处置。有关更多信息，请参阅 [中的错误重试和指数退避AWS中的Amazon Web Services 一般参考](#)。
- 如果您未使用 AWS 开发工具包功能，应允许暂停（例如，200 毫秒），然后重试 [ReceiveMessage](#) 操作时，未收到任何消息、超时或来自 Amazon SQS 的错误消息。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）。

设置长轮询

如果等待时间 [ReceiveMessageAPI](#) 操作大于 0，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（在没有消息可用时）来帮助降低您使用 Amazon SQS 的成本 [ReceiveMessage](#) 请求）和假的空响应（消息可用但未包含在响应中）。有关更多信息，请参阅 [Amazon SQS 短轮询和长轮询 \(p. 38\)](#)。

要获得最佳消息处理，请使用下列策略：

- 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）。如果您未使用 AWS 软件开发工具包访问 Amazon SQS，或者如果您配置 AWS 开发工具包的等待时间更短，则可能必须修改 Amazon SQS 客户端，才能允许时间更长的请求或对长轮询使用较短的等待时间。
- 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

要避免 HTTP 错误，请确保 [ReceiveMessage](#) 请求长于 [WaitTimeSeconds](#) 参数。有关更多信息，请参阅 [ReceiveMessage](#)。

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 \(p. 40\)](#)。

- 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。
- 在队列中包含毒丸消息可能会扭曲 [ApproximateAgeOfOldestMessage \(p. 135\)](#) CloudWatch 指标通过提供不正确的毒丸消息存在时间。配置死信队列有助于避免在使用此指标时发出错误警报。

设置死信队列保留期

消息的过期始终基于其原始入队时间戳。将消息移至死信队列时，入队时间戳将保持不变。这些区域有：[ApproximateAgeOfOldestMessage](#) 度量指示消息何时移动到死信队列，不是最初发送消息时。例如，假定邮件在原始队列中花费 1 天时间才移动到死信队列。如果死信队列的保留期为 4 天，则会在 3 天后从死信队列中删除该邮件，并且 [ApproximateAgeOfOldestMessage](#) 为 3 天。因此，最佳做法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。

避免不一致的消息处理

由于 Amazon SQS 是一个分布式系统，因此即使 Amazon SQS 将消息标记为已发送，使用者也可能不会收到消息。ReceiveMessageAPI 方法调用。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 (p. 40) 的最大接收数量设置为 1。

实施请求-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：

- 请勿为每个消息 创建回复队列。而是在启动时为每个创建者创建回复队列，使用关联 ID 消息属性将回复映射到请求。
- 不要让生成者共享回复队列。这可能会导致生成者收到针对另一个生成者的响应消息。

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）\(p. 46\)](#)。

降低 Amazon SQS 成本

以下最佳实践可帮助您降低成本并利用附加的潜在成本降低或几乎瞬时的响应。

批处理消息操作

要降低成本，可批处理您的消息操作：

- 要发送、接收和删除消息，并通过单一操作更改多条消息的消息可见性超时，请使用 [Amazon SQS 批处理 API 操作 \(p. 148\)](#)。
- 要将客户端缓冲与请求批处理结合起来，请将长轮询与 [缓冲异步客户端 \(p. 149\)](#) 包含在中 AWS SDK for Java。

Note

Amazon SQS 缓冲异步客户端当前不支持 FIFO 队列。

使用适当的轮询模式

- 长轮询使您能够在消息可用时立即使用 Amazon SQS 队列中的消息。
 - 要降低使用 Amazon SQS 的成本并减少空队列的空接收次数（对 ReceiveMessage 操作，不返回任何消息），请启用长轮询。有关更多信息，请参阅 [Amazon SQS 长轮询 \(p. 38\)](#)。
 - 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。

从 Amazon SQS 标准队列移动到 FIFO 队列

如果您未设置 DelaySeconds 参数，则可通过提供每条已发送消息的消息组 ID 来移动到 FIFO 队列。

有关更多信息，请参阅 [从标准队列移至 FIFO 队列 \(p. 28\)](#)。

针对 Amazon SQS FIFO 队列的其他建议

下列最佳实践可帮助您最佳地使用消息重复数据删除 ID 和消息组 ID。有关更多信息，请参阅。SendMessage和SendMessageBatch中的操作Amazon Simple Queue Service API 参考。

主题

- 使用 Amazon SQS 消息重复数据删除 ID (p. 52)
- 使用 Amazon SQS 消息组 ID (p. 53)
- 使用 Amazon SQS 接收请求尝试 ID (p. 53)

使用 Amazon SQS 消息重复数据删除 ID

消息重复数据删除 ID 为已发送消息的重复数据删除所使用的令牌。如果成功发送具有特定消息重复数据删除 ID 的邮件，则会成功接受具有相同消息重复数据删除 ID 的任何邮件，但不会在 5 分钟的重复数据删除间隔内传递。

Note

消息重复数据删除适用于整个队列，而不适用于单个邮件组。

即使在收到和删除消息之后，Amazon SQS 也会继续跟踪消息重复数据删除 ID。

提供消息重复数据删除 ID

在下列情况下，创建者应为每条消息提供消息重复数据删除 ID 值：

- Amazon SQS 必须将其视为唯一项的具有相同消息正文的已发送消息。
- Amazon SQS 必须将其视为唯一项的内容相同、但消息属性不同的已发送消息。
- Amazon SQS 必须将其视为重复项的内容不同（例如，包含在消息正文中的重试计数）的已发送消息。

为单一生产者/使用者系统使用重复数据删除

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。
- 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 FIFO 队列中的消息顺序，因此可重试发送或接收请求。

针对中断恢复场景进行设计

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，请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。

- 创建者必须知道队列的重复数据删除时间间隔。Amazon SQS 具有 minimum 5 分钟的重复数据删除时间间隔。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 (p. 40)。

处理可见性超时

要获得最佳性能，请将[可见性超时 \(p. 42\)](#)设置为大于 AWS 开发工具包读取超时。这适用于使用[ReceiveMessageAPI](#) 操作[短轮询 \(p. 39\)](#)或者[长轮询 \(p. 38\)](#)。

使用 Amazon SQS 消息组 ID

消息组 ID 是指定消息属于特定消息组的标记。属于同一消息组的邮件始终按照相对于消息组的严格顺序逐个处理（但是，属于不同消息组的邮件可能会失序）。

交错多个有序消息组

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是在 FIFO 方式处理的。

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。

在多生产者/使用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）。

避免具有相同消息组 ID 的消息大量积压

对于 FIFO 队列，最多可能有 20,000 个正在进行中的消息（由使用者从队列中接收，但尚未从队列中删除）。如果您达到此配额，Amazon SQS 不返回任何错误消息。FIFO 队列查看前 20k 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有积压消息，则在您成功处理完这些积压消息前，您无法使用在之后发送到队列的其他消息组中的消息。

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 \(p. 40\)](#)策略。

避免在虚拟队列中重复使用相同的消息组 ID

要防止发送到不同消息组 ID[虚拟队列 \(p. 45\)](#)，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 为[ReceiveMessage](#)调用。

在导致您的软件开发工具包和 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，并在软件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。

Amazon SQS Java 软件开发工具包示例

您可以使用AWS SDK for Java构建 Java 应用程序，以便与 Amazon Simple Queue Service (Amazon SQS) 和其他AWS服务。要安装并设置开发工具包，请参阅[入门](#)中的AWS SDK for Java 2.x开发人员指南。

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[使用 Amazon SQS 消息队列](#)中的AWS SDK for Java 2.x开发人员指南。

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

主题

- [使用服务器端加密 \(SSE\) \(p. 54\)](#)
- [为队列配置标签 \(p. 55\)](#)
- [发送消息属性 \(p. 57\)](#)
- [使用 Amazon S3 管理大型 Amazon SQS 消息 \(p. 58\)](#)

使用服务器端加密 (SSE)

您可以将AWS SDK for Java向 Amazon SQS 队列添加服务器端加密 (SSE)。每个队列都使用AWS Key Management Service(AWS KMS) 客户主密钥 (CMK) 以生成数据加密密钥。此示例使用AWSAmazon SQS 的托管 CMK。有关使用 SSE 和 CMK 角色的更多信息，请参阅[静态加密 \(p. 96\)](#)。

将 SSE 添加到现有队列

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。

以下代码示例将主密钥设置为AWSAmazon SQS 的托管 CMK。此示例还将[主密钥重用周期 \(p. 97\)](#)设置为 140 秒。

运行示例代码之前，请确保您已将AWS凭证。有关更多信息，请参阅[设置AWS促进发展的凭证和区域](#)中的AWS SDK for Java 2.x开发人员指南。

```
// 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 CMK 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 setAttrsRequest = SetQueueAttributesRequest.builder()  
    .queueUrl(queueUrl)
```

```
.attributes(attributes)
.build();

sqcClient.setQueueAttributes(set_attrs_request);
```

为队列禁用 SSE

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

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` 属性设置为 [CreateQueue API](#) 方法。

以下示例创建启用 SSE 的新队列。队列使用 AWS Amazon SQS 的托管 CMK。此示例还将 [主密钥重用周期 \(p. 97\)](#) 设置为 160 秒。

运行示例代码之前，请确保您已将 AWS 凭证。有关更多信息，请参阅[设置 AWS 促进发展的凭证和区域](#)中的 AWS SDK for Java 2.x 开发人员指南。

```
// 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 CMK 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[示例](#)中的 Amazon Simple Queue Service API 参考。

要检索特定队列的 CMK ID 或数据密钥重用期，请运行 [GetQueueAttributes](#) 方法并检查 `KmsMasterKeyId` 和 `KmsDataKeyReusePeriodSeconds` 值。

为队列配置标签

使用成本分配标签来帮助组织和标识您的 Amazon SQS 队列。以下示例演示如何使用配置标签 AWS SDK for Java。有关更多信息，请参阅[Amazon SQS 成本分配标签 \(p. 38\)](#)。

运行示例代码之前，请确保您已设置 AWS 凭证。有关更多信息，请参阅[设置 AWS 凭证和发展区域](#)中的 AWS SDK for Java 2.x 开发人员指南。

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

要添加或更新队列的标记值，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.
sqscClient.untagQueue(untagQueueRequest);
```

发送消息属性

您可将结构化元数据（如时间戳、地理空间数据、签名和标识符）与使用消息属性。有关更多信息，请参阅[Amazon SQS 消息属性 \(p. 35\)](#)。

运行示例代码之前，请确保您已将AWS凭证。有关更多信息，请参阅[设置AWS凭证和区域促进发展中的AWS SDK for Java 2.x开发人员指南](#)。

定义属性

要为消息定义属性，请添加使用[MessageAttributeValue](#)数据类型。有关更多信息，请参阅[消息属性组件 \(p. 35\)](#)和[消息属性数据类型 \(p. 35\)](#)。

这些区域有：AWS SDK for Java自动计算消息正文和消息属性校验和并将其与 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](#)和[计算消息属性的 MD5 消息摘要 \(p. 36\)](#)，了解其他编程语言。

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 230.0000000000000001。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccurateWeight", new MessageAttributeValue()
.withDataType("Number")
.withStringValue("230.0000000000000001"));
```

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](#)方法返回123456。

Binary (custom)

此示例定义自定义属性 Binary.JPG，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.JPG")
    .withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

发送带有属性的消息

此示例将属性添加到SendMessageRequest发送消息之前。

```
// 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执行方法晚于提供消息组 ID。如果您使用[SendMessageBatch](#)方法而非[SendMessage](#)，则必须指定批处理中每个消息的消息属性。

使用 Amazon S3 管理大型 Amazon SQS 消息

要管理大型 Amazon Simple Queue Service (Amazon SQS) 消息，您可以使用 Amazon Simple Storage Service (Amazon S3) 和 Amazon SQS Exue Queue Queue Queue Service (Amazon S3) 和 Amazon SQS 扩展客户端库。这在存储和使用高达 2 GB 的消息时特别有用。除非您的应用程序要求反复创建队列，并将其保持不活动状态或在队列中存储大量数据，否则请考虑使用 Amazon S3 来存储您的数据。

您可以使用 Amazon SQS 适用于 Java 的 Amazon SQS 扩展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 S3 存储桶检索消息对象
- 从 S3 存储桶中删除该消息对象

您可以使用适用于 Java 的 Amazon SQS 扩展客户端库使用 Amazon S3 管理 Amazon SQS 消息仅限使用 AWS SDK for Java。您无法使用 AWS CLI、Amazon SQS 控制台、Amazon SQS HTTP API 或任何其他 AWS 开发工具包。

这些区域有：[适用于 Java 的开发工具包](#)和适用于 Java 的 Amazon SQS 扩展客户端库要求使用 J2SE 开发工具包 8.0 或更高版本。

Prerequisites

以下示例使用的是 AWS Java 开发工具包。要安装和设置开发工具包，请参阅[设置AWS适用于 Java 的开发工具包](#)中的AWS SDK for Java开发人员指南。

运行示例代码之前，请配置您的 AWS 凭证。有关更多信息，请参阅[设置AWS凭证和发展区](#)中的AWS SDK for Java开发人员指南。

示例：使用 Amazon S3 管理大型 Amazon SQS 消息

以下示例创建具有随机名称的 Amazon S3 存储桶并添加生命周期规则以便在创建日期之后的 14 天后永久删除这些对象。它还会创建一个名为 MyQueue 并向该队列发送 256 KB 以上的随机消息。最后，代码会检索该消息，返回有关该消息的信息，然后删除该消息、队列和存储桶。

```
/*
 * Copyright 2010-2021 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.services.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazonaws.services.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-hmmss").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);
}
}
```

使用 JMS 和 Amazon SQS 的使用

Amazon SQS (JMS) 接口是 Amazon SQS 的一个 Java 消息服务 (JMS) 接口，可让您在已使用 JMS 的应用程序中充分利用 Amazon SQS。利用此接口，对代码稍作更改即可将 Amazon SQS 提供程序使用 Amazon 提供程序。以及 AWS SDK for Java，您可以创建 JMS 连接和会话以及与 Amazon SQS 队列之间收发消息的创建者和使用者。

此库支持依据 [JMS 1.1 规范](#) 的队列消息收发 (JMS 点对点模式)；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 队列消息库的功能的信息，请参阅[支持的 JMS 1.1 实施 \(p. 83\)](#)和[Amazon SQS 常见问题](#)。

主题

- [Prerequisites \(p. 62\)](#)
- [亚马逊 SQS Java 消息传递库入门 \(p. 63\)](#)
- [将 Amazon SQS Java 消息服务 \(JMS\) 客户端与其他 Amazon SQS 客户端一起使用 \(p. 68\)](#)
- [实际可用的 Java 示例：结合使用 JMS 与 Amazon SQS 标准队列 \(p. 69\)](#)
- [支持的 JMS 1.1 实施 \(p. 83\)](#)

Prerequisites

在开始之前，您必须满足以下先决条件：

- 适用于 Java 的 开发工具包

可通过两种不同的方式将适配 SDK for Java 包含在项目中：

- 下载并安装适配 SDK for Java。
- 使用 Maven 获取 Amazon SQS Java 消息传递库。

Note

适配 SDK for Java 作为一个依赖项包含在内。

这些区域有：[适用于 Java 的 开发工具包](#) 和 Amazon SQS 扩展客户端库要求使用 J2SE Demements 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 开发工具包](#)。

- Amazon SQS Java 消息传递库

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。有关下载此库的信息，请参阅[Amazon SQS Java 消息传递库](#)。

Note

Amazon SQS Java 消息传递库包括[Maven](#)和[Spring](#) 框架。

有关使用 Maven、Spring Framework 和 Amazon SQS Java 消息库的代码示例，请参阅[实际可用的 Java 示例：结合使用 JMS 与 Amazon SQS 标准队列 \(p. 69\)](#)。

```
<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 AWS Management ConsoleAmazon SQS , CreateQueueAPI 或 Amazon SQS Java 消息库中包装的 Amazon SQS 客户端。

- 有关使用 Amazon SQS 创建用于的队列的信息，请使用 AWS Management Console 或CreateQueueAPI，请参阅[创建队列 \(p. 11\)](#)。
- 有关使用 Amazon SQS Java 消息传递库的信息，请参阅[亚马逊 SQS Java 消息传递库入门 \(p. 63\)](#)。

亚马逊 SQS Java 消息传递库入门

要开始将 Java 消息服务 (JMS) 与 Amazon SQS 结合使用，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。

亚马逊 SQS Java 消息库中包含的包装 Amazon SQS 客户端对象检查是否存在 Amazon SQS 队列。如果该队列不存在，客户端将创建它。

创建 JMS 连接

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。但是，`getWrappedAmazonSQSClient` 方法还提供当前连接使用的 Amazon SQS 客户端的包装版本。该包装程序将客户端中的每个异常转变为 `JMSEException`，从而让预期有 `JMSEException` 发生的现有代码更方便地使用该异常。

2. 您可以使用 `getAmazonSQSClient` 和 `getWrappedAmazonSQSClient` 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 \(p. 71\)](#) 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 队列的名称必须以 .fifo 后缀。

有关 ContentBasedDeduplication 属性的更多信息，请参阅 [确切一次处理 \(p. 27\)](#)。

同步发送消息

- 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。

```
// Create the nontransacted session with AUTO_ACKNOWLEDGE mode
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

- 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);
```

- 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 [关键术语 \(p. 26\)](#)。

```
// 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

- 要接收消息，可为同一队列创建一个使用器，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();
```

- 在超时设为 1 秒钟的情况下对该使用器调用 `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 FIFO 特定于的消息属性，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 [关键术语 \(p. 26\)](#)。

```
// 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"));
}
```

- 关闭连接和会话。

```
// Close the connection (and the session).
connection.close();
```

输出看上去类似于以下内容：

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

Note

您可以使用 Spring Framework 初始化这些对象。

有关其他信息，请参阅 `SpringExampleConfiguration.xml`、`SpringExample.java`，以及 `ExampleConfiguration.java` 部分中的 `ExampleCommon.java` 和 [实际可用的 Java 示例：结合使用 JMS 与 Amazon SQS 标准队列 \(p. 69\)](#) 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 \(p. 71\)](#) 和 [SyncMessageReceiver.java \(p. 72\)](#)。

异步接收消息

在 [亚马逊 SQS Java 消息传递库入门 \(p. 63\)](#) 中的示例中，消息发送到 `MyQueue` 并被同步接收。

以下示例介绍如何通过监听器异步接收消息。

1. 实施 `MessageListener` 接口。

```
class MyListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        try {
            // Cast the received message as TextMessage and print the text to screen.
            System.out.println("Received: " + ((TextMessage) message).getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

在收到消息时，调用 `onMessage` 接口的 `MessageListener` 方法。在此监听器实现中，输出保存在消息中的文本。

2. 对 `receive` 实现实例设置使用者消息监听器，而不是对使用者显式调用 `MyListener` 方法。主线程会等待一秒钟。

```
// Create a consumer for the 'MyQueue'.
MessageConsumer consumer = session.createConsumer(queue);

// Instantiate and set the message listener for the consumer.
consumer.setMessageListener(new MyListener());

// Start receiving incoming messages.
connection.start();

// Wait for 1 second. The listener onMessage() method is invoked when a message is
// received.
Thread.sleep(1000);
```

其余步骤与 [亚马逊 SQS Java 消息传递库入门 \(p. 63\)](#)示例中的步骤相同。有关异步使用者的完整示例，请参阅[AsyncMessageReceiver.java](#)中的 [实际可用的 Java 示例：结合使用 JMS 与 Amazon SQS 标准队列 \(p. 69\)](#)。

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

中的示例[亚马逊 SQS Java 消息传递库入门 \(p. 63\)](#)使用AUTO_ACKNOWLEDGE模式，该模式会自动确认收到的每条消息（因此会从队列中删除消息）。

- 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);
```

- 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 [亚马逊 SQS Java 消息传递库入门 \(p. 63\)](#)示例中的步骤相同。有关具有客户端确认模式的同步使用者的完整示例，请参阅[SyncMessageReceiverClientAcknowledge.java](#)中的 [实际可用的 Java 示例：结合使用 JMS 与 Amazon SQS 标准队列 \(p. 69\)](#)。

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 \(p. 67\)](#)。

Amazon SQS Java 消息库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, Session.UNORDERED_ACKNOWLEDGE);
```

其余步骤与 [使用客户端确认模式 \(p. 67\)](#)示例中的步骤相同。有关具有 UNORDERED_ACKNOWLEDGE 模式的同时使用者的完整示例，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 Amazon SQS Java 消息服务 (JMS) 客户端与其他 Amazon SQS 客户端一起使用

将 Amazon SQS Java Message Service (JMS) 客户端与AWS开发工具包会将 Amazon SQS Message 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 Client 与 Amazon SQS 扩展客户端库配合使用，以发送包含对 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 Amazon S3 管理大型 Amazon SQS 消息 \(p. 58\)](#)。

以下 Java 代码示例将为创建 JMS 提供程序：

```
AmazonS3 s3 = new AmazonS3Client(credentials);
Region s3Region = Region.getRegion(Regions.US_WEST_2);
s3.setRegion(s3Region);

// Set the Amazon S3 bucket name, and set a lifecycle rule on the bucket to
// permanently delete objects a certain number of days after each object's creation date.
// Next, create the bucket, and enable message objects to be stored in the bucket.
BucketLifecycleConfiguration.Rule expirationRule = new BucketLifecycleConfiguration.Rule();
expirationRule.withExpirationInDays(14).withStatus("Enabled");
BucketLifecycleConfiguration lifecycleConfig = new
BucketLifecycleConfiguration().withRules(expirationRule);

s3.createBucket(s3BucketName);
s3.setBucketLifecycleConfiguration(s3BucketName, lifecycleConfig);
System.out.println("Bucket created and configured.");

// Set the SQS extended client configuration with large payload support enabled.
ExtendedClientConfiguration extendedClientConfig = new ExtendedClientConfiguration()
.withLargePayloadSupportEnabled(s3, s3BucketName);

AmazonSQS sqsExtended = new AmazonSQSExtendedClient(new AmazonSQSClient(credentials),
extendedClientConfig);
Region sqsRegion = Region.getRegion(Regions.US_WEST_2);
sqsExtended.setRegion(sqsRegion);
```

以下 Java 代码示例将创建连接工厂：

```
// Create the connection factory using the environment variable credential provider.
// Pass the configured Amazon SQS Extended Client to the JMS connection factory.
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
    new ProviderConfiguration(),
    sqsExtended
);

// Create the connection.
```

```
SQSConnection connection = connectionFactory.createConnection();
```

实际可用的 Java 示例：结合使用 JMS 与 Amazon SQS 标准队列

以下代码示例显示如何结合使用 Java 消息服务 (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 \(p. 64\)](#)、[同步发送消息 \(p. 64\)](#)，和[同步接收消息 \(p. 65\)](#)。（同步接收消息同时适用于标准队列和 FIFO 队列。但是，FIFO 队列中的消息包含更多属性。）

ExampleConfiguration.java

以下 Java 代码示例设置将用于其他 Java 示例的默认队列名称、区域和凭证。

```
/*
 * Copyright 2010-2021 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 2010-2021 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 2010-2021 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 2010-2021 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() ) );

ReceiverCallback callback = new ReceiverCallback();
consumer.setMessageListener( callback );

// No messages are processed until this is called
connection.start();

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 2010-2021 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 2010-2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *     https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

/**
 * An example class to demonstrate the behavior of UNORDERED_ACKNOWLEDGE mode for received
 * messages. This example
 * complements the example given in {@link SyncMessageReceiverClientAcknowledge} for
 * CLIENT_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 * messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the visibility
 * time out period, an attempt to
 * receive another message is made. It's shown that the first message received in the prior
 * attempt is returned again
 * for the second attempt. In UNORDERED_ACKNOWLEDGE mode, all the messages must be
 * explicitly acknowledged no matter what
 * the order they're received.
 *
 * This ISN'T the behavior for CLIENT_ACKNOWLEDGE mode. Please see {@link
 * SyncMessageReceiverClientAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverUnorderedAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue for
    this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
ExampleConfiguration.parseConfig("SyncMessageReceiverUnorderedAcknowledge", args);
```

```
// Setup logging for the example
ExampleCommon.setupLogging();

// Create the connection factory based on the config
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
    new ProviderConfiguration(),
    AmazonSQSClientBuilder.standard()
        .withRegion(config.getRegion().getName())
        .withCredentials(config.getCredentialsProvider())
);

// Create the connection
SQSConnection connection = connectionFactory.createConnection();

// Create the queue if needed
ExampleCommon.ensureQueueExists(connection, config.getQueueName());

// Create the session with unordered acknowledge mode
Session session = connection.createSession(false,
    SQSSession.UNORDERED_ACKNOWLEDGE);

// Create the producer and consume
MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

// Open the connection
connection.start();

// Send two text messages
sendMessage(producer, session, "Message 1");
sendMessage(producer, session, "Message 2");

// Receive a message and don't acknowledge it
receiveMessage(consumer, false);

// Receive another message and acknowledge it
receiveMessage(consumer, true);

// Wait for the visibility time out, so that unacknowledged messages reappear in
the queue
System.out.println("Waiting for visibility timeout...");
Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

// Attempt to receive another message and acknowledge it. This results in receiving
the first message since
// we have acknowledged only the second message. In the UNORDERED_ACKNOWLEDGE mode,
all the messages must
// be explicitly acknowledged.
receiveMessage(consumer, true);

// Close the connection. This closes the session automatically
connection.close();
System.out.println("Connection closed.");
}

/**
 * Sends a message through the producer.
 *
 * @param producer Message producer
 * @param session Session
 * @param messageText Text for the message to be sent
 * @throws JMSEException
 */

```

```
private static void sendMessage(MessageProducer producer, Session session, String messageText) throws JMSException {
    // Create a text message and send it
    producer.send(session.createTextMessage(messageText));
}

/**
 * Receives a message through the consumer synchronously with the default timeout
 (TIME_OUT_SECONDS).
 * If a message is received, the message is printed. If no message is received, "Queue
 is empty!" is
 * printed.
 *
 * @param consumer Message consumer
 * @param acknowledge If true and a message is received, the received message is
 acknowledged.
 * @throws JMSException
 */
private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSException {
    // Receive a message
    Message message = consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

    if (message == null) {
        System.out.println("Queue is empty!");
    } else {
        // Since this queue has only text messages, cast the message object and print
        the text
        System.out.println("Received: " + ((TextMessage) message).getText());

        // Acknowledge the message if asked
        if (acknowledge) message.acknowledge();
    }
}
}
```

SpringExampleConfiguration.xml

以下 XML 代码示例是 [SpringExample.java \(p. 80\)](#) 的 Bean 配置文件。

```
<!--
Copyright 2010-2021 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 2010-2021 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 (JMSException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
```

}

ExampleCommon.java

以下 Java 代码示例首先查看 Amazon SQS 队列是否存在，如果不存在，则创建一个。它还包含示例日历记录代码。

```
/*
 * Copyright 2010-2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *     https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

public class ExampleCommon {
    /**
     * A utility function to check the queue exists and create it if needed. For most
     * use cases this is usually done by an administrator before the application is run.
     */
    public static void ensureQueueExists(SQSConnection connection, String queueName) throws
JMSException {
        AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

        /**
         * In most cases, you can do this with just a createQueue call, but GetQueueUrl
         * (called by queueExists) is a faster operation for the common case where the
queue
         * already exists. Also many users and roles have permission to call GetQueueUrl
         * but don't have permission to call CreateQueue.
         */
        if( !client.queueExists(queueName) ) {
            client.createQueue( queueName );
        }
    }

    public static void setupLogging() {
        // Setup logging
        BasicConfigurator.configure();
        Logger.getRootLogger().setLevel(Level.WARN);
    }

    public static void handleMessage(Message message) throws JMSException {
        System.out.println( "Got message " + message.getJMSMessageID() );
        System.out.println( "Content: " );
        if( message instanceof TextMessage ) {
            TextMessage txtMessage = ( TextMessage ) message;
            System.out.println( "\t" + txtMessage.getText() );
        } else if( message instanceof BytesMessage ){
            BytesMessage byteMessage = ( BytesMessage ) message;
            // Assume the length fits in an int - SQS only supports sizes up to 256k so
that
            // should be true
            byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
            byteMessage.readBytes(bytes);
        }
    }
}
```

```
        System.out.println( "\t" + Base64.encodeAsString( bytes ) );
    } else if( message instanceof ObjectMessage ) {
        ObjectMessage objMessage = (ObjectMessage) message;
        System.out.println( "\t" + objMessage.getObject() );
    }
}
```

支持的 JMS 1.1 实施

Amazon SQS Java 消息传递库支持以下[JMS 1.1 实施](#)。有关 Amazon SQS Java 消息库支持的功能和特性的更多信息，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。此模式帮助 Amazon SQS 允许 JMS 客户端显式确认消息。

JMS 定义的标头和预留属性

发送消息

在发送消息时，您可以为每条消息设置以下标头和属性：

- JMSXGroupID (对于 FIFO 队列是必需的，对于标准队列是不允许的)
- JMS_SQS_DeduplicationId (对于 FIFO 队列是可选的，对于标准队列是不允许的)

在发送消息后，Amazon SQS 将为每条消息设置以下标头和属性：

- `JMSMessageID`
- `JMS_SQS_SequenceNumber` (仅适用于 FIFO 队列)

接收消息

在接收消息时，Amazon SQS 将为每条消息设置以下标头和属性：

- `JMSDestination`
- `JMSMessageID`
- `JMSRedelivered`
- `JMSXDeliveryCount`
- `JMSXGroupID` (仅适用于 FIFO 队列)
- `JMS_SQS_DeduplicationId` (仅适用于 FIFO 队列)
- `JMS_SQS_SequenceNumber` (仅适用于 FIFO 队列)

Amazon SQS 教程

本节提供您可用于探索 Amazon SQS 的功能和特性的教程。

主题

- [创建 Amazon SQS 队列 \(AWS CloudFormation \) \(p. 85\)](#)
- [教程：从 Amazon 虚拟私有云向 Amazon SQS 队列发送消息 \(p. 86\)](#)

创建 Amazon SQS 队列 (AWS CloudFormation)

您可以使用AWS CloudFormation控制台和 JSON (或 YAML) 模板创建 Amazon SQS 队列。有关更多信息，请参阅 [使用AWS CloudFormation模板和AWS::SQS::Queue资源](#)中的AWS CloudFormation用户指南。

使用AWS CloudFormation创建 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[确切一次处理 \(p. 27\)](#)。

Note

FIFO 队列的名称必须以.fifo后缀。

```
{  
    "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 Specify Template (指定模板) 面板上，选择 Upload a template file (上传模板文件)，选择您的 MyQueue.json 文件，然后选择 Next (下一步)。
4. 在存储库的指定详细信息页面，键入 MyQueue 对于来说为 Stack Name，然后选择下一步。
5. 在 Options (选项) 页面上，选择 Next (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。

Filter: Active ▾ By Stack Name		Showing 1 stack	
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 虚拟私有云向 Amazon SQS 队列发送消息

在本教程中，您将学习如何通过安全的私有网络将消息发送到 Amazon SQS 队列。此网络包含一个包含 Amazon EC2 实例的 VPC。该实 Amazon SQS 过接口 VPC 终端节点，即使网络与公共 Internet 断开连接，您也可以连接到 Amazon EC2 实例并将消息发送到 Amazon SQS 队列。有关更多信息，请参阅 [Amazon SQS 的 Amazon Virtual Private Cloud 终端节点 \(p. 101\)](#)。

Important

- 您只能将 Amazon Virtual Private Cloud 与 HTTPS Amazon SQS 终端节点结合使用。
- 在将 Amazon SQS 配置为从 Amazon VPC 发送消息时，您必须启用私有 DNS，并且指定格式为 `sqs.us-east-2.amazonaws.com`。
- 私有 DNS 不支持传统终端节点，例如 `queue.amazonaws.com` 或 `us-east-2.queue.amazonaws.com`。

主题

- 第 1 步：创建 Amazon EC2 key pair (p. 86)
- 第 2 步：CreateAWSresources (p. 87)
- 第 3 步：确认您的 EC2 实例不可公开访问 (p. 87)
- 第 4 步：为 Amazon SQS 创建一个亚马逊 VPC 终端节点 (p. 88)
- 第 5 步：向您的 Amazon SQS 队列发送消息 (p. 89)

第 1 步：创建 Amazon EC2 key pair

A密钥对允许您连接到 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. 登录到[Amazon EC2 控制台](#)。
2. 在导航菜单上的网络和安全性下，选择密钥对。
3. 选择 Create Key Pair。
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
```

第 2 步 : CreateAWSresources

要设置必要基础架构，您必须使用 AWS CloudFormation 模板，它是用于创建堆栈包括 AWS 资源，例如 Amazon EC2 实例和 Amazon SQS 队列。

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 AWS CloudFormation 模板名为 [SQS-VPCE-Tutorial-CloudFormation.yaml](#) 从 GitHub。
 2. 登录到 [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. 对于关键字名称，选择 SQS-VPCE-Tutorial-Key-Pair。
 - c. 选择 Next。
 6. 在 Options (选项) 页面上，选择 Next (下一步)。
 7. 在存储库的审核页面上，在 Capabilities 部分，选择我承认 AWS CloudFormation 可使用自定义名称创建 IAM 资源。, 然后选择 Create。

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。

第 3 步 : 确认您的 EC2 实例不可公开访问

您的 AWS CloudFormation 模板在 VPC 中启动名为 SOS-VPCE-Tutorial-EC2-Instance 的 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，您必须连接到实例，请尝试连接到公有终端节点，然后尝试将消息发送到 Amazon SQS。

1. 登录到[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 \(p. 86\)](#)的目录中，使用以下命令连接到实例，例如：

```
ssh -i SQS-VPCE-Tutorial-KeyPair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

6. 尝试连接到任何公有终端节点，例如：

```
ping amazon.com
```

正如预期的那样，连接尝试失败。

7. 登录到[Amazon SQS 控制台](#)。
8. 从队列列表中，选择由 AWS CloudFormation 模板（例如 VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFH2IJK）创建的队列。
9. 在存储库的详细信息表中，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 终端节点时，您的发送尝试将成功。

第 4 步：为 Amazon SQS 创建一个亚马逊 VPC 终端节点

要将 VPC 连接到 Amazon SQS，您必须定义一个接口 VPC 终端节点。添加终端节点后，您可以使用 VPC 中的 EC2 实例中的 Amazon SQS API。这允许您将消息发送到 AWS 网络中的队列，而无需跨公共 Internet。

Note

EC2 实例仍无法访问 Internet 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，选择终端节点 (Endpoints)。
3. 选择 Create Endpoint。
4. 在存储库的创建终端节点页面，用于服务名称中，选择 Amazon SQS 的服务名称。

Note

服务名称因当前AWS区域。例如，如果您位于美国东部（俄亥俄），则服务名称为com.amazonaws.**us-east-2**.sns。

5. 对于 VPC，选择 SQS-VPCE-Tutorial-VPC。
6. 对于子网 (Subnets)，选择子网 ID (Subnet ID) 包含 SQS-VPCE-Tutorial-Subnet 的子网。
7. 对于安全组 (Security group)，选择选择安全组 (Select security groups)，然后选择其组名称 (Group Name)包含 SQS VPCE Tutorial Security Group 的安全组。
8. 选择Create endpoint。

创建接口 VPC 终端节点并显示其 ID，例如 vpce-0ab1cdef2ghi3j456k。

9. 选择 Close (关闭)。

Amazon VPC 控制台将打开终端节点页。

Amazon VPC 开始创建终端节点，并显示 Pending 状态。在此过程完成后，Amazon VPC 将显示 available 状态。

第 5 步：向您的 Amazon SQS 队列发送消息

现在您的 VPC 包含一个用于 Amazon SQS 的终端节点，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-KeyPair.pem ec2-user@ec2-203-0-113-0.us-east-2.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"  
}
```

有关在由 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[接收和删除消息（控制台）\(p. 19\)](#)。

有关删除资源的信息，请参阅以下内容：

- [删除 VPC 终端节点](#)中的Amazon VPC 用户指南
- [删除 Amazon SQS 队列 \(p. 20\)](#)
- [终止您的实例](#)中的适用于 Linux 实例的 Amazon EC2 用户指南
- [删除 VPC](#)中的Amazon VPC 用户指南
- [删除AWS CloudFormation控制台](#)中的AWS CloudFormation用户指南
- [删除您的密钥对](#)中的适用于 Linux 实例的 Amazon EC2 用户指南

Amazon SQS 配额

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

主题

- [与队列相关的配额 \(p. 90\)](#)
- [与消息相关的配额 \(p. 91\)](#)
- [与策略相关的配额 \(p. 92\)](#)

与队列相关的配额

下表列出了与队列相关的配额。

配额	描述
延迟队列	队列的默认（最小）延迟为 0 秒。最长为 15 分钟。
列出的队列	每个 ListQueues 请求 1,000 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	FIFO 队列中的消息组数量没有限额。
每个队列的消息数（积压）	Amazon SQS 队列可以存储的消息数量是无限制的。
每个队列的消息数（传输中）	对于大多数标准队列（取决于队列流量和消息积压），最多可能有 120000 个传输中的消息（消费者从队列接收，但尚未从队列中删除）。如果您在使用 短轮询 (p. 39) ，Amazon SQS 将返回OverLimit 错误消息。如果您使用 长轮询 (p. 39) ，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，最多可能有 20000 个传输中的消息（消费者从队列接收，但尚未从队列中删除）。如果您达到此配额，则 Amazon SQS 不返回任何错误消息。
队列名称	队列名称可以包含最多 80 个字符。可接受以下字符：字母数字字符、连字符（-）和下划线（_）。
	<p>Note</p> <p>队列名称区分大小写（例如 Test-queue 和 test-queue 是不同的队列）。</p>
	FIFO 队列的名称必须以 .fifo 后缀。后缀计入 80 个字符的队列名称配额。若要确定队列是否 FIFO (p. 25) ，您可以检查队列名称是否以后缀结尾。
队列标签	我们建议不要向队列添加超过 50 个标签。
	标签 Key 是必需的，而标签 Value 是可选的。
	标签 Key 和标签 Value 区分大小写。

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

与消息相关的配额

下表列出了与消息相关的配额。

配额	描述
批处理消息 ID	批处理消息 ID 可以包含最多 80 个字符。可以使用以下字符：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 (p. 148) 部分中的 配置 AmazonSQSBufferedAsyncClient (p. 150) 。
消息内容	消息可以仅包含 XML、JSON 和非格式化的文本。允许以下 Unicode 字符：#x9 #xA #xD #x20 至 #xD7FF #xE000 至 #xFFFFD #x10000 至 #x10FFFF 此列表中未包含的任何字符将被拒绝。有关更多信息，请参阅 字符的 W3C 规范 。
消息组 ID	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. 53) 。
消息保留	默认情况下，消息将保留 4 天。最小值为 60 秒 (1 分钟)。最大值为 1209600 秒 (14 天)。
消息吞吐量	<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> <p>FIFO 队列</p> <ul style="list-style-type: none"> 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。 如果您使用批处理 (p. 148)，FIFO 队列的每个 API 方法每秒最多支持 3000 条消息

配额	描述
	<p>(SendMessageBatch、ReceiveMessage，或者DeleteMessageBatch)。3000 条消息代表 300 个 API 调用，每个调用带有包含 10 条消息的一个批处理。要申请提高配额，请提交支持请求。</p> <p>FIFO 队列的高吞吐量 (p. 28)</p> <ul style="list-style-type: none"> 不使用批处理 (SendMessage、ReceiveMessage, 和DeleteMessage)，则 FIFO 队列的高吞吐量每个 API 方法每秒最多支持 3000 条消息。为了获得最大吞吐量，请增加用于未进行批处理的邮件的消息组 ID 数。 您可以使用批处理 API (SendMessageBatch和DeleteMessageBatch)。每秒 30 000 条消息代表 3000 条 API 调用，每个调用带有包含 10 条消息的一个批处理。为了在使用时达到最大吞吐量SendMessageBatch和DeleteMessageBatch，则批处理请求中的所有消息必须使用相同的消息组 ID。 <p>有关更多信息，请参阅为 SQS FIFO 队列提供高吞吐量的分区和数据分配 (p. 29)。</p> <p>Note</p> <p>以上配额可在以下AWS区域：</p> <ul style="list-style-type: none"> 美国东部 (俄亥俄) 美国东部 (弗吉尼亚北部) 美国西部 (俄勒冈) 欧洲 (爱尔兰) <p>在所有其他AWS区域，每个 API 操作的最大吞吐量为每秒 1,500 (无批处理) 或 15,000 (使用批处理) 消息。</p>
消息定时器	消息的默认 (最小) 延迟为 0 秒。最长为 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 KB)。</p> <p>要发送大于 256 KB 的消息，可以使用适用于 Java 的 Amazon SQS 扩展客户端库。使用此库可以将包含引用的 Amazon SQS 消息发送到 Amazon S3 中的消息负载。最大负载大小为 2 GB。</p>
消息可见性超时	消息的默认可见性超时为 30 秒。最短值为 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 个委托人或 10 个条件。有关更多信息，请参阅 与策略相关的配额 (p. 92) 。

与策略相关的配额

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

Amazon SQS 队列自动化和问题排查

此部分提供了有关对 Amazon SQS 队列进行自动化和问题排查的信息。

主题

- [使用自动处理通知AWS使用 Amazon EventBridge 向 Amazon SQS 提供服务 \(p. 94\)](#)
- [使用对 Amazon Simple Queue Service 队列排查问题AWS X-Ray \(p. 94\)](#)

使用自动处理通知AWS使用 Amazon EventBridge 向 Amazon SQS 提供服务

Amazon EventBridge 让您实现自动化AWS服务并响应系统事件，例如应用程序可用性问题或资源更改。来自的事件AWS服务将近实时传输到 EventBridge。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 让您可以设置各种targets（例如 Amazon SQS 标准队列和 FIFO 队列），它们以 JSON 格式接收事件。有关更多信息，请参阅 [Amazon EventBridge 用户指南](#)。

使用对 Amazon Simple Queue Service 队列排查问题AWS X-Ray

AWS X-Ray 收集有关应用程序所服务请求的数据，并可让您查看和筛选数据以确定潜在的问题和进行优化的机会。对于任何被跟踪的对您应用程序的请求，您可以查看请求和响应的详细信息，以及您的应用程序对下游 AWS 资源、微服务、数据库和 HTTP Web API 进行的调用的详细信息。

要发送AWS X-Ray跟踪标 Amazon SQS，您可以执行下列操作之一：

- 使用 `X-Amzn-Trace-Id` 跟踪标头。
- 使用 `AWSTraceHeader` 消息系统属性 (p. 37)。

要收集有关错误和延迟的数据，您必须检测AmazonSQS客户端使用[AWSX-Ray SDK](#)。

您可以将AWS X-Ray控制台查看 Amazon SQS 与应用程序所用其他服务之间连接的映射。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 [Amazon SQS 和AWS X-Ray](#)中的AWS X-Ray开发人员指南。

Amazon SQS 中的安全性

本部分提供有关 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

- [数据保护 \(p. 95\)](#)
- [Amazon SQS 中的 Identity of Access Management \(p. 102\)](#)
- [Amazon SQS 中的日志记录和监控 \(p. 128\)](#)
- [Amazon SQS 的合规性验证 \(p. 139\)](#)
- [Amazon SQS 中的恢复能力 \(p. 140\)](#)
- [Amazon SQS 中的基础设施安全性 \(p. 140\)](#)
- [Amazon SQS 安全最佳实践 \(p. 140\)](#)

数据保护

这些区域有 : AWS 责任共担模式适用于 Amazon 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AWS云。您负责维护对托管在此基础设施上的内容的控制。此内容包括AWS您使用的服务。有关数据隐私的更多信息 , 请参阅[数据隐私常见问题](#)。有关欧洲数据保护的信息 , 请参阅[AWS责任共担模式](#)和[GDPR](#)博客帖子AWS安全博客。

出于数据保护目的 , 我们建议您保护AWS账户凭证并使用设置单独的用户账户AWS Identity and Access Management(IAM). 这仅向每个用户授予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您的数据 :

- 对每个账户使用 Multi-Factor Authentication (MFA)。
- 使用 SSL/TLS 与 AWS 资源进行通信。建议使用 TLS 1.2 或更高版本。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务中的所有默认安全控制。
- 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 中的个人数据。
- 如果在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-2 验证的加密模块 , 请使用 FIPS 终端节点。有关可用的 FIPS 终端节点的更多信息 , 请参阅[美国联邦信息处理标准 \(FIPS\) 第 140-2 版](#)。

我们强烈建议您切勿将敏感的可识别信息 (例如您客户的账号) 放入自由格式字段 (例如 Name (名称) 字段) 。这包括使用 Amazon SQS 或其他AWS服务使用控制台、API、AWS CLI , 或者AWS开发工具包。您输入到 Amazon SQS 或其他服务中的任何数据都可能被选取以包含在诊断日志中。当您向外部服务器提供 URL 时 , 请勿在 URL 中包含凭证信息来验证您对该服务器的请求。

以下部分提供有关 Amazon SQS 中数据保护的信息。

主题

- [数据加密 \(p. 95\)](#)
- [互联网络流量隐私 \(p. 101\)](#)

数据加密

数据保护指在数据传输 (发往和离开 Amazon SQS 时) 和处于静态 (存储在 Amazon SQS 数据中心的磁盘上时) 期间保护数据。您可以使用安全套接字层 (SSL) 或客户端加密保护传输中的数据。您可以在将消息保存到数据中心的磁盘之前请求 Amazon SQS 对其进行加密 , 然后在收到消息时对其进行解密 , 从而保护静态数据。

主题

- 静态加密 (p. 96)
- 密钥管理 (p. 98)

静态加密

借助服务器端加密 (SSE) , 您可以采用加密队列的方式传输敏感数据。SSE 使用 AWS Key Management Service(AWS KMS)。有关使用 AWS Management Console , 请参阅[为队列 \(控制台 \) 配置服务器端加密 \(SSE\) \(p. 13\)](#)。

有关使用 AWS SDK for Java (以及[CreateQueue](#)、[SetQueueAttributes](#), 和[GetQueueAttributes](#)操作) , 请参阅以下示例 :

- 使用服务器端加密 (SSE) (p. 54)
- 配置 AWS 服务的 KMS 权限 (p. 98)

一旦 Amazon SQS 收到消息 , SSE 就会对消息进行加密。这些消息以加密形式存储 , 仅当消息发送给授权使用者时 , Amazon SQS 才会对消息进行解密。

Important

对启用了 SSE 的队列的所有请求都必须使用 HTTPS 和[签名版本 4](#)。

一个[加密队列 \(p. 96\)](#) , 使用默认密钥 (AWS 托管 CMK) 无 Lambda 不同的 AWS 账户。的某些功能 AWS 服务 , 这些服 Amazon SQS 使用 AWS Security Token Service [AssumeRole](#) 操作与 SSE 兼容 , 但工作仅适用于标准队列 :

- 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[配置 AWS 服务的 KMS 权限 \(p. 98\)](#)和您的服务文档。

AWS KMS 将安全、高度可用的硬件和软件结合起来 , 提供可扩展到云的密钥管理系统。当您将 Amazon SQS 与 AWS KMS , [数据密钥 \(p. 97\)](#) 也将对您的消息数据进行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[客户主密钥 \(CMK\) \(p. 97\)](#)。
- 您也可以使用 AWS Amazon SQS 托管 CMK , 它对于每个账户和区域都是唯一的。
- AWS KMS 安全标准可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 。[是什么 AWS Key Management Service?](#)中的 AWS Key Management Service 开发人员指南。

主题

- 加密范围 (p. 96)
- 关键术语 (p. 97)

加密范围

SSE 将对 Amazon SQS 队列中的消息正文进行加密。

SSE 不对以下各项进行加密 :

- 队列元数据 (队列名称和属性)
- 消息元数据 (消息 ID、时间戳和属性)
- 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。这不会影响 Amazon SQS 的正常功能：

- 仅在启用队列加密后发送消息时对其进行加密。Amazon SQS 不会加密积压的消息。
- 任何加密的消息将保持加密状态，即使已禁用其队列的加密。

将消息移至死信队列 (p. 40) 不会影响其加密：

- 如果 Amazon SQS 将一条消息从加密的源队列移至未加密的死信队列，则该消息将保持加密状态。
- 如果 Amazon SQS 将一条消息从未加密的源队列移至加密的死信队列，则该消息将保持未加密状态。

关键术语

以下关键术语有助于您更好地了解 SSE 的功能。有关详细说明，请参阅[Amazon Simple Queue Service API 参考](#)。

数据密钥

数据加密密钥 (DEK) 负责加密 Amazon SQS 消息的内容。

有关更多信息，请参阅[数据密钥](#)中的AWS Key Management Service开发人员指南中的AWS Encryption SDK开发人员指南。

数据密钥重用周期

在调用之前，Amazon SQS 可以重用数据密钥来对消息进行加密或者解密的时间长度（以秒为单位）AWS KMS。一个表示秒数的证书，介于 60 秒 (1 分钟) 和 86400 秒 (24 小时) 之间。默认值为 300 (5 分钟)。有关更多信息，请参阅[了解数据密钥重用周期 \(p. 100\)](#)。

Note

在不太可能的情况下无法达到AWS KMS，Amazon SQS 将继续使用缓存数据密钥，直至重新建立连接。

客户主密钥 ID

您可以通过的AWS托管客户主密钥 (CMK) 或自定义 CMK — 在您的账户或其他账户中。虽然AWS适用于 Amazon SQS 的托管 CMK 始终是alias/aws/sqs，则自定义 CMK 的别名可以是，例如alias/[MyAlias](#)。您可以利用这些 CMK 保护 Amazon SQS 队列中的消息。

Note

记住以下内容：

- 如果未指定自定义 CMK，则 Amazon SQS 将使用AWS适用于 Amazon SQS 的托管 CMK。
- 第一次使用AWS Management Console指定AWS适用于 Amazon SQS 的管理 CMK，AWS KMS创建AWS适用于 Amazon SQS 的托管 CMK。
- 或者，您第一次使用SendMessage或者SendMessageBatch对启用 SSE 的队列上的操作，AWS KMS创建AWS适用于 Amazon SQS 的托管 CMK。

您可以创建 CMK、定义控制 CMK 的使用方式的策略和使用情况的审核 CMK 使用情况。客户托管密钥的部分AWS KMS控制台或[CreateKey](#) AWS KMSaction. 有关更多信息，请参阅[客户主密钥 \(CMK\)和创建密钥](#)中的AWS Key Management Service开发人员指南。有关 CMK 标识符的更多示例，请参阅[KeyId](#)中的AWS Key Management ServiceAPI 参考。有关查找 CMK 标识符的信息，请参阅[查找密钥 ID 和 ARN](#)中的AWS Key Management Service开发人员指南。

Important

使用 AWS KMS 无需支付额外费用。有关更多信息，请参阅[估算 AWS KMS 成本 \(p. 100\)](#)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 CMK 加密数据密钥，然后将加密的数据密钥与加密的消息一起存储。这种使用主密钥加密数据密钥的做法称为信封加密。

有关更多信息，请参阅[信封加密](#)中的AWS Encryption SDK开发人员指南。

密钥管理

Amazon SQS 与 AWS Key Management Service 管理[客户主密钥\(CMK\)](#) 用于服务器端加密 (SSE)。请参阅[静态加密 \(p. 96\)](#)，了解 SSE 信息和密钥管理定义。Amazon SQS 使用 CMK 来验证和保护用于加密和解密消息的数据密钥。以下部分提供有关在 Amazon SQS 服务中使用 CMK 和数据密钥的信息。

主题

- [配置 AWS KMS 权限 \(p. 98\)](#)
- [了解数据密钥重用周期 \(p. 100\)](#)
- [估算 AWS KMS 成本 \(p. 100\)](#)
- [AWS KMS 错误 \(p. 101\)](#)

配置 AWS KMS 权限

每个 CMK 都必须有一个密钥策略。请注意，您无法修改 AWS 适用于 Amazon SQS 的托管 CMK。此 CMK 的策略包括该账户（获授权可使用 Amazon SQS）中的所有委托人使用加密队列的权限。

对于客户托管的 CMK，您必须配置密钥策略，以便为每个队列创建者和使用者添加权限。为此，您将创建者和使用者指定为 CMK 密钥策略中的用户。有关的更多信息[AWS KMS 权限](#)，请参阅[AWS KMS 资源和操作](#)或者[AWS KMS API 权限参考](#)中的 AWS Key Management Service 开发人员指南。

或者，您可以在分配给委托人的 IAM 策略中指定所需的权限，而这些委托人可以创建和使用加密消息。有关更多信息，请参阅[将 IAM 策略与 AWS KMS 中的 AWS Key Management Service 开发人员指南](#)。

Note

尽管您可以配置从 Amazon SQS 发送到和接收的全局权限，AWS KMS 要求在特定区域内显式命名 CMK 的完整 ARN。Resource 部分。

配置 AWS 服务的 KMS 权限

几个 AWS 服务充当可以将事件发送到 Amazon SQS 队列的事件源。要允许这些事件源使用加密队列，您必须创建客户托管的 CMK 并在密钥策略中添加权限，以便服务使用所需的 AWS KMS API 方法。执行以下步骤来配置权限。

1. 创建客户托管 CMK。有关更多信息，请参阅[创建密钥](#)中的 AWS Key Management Service 开发人员指南。
2. 要允许 AWS 服务事件源使用 kms:GenerateDataKey 和 kms:Decrypt API 方法，请将以下语句添加到 CMK 密钥策略中。

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

```
        "Effect": "Allow",
        "Principal": {
            "Service": "service.amazonaws.com"
        },
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "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
Amazon CloudWatch Events	events.amazonaws.com
Amazon S3 事件通知	s3.amazonaws.com
Amazon SNS 主题订阅	sns.amazonaws.com

3. 配置现有 SSE 队列 ([p. 13](#)) 使用您的 CMK 的 ARN。
4. 向事件源提供加密队列的 ARN。

为创建者配置 KMS 权限

当数据密钥重用周期 ([p. 100](#)) 过期时，创建者下次调用 SendMessage 或 SendMessageBatch 时也会触发对 kms:GenerateDataKey 和 kms:Decrypt 的调用。对 kms:Decrypt 的调用是在使用新数据密钥之前验证它的完整性。因此，创建者必须具有客户主密钥 (CMK) 的 kms:GenerateDataKey 和 kms:Decrypt 权限。

将以下语句添加到创建者的 IAM 策略中。请记住，为密钥资源和队列资源使用正确的 ARN 值。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:SendMessage"
            ],
            "Resource": "arn:aws:sns:*:123456789012:MyQueue"
        }
    ]
}
```

为使用者配置 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，使用者必须对用于加密指定队列中的消息的任何客户主密钥 (CMK) 拥有 kms:Decrypt 权限。如果队列充当死信队列 ([p. 40](#))，则使用者必须还具有用于加密源队列中的消息的任何 CMK 的 kms:Decrypt 权限。将以下语句添加到使用者的 IAM 策略中。请记住，为密钥资源和队列资源使用正确的 ARN 值。

```
{  
    "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": [  
            "sns:ReceiveMessage"  
        ],  
        "Resource": "arn:aws:sns:*:123456789012:MyQueue"  
    }]  
}
```

了解数据密钥重用周期

这些区域有：[数据密钥重用周期 \(p. 97\)](#) 定义 Amazon SQS 重用相同数据密钥的最长持续时间。如果数据密钥重用周期结束，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。

- 较短的重用周期可提供更高的安全性，但会导致对 AWS KMS 进行多次调用，从而产生超出免费套餐的费用。
- 尽管用于加密和解密的数据密钥是单独缓存的，重用周期仍将应用于数据密钥的两个副本。
- 当数据密钥重用周期结束时，下一次调用 `SendMessage` 或 `SendMessageBatch` 时通常会触发对 AWS KMS `GenerateDataKey` 方法的调用以获取新数据密钥。此外，下一次调用 `SendMessage` 和 `ReceiveMessage` 时，每个调用都将触发调用 AWS KMS `Decrypt`，以便在使用数据密钥之前验证它的完整性。
- [委托人](#)(AWS 账户或 IAM 用户) 不会共享数据密钥（唯一委托人发送的消息将始终获得唯一数据密钥）。因此，对 AWS KMS 的调用数是数据密钥重用周期中正在使用的唯一委托人数的倍数：

估算 AWS KMS 成本

为了预测成本并更好地了解 AWS 账单，您可能想要了解 Amazon SQS 使用您的客户主密钥(CMK) 的频率。

Note

尽管以下公式可让您很好地了解预计成本，但由于 Amazon SQS 的分布式特性，实际成本可能更高。

计算 API 请求数(R) 每个队列，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 \(p. 97\)](#)(以秒为单位)。

P 是生产的数量[principals](#)(发送到 Amazon SQS 队列)。

C 是从 Amazon SQS 队列接收的使用委托人数。

Important

通常，创建委托人产生的费用是使用委托人的两倍。有关更多信息，请参阅[了解数据密钥重用周期 \(p. 100\)](#)。

如果创建者和使用者具有不同的 IAM 用户，则成本还增加。

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 AWS KMS 适用于 2 名委托人和 1 个队列的 API 调用

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 个创建委托人和 1 个使用委托人。

$$2,678,400 / 300 * (2 * 1 + 1) = 26,784$$

示例 2：计算 AWS KMS 多个创建者和使用者以及 2 个队列的 API 调用

此示例假定：

- 账单周期为 2 月 1 日 - 28 日 (2419200 秒)。
- 数据密钥重用周期设置为 24 小时 (86400 秒)。
- 有 2 个队列。
- 第一个队列有 3 个创建委托人和 1 个使用委托人。
- 第二个队列有 5 个创建委托人和 2 个使用委托人。

$$(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532$$

AWS KMS 错误

当您使用 Amazon SQS 和 AWS KMS，您可能会遇到错误。以下参考描述错误和可能的故障排除解决方案。

- [公用 AWS KMS 错误](#)
- [AWS KMS Decrypt 错误](#)
- [AWS KMS GenerateDataKey 错误](#)

互联网络流量隐私

Amazon SQS 的 Amazon Virtual Private Cloud (Amazon VPC) 终端节点是 VPC 内的逻辑实体，仅允许连接到 Amazon SQS。VPC 将请求路由到 Amazon SQS 并将响应路由回 VPC。以下部分提供有关使用 VPC 终端节点和创建 VPC 终端节点策略的信息。

主题

- [Amazon SQS 的 Amazon Virtual Private Cloud 终端节点 \(p. 101\)](#)
- [为 Amazon SQS 创建 Amazon VPC 终端节点策略 \(p. 102\)](#)

Amazon SQS 的 Amazon Virtual Private Cloud 终端节点

如果您使用 Amazon VPC 托管 AWS 资源，您可以在 VPC 和 Amazon SQS 之间建立连接。您可以使用此连接将消息发送到 Amazon SQS 队列，而无需跨公共 Internet。

亚马逊 VPC 允许您启动 AWS 自定义虚拟网络中的资源。可以使用 VPC 控制您的网络设置，例如 IP 地址范围、子网、路由表和网络网关。有关 VPC 的更多信息，请参阅 [Amazon VPC 用户指南](#)。

要将 VPC 连接到 Amazon SQS，您必须首先定义接口 VPC 终端节点，您可以通过它将 VPC 连接到其他 AWS 服务。该终端节点提供了到 Amazon SQS 的可靠、可扩展的连接，无需互联网网关、网络地址转换 (NAT) 实例或 VPN 连接。有关更多信息，请参阅 [教程：从 Amazon 虚拟私有云向 Amazon SQS 队列发送消息 \(p. 86\)](#) 和 [示例 5：如果不是来自 VPC 终端节点，则拒绝访问 \(p. 124\)](#) (在本指南中) [接口 VPC 终端节点 \(AWS PrivateLink\)](#) 中的 Amazon VPC 用户指南。

Important

- 您只能将 Amazon Virtual Private Cloud 与 HTTPS Amazon SQS 终端节点结合使用。
- 如果将 Amazon SQS 配置为从 Amazon VPC 发送消息，则必须启用私有 DNS 并采用 `sqs.us-east-2.amazonaws.com`。
- 私有 DNS 不支持传统终端节点，例如 `queue.amazonaws.com` 或 `us-east-2.queue.amazonaws.com`。

为 Amazon SQS 创建 Amazon VPC 终端节点策略

您可以为 Amazon SQS 的 Amazon VPC 终端节点创建一个策略，在该策略中可以指定以下内容：

- 可执行操作的委托人。
- 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 [使用 VPC 终端节点控制对服务的访问](#) 中的 Amazon VPC 用户指南

以下 VPC 终端节点策略示例指定 IAM 用户 MyUser 允许向 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：

- Amazon SQS API 操作，例如 `sqs:CreateQueue` 和 `sqs:DeleteQueue`。
- 其他尝试使用该 VPC 终端节点的 IAM 用户和规则。
- MyUser 将消息发送至不同的 Amazon SQS 队列。

Note

IAM 用户仍然可以使用在外 VPC。有关更多信息，请参阅 [示例 5：如果不是来自 VPC 终端节点，则拒绝访问 \(p. 124\)](#)。

Amazon SQS 中的 Identity of Access Management

访问 Amazon SQS 需要具有 AWS 可以用于对您的请求进行身份验证。这些凭证必须有权访问 AWS 资源，例如 Amazon SQS 队列和消息。以下章节详细说明如何使用 [AWS Identity and Access Management \(IAM\)](#) 和 Amazon SQS，以控制访问您的资源的权限，从而对这些资源进行保护。

主题

- [Authentication \(p. 103\)](#)
- [访问控制 \(p. 103\)](#)
- [Amazon SQS 中的访问管理概述 \(p. 104\)](#)
- [将基于身份的策略用于 Amazon SQS \(p. 108\)](#)
- [将自定义策略与 Amazon SQS 访问策略语言一起使用 \(p. 116\)](#)
- [将临时安全凭证用于 Amazon SQS \(p. 125\)](#)
- [Amazon SQS API 权限：操作和资源参考 \(p. 126\)](#)

Authentication

您可以以下面任一类型的身份访问 AWS：

- AWS 账户根用户— 当您第一次创建AWS账户中，最初使用的是一个对所有的完全访问权限的单点登录身份。AWS服务和资源。此身份称为AWS账户root 用户，可以通过使用您用于创建账户的电子邮件地址和密码进行登录来访问。强烈建议您不使用根用户执行日常任务，即使是管理任务。相反，请遵循[仅使用根用户创建您的第一个 IAM 用户的最佳实践](#)。然后请妥善保存根用户凭证，仅用它们执行少数账户和服务管理任务。
- IAM 用户— 一个[IAM 用户](#)是您的AWS账户，该账户具有特定的自定义权限（例如，在 Amazon SQS 中创建队列的权限）。您可以使用 IAM 用户名和密码登录以保护AWS网页，如[AWS Management Console](#)、[AWS论坛](#)，或[AWS SupportCenter](#)。

除了用户名和密码之外，您还可以为每个用户生成[访问密钥](#)。您可以在访问时使用这些键AWS服务，无论是通过[几个 SDK 中的一个](#)或通过使用[AWS Command Line Interface\(CLI\)](#)。SDK 和 CLI 工具使用访问密钥对您的请求进行加密签名。如果您不使用 AWS 工具，则必须自行对请求签名。Amazon SQS 支持签名版本 4，这是用于对入站 API 请求进行验证的协议。有关对请求进行身份验证的更多信息，请参阅[签名版本 4 签名流程中的AWS一般参考](#)。

- IAM 角色 – [IAM 角色](#)是可在账户中创建的一种具有特定权限的 IAM 身份。IAM 角色类似于 IAM 用户，因为它是AWS身份，该身份拥有的权限策略可确定其在AWS。但是，角色旨在让需要它的任何人代入，而不是唯一地与某个人员关联。此外，角色没有关联的标准长期凭证（如密码或访问密钥）。相反，当您代入角色时，它会为您提供角色会话的临时安全凭证。具有临时凭证的 IAM 角色在以下情况下很有用：
 - 联合身份用户访问— 您可以不创建 IAM 用户，而是使用来自AWS Directory Service、您的企业用户目录或 Web 身份提供商。这些被称为联合身份用户。AWS在通过访问请求访问权限时，将为联合用户分配角色。[身份提供商](#)。有关联合身份用户的更多信息，请参阅[联合身份用户和角色](#)中的IAM 用户指南。
 - AWS服务访问— 服务角色是[IAM 角色](#)服务假定为了代表您执行操作而执行操作。服务角色只在您的账户内提供访问权限，不能用于为访问其他账户中的服务授权。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅。[创建向AWS服务](#)中的IAM 用户指南。
 - 在 Amazon EC2 上运行的应用程序— 您可以使用 IAM 角色管理在 EC2 实例上运行的应用程序的临时凭证，然后使用AWS CLI或者AWSAPI 请求. 这优先于在 EC2 实例中存储访问密钥。要将 AWS 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅。[使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限](#)中的IAM 用户指南。

访问控制

Amazon SQS 拥有自己的基于资源的权限系统，该系统使用了以用于AWS Identity and Access Management(IAM) 策略。这意味着 Amazon SQS 策略和 IAM 策略的作用类似。

Note

重要的是要了解，所有 AWS 账户可以向其账户下的用户授予权限。跨账户访问允许您共享对 AWS 资源的访问，而不需要管理其他用户。有关使用跨账户访问的信息，请参阅[启用跨账户访问权限](#)中的 IAM 用户指南。

跨账户权限不适用于以下操作：

- [AddPermission](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [TagQueue](#)
- [UntagQueue](#)

目前 Amazon SQS 仅支持[IAM 中可用的条件密钥](#)。有关更多信息，请参阅[Amazon SQS API 权限：操作和资源参考 \(p. 126\)](#)。

Amazon SQS 中的访问管理概述

EVEREAWS资源归一个AWS 账户，创建或访问资源的权限由权限策略进行管理。账户管理员可以向 IAM 身份（用户、组和角色）附加权限策略，某些服务（如 Amazon SQS）也支持向资源附加权限策略。

Note

账户管理员（或管理员用户）是具有管理权限的用户。有关更多信息，请参阅[IAM 用户指南](#)中的 IAM 最佳实践。

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 \(p. 104\)](#)
- [了解资源所有权 \(p. 105\)](#)
- [管理对资源的访问 \(p. 105\)](#)
- [指定策略元素：操作、效果、资源和委托人 \(p. 108\)](#)
- [在策略中指定条件 \(p. 108\)](#)

Amazon Simple Queue Service 资源和操作

在 Amazon SQS 中，唯一的资源是queue。在策略中，可使用 Amazon 资源名称 (ARN) 标识策略应用到的资源。以下资源具有与之关联的唯一 ARN：

资源类型	ARN 格式
Queue	<code>arn:aws:sqs:<i>region</i>:<i>account_id</i>:<i>queue_name</i></code>

以下是队列的 ARN 格式的示例：

- 名为的队列的 ARN ARN`my_queue`在美国东部（俄亥俄）区域，属于AWS账户 123456789012：

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- 名为的队列的 ARN ARN`my_queue`Amazon SQS 支持的每个不同区域：

```
arn:aws:sqs:*:123456789012:my_queue
```

- 使用 ? 或 * 作为队列名称通配符的 ARN。在以下示例中，ARN 匹配前缀为 `my_prefix_` 的所有队列：

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

调用 [GetQueueAttributes](#) 操作可以获取现有队列的 ARN 值。QueueArn 属性的值即为队列的 ARN。有关 ARN 的更多信息，请参阅[IAM ARN](#)中的IAM 用户指南。

Amazon SQS 提供了用于处理队列资源的一组操作。有关更多信息，请参阅[Amazon SQS API 权限：操作和资源参考 \(p. 126\)](#)。

了解资源所有权

这些区域有：AWS 账户对在该账户下创建的资源具有所有权，而无论创建资源的人员是谁。具体而言，资源所有者是AWS 账户的委托人实体（即根账户、IAM 用户或 IAM 角色）对资源创建请求进行身份验证。以下示例说明了它的工作原理：

- 如果您使用的根账户凭证AWS 账户创建 Amazon SQS 队列，您的AWS 账户是资源的所有者（在 Amazon SQS 中，资源是 Amazon SQS 队列）。
- 如果您在AWS 账户并向用户授予创建队列的权限，则用户即可创建队列。但是，您的AWS 账户（用户所属的）拥有队列资源。
- 如果您在AWS 账户，则能够 Amazon SQS 入该角色的任何人都可以创建队列。您的AWS 账户（角色所属的）拥有队列资源。

管理对资源的访问

A权限策略描述了授予账户的权限。下一节介绍创建权限策略时的可用选项。

Note

本节讨论如何在 Amazon SQS 范围内使用 IAM。这里不提供有关 IAM 服务的详细信息。有关完整的 IAM 文档，请参阅[什么是 IAM？](#)中的IAM 用户指南。有关 IAM 策略语法和说明的信息，请参阅[AWS IAM 策略参考](#)中的IAM 用户指南。

附加到 IAM 身份的策略称为基于身份的策略 (IAM 策略) 和附加到资源的策略称为基于资源的策略。

基于身份的策略 (IAM 策略和 Amazon SQS 策略)

可通过两种方式向用户授予访问 Amazon SQS 队列的权限：使用 Amazon SQS 策略系统和使用 IAM 策略系统。您可以使用任一系统或这两种系统来将策略附加到用户或角色。在大多数情况下，使用任一系统都能获得相同的结果。例如，您可以执行以下操作：

- 将权限策略附加到账户中的用户或组—要为用户授予创建 Amazon SQS 队列的权限，请将权限策略附加到用户或用户所属的组。
- 将权限策略附加到另一个AWS 账户—要为用户授予创建 Amazon SQS 队列的权限，请将 Amazon SQS 权限策略附加到其他用户中的用户AWS 账户。

跨账户权限不适用于以下操作：

- [AddPermission](#)
 - [CreateQueue](#)
 - [DeleteQueue](#)
 - [ListQueues](#)
 - [ListQueueTags](#)
 - [RemovePermission](#)
 - [SetQueueAttributes](#)
 - [TagQueue](#)
 - [UntagQueue](#)
- 将权限策略附加到角色（授予跨账户权限）— 要授予跨账户权限，可将基于身份的权限策略附加到 IAM 角色。例如，AWS 账户管理员可以创建角色，以授予跨账户权限：AWS 账户 B（或 AWS 服务），具体如下：
- 账户 A 管理员可以创建一个 IAM 角色，然后向该角色附加授予其访问账户 A 中资源的权限策略。
 - 账户 A 管理员向将账户 B 标识为能够担任该角色的委托人的角色附加信任策略。
 - 账户 B 管理员向账户 B 中的任何用户委派担任该角色的权限。这将允许账户 B 中的用户创建或访问账户 A 中的队列。

Note

如果要向某项 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 IAM 委派权限的更多信息，请参阅[访问控制](#)中的 IAM 用户指南。

Amazon SQS 与 IAM 策略结合使用，但它拥有自己的策略基础架构。您可以对队列使用 Amazon SQS 策略，以指定哪些 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）。您可以为其授予权限的特定操作是 Amazon SQS 操作总体列表的子集。当您编写 Amazon SQS 策略并指定 * 来“允许任何 Amazon SQS 操作”，这意味着用户可以在此子集中执行所有操作。

下图说明了这些基本 Amazon SQS 策略中涵盖操作子集的一个策略的概念。该策略用于 queue_xyz，并且向 AWS 账户 1 和 AWS 账户 2 授予对指定队列使用任何允许的操作的权限。

Note

该策略中的资源被指定为 123456789012/queue_xyz，其中 123456789012 是拥有该队列的账户的 AWS 账户 ID。



随着 IAM 的引入和用户和 Amazon 资源名称 (ARN) , 一些关于 SQS 策略的事情已经发生了变化。以下示意图和表格描述了这些变化。



①有关为不同账户中用户授予权限的信息，请参阅。[教程：委托跨的访问权限AWS使用 IAM 角色的账户](#)中的IAM 用户指南。

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 \(p. 126\)](#)。

③您可以使用 Amazon 资源名称 (ARN) 指定资源，这是在 IAM 策略中指定资源的标准方法。有关 Amazon SQS 队列的 ARN 格式的信息，请参阅[Amazon Simple Queue Service 资源和操作 \(p. 104\)](#)。

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 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 IAM 之前，Amazon SQS 会自动向某个队列的创建者授予对该队列的完全控制权限（即访问该队列中所有可能的 Amazon SQS 操作）。现在，除非创建者使用 AWS 安全凭证，否则上述情况将不再出现。任何有权创建队列的用户如希望对所创建的队列执行任何操作，还必须拥有使用其他 Amazon SQS 操作的权限。

以下是一个示例策略，该策略允许用户使用所有 Amazon SQS 操作，但是仅限于名称前缀为文本字符串的队列。bob_queue_。

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Action": "sqs:*",  
        "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"  
    }]  
}
```

有关更多信息，请参阅。[将基于身份的策略用于 Amazon SQS \(p. 108\)](#), 和[身份 \(用户、组和角色\)中的 IAM 用户指南](#)。

指定策略元素：操作、效果、资源和委托人

对于每个Amazon Simple Queue Service 资源 (p. 104)时，该服务定义了一组操作。为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) 标识策略应用到的资源。
- 操作— 您可以使用操作关键字标识要允许或拒绝的资源操作。例如，`sqs:CreateQueue`权限允许用户执行 Amazon Amazon Amazon Amazon Simple Queue Service`CreateQueue`action。
- Effect— 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，附加了策略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、账户、服务或其他实体 (仅适用于基于资源的策略)。

要详细了解 Amazon SQS 策略语法和说明，请参阅[AWS IAM 策略参考](#)中的IAM 用户指南。

有关所有 Amazon 简单队列服务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 \(p. 126\)](#)。

在策略中指定条件

当您授予权限时，可使用 Amazon SQS 访问策略语言来指定策略何时生效的条件。例如，您可能希望策略仅在特定日期后应用。有关使用策略语言指定条件的更多信息，请参阅。[Condition](#)中的IAM 用户指南。

要表示条件，您可以使用预定义的条件键。Amazon SQS 没有特定的条件密钥。不过，其中有AWS范围内的条件密钥，您可以与 Amazon SQS 配合使用。目前，Amazon SQS 只支持 IAM 中可用条件密钥的有限子集：请参阅 [the section called “API 权限参考” \(p. 126\)](#)。

将基于身份的策略用于 Amazon SQS

本主题提供基于身份的策略的示例，在这些策略中，账户管理员可以向 IAM 身份（用户、组和角色）挂载权限策略。

Important

我们建议您首先阅读以下介绍性主题，这些主题讲解了管理 Amazon 简单队列服务资源访问权限的基本概念和选项。有关更多信息，请参阅[Amazon SQS 中的访问管理概述 \(p. 104\)](#)。

除了[ListQueues](#)，所有 Amazon SQS 操作均支持资源级权限。有关更多信息，请参阅[Amazon SQS API 权限：操作和资源参考 \(p. 126\)](#)。

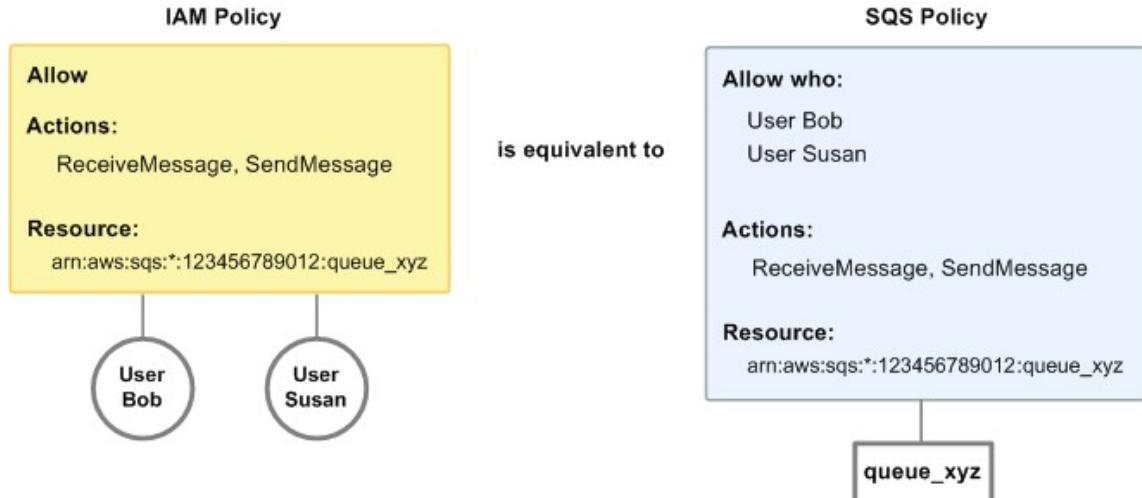
主题

- [使用 Amazon SQS 和 IAM 策略 \(p. 109\)](#)
- [Amazon SQS 控制台所需的权限 \(p. 110\)](#)
- [AWS适用于 Amazon SQS 的托管 \(预定义 \) 策略 \(p. 111\)](#)
- [适用于 Amazon SQS 的 IAM 策略的基本示例 \(p. 111\)](#)
- [Amazon SQS 策略的基本示例 \(p. 112\)](#)

使用 Amazon SQS 和 IAM 策略

可通过两种方式向用户授予访问 Amazon SQS 资源的权限：使用 Amazon SQS 策略系统和使用 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，下图显示了等效的 IAM 策略和 Amazon SQS 策略。IAM 策略授予 Amazon SQS 的权限 `ReceiveMessage` 和 `SendMessage` 的队列的操作 `queue_xyz` 在您的 AWS 账户，并且该策略已附加到用户 Bob 和 Susan（Bob 和 Susan 拥有策略所述的权限）。Amazon SQS 策略还授予 Bob 和 Susan 对 `ReceiveMessage` 和 `SendMessage` 操作相同队列。



IAM 策略和 Amazon SQS 策略之间有一个主要区别：Amazon SQS 策略系统允许您向其他 AWS 账户，而 IAM 则没有。

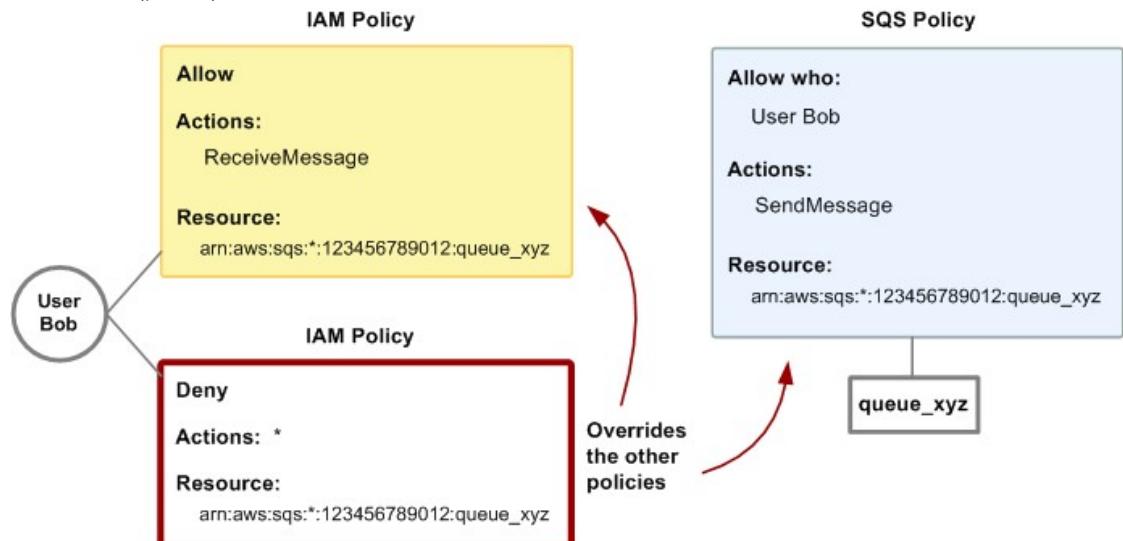
您可以自行决定如何综合使用上述两种系统来管理您的权限。以下示例展示这两种策略系统是如何共同运行的。

- 在第一个示例中，Bob 同时拥有适用于其账户的 IAM 策略和 Amazon SQS 策略。IAM 策略授予其账户对 `ReceiveMessage` 上的操作操作 `queue_xyz`，而 Amazon SQS 策略授予其账户对 `SendMessage` 在同一队列上的操作。下图阐明了这一概念。



如果 Bob 发送 ReceiveMessage 请求至 queue_xyz 时，IAM 策略允许操作。如果 Bob 发送 SendMessage 请求至 queue_xyz 时，Amazon SQS 策略允许操作。

- 在第二个示例中，Bob 滥用他对 queue_xyz 的访问权限，因此有必要删除他对该队列的所有访问权限。最简单的方法是添加一个策略，拒绝他访问该队列的所有操作。此策略会覆盖另外两个策略，因为显式 deny 始终覆盖 allow。有关策略评估逻辑的更多信息，请参阅[将自定义策略与 Amazon SQS 访问策略语言一起使用 \(p. 116\)](#)。下图阐明了这一概念。



您还可以向 Amazon SQS 策略中添加一条额外的语句，拒绝 Bob 以任何方式访问该队列。添加拒绝 Bob 访问该队列的 IAM 策略具有同样的效果。有关涉及 Amazon SQS 操作和资源的策略示例，请参阅[Amazon SQS 策略的基本示例 \(p. 112\)](#)。有关编写 Amazon SQS 策略的更多信息，请参阅[将自定义策略与 Amazon SQS 访问策略语言一起使用 \(p. 116\)](#)。

Amazon SQS 控制台所需的权限

希望使用 Amazon SQS 控制台的用户必须具有在用户的 AWS 账户。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 Amazon SQS 权限之外，要为 Amazon SQS 队列订阅 Amazon SNS 主题，控制台还需要 Amazon SNS 操作的权限。

如果创建比必需的最低权限更为严格的 IAM 策略，对于附加了该 IAM 策略的用户，控制台可能无法按预期正常运行。

对于只需要调用的用户，您无需为其提供最低控制台权限。AWS CLI 或 Amazon SQS 操作。

AWS适用于 Amazon SQS 的托管（预定义）策略

AWS 通过提供单独的 AWS 托管 IAM 策略。这些 AWS 托管策略通过授予常用案例所需的权限来简化对权限的使用。有关更多信息，请参阅 [AWS 管理的策略](#) 中的 IAM 用户指南。

以下 AWS 托管策略（可将它们附加到您的账户中的用户）是特定于 Amazon SQS 的：

- `AmazonSQSReadOnlyAccess`— 授予 Amazon SQS 队列的只读访问权限，使用 AWS Management Console。
- `AmazonSQSFullAccess`— 授予 Amazon SQS 队列的完全访问权限，使用 AWS Management Console。

您可以在 IAM 控制台上搜索和查看可用策略。此外，您还可以创建自定义 IAM 策略，以授予 Amazon SQS 操作和队列的相关权限。您可以将这些自定义策略附加到需要权限的 IAM 用户或组。

适用于 Amazon SQS 的 IAM 策略的基本示例

以下示例介绍了 Amazon SQS 权限策略。

Note

当您为 Amazon EC2 Auto Scaling 配置生命周期挂钩时，您无需编写策略将消息发送至 Amazon SQS 队列。有关更多信息，请参阅 [Amazon EC2 Auto Scaling 生命周期钩子](#) 中的适用于 Linux 实例的 Amazon EC2 用户指南。

示例 1：允许用户创建队列

在以下示例中，我们为 Bob 创建了一条策略，允许他访问所有 Amazon SQS 操作，但是仅限于名称前缀为文本字符串的队列。alice_queue_。

Amazon SQS 不会自动向队列创建者授予使用该队列的权限。因此，除了使用 Amazon SQS 操作，我们还必须向 Bob 显式授予使用所有 Amazon SQS 操作的权限。CreateQueue 操作在 IAM 策略中。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "sns:SendMessage",  
            "Resource": "arn:aws:sns:us-east-1:123456789012:alice_queue_*"  
        }  
    ]  
}
```

示例 2：允许开发人员向共享队列写入消息

在以下示例中，我们为开发人员创建了一个组，并挂载了一条策略，允许该组使用 Amazon SQSSendMessage 操作，但仅与属于指定 AWS 账户并被命名为 MyCompanyQueue。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "sns:SendMessage",  
            "Resource": "arn:aws:sns:us-east-1:123456789012:MyCompanyQueue"  
        }  
    ]  
}
```

```
    }]
}
```

您可以使用 * (而不是 SendMessage) 向委托人授予对共享队列执行以下操作的权限：ChangeMessageVisibility、DeleteMessage、GetQueueAttributes、GetQueueUrl、ReceiveMessage 和 SendMessage。

Note

虽然*包含其他权限类型提供的访问权限，Amazon SQS 会单独考虑这些权限。例如，可以向用户同时授予 * 和 SendMessage 权限，即使 * 包含 SendMessage 提供的访问权限，也是如此。此概念在您删除权限时也适用。如果委托人只有 * 权限，则请求删除 SendMessage 权限不会为委托人留下除此以外的一切权限。相反，该请求不起作用，因为委托人不具有显式 SendMessage 权限。要只为委托人留下 ReceiveMessage 权限，请先添加 ReceiveMessage 权限，然后删除 * 权限。

示例 3：允许管理人员获取队列的一般大小

在以下示例中，我们为管理人员创建了一个组，并挂载了一条策略，允许该组使用 Amazon SQSGetQueueAttributes 操作与属于指定 AWSAccount。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sns:GetQueueAttributes",
      "Resource": "*"
    }
  ]
}
```

示例 4：允许合作伙伴向特定的队列发送消息

您可以使用 Amazon SQS 策略或 IAM 策略完成此任务。如果您的合作伙伴有 AWS 账户，这样会比使用 Amazon SQS 策略简单。但是，合作伙伴公司中拥有 AWS 安全凭证的任何用户都可以向该队列发送消息。如果需要仅向特定用户或应用程序授予访问权限，则必须像对待公司内部的用户那样对待合作伙伴，使用 IAM 策略而不是 Amazon SQS 策略。

本示例将执行以下操作：

1. 创建名为 WidgetCo 的组，代表合作伙伴公司。
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": "sns:SendMessage",
      "Resource": "arn:aws:sns::123456789012:WidgetPartnerQueue"
    }
  ]
}
```

Amazon SQS 策略的基本示例

本部分显示了 Amazon SQS 常用案例的示例策略。

在将每个策略附加到用户时，可使用控制台验证该策略的效果。最初，用户没有权限并且无法在控制台中执行任何操作。在将策略附加到用户时，可以验证用户是否能在控制台中执行各种操作。

Note

建议使用两个浏览器窗口：一个浏览器窗口用于授予权限，另一个浏览器窗口用于使用用户凭证登录 AWS Management Console，以便在向用户授予权限时验证这些权限。

示例 1：向一项权限授予一项权限AWS 账户

以下示例策略授予AWS 账户number111122223333的SendMessage的队列的权限444455556666/queue1在美国东部（俄亥俄）区域。

```
{  
    "Version": "2012-10-17",  
    "Id": "Queue1_Policy_UUID",  
    "Statement": [  
        {  
            "Sid": "Queue1_SendMessage",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": [  
                    "111122223333"  
                ]  
            },  
            "Action": "sns:SendMessage",  
            "Resource": "arn:aws:sns:us-east-2:444455556666:queue1"  
        }]  
}
```

示例 2：向一项权限授予两项权限AWS 账户

以下示例策略授予AWS 账户number111122223333这两者的SendMessage和ReceiveMessage的队列的权限444455556666/queue1。

```
{  
    "Version": "2012-10-17",  
    "Id": "Queue1_Policy_UUID",  
    "Statement": [  
        {  
            "Sid": "Queue1_Send_Receive",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": [  
                    "111122223333"  
                ]  
            },  
            "Action": [  
                "sns:SendMessage",  
                "sns:ReceiveMessage"  
            ],  
            "Resource": "arn:aws:sns:us-east-2:444455556666:queue1"  
        }]  
}
```

示例 3：向两项权限授予所有权限AWS 账户

以下示例策略授予两个不同的AWS 账户数字（数字）111122223333和444455556666）权限对名为的队列使用 Amazon SQS 允许共享访问的所有操作123456789012/queue1在美国东部（俄亥俄）区域。

```
{  
    "Version": "2012-10-17",
```

```
"Id": "Queue1_Policy_UUID",
"Statement": [
    {
        "Sid": "Queue1_AllActions",
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                "111122223333",
                "44445556666"
            ]
        },
        "Action": "sns:*",
        "Resource": "arn:aws:sns: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role1和username1在AWS账户number111122223333跨账户权限，可使用Amazon SNS允许共享访问名为的队列的所有操作。123456789012/queue1在美国东部（俄亥俄）区域。

跨账户权限不适用于以下操作：

- [AddPermission](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [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": "sns:*",
            "Resource": "arn:aws:sns: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": "sns:ReceiveMessage",
    "Resource": "arn:aws:sns::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": "sns:ReceiveMessage",
        "Resource": "arn:aws:sns::111122223333:queue1",
        "Condition": {
            "DateGreaterThan": {
                "aws:CurrentTime": "2009-01-31T12:00Z"
            },
            "DateLessThan": {
                "aws:CurrentTime": "2009-01-31T15:00Z"
            }
        }
    }]
}
```

示例 7：向 CIDR 范围内的所有用户授予所有权限

以下示例策略向所有用户（匿名用户）授予对名为的队列使用可以共享的所有可能的 Amazon SQS 操作的权限。111122223333/queue1，但只有当请求来自 192.168.143.0/24 CIDR 范围。

```
{
    "Version": "2012-10-17",
    "Id": "Queue1_Policy_UUID",
    "Statement": [{
        "Sid": "Queue1_AnonymousAccess_AllActions_AllowlistIP",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "sns:*",
        "Resource": "arn:aws:sns::111122223333:queue1",
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": "192.168.143.0/24"
            }
        }
    }]
}
```

示例 8：不同 CIDR 范围内用户的允许列表和阻止列表权限

以下策略示例具有两项陈述：

- 第一个语句向 192.168.143.0/24 CIDR 范围 (192.168.143.188 除外) 内的所有用户 (匿名用户) 授予对名为 111122223333/queue1 的队列使用 SendMessage 操作的权限。
- 第二个语句阻止 10.1.2.0/24 CIDR 范围内的所有用户 (匿名用户) 使用该队列。

```
{  
    "Version": "2012-10-17",  
    "Id": "Queue1_Policy_UUID",  
    "Statement": [  
        {  
            "Sid": "Queue1_AnonymousAccess_SendMessage_IPLimit",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "sns:SendMessage",  
            "Resource": "arn:aws:sns::111122223333:queue1",  
            "Condition": {  
                "IpAddress": {  
                    "aws:SourceIp": "192.168.143.0/24"  
                },  
                "NotIpAddress": {  
                    "aws:SourceIp": "192.168.143.188/32"  
                }  
            }  
        },  
        {  
            "Sid": "Queue1_AnonymousAccess_AllActions_IPLimit_Deny",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "sns:*",  
            "Resource": "arn:aws:sns::111122223333:queue1",  
            "Condition": {  
                "IpAddress": {  
                    "aws:SourceIp": "10.1.2.0/24"  
                }  
            }  
        }  
    ]  
}
```

将自定义策略与 Amazon SQS 访问策略语言一起使用

如果您只希望允许 Amazon SQS 访问基于 AWS 账户 ID 和基本权限 (例如[SendMessage](#)或[ReceiveMessage](#))，您无需编写自己的策略。您可以只使用 Amazon SQS[AddPermission](#)action。

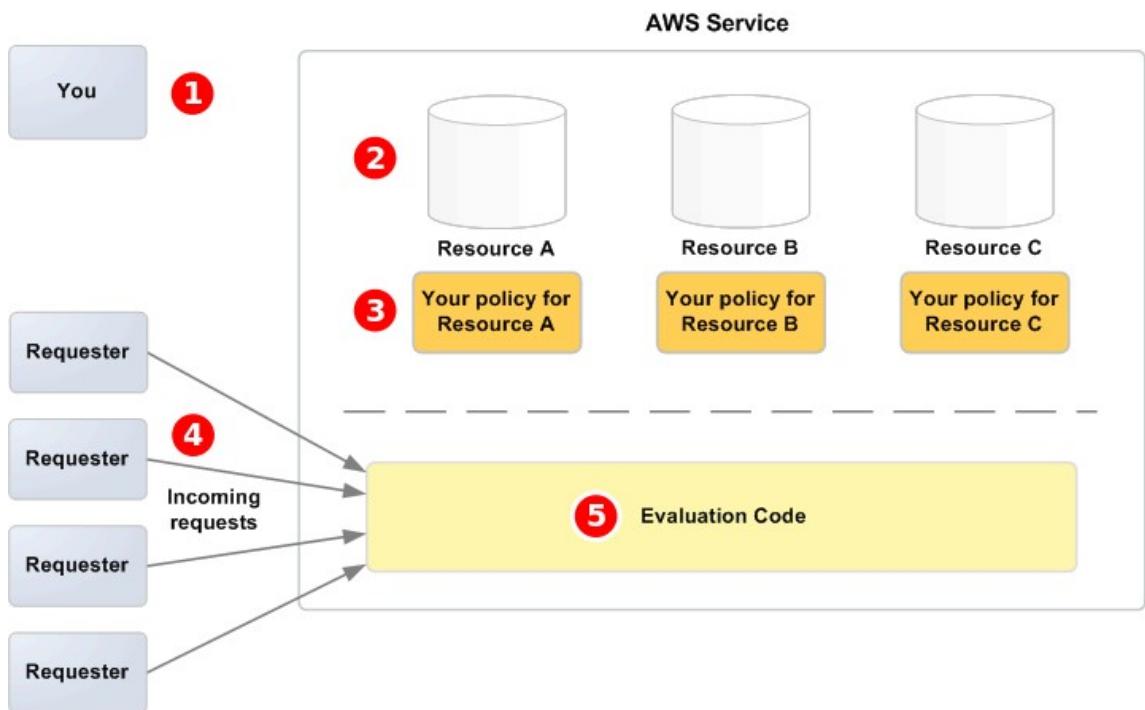
如果需要基于更具体的条件 (例如，请求到达的时间或请求者的 IP 地址) 来显式拒绝或允许访问，则需要自行编写 Amazon SQS 策略并将其上传到 AWS 系统使用 Amazon SQS[SetQueueAttributes](#)action。

主题

- [Amazon SQS 访问控制架构 \(p. 116\)](#)
- [Amazon SQS 访问控制处理工作流程 \(p. 117\)](#)
- [Amazon SQS 访问策略语言关键概念 \(p. 118\)](#)
- [Amazon SQS 访问策略语言评估逻辑 \(p. 119\)](#)
- [Amazon SQS 访问策略语言中显式拒绝和默认拒绝之间的关系 \(p. 121\)](#)
- [自定义 Amazon SQS 访问策略语言示例 \(p. 122\)](#)

Amazon SQS 访问控制架构

下图介绍了 Amazon SQS 资源的访问控制。



① 您，资源所有者。

②您的资源包含在AWS服务（例如，Amazon SQS队列）。

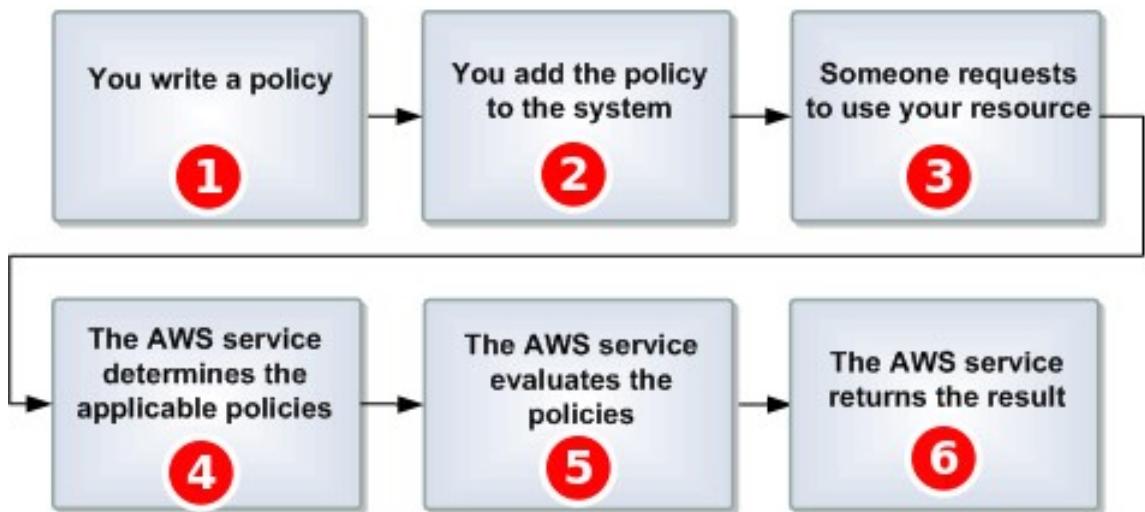
③ 您的策略。比较好的做法是为每个资源提供一个策略。AWS服务提供了一个API可用来上传和管理策略。

④ 请求者和他们向AWS服务传入的请求。

⑤访问策略语言评估代码。这是一组在AWS服务内能根据适用的策略对传入的请求进行评估并决定是否允许该请求者访问资源的代码。

Amazon SQS 访问控制处理工作流程

下图介绍了使用Amazon SQS访问策略语言进行访问控制的一般工作流程。



- ① 您为您的队列编写 Amazon SQS 策略。
- ② 您将策略上传到AWS。AWS 服务提供一个用来上传策略的 API。例如，您使用 `AmazonSQS.setQueueAttributes` 操作，以便为特定 Amazon SQS 队列上传策略。
- ③ 某人向您发出使用 Amazon SQS 队列的请求。
- ④ Amazon SQS 检查所有可用的 Amazon SQS 策略并决定哪些策略适用。
- ⑤ Amazon SQS 对这些策略进行评估，确定是否允许请求者使用您的队列。
- ⑥ Amazon SQS 根据策略评估结果返回 `Access denied` 错误消息，或者继续处理该请求。

Amazon SQS 访问策略语言关键概念

要自行编写策略，您必须熟悉 [JSON](#) 和一些关键概念。

允许

将 [Statement](#) (p. 119) 设为 [效果](#) (p. 118) 时 `allow` 的结果。

操作

该活动 [委托人](#) (p. 119) 具有执行的权限，通常是对 AWS。

默认拒绝

没有 [允许](#) (p. 118) 或 [显式拒绝](#) (p. 118) 设置的 [Statement](#) (p. 119) 的结果。

Condition

有关 [许可](#) (p. 119) 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](#) (p. 119) 的 [Statement](#) (p. 119) 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](#) (p. 118)、[允许](#) (p. 118) 和 [显式拒绝](#) (p. 118)。

显式拒绝

将 [Statement](#) (p. 119) 设为 [效果](#) (p. 118) 时 `deny` 的结果。

评估

Amazon SQS 根据确定是否拒绝或允许传入请求的过程 [Policy](#) (p. 119)。

Issuer

编写[Policy \(p. 119\)](#)以对资源授予权限的用户。根据定义，发布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 \(p. 118\)](#)和[密钥 \(p. 119\)](#)允许或拒绝对资源的访问的概念。

Policy

充当一条或多条[语句 \(p. 119\)](#)容器的文档。



Amazon SQS 使用策略确定是否授予用户访问资源的权限。

委托人

收到[许可 \(p. 119\)](#)中[Policy \(p. 119\)](#)的用户。

资源

[委托人 \(p. 119\)](#)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 \(p. 119\)](#)文档。

请求者

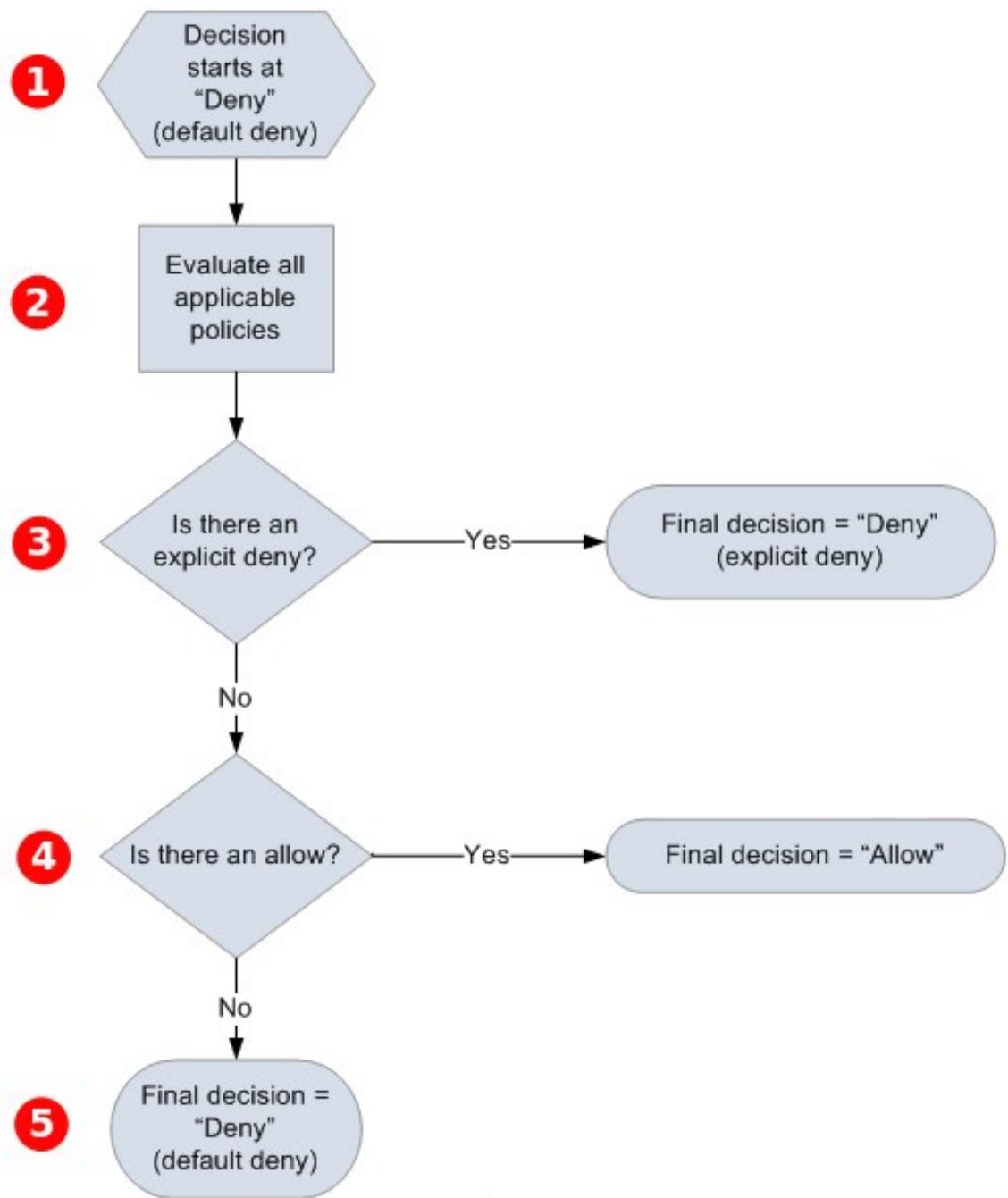
发送[访问资源 \(p. 119\)](#)请求的用户。

Amazon SQS 访问策略语言评估逻辑

在评估期间，Amazon SQS 确定应允许还是拒绝资源所有者以外的某人发送的请求。评估逻辑遵循多个基本规则：

- 在默认情况下，您拒绝任何人发送的所有使用资源的请求。
- [允许 \(p. 118\)](#) 将覆盖任意[默认拒绝 \(p. 118\)](#)。
- [显式拒绝 \(p. 118\)](#) 将覆盖任意允许。
- 策略评估的顺序并不重要。

下图详述了 Amazon SQS 如何评估访问权限的决策。



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、委托人、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。

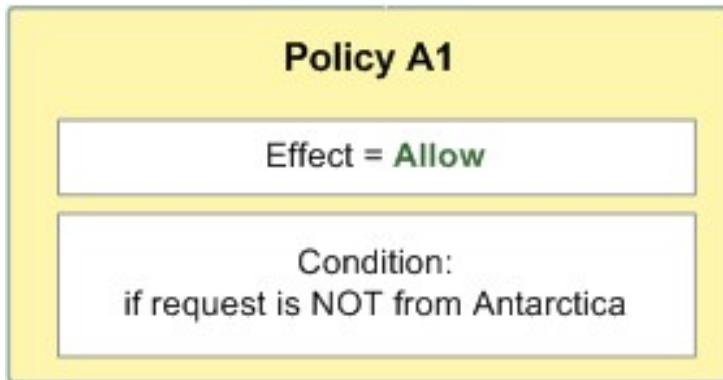
④ 如果没有找到显式拒绝指令，那么执行代码将寻找应用于请求的任何允许指令。即使执行代码只找到了一个，也会返回允许决策，整个过程完成 (服务将继续处理该请求)。

如果未找到任何允许指令，那么最终决策将是拒绝（因为没有显式拒绝或允许，所以这将被视为一个默认拒绝）。

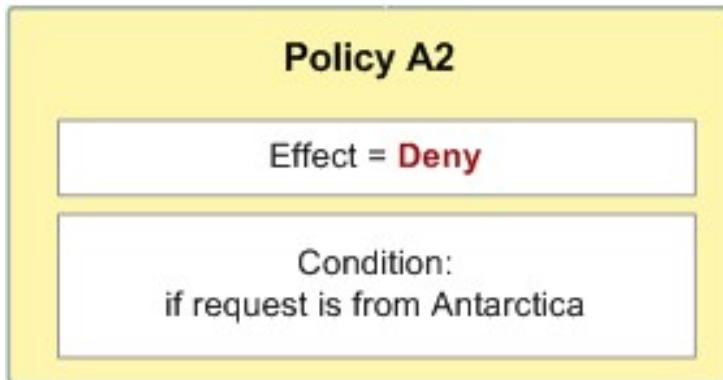
Amazon SQS 访问策略语言中显式拒绝和默认拒绝之间的关系

如果 Amazon SQS 策略不直接应用于请求，该请求的结果为[默认拒绝 \(p. 118\)](#)。例如，如果用户请求使用 Amazon SQS 的权限，但应用于该用户的唯一策略可使用 DynamoDB，则该请求的结果为默认拒绝。

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 \(p. 118\)](#)或[显式拒绝 \(p. 118\)](#)。[效果 \(p. 118\)](#)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。下列示意图说明 Amazon SQS 策略。

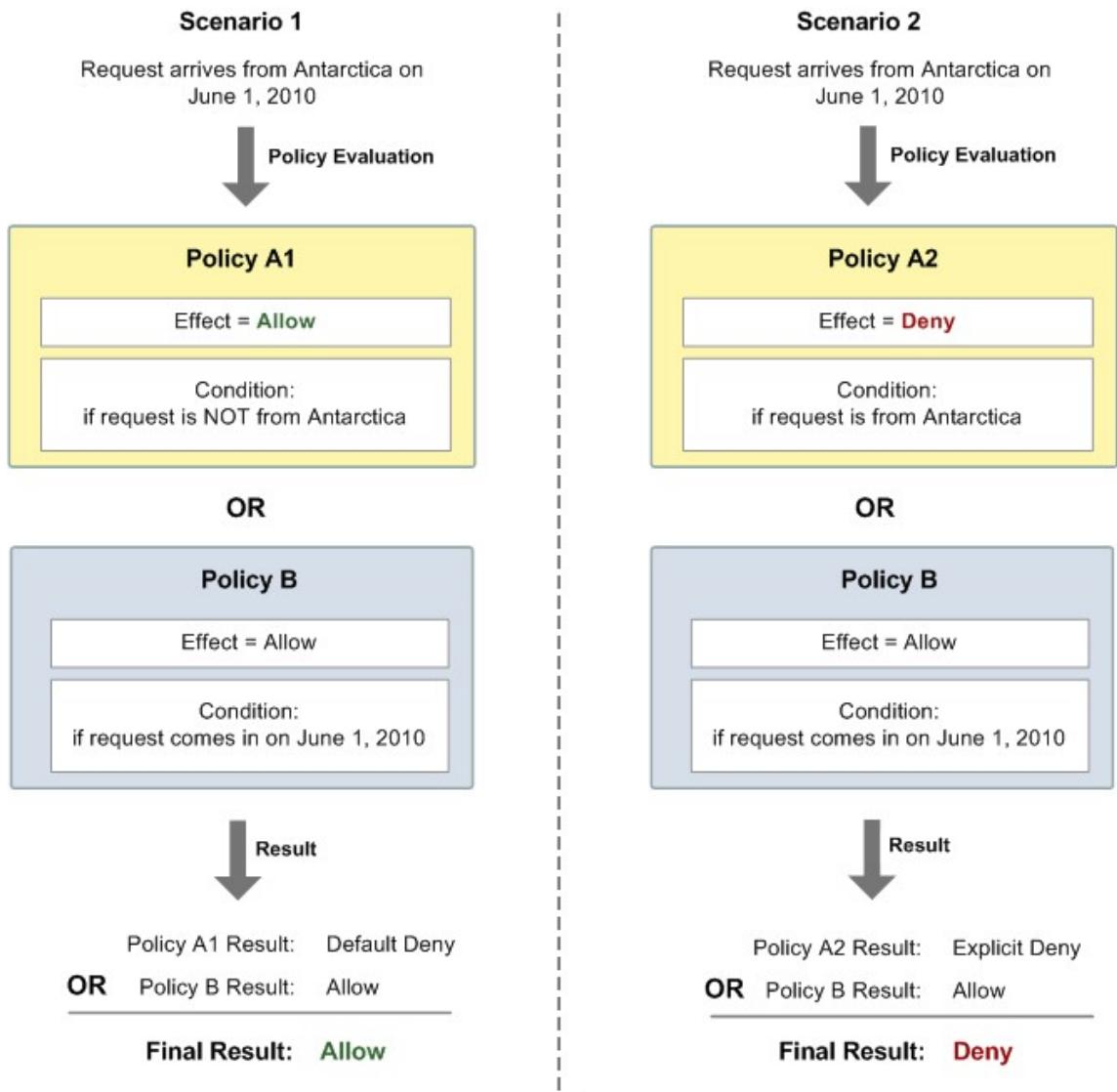


如果用户从美国发出请求，那么条件已经满足（该请求不是来自南极洲），则该请求的结果是允许。但是，如果用户从南极洲发出请求，那么条件未满足，该请求的默认结果为默认拒绝。您可以编写策略 A2，显式拒绝来自南极洲的请求，这样结果就变为显式拒绝。下列示意图说明了该策略。



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，Policy A1 的结果为默认拒绝，Policy 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。Policy B 返回的允许结果将置换 Policy A1 的默认拒绝结果，因此，请求获得允许。

在场景 2 中，Policy B2 的结果为显式拒绝，Policy B 的结果为允许。从 Policy A2 发出的显式拒绝将覆盖从 Policy B 发出的允许，因此该请求会被拒绝。

自定义 Amazon SQS 访问策略语言示例

以下示例是典型的 Amazon SQS 访问策略。

示例 1：授予一个账户的权限

以下示例 Amazon SQS 策略提供 AWS 账户 111122223333 向发送和接收来自发送的权限 queue2 拥有的 AWS 账户 444455556666.

```
{  
    "Version": "2012-10-17",  
    "Id": "UseCase1",  
    "Statement": [  
        {"Sid": "AllowQueue1", "Effect": "Allow", "Principal": "444455556666", "Action": "SQS:SendMessage", "QueueArn": "arn:aws:sqs:us-east-1:111122223333:queue1"},  
        {"Sid": "AllowQueue2", "Effect": "Allow", "Principal": "444455556666", "Action": "SQS:SendMessage", "QueueArn": "arn:aws:sqs:us-east-1:444455556666:queue2"},  
        {"Sid": "AllowQueue1", "Effect": "Allow", "Principal": "444455556666", "Action": "SQS:ReceiveMessage", "QueueArn": "arn:aws:sqs:us-east-1:111122223333:queue1"}  
    ]  
}
```

```
"Statement" : [ {
    "Sid": "1",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "111122223333"
        ]
    },
    "Action": [
        "sns:SendMessage",
        "sns:ReceiveMessage"
    ],
    "Resource": "arn:aws:sns:us-east-2:44445556666:queue2"
} ]  
}
```

示例 2：授予一个或多个账户的权限

以下示例 Amazon SQS 策略给出了一个或多个 AWS 账户对特定时间段内由账户所拥有的队列的访问权限。有必要编写此策 Amazon SQS 用 [SetQueueAttributes](#) 操作，因为 [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",
                "44445556666"
            ]
        },
        "Action": [
            "sns:SendMessage",
            "sns:ReceiveMessage"
        ],
        "Resource": "arn:aws:sns:us-east-2:44445556666:queue2",
        "Condition": {
            "DateLessThan": {
                "AWS:CurrentTime": "2009-06-30T12:00Z"
            }
        }
    } ]
}
```

示例 3：对 Amazon EC2 实例的请求授予权限

以下示例 Amazon SQS 策略对来自 Amazon EC2 实例的请求授予权限。此示例根据“[示例 2：授予一个或多个账户的权限 \(p. 123\)](#)”示例编写：它将访问时间限制在 2009 年 6 月 30 日中午 12 点 (UTC) 之前，将访问 IP 的范围限制在 203.0.113.0/24。有必要编写此策 Amazon SQS 用 [SetQueueAttributes](#) 操作，因为 [AddPermission](#) 操作不允许在授予队列访问权限时指定 IP 地址限制。

```
{
    "Version": "2012-10-17",
    "Id": "UseCase3",
    "Statement" : [ {
        "Sid": "1",
        "Effect": "Allow",
        "Principal": {
            "AWS": [

```

```
        "111122223333"
    ],
},
"Action": [
    "sns:SendMessage",
    "sns:ReceiveMessage"
],
"Resource": "arn:aws:sns:us-east-2:44445556666:queue2",
"Condition": {
    "DateLessThan": {
        "AWS:CurrentTime": "2009-06-30T12:00Z"
    },
    "IpAddress": {
        "AWS:SourceIp": "203.0.113.0/24"
    }
}
}]
```

示例 4：拒绝访问特定账户

以下示例 Amazon SQS 策略拒绝特定的 AWS 账户对队列的访问权限。此示例建立在[“示例 1：授予一个账户的权限 \(p. 122\)”](#)示例：它拒绝访问指定的 AWS 账户。有必要编写此策 Amazon SQS 用 `SetQueueAttributes` 操作，因为 `AddPermission` 操作不允许拒绝对队列的访问（它只允许授予对队列的访问权限）。

```
{
    "Version": "2012-10-17",
    "Id": "UseCase4",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Deny",
            "Principal": {
                "AWS": [
                    "111122223333"
                ]
            },
            "Action": [
                "sns:SendMessage",
                "sns:ReceiveMessage"
            ],
            "Resource": "arn:aws:sns:us-east-2:44445556666:queue2"
        }
    ]
}
```

示例 5：如果不是来自 VPC 终端节点，则拒绝访问

以下示例 Amazon SQS 策略限制访问 queue1：可以执行 `SendMessage` 和 `ReceiveMessage` 仅从 VPC 终端节点 ID 执行的操作 `vpce-1a2b3c4d`（使用指定的 `aws:sourceVpc` 条件）。有关更多信息，请参阅 [Amazon SQS 的 Amazon Virtual Private Cloud 终端节点 \(p. 101\)](#)。

Note

- `aws:sourceVpc` 条件不需要 VPC 终端节点资源的 ARN，而只需要 VPC 终端节点 ID。
- 您可以通过修改以下示例，通过拒绝所有 Amazon SQS 操作 (`sqs:*`) 在第二个语句中。但是，此类策略声明将规定所有操作（包括修改队列权限所需的管理操作）必须通过在策略中定义的特定 VPC 终端节点进行，这可能会阻止用户以后修改队列权限。

```
{
    "Version": "2012-10-17",
    "Id": "UseCase5",
```

```
"Statement": [{

    "Sid": "1",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "111122223333"
        ]
    },
    "Action": [
        "sns:SendMessage",
        "sns:ReceiveMessage"
    ],
    "Resource": "arn:aws:sns:us-east-2:111122223333:queue1"
},
{
    "Sid": "2",
    "Effect": "Deny",
    "Principal": "*",
    "Action": [
        "sns:SendMessage",
        "sns:ReceiveMessage"
    ],
    "Resource": "arn:aws:sns:us-east-2:111122223333:queue1",
    "Condition": {
        "StringNotEquals": {
            "aws:sourceVpce": "vpce-1a2b3c4d"
        }
    }
}
]
```

将临时安全凭证用于 Amazon SQS

除了创建有其自己的安全凭证的 IAM 用户之外，IAM 还允许您向任何用户授予临时安全证书，从而使该用户可以访问您的 AWS 服务和资源。您可以管理具有 AWS 账户（IAM 用户）。您还可以对系统中没有 AWS 账户（联合用户）。此外，您创建的能访问 AWS 资源的应用程序也可被视为用户。

您可以使用上述临时安全证书向 Amazon SQS 发出请求。API 库会使用这些凭证计算必要的签名值，以便对您的请求进行身份验证。如果您使用过期凭证发送请求，Amazon SQS 会拒绝该请求。

Note

您不能基于临时凭证设置策略。

Prerequisites

1. 使用 IAM 创建临时安全凭证：

- 安全令牌
 - 访问密钥 ID
 - 秘密访问密钥
2. 使用临时访问密钥 ID 和安全令牌准备待签字符串。
3. 使用临时秘密访问密钥（而不是您自己的秘密访问密钥）对您的查询 API 请求签名。

Note

在提交已签名的查询 API 请求时，请使用临时访问密钥 ID（而不是您自己的访问密钥 ID），并且包含安全令牌。有关 IAM 临时安全凭证支持的更多信息，请参阅[授予针对的临时访问权限 AWS 资源中的 IAM 用户指南](#)。

使用临时安全凭证调用 Amazon SQS 查询 API 操作

1. 使用 AWS Identity and Access Management 请求临时安全令牌。有关更多信息，请参阅 。[创建临时安全证书以启用 IAM 用户访问](#)中的IAM 用户指南。
IAM 会返回一个安全令牌、一个访问密钥 ID 和一个秘密访问密钥。
2. 使用临时访问密钥 ID (而不是您自己的访问密钥 ID) 准备查询，并且包含安全令牌。使用临时秘密访问密钥 (而不是您自己的秘密访问密钥) 对请求签名。
3. 提交已使用临时访问密钥 ID 和安全令牌签名的查询字符串。

以下示例展示了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 。[签署条件AWSAPI 请求](#)中的Amazon Web Services 一般参考。

```
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
```

Amazon SQS API 权限：操作和资源参考

当您设置[访问控制 \(p. 103\)](#)并编写您可以附加到 IAM 身份的权限策略时，可以使用下表作为参考。这些区域有：列表包含每个 Amazon 简单队列服务操作、您可授予执行该操作的权限的对应操作以及AWS资源，您可以授予权限。

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，sqss:CreateQueue)。

目前 Amazon SQS 仅支持[IAM 中可用的条件密钥](#)：

- aws:CurrentTime
- aws:EpochTime
- aws:SecureTransport
- aws:SourceAccount

- aws:SourceArn

Note

此条件可确保 AWS 服务仅代表资源授予访问权限 AWS 账户拥有。您无法将 IAM 角色的 ARN 指定为源 ARN，因为 IAM 角色既不是源也不是服务。

- aws:SourceIP
- aws:UserAgent
- aws:MultiFactorAuthAge
- aws:MultiFactorAuthPresent
- aws:PrincipalOrgID
- aws:RequestTag
- aws:sourceVpc
- aws:TagKeys
- aws:TokenAge

Amazon Amazon Amazon Simple Queue Service API 和必需的操作权限

[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*

[删除消息](#)

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

此部分提供有关对 Amazon SQS 队列进行日志记录和监控的信息。

主题

- 使用记录 Amazon SQS API 调用 AWS CloudTrail (p. 129)
- 使用 CloudWatch 监控 Amazon SQS 队列 (p. 132)

使用记录 Amazon SQS API 调用 AWS CloudTrail

Amazon SQS 与 AWS CloudTrail，后者是一项提供 Amazon SQS 记录的服务，该服务会调用用户、角色或 AWS 服务制造商。CloudTrail 将与 Amazon SQS 队列相关的 API 调用作为事件捕获，包括来自 Amazon SQS 控制台的调用和来自 Amazon SQS API 的代码调用。有关 CloudTrail 的更多信息，请参阅[AWS CloudTrail 用户指南](#)。

Note

标准队列和 FIFO 队列支持 CloudTrail 日志记录。

使用 CloudTrail 所收集的信息，您可以确定向 Amazon SQS API 发出的特定请求、请求者的 IP 地址、请求者的身份、请求的日期和时间等。如果您配置跟踪中，您可以使 CloudTrail 事件持续传送至 Amazon S3 存储桶。如果您不配置跟踪，则可以在 CloudTrail 控制台中的事件历史记录中查看最新事件。有关更多信息，请参阅[创建跟踪概述中的 AWS CloudTrail 用户指南](#)。

CloudTrail 中的 Amazon SQS 信息

在创建 AWS 账户，则启用 CloudTrail。当发生受支持的 Amazon SQS 事件活动时，它会记录在 CloudTrail 事件中，并与其他 AWS 事件历史记录中的服务事件。您可以查看、搜索和下载最新事件 AWS 账户。有关更多信息，请参阅[使用 CloudTrail 事件历史记录查看事件](#)中的 AWS CloudTrail 用户指南。

通过跟踪，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。您可以创建跟踪以持续记录事件 AWS 账户。默认情况下，在使用 AWS Management Console 创建跟踪时，此跟踪将应用于所有 AWS 区域。跟踪记录来自所有 AWS 区域并将日志文件传送至指定的 Amazon S3 存储桶。您还可以配置其他 AWS 服务，进一步分析 CloudTrail 日志中收集的事件数据并采取措施。有关更多信息，请参阅 AWS CloudTrail 用户指南 中的以下主题：

- [CloudTrail 支持的服务和集成](#)
- [为 CloudTrail 配置 Amazon SNS 通知](#)
- [接收多个区域中的 CloudTrail 日志文件](#)
- [接收多个账户中的 CloudTrail 日志文件](#)

Amazon SQS 支持记录以下操作的日志：

- [AddPermission](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [PurgeQueue](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [TagQueue](#)
- [UntagQueue](#)

每个事件或日志条目都包含有关请求者的信息。此信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 IAM 用户凭证发出的？
- 请求是使用角色还是联合身份用户的临时安全凭证发出的？
- 请求是否由其他 AWS 服务发出？

有关更多信息，请参阅 。[CloudTrail userIdentity 元素](#)中的AWS CloudTrail用户指南。

Amazon SQS 日志文件条目示例

CloudTrail 日志文件包含一个或多个日志条目，每个条目由多个 JSON 格式的事件组成。一个日志条目表示来自任何源的一个请求，包括有关所请求的操作、所有参数以及操作的日期和时间等信息。日志条目不一定具有任何特定顺序。即，它们不是公共 API 调用的有序堆栈跟踪。

AddPermission

下面的示例显示了一个 CloudTrail 日志条目，该条目是AddPermissionAPI 调用。

```
{  
    "Records": [  
        {  
            "eventVersion": "1.06",  
            "userIdentity": {  
                "type": "IAMUser",  
                "principalId": "AKIAI44QH8DHBEEXAMPLE",  
                "arn": "arn:aws:iam::123456789012:user/Alice",  
                "accountId": "123456789012",  
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",  
                "userName": "Alice"  
            },  
            "eventTime": "2018-06-28T22:23:46Z",  
            "eventSource": "sns.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://sns.us-east-2.amazonaws.com/123456789012/MyQueue"  
            },  
            "responseElements": null,  
            "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",  
            "eventId": "0987g654-32f1-09e8-d765-c4f3fb2109fa"  
        }  
    ]  
}
```

CreateQueue

下面的示例显示了一个 CloudTrail 日志条目，该条目的是CreateQueueAPI 调用。

```
{  
    "Records": [  
        {  
            "eventVersion": "1.06",  
            "userIdentity": {  
                "type": "IAMUser",  
                "principalId": "AKIAI44QH8DHBEEXAMPLE",  
                "arn": "arn:aws:iam::123456789012:user/Alejandro",  
                "accountId": "123456789012",  
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",  
                "userName": "Alejandro"  
            },  
            "eventTime": "2018-06-28T22:23:46Z",  
            "eventSource": "lambda.amazonaws.com",  
            "eventName": "CreateQueue",  
            "awsRegion": "us-east-2",  
            "sourceIPAddress": "203.0.113.0",  
            "userAgent": "Amazon Lambda Function/1.0",  
            "requestParameters": {  
                "queueName": "MyQueue",  
                "attributes": {  
                    "K": "V"  
                }  
            },  
            "responseElements": null,  
            "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",  
            "eventId": "0987g654-32f1-09e8-d765-c4f3fb2109fa"  
        }  
    ]  
}
```

```
        "userName": "Alejandro"
    },
    "eventTime": "2018-06-28T22:23:46Z",
    "eventSource": "sns.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://sns.us-east-2.amazonaws.com/123456789012/MyQueue"
    },
    "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
    "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
}
]
```

DeleteQueue

下面的示例显示了一个 CloudTrail 日志条目，该条目的是DeleteQueueAPI 调用。

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AKIAI44QH8DHBEEXAMPLE",
                "arn": "arn:aws:iam::123456789012:user/Carlos",
                "accountId": "123456789012",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "Carlos"
            },
            "eventTime": "2018-06-28T22:23:46Z",
            "eventSource": "sns.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://sns.us-east-2.amazonaws.com/123456789012/MyQueue"
            },
            "responseElements": null,
            "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
            "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
        }
    ]
}
```

RemovePermission

下面的示例显示了一个 CloudTrail 日志条目，该条目的是RemovePermissionAPI 调用。

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "IAMUser",

```

```
"principalId": "AKIAI44QH8DHBEEXAMPLE",
"arn": "arn:aws:iam::123456789012:user/Jane",
"accountId": "123456789012",
"accessKeyId": "AKIAIOSFODNN7EXAMPLE",
"userName": "Jane"
},
"eventTime": "2018-06-28T22:23:46Z",
"eventSource": "sns.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://sns.us-east-2.amazonaws.com/123456789012/MyQueue"
},
"responseElements": null,
"requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
"eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
}
]
```

SetQueueAttributes

下面的示例显示了一个 CloudTrail 日志条目，该条目是SetQueueAttributes：

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AKIAI44QH8DHBEEXAMPLE",
                "arn": "arn:aws:iam::123456789012:user/Maria",
                "accountId": "123456789012",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "Maria"
            },
            "eventTime": "2018-06-28T22:23:46Z",
            "eventSource": "sns.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://sns.us-east-2.amazonaws.com/123456789012/MyQueue"
            },
            "responseElements": null,
            "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
            "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
        }
    ]
}
```

使用 CloudWatch 监控 Amazon SQS 队列

Amazon SQS 和 Amazon CloudWatch 集成在一起，因此您可以使用 CloudWatch 查看和分析您的 Amazon SQS 队列的指标。您可以查看和分析队列的指标，从Amazon SQS 控制台 (p. 133)，CloudWatch 控制

台 (p. 133) , 使用 AWS CLI (p. 135) , 或者使用 CloudWatch API (p. 135)。您还可以设置 CloudWatch 警报 (p. 135)Amazon SQS 指标。

Amazon SQS 队列的 CloudWatch 指标将以一分钟为间隔自动收集并推送到 CloudWatch。系统会对满足 CloudWatch 准则的所有队列收集这些指标。active。如果某个队列包含任何消息或任何操作访问该队列，则 CloudWatch 最多在 6 小时内将其视为有效。

Note

- CloudWatch 中报告的 Amazon SQS 指标是不收费的。它们作为 Amazon SQS 服务的一部分提供。
- 标准队列和 FIFO 队列均支持 CloudWatch 指标。

主题

- 访问 Amazon SQS 的 CloudWatch 指标 (p. 133)
- 为 Amazon SQS 指标创建 CloudWatch 警报 (p. 135)
- 适用于 Amazon SQS 的可用 CloudWatch 指标 (p. 135)

访问 Amazon SQS 的 CloudWatch 指标

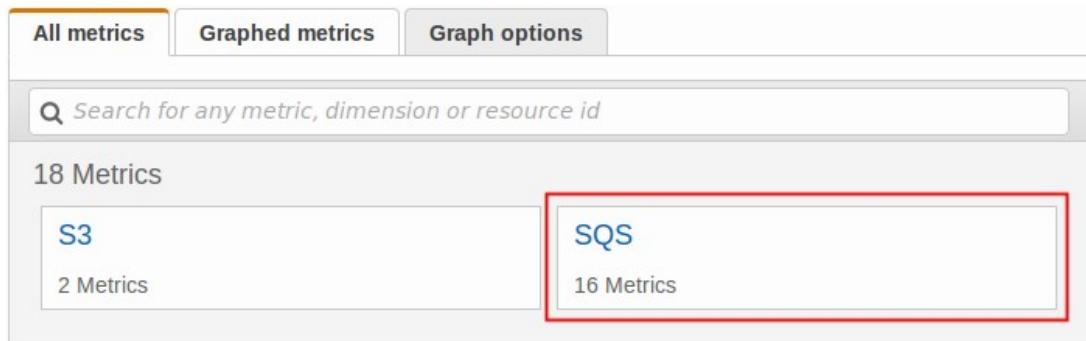
Amazon SQS 和 Amazon CloudWatch 集成在一起，因此您可以使用 CloudWatch 查看和分析您的 Amazon SQS 队列的指标。您可以查看和分析队列的指标，从 Amazon SQS 控制台 (p. 133) , CloudWatch 控制台 (p. 133) , 使用 AWS CLI (p. 135) , 或者使用 CloudWatch API (p. 135)。您还可以设置 CloudWatch 警报 (p. 135)Amazon SQS 指标。

Amazon SQS 控制台

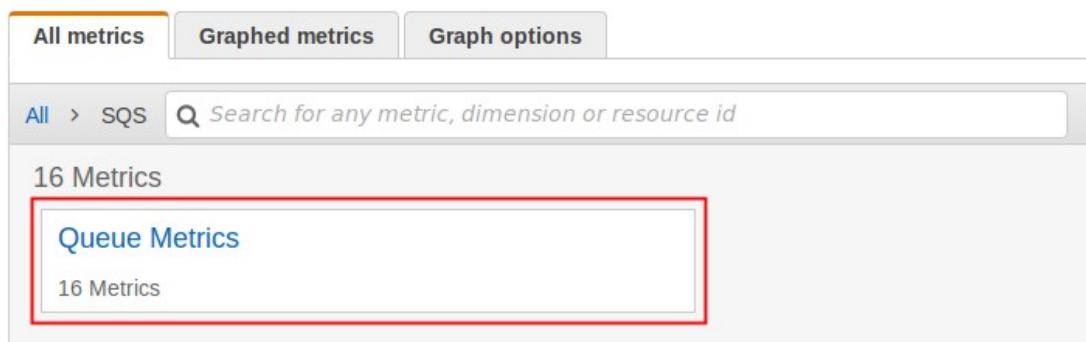
1. 登录到 Amazon SQS 控制台。
 2. 在队列列表中，选择 (选中) 与您要访问其指标的队列相对应的框。您最多可以显示 10 个队列的指标。
 3. 选择 Monitoring 选项卡。
- 将会在 SQS metrics 部分中显示多个图形。
4. 要了解特定图形表示的内容，请将光标悬停在所需图形旁的 ⓘ 上，或参阅 适用于 Amazon SQS 的可用 CloudWatch 指标 (p. 135)。
 5. 要同时更改所有图形的时间范围，请针对 Time Range 选择所需时间范围 (例如，Last Hour)。
 6. 要查看单个图形的其他统计数据，请选择该图形。
 7. 在 CloudWatch 监控详细信息对话框上，选择 Statistic , (例如总计)。有关受支持的统计数据的列表，请参阅 适用于 Amazon SQS 的可用 CloudWatch 指标 (p. 135)。
 8. 要更改单个图形显示的时间范围和时间间隔 (例如，显示最近 24 小时的时间范围而不是前 5 分钟，或者显示每小时时间段而不是每 5 分钟)，并且仍然显示图形的对话框，则对于 Time Range，选择所需时间范围 (例如，Last 24 Hours)。对于 Period，在指定的时间范围内选择所需时间段 (例如，1 Hour)。查看完图形后，请选择 Close。
 9. (可选) 要使用其他 CloudWatch 功能，请在监控选项卡上，选择查看全部 CloudWatch 指标，然后按照中的说明操作。Amazon CloudWatch 控制台 (p. 133) 过程。

Amazon CloudWatch 控制台

1. 登录到 CloudWatch 控制台。
2. 在导航面板上，选择 Metrics。
3. 选择 SQS 指标命名空间。

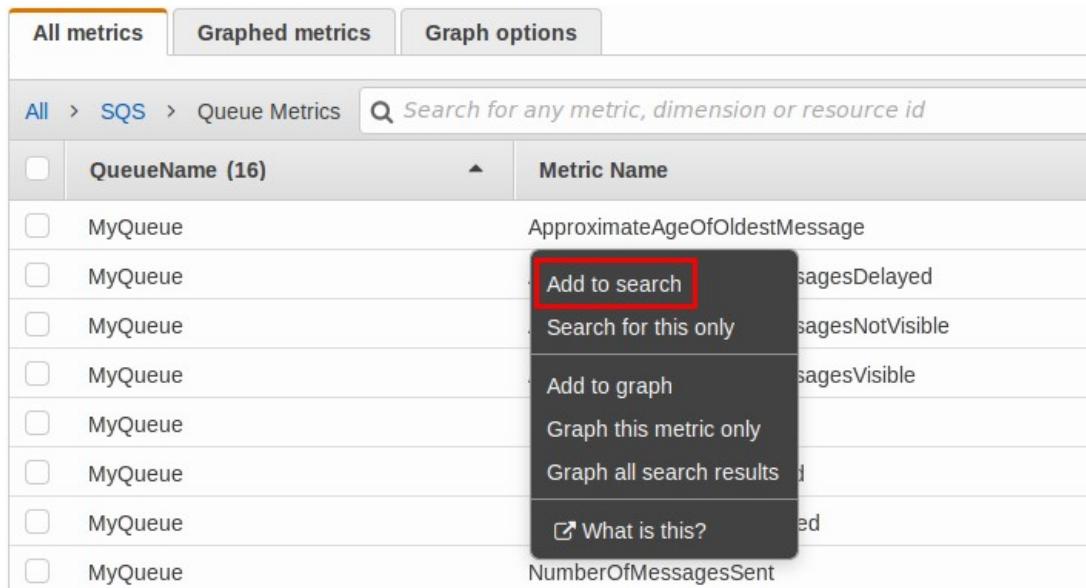


4. 选择 Queue Metrics 指标维度。



5. 您现在可检查您的 Amazon SQS 指标：

- 要对指标进行排序，请使用列标题。
- 要为指标绘制图表，请选中该指标旁的复选框。
- 要按指标进行筛选，请选择指标名称，然后选择 Add to search。



有关更多信息和其他选项，请参阅。绘制指标图形和使用 Amazon CloudWatch 仪表板中的Amazon CloudWatch 用户指南。

AWS Command Line Interface

要使用访问 Amazon SQS 指标AWS CLI，运行`get-metric-statistics`命令。

有关更多信息，请参阅。获取指标的统计数据中的Amazon CloudWatch 用户指南。

CloudWatch API

要使用 CloudWatch API 访问 Amazon SQS 指标，请使用`GetMetricStatistics`action.

有关更多信息，请参阅。获取指标的统计数据中的Amazon CloudWatch 用户指南。

为 Amazon SQS 指标创建 CloudWatch 警报

CloudWatch 允许您基于指标阈值触发警报。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。创建 Amazon CloudWatch 警报中的Amazon CloudWatch 用户指南。

1. 登录到AWS Management Console并在以下网址打开 CloudWatch 控制台：<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`，请选择要为其设置警报的队列名称和指标名称，然后选择下一步。有关可用指标的列表，请参阅 适用于 Amazon SQS 的可用 CloudWatch 指标 (p. 135)。

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”。

如果您希望 CloudWatch 在触发警报时发送通知，请选择一个现有的 Amazon SNS 主题，或者选择新列表并输入电子邮件地址（用逗号分隔）。

Note

如果您创建一个新 Amazon SNS 主题，那么电子邮件地址在接收任何通知之前必须通过验证。如果在验证电子邮件地址之前警报状态发生了变化，则不会发送通知。

6. 选择 Create Alarm (创建警报)。

将创建警报。

适用于 Amazon SQS 的可用 CloudWatch 指标

Amazon SQS 将以下指标发送到 CloudWatch。

Note

对于标准队列，由于 Amazon SQS 的分布式体系结构，结果是近似的。在大多数情况下，计数应接近队列中的实际消息数。

对于 FIFO 队列，结果是精确的。

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> <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 (p. 40)。当消息移至死信队列，死信队列的 ApproximateAgeOfOldestMessage 指标表示该消息移至死信队列的时间（而不是该消息发送的原始时间）。 <p>报告标准：报告一个非负值如果队列处于活动状态 (p. 132)。</p> <p>单位：秒</p> <p>有效统计数据：Amazon SQS 控制台中的 A、Sum、Data Sum、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data S</p>
ApproximateNumberOfMessagesDelayed	<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：报告一个非负值如果队列处于活动状态 (p. 132)。</p> <p>单位：计数</p> <p>有效统计数据：Amazon SQS 控制台中的 A、Sum、Data Sum、Data Sam、Data</p>

指标	描述
	Sam、Data Sam、Data Sum、Data Sam、Data S
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：报告一个非负值如果队列处于活动状态 (p. 132)。</p> <p>单位：计数</p> <p>有效统计数据：Amazon SQS 控制台中的 A、Sum、Data Sum、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data S</p>
ApproximateNumberOfMessagesVisible	<p>可从队列取回的消息数量。</p> <p>报告标准：报告一个非负值如果队列处于活动状态 (p. 132)。</p> <p>单位：计数</p> <p>有效统计数据：Amazon SQS 控制台中的 A、Sum、Data Sum、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data S</p>
NumberOfEmptyReceives ¹	<p>未返回消息的 ReceiveMessage API 调用数量。</p> <p>报告标准：报告一个非负值如果队列处于活动状态 (p. 132)。</p> <p>单位：计数</p> <p>有效统计数据：Amazon SQS 控制台中的 A、Sum、Data Sum、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data S</p>

指标	描述
NumberOfMessagesDeleted ¹	<p>从队列删除的消息数量。</p> <p>报告标准：报告一个非负值如果队列处于活动状态 (p. 132)。</p> <p>单位：计数</p> <p>有效统计数据：Amazon SQS 控制台中的 A、Sum、Data Sum、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data S</p> <p>Amazon SQS 发出NumberOfMessagesDeleted度量的每个成功删除操作使用有效接收句柄，包括重复删除。以下情形可能会使 NumberOfMessagesDeleted 指标值高于预期：</p> <ul style="list-style-type: none"> 调用DeleteMessage对属于同一消息的不同收据句柄执行操作：如果消息未在可见性超时过期，则该消息将对其他可对其执行处理和再次删除操作的用户可用，从而使NumberOfMessagesDeleted指标。 调用DeleteMessage在同一收款句柄上的操作：如果消息已处理并删除，但您调用DeleteMessage操作时，将返回一个成功状态，从而使NumberOfMessagesDeleted指标。
NumberOfMessagesReceived ¹	<p>调用 ReceiveMessage 操作返回的消息数量。</p> <p>报告标准：报告一个非负值如果队列处于活动状态 (p. 132)。</p> <p>单位：计数</p> <p>有效统计数据：Amazon SQS 控制台中的 A、Sum、Data Sum、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data S</p>
NumberOfMessagesSent ¹	<p>添加到队列的消息数量。</p> <p>报告标准：报告一个非负值如果队列处于活动状态 (p. 132)。</p> <p>单位：计数</p> <p>有效统计数据：Amazon SQS 控制台中的 A、Sum、Data Sum、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data S</p>

指标	描述
SentMessageSize ¹	<p>添加到队列的消息大小。</p> <p>报告标准：报告一个非负值如果队列处于活动状态 (p. 132)。</p> <p>单位：字节</p> <p>有效统计数据：Amazon SQS 控制台中的 A、Sum、Data Sum、Data Sam、Data Sam、Data Sam、Data Sam、Data Sam、Data S</p> <p>Note</p> <p>SentMessageSize直到至少一条消息发送至相应的队列之前，CloudWatch 控制台中不会显示为可用指标。</p>

¹ 这些指标是从服务角度计算的，可以包括重试次数。不要依赖这些指标的绝对值，也不要使用它们来估计当前队列状态。

Amazon SQS 指标的维度

Amazon SQS 发送至 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 SQS 的合规性验证

作为多个项目的一部分，第三方审计员将评估 Amazon SQS 的安全性和合规性AWS合规性计划，包括以下内容：

- 支付卡行业数据安全标准 (PCI DSS)
- 健康保险流通与责任法案 (HIPAA)

有关特定合规性计划范围内的 AWS 服务列表，请参阅[合规性计划范围内的 AWS 服务](#)。有关常规信息，请参阅 [AWS 合规性计划](#)。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅[下载 AWS Artifact 中的报告](#)。

您在使用 Amazon SQS 时的合规性责任由您的数据的敏感性、您公司的合规性目标以及适用的法律法规决定。AWS提供以下资源来帮助实现合规性：

- [安全性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，并提供了在上部署基于安全性和合规性的基准环境的步骤。AWS。
- [HIPAA 安全性与合规性架构设计白皮书](#)— 本白皮书介绍了公司如何使用AWS来创建符合 HIPAA 要求的应用程序。
- [AWS合规性资源](#)— 此业务手册和指南集合可能适用于您的行业和位置。
- [使用规则评估资源中的AWS Config开发人员指南](#)— AWS Config服务评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- [AWS Security Hub](#)— 此AWS服务向您提供中安全状态的全面视图AWS，可帮助您检查是否符合安全行业标准和最佳实践。

Amazon SQS 中的恢复能力

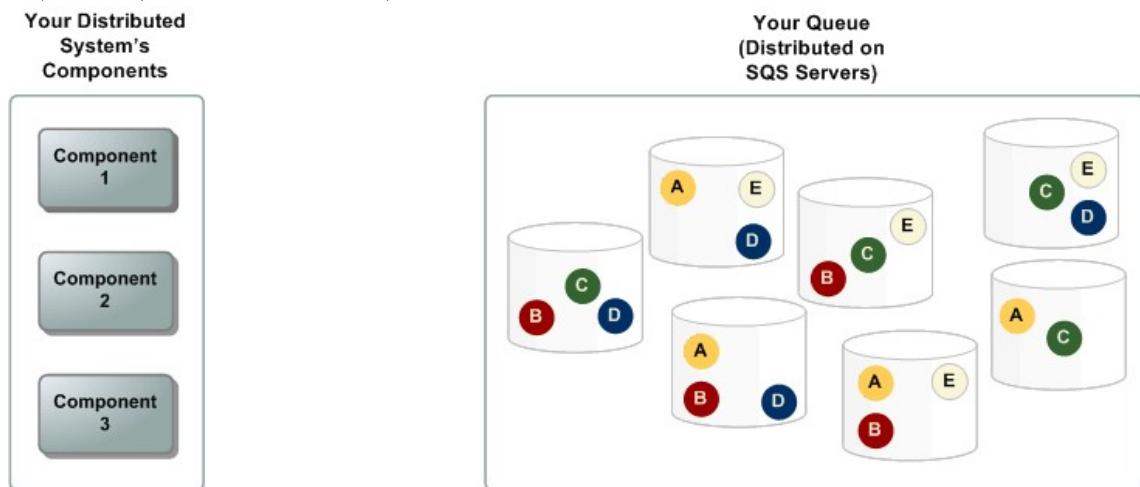
这些区域有：AWS全球基础设施围绕AWS区域和可用区。AWS区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。有关 AWS 区域和可用区的更多信息，请参阅 [AWS 全球基础设施](#)。

除了AWSAmazon SQS 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。

在下面的情况下，您的系统具有多个创建者（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

作为托管服务，Amazon SQS 受 AWS 全局网络安全过程 [Amazon Web Services：安全过程概述](#) 白皮书。

您可以使用 AWS 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) 1.0 或更高版本。建议使用 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 委托人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。此外，您可以使用 Amazon SQS 策略来控制来自特定 Amazon VPC 终端节点或特定 VPC 的访问。这将有效隔离在一个特定 VPC 到给定 Amazon SQS 队列的网络访问。AWS 网络。有关更多信息，请参阅 [示例 5：如果不是来自 VPC 终端节点，则拒绝访问 \(p. 124\)](#)。

Amazon SQS 安全最佳实践

AWS 提供了针对 Amazon SQS 的多项安全功能，您应在自己的安全策略的上下文中查看这些功能。

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。

预防性最佳实践

以下是针对 Amazon SQS 的预防性安全最佳实践。

主题

- 确保队列不可公开访问 (p. 141)
- 实施最低权限访问 (p. 141)
- 对应用程序使用 IAM 角色和 AWS 需要访问 Amazon SQS 的服务 (p. 141)
- 实施服务器端加密 (p. 142)
- 实施传输中数据加密 (p. 142)
- 考虑使用 VPC 终端节点来访问 Amazon SQS (p. 142)

确保队列不可公开访问

除非您明确要求 Internet 上的任何人都可以读取或写入 Amazon SQS 队列，否则应确保队列不可公开访问（可供世界上的每个人或任何经过身份验证的经过身份验证的任何人访问）AWS 用户）。

- 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，并且需要三类用户账户访问：

- Administrators—用于创建、修改和删除队列的访问权。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。
- 使用者—用于接收和删除来自队列的消息的访问权。

有关详细信息，请参阅以下章节：

- [Amazon SQS 中的 Identity of Access Management \(p. 102\)](#)
- [Amazon SQS API 权限：操作和资源参考 \(p. 126\)](#)
- [将自定义策略与 Amazon SQS 访问策略语言一起使用 \(p. 116\)](#)

对应用程序使用 IAM 角色和 AWS 需要访问 Amazon SQS 的服务

用于应用程序或 AWS 服务（如 Amazon EC2）才能访问 Amazon SQS 队列，则必须使用有效的 AWS 凭证中的凭证 AWS API 请求。由于这些凭证不会自动轮换，因此您不应将 AWS 凭证直接存储在应用程序或 EC2 实例中。

您应使用 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。在使用角色时，您不需要将长期凭证（如用户名、密码和访问密钥）分配给 EC2 实例或 AWS 服务（例如 AWS Lambda）。相反，角色可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 角色和针对角色的常见情形：用户、应用程序和服务](#) 中的 IAM 用户指南。

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 AWS Key Management Service 中托管的密钥。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 [静态加密 \(p. 96\)](#) 和 [密钥管理 \(p. 98\)](#)。

实施传输中数据加密

如果没有 HTTPS (TLS)，基于网络的攻击者便能使用中间人之类的攻击来窃听或操纵网络流量。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 终端节点来访问 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 终端节点来仅为对特定 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 终端节点或特定 VPC 对队列的访问。

Amazon SQS VPC 终端节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 终端节点访问的请求、用户或组。
- 您可以使用队列策略控制哪些 VPC 或 VPC 终端节点有权访问您的队列。

有关更多信息，请参阅 [Amazon SQS 的 Amazon Virtual Private Cloud 终端节点 \(p. 101\)](#) 和 [为 Amazon SQS 创建 Amazon VPC 终端节点策略 \(p. 102\)](#)。

使用 Amazon SQS API

本节提供了有关构建 Amazon SQS 终端节点以及通过 GET 和 POST 方法以及使用批处理 API 操作。有关 Amazon SQS 的详细信息操作—包括参数、错误、示例和数据类型—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 [AWS 开发工具包](#)，其中包含以下自动功能：

- 使用密码对服务请求签名
- 重试请求
- 处理错误响应

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](#)。

主题

- [提出查询 API 请求 \(p. 143\)](#)
- [Amazon SQS 批处理操作 \(p. 148\)](#)

提出查询 API 请求

在本节中，您将了解如何构建 Amazon SQS 终端节点，如何将 GET 和 POST 请求和解释答复。

主题

- [构建终端节点 \(p. 143\)](#)
- [提出 GET 请求 \(p. 144\)](#)
- [提出 POST 请求 \(p. 144\)](#)
- [对请求进行身份验证 \(p. 145\)](#)
- [解释响应 \(p. 147\)](#)

构建终端节点

为了使用 Amazon SQS 队列，您必须构建一个终端节点。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：

- [区域终端节点](#)
- [Amazon Simple Queue Service 终端节点和配额](#)

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

以下是一个提出创建队列的请求的终端节点的示例。

```
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。

该结构 **AUTHPARAMS** 取决于 API 请求的签名。有关更多信息，请参阅 [签名 AWS API 请求中 Amazon Web Services 一般参考](#)。

提出 GET 请求

Amazon SQSGET请求被构建为由以下部分组成的 URL：

- 终端节点— 请求对其执行操作的资源 ([队列名称和 URL \(p. 33\)](#))，例如：<https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue>
- 操作—[操作](#)，该终端节点要执行。终端节点与操作之前用问号 (?) 分隔，例如：?
`Action=SendMessage&MessageBody=Your%20message%20Text`
- 参数— 任何请求参数 — 每个参数之间用“和”号隔开 (&)，例如：
`&Version=2012-11-05&AUTHPARAMS`

以下是一个示例。GET请求，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 SQSPOST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 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](#) POST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。

对请求进行身份验证

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[第 1 步：创建 AWS 账户 \(p. 4\)](#)和[第 2 步：创建 IAM 用户 \(p. 4\)](#)）。接下来，AWS 将按照以下步骤操作：

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. 将出现以下三种情况之一：
 - 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 \(p. 145\)](#)
- [第 1 部分：来自用户的请求 \(p. 146\)](#)
- [第 2 部分：来自响应的 AWS \(p. 147\)](#)

使用 HMAC-SHA 的基本身份验证过程

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

- 这些区域有：AWS 访问密钥 ID，可确定您的 AWS 账户，其中 AWS 用于查找您的秘密访问密钥。
- 这些区域有：HMAC-SHA 请求签名，该密钥是使用您的秘密访问密钥计算得出的。AWS—有关更多信息，请参阅[RFC2104](#)。这些区域有：[AWS 开发工具包](#) 处理签名过程；但是，如果您通过 HTTP 或 HTTPS 提交查询请求，则必须在每个查询请求中包含一个签名。

1. 派生签名版本 4 签名密钥。有关更多信息，请参阅[使用 Java 派生签名密钥](#)。

Note

Amazon SQS 支持签名版本 4，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，应使用签名版本 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。如果指定时间戳 (而不是过期时间)，则请求会在指定的时间过后 15 分钟自动过期 (如果请求的时间戳比 AWS 服务器上的当前时间早 15 分钟以上，则 AWS 不会处理请求)。

如果您使用的是 .NET，则不得发送过于具体的时间戳 (因为 AWS 和 .NET 对如何确定额外时间精度有不同的解释)。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

以下是在使用 HMAC-SHA 请求签名对 AWS 请求进行身份验证时必须执行的过程。

Create a request:

1

Request

AccessKeyId = ...
Action = ...
Timestamp = ...
ParameterA = ...

Create an HMAC-SHA signature:

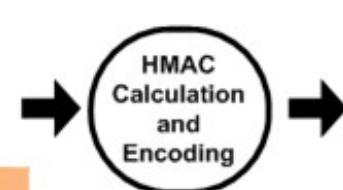
2

String based on request contents



Your Secret Access Key

wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY



Your Signature

Send the request and signature to AWS:

3

Request

AccessKeyId = ...
Action = ...
Timestamp = ...
ParameterA = ...

Your Signature

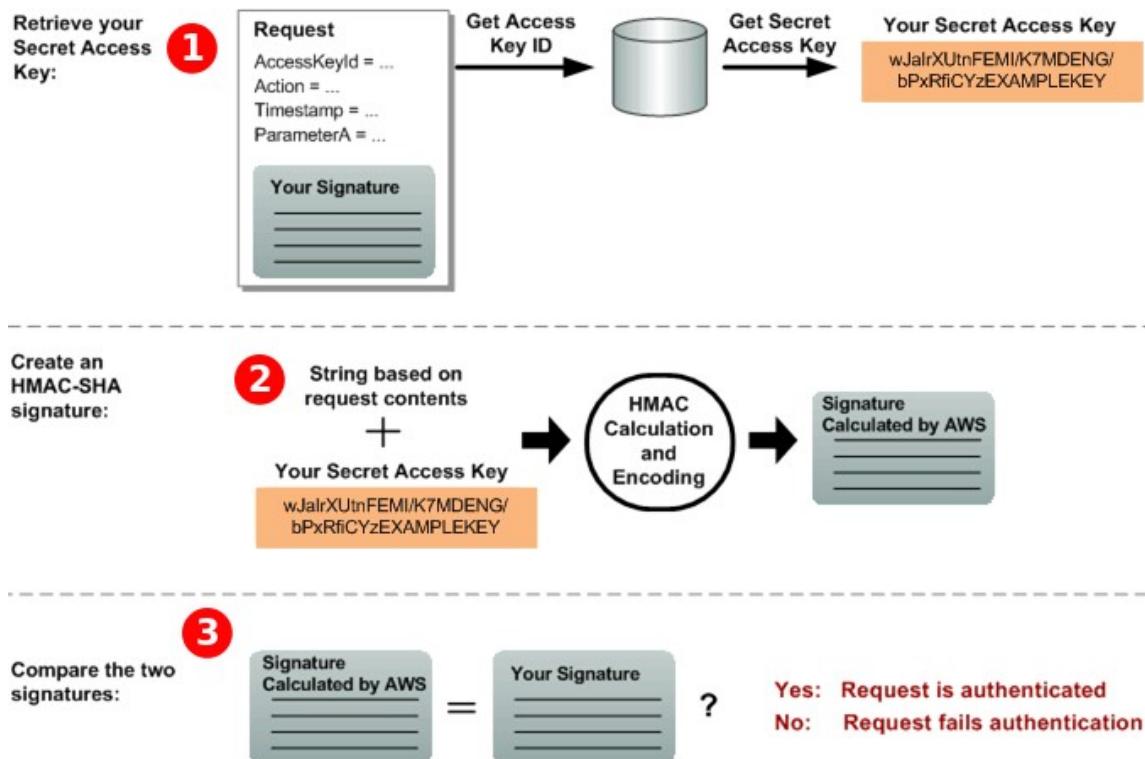


AWS

1. 构建要发送到 AWS 的请求。
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，然后将请求发送给 AWS。

第 2 部分：来自的响应AWS

AWS 在响应时开始以下过程。



1. AWS 使用访问密钥 ID 来查找秘密访问密钥。
2. 通过与用于计算您在请求中发送的签名相同的算法，AWS 可根据请求数据和秘密访问密钥生成签名。
3. 将出现以下三种情况之一：
 - 如果 AWS 生成的签名与您在请求中发送的签名相匹配，AWS 将认为请求是真实的。
 - 如果比较失败，则 AWS 将放弃请求并返回错误。

解释响应

为响应操作请求，Amazon SQS 会返回包含请求结果的 XML 数据结构。有关更多信息，请参阅中的相关部分。[Amazon Simple Queue Service API 参考](#)。

主题

- [成功的响应结构 \(p. 147\)](#)
- [错误响应结构 \(p. 148\)](#)

成功的响应结构

如果请求成功，则主要响应元素将以请求的操作命名并附加上 Response (*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，后者又包含所创建的队列的 URL。
- **ResponseMetadata**— 包含RequestId，后者又包含请求的 UUID。

以下是 XML 格式的成功响应的示例：

```
<CreateQueueResponse
    xmlns="https://sns.us-east-2.amazonaws.com/doc/2012-11-05/
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="CreateQueueResponse">
    <CreateQueueResult>
        <QueueUrl>https://sns.us-east-2.amazonaws.com/770098461991/queue2</QueueUrl>
    </CreateQueueResult>
    <ResponseMetadata>
        <RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
    </ResponseMetadata>
</CreateQueueResponse>
```

错误响应结构

如果请求不成功，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条件。
- **Detail**— (可选) 指定有关错误的更多详细信息。

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>
```

Amazon SQS 批处理操作

要减少成本或通过单个操作来处理多达 10 条消息，可以使用以下操作：

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

可以使用查询 API 或使用 AWS 支持 Amazon SQS 批处理操作的开发工具包。

Note

您在单个 `SendMessageBatch` 调用中发送的所有消息的总大小不能超过 262144 字节 (256 KB)。您无法显式设置针对 `SendMessageBatch`、`DeleteMessageBatch` 或 `ChangeMessageVisibilityBatch` 的权限。在设置针对 `sendMessage`、`DeleteMessage` 或 `ChangeMessageVisibility` 的权限时，将会设置针对这些操作所对应的批处理版本的权限。Amazon SQS 控制台不支持批处理操作。

主题

- [启用客户端缓冲和请求批处理 \(p. 149\)](#)
- [利用水平扩展和操作批处理来提高吞吐量 \(p. 152\)](#)

启用客户端缓冲和请求批处理

这些区域有：[AWS SDK for Java](#) includes `AmazonSQSBufferedAsyncClient`，它访问 Amazon SQS。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 — 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 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 \(p. 38\)](#) 的支持还有助于提高吞吐量。有关更多信息，请参阅[利用水平扩展和操作批处理来提高吞吐量 \(p. 152\)](#)。

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 缓冲异步客户端当前不支持 FIFO 队列。

主题

- [使用 `AmazonSQSBufferedAsyncClient` \(p. 149\)](#)
- [配置 `AmazonSQSBufferedAsyncClient` \(p. 150\)](#)

使用 `AmazonSQSBufferedAsyncClient`

在开始之前，请完成[设置 Amazon SQS \(p. 4\)](#) 中的步骤。

Important

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

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`，则可以使用它来发送多个请求到 Amazon SQS（就像使用 `AmazonSQSAsyncClient`），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 SendMessageResult sendResult = bufferedSqs.sendMessageAsync(request);

final ReceiveMessageRequest receiveRq = new ReceiveMessageRequest()
    .withMaxNumberOfMessages(1)
    .withQueueUrl(queueUrl);
final ReceiveMessageResult rx = bufferedSqs.receiveMessage(receiveRq);
```

配置 AmazonSQSBufferedAsyncClient

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，AmazonSQSBufferedAsyncClient 会在使用消息时尝试使用长轮询。
longPollWaitTimeoutSeconds	20 s	在返回空接收结果前，ReceiveMessage 调用在服务器上阻塞以等待消息显示在队列中的最长时间 (以秒为单位)。 Note 如果禁用长轮询，则此设置不起作用。
maxBatchOpenMs	200ms	传出调用等待其他要一起对同类型的消息进行批处理的调用的最长时间 (以毫秒为单位)。 设置的时间越长，则执行等量工作所需的批处理次数就越少 (但是，批处理中的首次调用必须等待更长的时间)。

参数	默认值	描述
		如果将此参数设置为 0，则提交的请求不会等待其他请求，从而有效地禁用批处理。
maxBatchSize	每批 10 个请求	<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> <p>Note</p> <p>Amazon SQS 允许的最大值为每批 10 个请求。</p>
maxBatchSizeBytes	256 KB	<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> <p>Note</p> <p>Amazon SQS 允许的最大值为 256 KB。</p>
maxDoneReceiveBatches	10 个批处理	<p><code>AmazonSQSBufferedAsyncClient</code> 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> <p>Note</p> <p>0 表示所有消息预取操作将被禁用，消息只能按需使用。</p>
maxInflightOutboundBatches	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，发出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 <code>AmazonSQSBufferedAsyncClient</code> 使用的线程就越多。</p>

参数	默认值	描述
maxInflightReceiveBatches	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> <p>Note</p> <p>0 表示所有消息预取操作将被禁用，消息只能按需使用。</p>
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> <p>Note</p> <p>-1 表示为队列选择默认设置。 不能将可见性超时设置为 0。</p>

利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[与消息相关的配额 \(p. 91\)](#)。

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

主题

- [横向扩展 \(p. 152\)](#)
- [操作批处理 \(p. 153\)](#)
- [单一操作和批处理请求的有效 Java 示例 \(p. 153\)](#)

横向扩展

由于您通过 HTTP 请求响应协议访问 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，请确保 Amazon SQS 客户端具有足够的连接或线程，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，AWS SDK for Java [AmazonSQSClient](#) 类最多会维持 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。

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 批处理操作是 [SendMessageBatch](#)、[DeleteMessageBatch](#)，和 [ChangeMessageVisibilityBatch](#)。若要在不更改创建者或使用者的情况下利用批处理，可以使用 [Amazon SQS 缓冲异步客户端 \(p. 149\)](#)。

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

Prerequisites

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 2010-2021 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 (Exception e) {  
            log.error("Producer: error processing message: " + e.getMessage());  
        }  
    }  
}
```

```
        }
    }
} 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 会针对已发送、接收和删除的消息自动生成数量指标。可以通过监控选项卡或[CloudWatch 控制台](#)。

Note

队列启动后，这些指标最多可能需要花费 15 分钟才可用。

相关 Amazon SQS 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。
Amazon SQAWS CLI命令参考	可用于使用队列的 AWS CLI 命令的说明。
区域和终端节点	Amazon SQS 区域和终端节点的相关信息
产品页面	提供 Amazon SQS 相关信息的主要网页。
开发论坛	由开发人员组成的社区形式的论坛，在这里开发人员可以讨论与 Amazon SQS 有
AWS Premium Support 信息	提供相关信息的主要网页AWSPremium Support 是一种一对一、快速响应的支持渠道，可帮助您在AWS基础设施服务.

文档历史记录

下表介绍了对Amazon Simple Queue Service 开发者指南自 2019 年 1 月起。如需有关此文档更新的通知，请订阅[RSS 源](#)。

update-history-change	更新-历史记录-描述	更新-历史记录-日期
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多的每秒事务 (TPS)。有关吞吐量配额的信息，请参阅 与消息相关的配额 。	2019 年 5 月 27 日
预览版本中提供了 FIFO 队列中的消息的高吞吐量	Amazon SQS FIFO 队列的高吞吐量目前为预览版，可能会发生变化。此功能为 FIFO 队列中的消息提供更多的每秒事务 (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
全新的 Amazon SQS 控制台设计	为了简化开发和生产工作流程，Amazon SQS 控制台具有 新用户体验 。	2020 年 7 月 8 日
Amazon SQS 支持对 ListQueues 和 ListDeadLetterSourceQueues 分页	您可以指定从 <code>ListQueues</code> 或 <code>ListDeadLetterSourceQueues</code> 请求返回的最大结果数。	2020 年 6 月 22 日
Amazon SQS 在所有提供的 Amazon CloudWatch 指标中支持 1 分钟AWS区域，除 AWSGovCloud (US) 区域	Amazon SQS 的一分钟 CloudWatch 指标适用于所有区域，除了 AWS GovCloud (US) 区域。	2020 年 1 月 9 日
Amazon SQS 支持 1 分钟 CloudWatch 指标	Amazon SQS 的一分钟 CloudWatch 指标目前仅在以下区域提供：美国东部（俄亥俄）、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 可用于 Amazon SQS FIFO 队列的触发器	您可以将到达中的消息配置为 Lambda 函数触发器。	2019 年 11 月 25 日
Amazon SQS 的服务器端加密 (SSE) 已在中国区域提供。	Amazon SQS SSE 在中国区域可用。	2019 年 11 月 13 日
FIFO 队列在中东（巴林）区域提供	FIFO 队列在中东（巴林）区域提供。	2019 年 10 月 10 日
Amazon SQS 的 Amazon Virtual Private Cloud (Amazon VPC) 终端节点已在推出。AWSGovCloud (美国东部) 和AWSGovCloud (美国西部) 区域	您可以在推出的 Amazon VPC 将消息发送到 Amazon SQS 队列。AWSGovCloud (美国东部) 和AWSGovCloud (美国西部) 区域	2019 年 9 月 5 日

Amazon SQS 允许使用 AWS X-Ray 使用消息系统属性	您可以使用 X-Ray 对通过 Amazon SQS 队列传递的消息进行问题排查。此版本添加了 <code>MessageSystemAttribute</code> 请求参数（允许您通过 Amazon SQS 将 X-Ray 跟踪标头发送到 <code>SendMessage</code> 和 <code>SendMessageBatch</code> API 操作， <code>AWSTraceHeader</code> 属性添加到 <code>ReceiveMessage</code> API 操作，以及 <code>MessageSystemAttributeValue</code> 数据类型。	2019 年 8 月 28 日
您可以在创建时标记 Amazon SQS 队列	您可以使用单个 Amazon SQS API 调用 AWS 软件开发工具包函数，或 AWS Command Line Interface(AWS CLI) 命令同时创建队列并指定其标签。此外，Amazon SQS 还支持 <code>aws:TagKeys</code> 和 <code>aws:RequestTag</code> AWS Identity and Access Management(IAM) 密钥。	2019 年 8 月 22 日
Amazon SQS 的临时队列客户端现已推出	临时队列可帮助您节省开发时间和部署成本，使用常见消息模式（例如请求-响应）。您可以将 临时队列客户端 创建高吞吐量、经济高效、由应用程序管理的临时队列。	2019 年 7 月 25 日
适用于 Amazon SQS 的 SSE 可在 AWS GovCloud (美国东部) 区域	Amazon SQS 的服务端加密 (SSE) 已在推出。AWS GovCloud (美国东部) 区域	2019 年 6 月 20 日
FIFO 队列在亚太地区 (香港)、中国 (北京)、AWS GovCloud (美国东部) 和 AWS GovCloud (美国西部) 区域	FIFO 队列在亚太地区 (香港)、中国 (北京)、AWS GovCloud (美国东部) 和 AWS GovCloud (美国西部) 区域	2019 年 5 月 15 日
Amazon VPC 终端节点策略可用于 Amazon SQS	您可以为 Amazon SQS 创建 Amazon VPC 终端节点策略。	2019 年 4 月 4 日
在欧洲 (斯德哥尔摩) 和中国 (宁夏) 区域提供 FIFO 队列	FIFO 队列在欧洲 (斯德哥尔摩) 和中国 (宁夏) 区域提供。	2019 年 3 月 14 日
FIFO 队列在所有提供 Amazon SQS 的区域中可用	FIFO 队列在美国东部 (俄亥俄)、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 (新加坡)、亚太地区 (悉尼)、加拿大 (中部)、欧洲 (法兰克福)、欧洲 (爱尔兰)、欧洲 (欧洲 (欧洲)) 巴黎和南美洲 (圣保罗) 区域。	2019 年 2 月 7 日

AWS术语表

最新的AWS术语，请参阅[AWS术语表](#)中的AWS一般参考。

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。