
Amazon Simple Queue Service

开发人员指南

API Version 2012-11-05



Amazon Simple Queue Service: 开发人员指南

Copyright © 2016 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Abstract

应用程序中的一个组件生成的、将由使用 Amazon SQS 的另一个组件使用的队列消息。

Table of Contents

什么是 Amazon Simple Queue Service ?	1
架构概述	1
Amazon SQS 功能	2
Amazon SQS 队列的工作原理	3
分布式队列的属性	4
消息顺序	4
至少一次传递	4
消息示例	4
队列和消息标识符	5
队列 URL	5
消息 ID	6
接收句柄	6
处理消息所需的资源	6
可见性超时	7
可见性超时的一般建议	7
延长消息的可见性超时	8
终止消息的可见性超时	8
与可见性超时相关的 API 操作	8
消息生命周期	8
Amazon SQS 与 JMS	10
Amazon SQS 和 JMS 先决条件	10
Amazon SQS Java Messaging Library 入门	11
创建 JMS 连接	11
创建 Amazon SQS 队列	12
同步发送消息	12
同步接收消息	12
异步接收消息	13
客户端确认模式	14
无序确认模式	15
代码示例	15
ExampleConfiguration.java	15
TextMessageSender.java	17
AsyncMessageReceiver.java	19
SyncMessageReceiver.java	20
SyncMessageReceiverClientAcknowledge.java	21
SyncMessageReceiverUnorderedAcknowledge.java	24
SpringExampleConfig.xml	26
SpringExample.java	27
ExampleCommon.java	28
参考/附录	29
支持的 JMS 1.1 常用接口 :	30
支持的消息类型	30
支持的消息确认模式	30
JMS 定义标头和预留属性	30
Amazon S3 中的 Amazon SQS 消息	31
先决条件	31
使用适用于 Java 的 Amazon SQS 扩展客户端库	32
消息属性	36
消息属性项目和验证	36
数据类型	37
通过 AWS 管理控制台使用消息属性	37
利用 AWS 软件开发工具包来使用消息属性	41
通过 Amazon SQS 查询 API 使用消息属性	43
MD5 消息摘要计算	44
长轮询	47

使用 AWS 管理控制台启用长轮询	48
使用查询 API 启用长轮询	50
延迟队列	52
使用 AWS 管理控制台创建延迟队列	53
使用查询 API 创建延迟队列	55
消息定时器	57
使用控制台创建消息定时器	57
使用查询 API 创建消息定时器	61
批处理 API 操作	63
SendMessageBatch 的最大消息大小	63
客户端缓冲和请求批处理	64
AmazonSQSBufferedAsyncClient 入门	64
高级配置	65
通过水平扩展和批处理提高吞吐量	66
水平扩展	66
批处理	66
示例	67
运行示例	68
提出 API 请求	71
终端节点	72
提出查询请求	73
GET 请求的结构	73
POST 请求的结构	74
相关主题	75
提出 SOAP 请求	75
请求身份验证	76
什么是身份验证？	76
您的 AWS 账户	78
您的访问密钥	78
HMAC-SHA 签名	78
查询请求身份验证	83
响应	84
成功响应的结构	84
错误响应的结构	84
相关主题	85
共享队列	86
共享队列的简单 API	86
共享队列的高级 API	86
了解资源级权限	86
授予对队列的匿名访问权限	87
编程语言	89
死信队列	90
使用 AWS 管理控制台设置死信队列	90
通过 Amazon SQS API 使用死信队列	92
问题：使用 Amazon SQS 控制台查看消息可能会导致消息移至死信队列	93
使用 Access Policy Language	95
概述	96
何时使用访问控制	96
主要概念	96
架构概述	99
使用 Access Policy Language	100
评估逻辑	102
关于访问控制的基本使用案例	106
Amazon SQS 策略示例	109
Amazon SQS 策略的特别信息	113
访问控制	114
Amazon SQS 策略的 IAM 相关功能	115
IAM 和 Amazon SQS 策略一起使用	116

Amazon SQS ARN	118
Amazon SQS 操作	119
Amazon SQS 密钥	119
Amazon SQS 的 IAM 策略示例	120
使用临时安全证书	121
为队列订阅 Amazon SNS 主题	123
使用 AWS 管理控制台为队列订阅 Amazon SNS 主题	123
使用 CloudWatch 监控 Amazon SQS	125
访问 Amazon SQS 的 CloudWatch 指标	125
为 Amazon SQS 指标设置 CloudWatch 报警器	126
Amazon SQS 指标	126
使用 CloudTrail 记录 Amazon SQS API 调用	128
CloudTrail 中的 Amazon SQS 信息	128
了解 Amazon SQS 日志文件条目	129
AddPermission	129
CreateQueue	130
DeleteQueue	130
RemovePermission	131
SetQueueAttributes	131
Resources	133
文档历史记录	134

什么是 Amazon Simple Queue Service ?

Abstract

获取可靠且可扩展的托管队列，使用 Amazon SQS 存储计算机之间传输的消息。

Amazon Simple Queue Service (Amazon SQS) 提供了可靠且可扩展的托管队列，用于存储计算机之间传输的消息。通过使用 Amazon SQS，您可以在执行不同任务的应用程序的分布式组件之间移动数据，既不会丢失消息，也不要求各个组件始终处于可用状态。

Amazon SQS 是分布式队列系统，可以让 Web 服务应用程序快速可靠地对应用程序中的一个组件生成给另一组件使用的消息进行排队。队列是等待处理的消息的临时储存库。

使用 Amazon SQS，您可以分离应用程序的组件以便其独立运行，Amazon SQS 同时还可以简化组件间的消息管理。分布式应用程序的任何组件均可将消息存储在拥有故障保护功能的队列中。消息可包含高达 256 KB 的任何格式文本。任何组件均可在之后利用 Amazon SQS API 以编程方式检索消息。对于大于 256 KB 的消息，可以通过使用 Amazon S3 来存储更大负载且适用于 Java 的 Amazon SQS 扩展客户端库进行管理。

队列在产生并保存数据的组件以及接收数据进行处理组件之间起到缓冲作用。这意味着，如果产生消息的组件工作速度快于接收处理消息的组件，或如果产生消息的组件或接受处理消息的组件仅间歇性地连接到网络，队列可解决因此而产生的问题。

Amazon SQS 确保每条消息至少传送一次，并且支持与同一队列交互的多个读取器和写入器。单个队列可由多个分布式应用程序组件同时使用，无需这些组件互相协作以共享队列。

Amazon SQS 设计为始终可用并传送消息。因此而放弃的一项功能是不保证消息的先进先出传送。对许多分布式应用程序而言，每条消息均可独立存在，并且只要所有的消息得以传送，次序并不重要。如果您的系统要求保留次序，您可以在每条消息中放置序列信息，以便队列返回消息时将其重新排序。

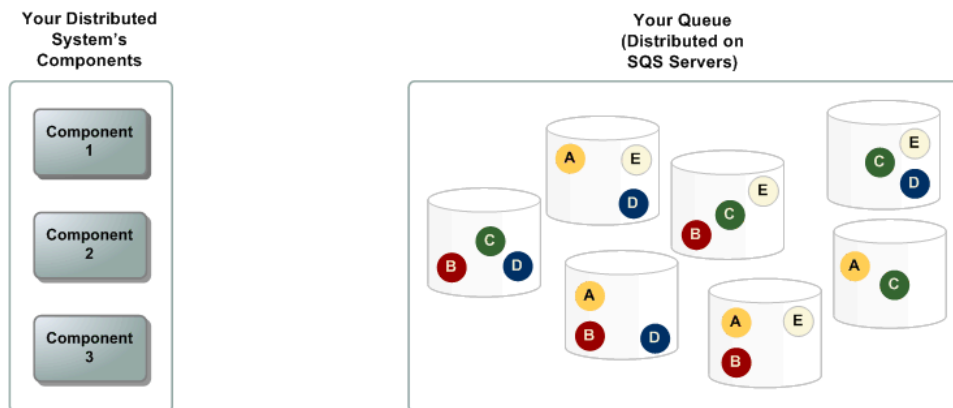
请务必阅读有关分布式队列的信息，这有助于您了解如何设计与 Amazon SQS 正确搭配工作的应用程序。有关更多信息，请参阅 [分布式队列的属性 \(p. 4\)](#)。

架构概述

整个系统中有三个主要的参与者：

- 分布式系统的组件
- 队列
- 队列中的消息

在下图中，您的系统有多个向队列发送消息以及从队列接收消息的组件。该图显示，具有消息（标记为 A-E）的单一队列冗余地保存在多台 Amazon SQS 服务器中。



Amazon SQS 功能

Amazon SQS 提供以下主要功能：

- 冗余基础设施 – 确保将您的消息至少传输一次、对消息的高度并发访问以及发送和检索消息的高度可用性
- 多个写入器和读取器 – 您的系统的多个部分可以同时发送或接收消息
Amazon SQS 在处理期间锁定消息，防止您的系统的其他部分同时处理消息。
- 每个队列的设置均可配置 – 并非您的所有队列都要完全相同
例如，一个队列可以针对比其他消息需要更长处理时间的消息进行优化。
- 可变消息大小 – 您的消息大小可高达 262 144 字节 (256 KB)
对于较大的消息，您可以使用 Amazon Simple Storage Service (Amazon S3) 或 Amazon DynamoDB 存储消息内容，并使用 Amazon SQS 保留指向 Amazon S3 对象的指针。有关更多信息，请参阅[通过 Amazon S3 管理 Amazon SQS 消息](#)。此外，您也可以将更大的消息拆分为较小的消息。
- 访问控制 – 您可以控制谁可以从队列发送和收取消息
- 延迟队列 – 延迟队列即用户对其设置默认延迟的队列，从而所有排队消息的传送会推迟那一段时间。使用 `CreateQueue` 创建队列时您可以设置延迟值，您也可以用 `SetQueueAttributes` 更新该值。如果您更新该值，新的值仅影响更新后排队的消息。
- PCI 合规性 – Amazon SQS 支持由商家或服务提供商处理、存储和传输信用卡数据，而且已经验证符合支付卡行业 (PCI) 数据安全标准 (DSS)。有关 PCI DSS 的更多信息，包括如何请求 AWS PCI Compliance Package 的副本，请参阅[PCI DSS 第 1 级](#)。

Amazon SQS 队列的工作原理

Abstract

介绍 Amazon SQS 队列、队列和消息标识符、队列的一般大小以及在队列中管理消息的基础知识。

主题

- [分布式队列的属性 \(p. 4\)](#)
- [队列和消息标识符 \(p. 5\)](#)
- [处理消息所需的资源 \(p. 6\)](#)
- [可见性超时 \(p. 7\)](#)
- [消息生命周期 \(p. 8\)](#)

本部分介绍了 Amazon SQS 队列的基本属性、队列和消息的标识符、确定队列一般大小的方法，以及管理队列中消息的方法。

如果您尚未向队列发送任何消息，或者如果您从队列删除了所有消息，则队列可能为空。

您必须为每个队列指定一个名称（有关更多信息，请参阅[“队列 URL \(p. 5\)”](#)）。您可以获取名称的前几个字符相同的所有队列或队列子集的列表（例如，您可以获取名称以“T3”开头的所有队列的列表）。

无论队列是否为空，您都可以随时删除队列。但请注意，队列会在设置的时间段内保留消息。默认情况下，队列会将消息保留四天。但是，您可以将队列配置为在发送消息后，最多将消息保留 14 天。

如果在连续 30 天内没有对队列执行以下操作之一，则 Amazon SQS 可能会在不通知的情况下删除该队列：SendMessage、ReceiveMessage、DeleteMessage、GetQueueAttributes、SetQueueAttributes、AddPermission 和 RemovePermission。



Important

如果您重复创建队列，然后让其处于非活跃状态，或者如果您在队列中存储过量数据，则会违背 Amazon SQS 的预期用途。

下表列出了要使用的 API 操作。

要执行以下任务……	请使用以下操作
创建队列	CreateQueue

要执行以下任务.....	请使用以下操作
获取现有队列的 URL	GetQueueUrl
列出队列	ListQueues
删除队列	DeleteQueue

分布式队列的属性

Abstract

介绍 Amazon SQS 分布式队列的属性以设计您的应用程序。

以下信息有助于您将您的应用程序设计为与 Amazon SQS 正确地搭配工作。

消息顺序

Amazon SQS 会尽量保持消息顺序，但是由于队列的分布式特性，我们无法保证您将以发送消息的先后顺序接收这些消息。如果您的系统要求保持顺序，则我们建议您在每条消息中放置排序信息，以便在收到消息时对消息重新排序。

至少一次传递

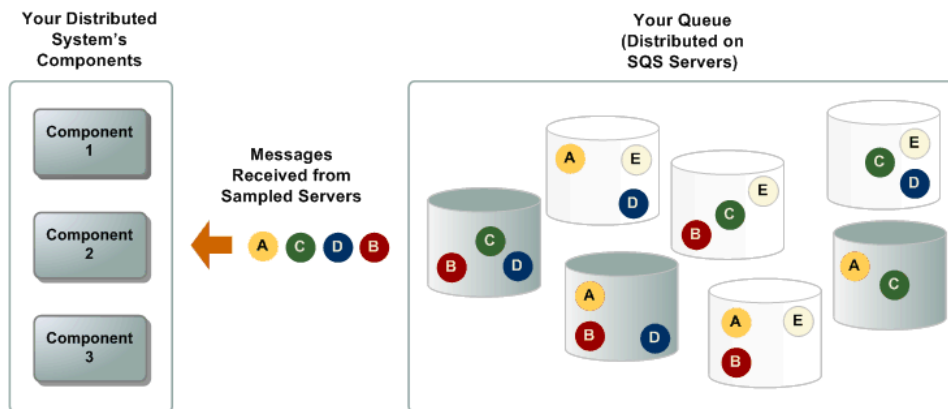
Amazon SQS 会在多台服务器上存储消息的副本，以实现冗余和高可用性。在极少数情况下，当您接收或删除消息时，存储消息副本的服务器之一可能不可用。如果出现这种情况，则该不可用服务器上的消息副本将不会被删除，并且您在接收消息时可能会再次获得该消息副本。因此，您必须将您的应用程序设计为幂等的应用程序（即，如果您的应用程序多次处理同一条消息，则不得受到不利影响）。

消息示例

从队列取回消息的行为取决于您使用的是短（标准）轮询（默认行为）还是长轮询。有关长轮询的更多信息，请参阅“[Amazon SQS 长轮询 \(p. 47\)](#)”。

如果使用短轮询，则您从队列中检索消息时，Amazon SQS 会对服务器的一个子集（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。这意味着，特定接收请求可能不会返回您的所有消息。或者，如果您的队列中有少量消息（少于 1000 条），则意味着，特定请求可能不会返回您的任何消息，而后续请求则会返回您的任何消息。如果您继续从您的队列中检索消息，则 Amazon SQS 会对所有服务器进行采样，您会收到您的所有消息。

下图显示了您的系统组件之一提出接收请求后返回消息的短轮询行为。Amazon SQS 对若干服务器（显示为灰色）进行采样并从这些服务器返回消息（消息 A、C、D 和 B）。系统不会为此特定请求返回消息 E，但会为后续请求返回该消息。



队列和消息标识符

Abstract

介绍 Amazon SQS 的三个标识符：队列 URL、消息 ID 和接收句柄。

Amazon SQS 使用您需要熟悉的以下三个标识符：

- 队列 URL
- 消息 ID
- 接收句柄

队列 URL

创建新队列时，您必须提供在您的所有队列范围内唯一的队列名称。如果您同时使用最新的 WSDL 和之前的版本创建队列，则您的所有队列仍具有单一命名空间。Amazon SQS 会为您创建的每个队列分配一个名为 *队列 URL* 的标识符，该标识符包括队列名称以及 Amazon SQS 确定的其他组件。每当您要对队列执行操作时，都需要提供其队列 URL。

以下是名为“queue2”的队列的队列 URL，该队列由 AWS 账户号为“123456789012”的人员所拥有。

```
http://sqs.us-east-1.amazonaws.com/123456789012/queue2
```



Important

在您的系统中，请始终存储 Amazon SQS 在您创建队列时向您返回的整个队列 URL（例如，`http://sqs.us-east-1.amazonaws.com/123456789012/queue2`）。每当您需要在请求中指定队列 URL 时，请勿从它的各个组件构建队列 URL，因为 Amazon SQS 可能会更改构成队列 URL 的组件。

此外，您还可以通过列出您的队列来获取队列的队列 URL。即使您的所有队列具有单一命名空间，但是返回的队列列表仍取决于您用于请求的 WSDL。有关更多信息，请参阅“[ListQueues](#)”。

消息 ID

每条消息都会收到一个系统分配的消息 ID，该 ID 由 Amazon SQS 在 [SendMessage](#) 响应中返回给您。此标识符对于识别消息很有用，但要删除消息，您需要消息的接收句柄，而不是消息 ID。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。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。这意味着，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。

```
MbZj6wDWli+JvwwJaBV+3dcjk2YW2vA3+STFFljTM8tJJg6HRG6PYSasuWXPJB+Cw  
LjlFjgXUvluSjlGUPAWV66FU/WeR4mq2OKpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE  
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

处理消息所需的资源

Abstract

介绍处理 Amazon SQS 消息所需的资源。

为了帮助您估计处理排队消息所需的资源，Amazon SQS 可以为您提供队列中消息的大概数量。您可以查看可见的消息数量，也可以查看不可见的消息数量。有关可见性的更多信息，请参阅“[可见性超时 \(p. 7\)](#)”。



Important

由于 Amazon SQS 采用的是分布式架构，因此，该结果并不是队列中消息数量的准确计数。在大多数情况下，该结果应接近于队列中消息的实际数量，但是您不应指望该计数精确无误。

下表列出了要使用的 API 操作。

要执行以下任务……	请使用以下操作	将 AttributeName 设置为以下值
获取队列中消息的大概数量	GetQueueAttributes	ApproximateNumberOfMessages
获取队列中不可见消息的大概数量	GetQueueAttributes	ApproximateNumberOfMessages-NotVisible

可见性超时

Abstract

介绍可见性超时的概念，Amazon SQS 使用可见性超时防止其他组件再次处理消息。

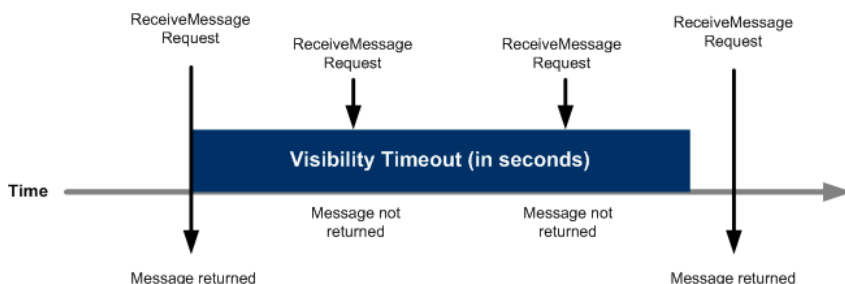
主题

- [可见性超时的一般建议](#) (p. 7)
- [延长消息的可见性超时](#) (p. 8)
- [终止消息的可见性超时](#) (p. 8)
- [与可见性超时相关的 API 操作](#) (p. 8)

您系统中的使用组件接收并处理来自队列的消息时，该消息会保留在队列中。为什么 Amazon SQS 不自动删除它？

因为您的系统是分布式系统，所以无法保证该组件会真正收到该消息（在收到该消息前，连接可能会中断，或者该组件可能会发生故障）。因此，Amazon SQS 不会删除该消息，而您的使用组件则必须在收到并处理该消息后从队列中删除该消息。

该组件收到该消息后，该消息仍在队列中。但是，您不希望系统中的其他组件再次接收并处理该消息。因此，Amazon SQS 会使用 *可见性超时*（这是 Amazon SQS 阻止其他使用组件接收并处理该消息的一段时间）来阻止这些组件。以下图片和讨论说明了这一概念。



Note

每个队列处于飞行状态的消息数限制为 120 000。消息的飞行状态是指该消息已由使用组件从队列接收，但尚未从队列中删除。如果您达到 120 000 的限制，将会收到一条来自 Amazon SQS 的“OverLimit”错误消息。为避免达到限制，您应该在处理消息后将其从队列删除。您还可以增加用来处理消息的队列数量。

可见性超时的一般建议

在 Amazon SQS 返回该消息后，可见性超时时钟开始计时。在这段时间里，该组件会处理并删除该消息。但是，如果该组件在删除该消息前发生故障，则会出现什么情况？如果您的系统在可见性超时过期之前没有为该消息调用 [DeleteMessage](#)，则该消息会再次变为对您系统中的组件发出的 [ReceiveMessage](#) 调用可见，并且会再次被接收。如果某条消息只应被接收一次，则您的系统应在可见性超时期间删除该消息。

每个队列的起始可见性超时的默认设置为 30 秒。您可以为整个队列更改该设置。通常，您会将可见性超时设置为处理消息并从队列中删除消息的平均时间。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。

如果您有一套生成消息的系统，这些消息需要不同的时间来处理并删除，则我们建议您创建多个队列，并且每个队列具有不同的可见性超时设置。随后，您的系统可以将所有消息发送到单一队列，该队列会根据每条消息的预期处理和删除时间，将该消息转发到具有适当可见性超时的另一个队列。

延长消息的可见性超时

收到来自队列的消息并开始处理该消息时，您可能会发现，该队列的可见性超时不足以完全处理并删除该消息。要为自己提供更多时间来处理该消息，您可以通过使用 [ChangeMessageVisibility](#) 操作指定新超时值来延长该消息的可见性超时。Amazon SQS 会使用新值重新启动超时期间。

例如，假设该队列的超时为 30 秒，并且您从该队列接收一条消息。在到达该消息超时的 20 秒（即，还剩 10 秒）时，您希望为自己额外提供 60 秒，于是，您立即为该消息调用 [ChangeMessageVisibility](#)，并将 *VisibilityTimeout* 设置为 60 秒。这意味着，您将该消息的可见性超时从 30 秒延长到 80 秒：初始超时设置中的 20 秒加上更改超时后的 60 秒。

延长消息的可见性超时后，新的超时仅会应用于该消息的特定接收。[ChangeMessageVisibility](#) 不会影响该队列的超时或该消息的后续接收。如果出于某个原因，您没有删除该消息，并且再次接收它，则该消息的可见性超时会是为该队列设置的原始值。

终止消息的可见性超时

收到来自该队列的某条消息时，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，这样会立即使该消息对系统中的其他组件可见以得到处理。要执行此操作，您可以调用 [ChangeMessageVisibility](#)，并将 *VisibilityTimeout* 设置为 0 秒。

与可见性超时相关的 API 操作

下表列出了用于操纵可见性超时的 API 操作。请使用每个操作的 *VisibilityTimeout* 参数来设置或获取值。

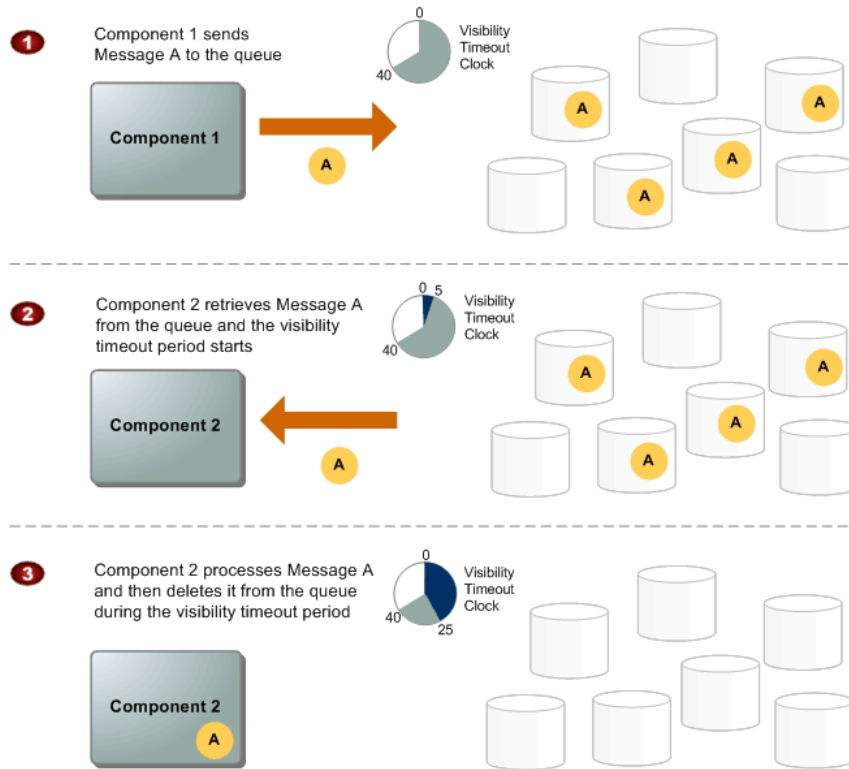
要执行以下任务……	请使用以下操作
设置队列的可见性超时	SetQueueAttributes
获取队列的可见性超时	GetQueueAttributes
设置所收到消息的可见性超时，而不影响队列的可见性超时	ReceiveMessage
延长或终止消息的可见性超时	ChangeMessageVisibility
延长或终止多达十条消息的可见性超时	ChangeMessageVisibilityBatch

消息生命周期

Abstract

介绍 Amazon SQS 消息的生命周期并显示消息的架构。

下面的示意图和流程描述了称为 *Message A* 的 Amazon SQS 消息从创建到删除的生命周期。假定一个队列已存在。



消息生命周期

1	Component 1 将 Message A 发送到一个队列，该消息在 SQS 服务器间冗余分布。
2	当 Component 2 准备好处理一条消息时，就从队列检索消息，然后 Message A 返回。在 Message A 处理期间，它仍然停留在队列中并且在 <i>可见性超时</i> 期间不返回给后继的接收请求。
3	Component 2 从队列删除 Message A，以避免一旦可见性超时过期该消息被再次接受并处理。



Note

SQS 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，您可以借助 [SetQueueAttributes](#) 将消息保存期设为 60 秒到 1 209 600 秒（14 天）之间的值。

结合使用 JMS 与 Amazon SQS

Abstract

介绍结合使用 Amazon SQS 与 JMS (Java Message Service) 的好处。

Amazon SQS Java Messaging Library 是连接 Amazon SQS 的一个 JMS 接口，其可使您在已经使用 JMS 的应用程序中充分利用 Amazon SQS。通过这个接口，您对代码稍作更改即可将 Amazon SQS 用作 JMS 提供程序。结合使用 AWS SDK for Java 与 Amazon SQS Java Messaging Library，您可以创建 JMS 连接、会话、创建者和使用者，以便与 Amazon SQS 队列之间收发消息。

Amazon SQS Java Messaging Library 支持队列消息收发（JMS 点对点模式），收发方式符合 [JMS 1.1 规范](#)。更具体地讲，Amazon SQS Java Messaging Library 支持将文本、字节或对象消息同步发送到 Amazon SQS 队列，以及同步或异步接收这些消息。有关 Amazon SQS Java Messaging Library 中支持的 JMS 1.1 规范功能的更多信息，请参阅[参考/附录 \(p. 29\)](#)和 [Amazon SQS 常见问题](#)。

您必须创建一个 Amazon SQS 队列与 JMS 配合使用。要创建 Amazon SQS 队列，您可以使用用于 Amazon SQS 的 AWS 管理控制台、CreateQueue API，或者 Amazon SQS Java Messaging Library 中包装提供的 Amazon SQS 客户端。有关使用 AWS 管理控制台或 CreateQueue API 创建用于 Amazon SQS 的队列的信息，请参阅[创建队列](#)。有关使用 Amazon SQS Java Messaging Library 的信息，请参阅 [Amazon SQS Java Messaging Library 入门 \(p. 11\)](#)。

主题

- [Amazon SQS 和 JMS 先决条件 \(p. 10\)](#)
- [Amazon SQS Java Messaging Library 入门 \(p. 11\)](#)
- [异步接收消息 \(p. 13\)](#)
- [客户端确认模式 \(p. 14\)](#)
- [无序确认模式 \(p. 15\)](#)
- [代码示例 \(p. 15\)](#)
- [参考/附录 \(p. 29\)](#)

Amazon SQS 和 JMS 先决条件

要结合使用 Amazon SQS 与 JMS，您必须有以下组件：

- 适用于 Java 的开发工具包 – 可通过两种不同的方式在您的项目中包含 适用于 Java 的开发工具包。您可以下载并安装 适用于 Java 的开发工具包；或者，如果您使用 Maven 获取 Amazon SQS Java

Messaging Library，则可以包含 适用于 Java 的开发工具包 作为依赖项。适用于 Java 的开发工具包和 Amazon SQS Java Messaging Library 要求有 J2SE Development Kit 6.0 或更高版本。有关下载 适用于 Java 的开发工具包 的信息，请转到 [适用于 Java 的开发工具包](#)。有关使用 Maven 的更多信息，请参见以下注释。

- Amazon SQS Java Messaging Library – 如果您不使用 Maven，则必须将包文件 `amazon-sqs-java-messaging-lib.jar` 添加到 Java 生成类路径中。有关下载的信息，请参阅 [Amazon SQS Java Messaging Library](#)。



Note

Amazon SQS Java Messaging Library 包含对 [Maven](#) 和 [Spring Framework](#) 的支持。有关使用 Maven、Spring Framework 和 Amazon SQS Java Messaging Library 的代码示例，请参阅[代码示例 \(p. 15\)](#)。

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
  <version>1.0.0</version>
  <type>jar</type>
</dependency>
```

Amazon SQS Java Messaging Library 入门

满足先决条件以后，您可以使用以下代码示例开始结合使用 Amazon SQS 与 JMS。

此示例将创建 JMS 连接和会话，然后收发一条消息。Amazon SQS Java Messaging Library 中包装提供的 Amazon SQS 客户端还会检查是否存在 Amazon SQS 队列。此外，它还使用包装的 Amazon SQS 客户端检查是否存在 Amazon SQS 队列。如果不存在，则创建该队列。

创建 JMS 连接

为创建 JMS 连接，必须先创建一个连接工厂，然后对该工厂调用 `createConnection` 方法：

```
// Create the connection factory using the environment variable credential
// provider.
// Connections this factory creates can talk to the queues in us-east-1 region.

SQSConnectionFactory connectionFactory =
    SQSConnectionFactory.builder()
        .withRegion(Region.getRegion(Regions.US_EAST_1))
        .withAWSCredentialsProvider(new EnvironmentVariableCredentialsProvider())

        .build();

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



Note

该示例中的 `EnvironmentVariableCredentialsProvider` 类假定已设置了 `AWS_ACCESS_KEY_ID` (或 `AWS_ACCESS_KEY`) 和 `AWS_SECRET_KEY` (或 `AWS_SECRET_ACCESS_KEY`) 环境变量。有关更多信息，包括为该工厂提供所需证书的其他选项，请参阅 [AWSCredentialsProvider](#)。

SQSConnection 类扩展 javax.jms.Connection。除了 JMS 标准连接方法，SQSConnection 还提供其他一些方法，例如 getAmazonSQSClient 和 getWrappedAmazonSQSClient。从 getAmazonSQSClient 和 getWrappedAmazonSQSClient 返回的客户端对象可用于执行未包含在 JMS 规范中的管理操作 – 例如创建 Amazon SQS 队列。

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。另一方面，如果您使用 getAmazonSQSClient，则这些异常将为 Amazon SQS 异常。因此，如果您现有的代码需要 JMS 异常，则应使用 getWrappedAmazonSQSClient。

创建 Amazon SQS 队列

此示例检查 Amazon SQS 队列是否存在，如果不存在，则创建一个。接下来使用包装的客户端对象检查 Amazon SQS 队列是否存在，如果不存在，则创建一个新队列：

```
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an SQS queue named 'TestQueue' - if it does not already exist.
if (!client.queueExists("TestQueue")) {
    client.createQueue("TestQueue");
}
```

同步发送消息

连接和基础 Amazon SQS 队列准备就绪以后，将创建一个具有 `AUTO_ACKNOWLEDGE` 模式的非事务性 JMS 会话：

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

下一步，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 with name 'TestQueue' in the session
Queue queue = session.createQueue("TestQueue");

// Create a producer for the 'TestQueue'.
MessageProducer producer = session.createProducer(queue);
```

创建文本消息并将其发送到队列：

```
// Create the text message.
TextMessage message = session.createTextMessage("Hello World!");

// Send the message.
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

同步接收消息

要接收消息，将在同一队列上创建一个使用者，然后调用 `start` 方法。可随时调用连接上的 `start` 方法，但直到调用该方法时使用者才会接收消息。

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

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

将超时设为 1 秒钟，对使用者调用 `receive` 方法，然后输出收到的消息内容：

```
// Receive a message from 'TestQueue' and wait up to 1 second
Message receivedMessage = consumer.receive(1000);

// Cast the received message as TextMessage and print the text to screen.
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
}
```

最后，关闭连接，这也会关闭会话：

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

输出将与以下类似：

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



Note

您也可以使用 Spring 初始化这些对象。有关更多信息，请参阅 `SpringExampleConfig.xml`、`SpringExample.java`，以及[代码示例 \(p. 15\)](#)中提供的 `ExampleConfiguration.java` 和 `ExampleCommon.java` 中的其他帮助程序类。

有关发送和接收的完整示例，请参阅 `TextMessageSender.java` 和 `SyncMessageReceiver.java`。

异步接收消息

在 [Amazon SQS Java Messaging Library 入门 \(p. 11\)](#)中提供的示例中，消息发送到 `TestQueue` 并被同步接收。此示例介绍如何通过监听器异步接收消息。

首先，实现 `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.
            if (message != null) {
                System.out.println("Received: " + ((TextMessage) message).get
                Text());
            }
        } catch (Exception e) {
            // Handle exception
        }
    }
}
```

```
    }  
    } catch (JMSEException e) {  
        e.printStackTrace();  
    }  
}  
}
```

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

下一步并不是对使用者显式调用 `receive` 方法，而是对 `MyListener` 实现实例设置使用者消息监听器。主线程会睡眠一秒钟：

```
// Create a consumer for the 'TestQueue'.  
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 will be invoked when a  
message is received.  
Thread.sleep(1000);
```

其余步骤与 [Amazon SQS Java Messaging Library 入门 \(p. 11\)](#) 中的示例所提供的步骤相同。有关异步收件人的完整示例，请参阅[代码示例 \(p. 15\)](#)中的 `AsyncMessageReceiver.java`。

此示例的输出将与以下类似：

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

客户端确认模式

[Amazon SQS Java Messaging Library 入门 \(p. 11\)](#) 中提供的示例使用了 `AUTO_ACKNOWLEDGE` 模式，该模式会自动确认收到的每条消息 – 因此会从 Amazon SQS 队列中删除消息。如果您希望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);
```

下一步，当收到消息时，显示并显式确认该消息：

```
// 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，当确认消息时，也会隐式确认先前收到的所有消息。例如，如果收到 10 条消息，则确认第 10 条消息（按接收它们的顺序）也会同时确认先前的所有 9 条消息。

其余步骤与 [Amazon SQS Java Messaging Library 入门 \(p. 11\)](#) 中的示例所提供的步骤相同。有关具有客户端确认模式的同步收件人的完整示例，请参阅[代码示例 \(p. 15\)](#)中的 `SyncMessageReceiverClientAcknowledge.java`。

此示例的输出将与以下类似：

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5  
Received: Hello World!  
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

无序确认模式

在 [客户端确认模式 \(p. 14\)](#) 中先前显式的示例中，（当使用 `CLIENT_ACKNOWLEDGE` 模式时）在显式确认的消息之前收到的所有消息均会自动进行确认。在 JMS 1.1 规范中，这称为 `ACK-RANGE`。Amazon SQS Java Messaging Library 提供了另一种确认模式 `UNORDERED_ACKNOWLEDGE`，在该模式中，无论接收消息的顺序如何，客户端都必须显式确认所有收到的消息。换句话说，使用 `UNORDERED_ACKNOWLEDGE` 模式时，如果确认了某条消息，将仅确认该特定消息。

此示例中的会话采用 `UNORDERED_ACKNOWLEDGE` 模式创建：

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.  
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

其余步骤与 [客户端确认模式 \(p. 14\)](#) 中的示例所提供的步骤相同。有关具有无序确认模式的同步收件人的完整示例，请参阅[代码示例 \(p. 15\)](#)中的 `SyncMessageReceiverUnorderedAcknowledge.java`。

此示例的输出将与以下类似：

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7  
Received: Hello World!  
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

代码示例

以下代码示例显示如何结合使用 JMS 与 Amazon SQS。

ExampleConfiguration.java

以下 Java 代码是可用于所有后续 Java 示例的配置示例。例如，此代码设置了将在其他 java 示例中使用的默认队列名称、区域和证书。

```
public class ExampleConfiguration {
    public static final String DEFAULT_QUEUE_NAME = "SQSJMSClientExampleQueue";

    public static final Region DEFAULT_REGION = Region.getRegion(Regions.US_EAST_1);

    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 PropertiesFileCredentialsPro
vider(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 DefaultAWS creden
tialsProviderChain();

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 credentialsPro
vider) {
    // Make sure they're usable first
    credentialsProvider.getCredentials();
    this.credentialsProvider = credentialsProvider;
}
}
```

TextMessageSender.java

以下 Java 代码显示如何创建文本消息发件人。

```
public class TextMessageSender {
    public static void main(String args[]) throws JMSEException {
        ExampleConfiguration config = ExampleConfiguration.parseConfig("TextMes
sageSender", args);
    }
}
```

```
ExampleCommon.setupLogging();

// Create the connection factory based on the config
SQSConnectionFactory connectionFactory =
    SQSConnectionFactory.builder()
        .withRegion(config.getRegion())
        .withAWSCredentialsProvider(config.getCredentialsProvider())

        .build();

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

private static void sendMessages( Session session, MessageProducer producer
) {
    BufferedReader inputReader = new BufferedReader(
        new InputStreamReader( System.in, Charset.defaultCharset() )
    );

    try {
        String input;
        while( true ) {
            System.out.print( "Enter message to send (leave empty to exit):
" );

            input = inputReader.readLine();
            if( input == null || input.equals("") ) break;

            TextMessage message = session.createTextMessage(input);
            producer.send(message);
            System.out.println( "Send message " + message.getJMSMessageID()
        );

        }
    } catch (EOFException e) {
        // Just return on EOF
    } catch (IOException e) {
        System.err.println( "Failed reading input: " + e.getMessage() );
    } catch (JMSException e) {
        System.err.println( "Failed sending message: " + e.getMessage() );
        e.printStackTrace();
    }
}
```

```
}  
}
```

AsyncMessageReceiver.java

以下 Java 代码显示如何创建异步消息收件人。

```
public class AsyncMessageReceiver {  
    public static void main(String args[]) throws JMSEException, InterruptedException {  
        ExampleConfiguration config = ExampleConfiguration.parseConfig("AsyncMessageReceiver", args);  
  
        ExampleCommon.setupLogging();  
  
        // 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 will be processed until this is called  
        connection.start();  
  
        callback.waitForOneMinuteOfSilence();  
        System.out.println( "Returning after one minute of silence" );  
  
        // Close the connection. This will close 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();
            }
        }
    }
}
```

SyncMessageReceiver.java

以下 Java 代码显示如何创建同步消息收件人。

```
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 =
            SQSConnectionFactory.builder()
                .withRegion(config.getRegion())
                .withAWSCredentialsProvider(config.getCredentialsProvider())
                .build();

        // 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 will close 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();
        }
    }
}
```

SyncMessageReceiverClientAcknowledge.java

以下 Java 代码显示了具有客户端确认模式的同步收件人的示例。

```
/**
 * 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 is 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 is NOT 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 =
        SQSConnectionFactory.builder()
            .withRegion(config.getRegion())
            .withAWSCredentialsProvider(config.getCredentialsProvider())
            .build();

    // 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 will result in receiving no messages since
    // we have acknowledged the second message. Although we did not 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 will close 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 re
     * ceived, "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 acknow
        ledge) throws JMSEException {
        // Receive a message
        Message message = consumer.receive(TimeUnit.SECONDS.to
        Millis(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 代码显示了具有无序确认模式的同步收件人的示例。

```
/**
 * 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 is 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 are received.
 *
 * This is NOT the behavior for CLIENT_ACKNOWLEDGE mode. Please see {@link
 * SyncMessageReceiverClientAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverUnorderedAcknowledge {

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

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

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

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory =
            SQSConnectionFactory.builder()
                .withRegion(config.getRegion())
                .withAWSCredentialsProvider(config.getCredentialsProvider())
                .build();

        // 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(con
fig.getQueueName()));
MessageConsumer consumer = session.createConsumer(session.createQueue(con
fig.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 will
result in receiving the first message since
// we have acknowledged only the second message. In the UNORDERED_AC
KNOWLEDGE mode, all the messages must
// be explicitly acknowledged.
receiveMessage(consumer, true);

// Close the connection. This will close 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 re
ceived, "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 acknow
    ledge) throws JMSEException {
        // Receive a message
        Message message = consumer.receive(TimeUnit.SECONDS.to
        Millis(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();
        }
    }
}
```

SpringExampleConfig.xml

下面显示了一个示例 Spring 配置文件。

```
<?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.springframe
        work.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/util http://www.springframe
        work.org/schema/util/spring-util-3.0.xsd
    ">

    <bean id="CredentialsProviderBean" class="com.amazonaws.auth.DefaultAWSCre
    dentialsProviderChain"/>

    <bean id="ConnectionFactoryBuilder" class="com.amazon.sqs.javames
    saging.SQSConnectionFactory$Builder">
        <property name="regionName" value="us-east-1"/>
        <property name="numberOfMessagesToPrefetch" value="5"/>

        <property name="awsCredentialsProvider" ref="CredentialsProviderBean"/>
    </bean>

    <bean id="ConnectionFactory" class="com.amazon.sqs.javamessaging.SQSConnection
    Factory"
        factory-bean="ConnectionFactoryBuilder"
        factory-method="build" />
```

```
<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 代码显示了使用 [SpringExampleConfig.xml](#) (p. 26) 的示例。

```
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] + " does not exist or is not
readable." );
            System.exit(2);
        }

        ExampleCommon.setupLogging();

        FileSystemXmlApplicationContext context =
            new FileSystemXmlApplicationContext( "file://" + springFile.get
AbsolutePath() );

        Connection connection;
        try {
            connection = context.getBean(Connection.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Could not 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( "Could not find the name of the queue to use:
" + e.getMessage() );
            System.exit(3);
            return;
        }
    }
}
```



```
        }

        if( connection instanceof SQSConnection ) {
            ExampleCommon.ensureQueueExists( (SQSConnection) connection,
queueName );
        }

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer = session.createConsumer( session.createQueue(
queueName) );

        receiveMessages(session, consumer);

        // The context can be setup to close the connection for us
        context.close();
        System.out.println( "Context closed" );
    }

    private static void receiveMessages( Session session, MessageConsumer consumer ) {
        try {
            while( true ) {
                System.out.println( "Waiting for messages");
                // Wait 1 minute for a message
                Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));

                if( message == null ) {
                    System.out.println( "Shutting down after 1 minute of silence" );
                }

                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message" );
        } catch (JMSEException e) {
            System.err.println( "Error receiving from SQS: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

ExampleCommon.java

以下 Java 代码首先查看 Amazon SQS 队列是否存在，如果不存在，则创建一个。此外还提供了日志记录代码的示例。

```
public class ExampleCommon {
    /**
     * A utility function to check the queue exists and create it if needed.
     For most
     * use cases this will usually be done by an administrator before the ap
     plication
    */
}
```

```
    * is run.
    */
    public static void ensureQueueExists(SQSConnection connection, String
queueName) throws JMSEException {
        AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazon
SQSClient();

        /**
         * For most cases this could be done 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 do not have permission to call CreateQueue.
         */
        if( !client.queueExists(queueName) ) {
            client.createQueue( queueName );
        }
    }

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

    public static void handleMessage(Message message) throws JMSEException {
        System.out.println( "Got message " + message.getJMSMessageID() );
        System.out.println( "Content: " );
        if( message instanceof TextMessage ) {
            TextMessage txtMessage = ( TextMessage ) message;
            System.out.println( "\t" + txtMessage.getText() );
        } else if( message instanceof BytesMessage ){
            BytesMessage byteMessage = ( BytesMessage ) message;
            // Assume the length fits in an int - SQS only supports sizes up
to 256k so that
            // should be true
            byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
            byteMessage.readBytes(bytes);
            System.out.println( "\t" + Base64.encodeAsString( bytes ) );
        } else if( message instanceof ObjectMessage ) {
            ObjectMessage objMessage = (ObjectMessage) message;
            System.out.println( "\t" + objMessage.getObject() );
        }
    }
}
```

参考/附录

下面列出了 Amazon SQS Java Messaging Library 中支持的一些 [JMS 1.1 规范](#) 实现。有关 Amazon SQS Java Messaging Library 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 JMS 1.1 常用接口：

- ConnectionFactory
- 连接
- 目标
- Session
- MessageProducer
- MessageConsumer

支持的消息类型

- TextMessage
- ByteMessage
- ObjectMessage

支持的消息确认模式

- DUPS_OK_ACKNOWLEDGE
- AUTO_ACKNOWLEDGE
- CLIENT_ACKNOWLEDGE
- *UNORDERED_ACKNOWLEDGE*



Note

UNORDERED_ACKNOWLEDGE 确认模式不属于 JMS 1.1 规范内容。该模式由 Amazon SQS 添加，旨在允许 JMS 客户端显式确认消息。

JMS 定义标头和预留属性

Amazon SQS JMS 客户端设置以下 JMS 定义标头：

- JMSTDestination
- JMSMessageID
- JMSRedelivered

Amazon SQS JMS 客户端设置以下 JMS 预留属性：

- JMSXDeliveryCount

使用 Amazon S3 管理 Amazon SQS 消息

Abstract

介绍使用 Amazon S3 管理 Amazon SQS 消息的好处。

您可以使用 Amazon S3 管理 Amazon SQS 消息。这在存储和检索大小高达 2 GB 的消息时特别有用。要使用 Amazon S3 管理 Amazon SQS 消息，请使用适用于 Java 的 Amazon SQS 扩展客户端库。具体来说，您可使用该库执行以下操作：

- 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 Amazon S3 存储桶中的单个消息对象的消息。
- 从 Amazon S3 存储桶中获取相应的消息对象。
- 从 Amazon S3 存储桶中删除相应的消息对象。



Note

您只能使用适用于 Java 的 Amazon SQS 扩展客户端库来通过 Amazon S3 管理 Amazon SQS 消息；无法使用 AWS CLI、Amazon SQS 控制台、Amazon SQS HTTP API 或任一 AWS 软件开发工具包（适用于 Java 的开发工具包除外，如本主题后面所述）。

先决条件

要使用 Amazon S3 管理 Amazon SQS 消息，您需要：

- AWS SDK for Java – 可通过两种不同的方式将适用于 Java 的开发工具包包含在您的项目中。您可以下载并安装适用于 Java 的开发工具包，或者，如果您使用 Maven 获取适用于 Java 的 Amazon SQS 扩展客户端库，则可以包含适用于 Java 的开发工具包作为依赖项。适用于 Java 的开发工具包和适用于 Java 的 Amazon SQS 扩展客户端库要求有 J2SE Development Kit 6.0 或更高版本。有关下载适用于 Java 的开发工具包的信息，请转到 [适用于 Java 的开发工具包](#)。有关使用 Maven 的更多信息，请参阅此列表后面的注释。
- 适用于 Java 的 Amazon SQS 扩展客户端库 – 如果您不使用 Maven，则必须将包文件 `amazon-sqs-java-extended-client-lib.jar` 添加到 Java 生成类路径。有关下载的信息，请转到 [适用于 Java 的 Amazon SQS 扩展客户端库](#)。

- Amazon S3 存储桶 – 您必须创建新的 Amazon S3 存储桶或使用现有的存储桶来存储消息。建议您创建新的存储桶来实现此目的。要控制存储桶空间和 AWS 账户产生的费用，您还应在存储桶上设置生命周期配置规则，以便在消息对象创建日期之后的一段特定时间后永久删除该对象。有关说明，请参阅[管理生命周期配置](#)或本部分后面的[示例 \(p. 32\)](#)。



Note

适用于 Java 的 Amazon SQS 扩展客户端库包含对 [Maven](#) 的支持，如下所示：

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-sqs</artifactId>
    <version>${aws-java-sdk.version}</version>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-s3</artifactId>
    <version>${aws-java-sdk.version}</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>1.10.8</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

使用适用于 Java 的 Amazon SQS 扩展客户端库

在满足[先决条件 \(p. 31\)](#)后，使用以下 Java 代码示例开始通过 Amazon S3 管理 Amazon SQS 消息。

该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 天后永久删除该存储桶。然后，该示例会创建一个队列并向该队列发送 256 KB 以上的随机消息。此消息将存储在 Amazon S3 存储桶中。然后，该示例会检索此消息并打印有关检索到的消息的信息。然后，该示例会删除该消息、队列和存储桶。

```
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
import java.util.UUID;
import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClient;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration;
import com.amazonaws.services.s3.model.ListVersionsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.S3ObjectSummary;
import com.amazonaws.services.s3.model.S3VersionSummary;
import com.amazonaws.services.s3.model.VersionListing;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.DeleteMessageRequest;
import com.amazonaws.services.sqs.model.DeleteQueueRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.SendMessageRequest;
import com.amazonaws.sqs.javamessaging.ExtendedClientConfiguration;

public class SQSExtendedClientExample {

    private static final String s3BucketName = UUID.randomUUID() + "-"
        + DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new DateTime());

    public static void main(String[] args) {

        AWSCredentials credentials = null;

        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                "Cannot load the AWS credentials from the expected AWS credential pro-
files file. "
                + "Make sure that your credentials file is at the correct "
                + "location (/home/$USER/.aws/credentials) and is in a valid format.",
                e);
        }

        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.
        //   Then create the bucket, and enable message objects to be stored in the
        bucket.
        BucketLifecycleConfiguration.Rule expirationRule = new BucketLifecycleCon-
figuration.Rule();
        expirationRule.withExpirationInDays(14).withStatus("Enabled");
        BucketLifecycleConfiguration lifecycleConfig = new BucketLifecycleConfig-
uration().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 ExtendedClientConfig
uration()
    .withLargePayloadSupportEnabled(s3, s3BucketName);

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

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

// Create a message queue for this example.
String QueueName = "QueueName" + UUID.randomUUID().toString();
CreateQueueRequest createQueueRequest = new CreateQueueRequest(QueueName);

String myQueueUrl = sqsExtended.createQueue(createQueueRequest).getQueueUrl();
System.out.println("Queue created.");

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

// Receive messages, and then print general information about them.
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest(myQueueUrl);
List<Message> messages = sqsExtended.receiveMessage(receiveMessageRequest).getMessages();

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.
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(s3BucketName);

    while (true) {
        for (Iterator<?> iterator = objectListing.getObjectSummaries().iterator();
            iterator.hasNext(); ) {
            S3ObjectSummary objectSummary = (S3ObjectSummary) iterator.next();
            client.deleteObject(s3BucketName, objectSummary.getKey());
        }

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

    VersionListing list = client.listVersions(new ListVersionsRequest().with
        BucketName(s3BucketName));

    for (Iterator<?> iterator = list.getVersionSummaries().iterator(); iterat
        or.hasNext(); ) {
        S3VersionSummary s = (S3VersionSummary) iterator.next();
        client.deleteVersion(s3BucketName, s.getKey(), s.getVersionId());
    }

    client.deleteBucket(s3BucketName);

}

}
```


使用 Amazon SQS 消息属性

Abstract

包含可选的 Amazon SQS 消息属性 (用于包含有关消息的结构化元数据)。

Amazon Simple Queue Service (Amazon SQS) 为消息属性提供支持。消息属性可使您提供消息相关的结构化元数据项目 (如时间戳、地理空间数据、签名和标识符)。消息属性是可选的, 并独立于消息正文 (但随之一起发送)。此信息可以由消息的接收方用于帮助决定如何处理消息, 而不必先处理消息正文。每条消息最多可以包含 10 个属性。要指定消息属性, 您可以使用 AWS 管理控制台、AWS 软件开发工具包或查询 API。

主题

- [消息属性项目和验证 \(p. 36\)](#)
- [消息属性数据类型和验证 \(p. 37\)](#)
- [通过 AWS 管理控制台使用消息属性 \(p. 37\)](#)
- [利用 AWS 软件开发工具包来使用消息属性 \(p. 41\)](#)
- [通过 Amazon SQS 查询 API 使用消息属性 \(p. 43\)](#)
- [MD5 消息摘要计算 \(p. 44\)](#)

消息属性项目和验证

每个消息属性包含以下项目：

- 名称 – 消息属性名称可以包含以下字符：A-Z、a-z、0-9、下划线 (_)、连字符 (-) 和句点 (.)。名称不得以句点开头或结尾, 并且不应包含连续句点。名称区分大小写, 且必须在消息的所有属性名称中是唯一的。名称最多可以有 256 个字符。名称不能以“AWS.”或“Amazon.” (或任何大小写变体) 开头, 因为这些前缀是预留供 Amazon Web Services 使用的。
- 类型 – 受支持的消息属性数据类型是字符串、数字和二进制。您也可以提供有关类型的自定义信息。数据类型在内容方面具有与消息正文相同的限制。数据类型区分大小写, 长度可以最多为 256 字节。想要了解更多信息, 请参阅[消息属性数据类型和验证 \(p. 37\)](#)部分。
- 值 – 用户指定的消息属性值。对于字符串数据类型, 值属性在内容方面具有与消息正文相同的限制。有关更多信息, 请参阅[SendMessage](#)。

名称、类型和值都不得为空或 null。此外, 消息正文也不应为空或 null。消息属性的所有部分 (包括名称、类型和值) 都包含在消息大小限制中, 该限制当前是 256 KB (262 144 个字节)。

消息属性数据类型和验证

消息属性数据类型标识 Amazon SQS 处理消息属性值的方式。例如，如果类型是数字，则 Amazon SQS 会验证它是否为数字。

Amazon SQS 支持以下逻辑数据类型（使用可选的自定义类型标签）：

字符串	.<自定义类型>（可选）
数字	.<自定义类型>（可选）
二进制	.<自定义类型>（可选）

- 字符串 – 字符串是使用 UTF-8 二进制编码的 Unicode。若要获取代码值的列表，请访问 http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters。
- 数字 – 数字是正或负整数或是浮点数。数字具有足够的范围和精度，以便包含整数、浮点数和双精度数通常支持的大多数可能值。数字最多可有 38 位数字精度，并且介于 10^{-128} 到 10^{+126} 之间。系统会删减开头和结尾的 0。
- 二进制 – 二进制类型属性可以存储任何二进制数据，例如压缩数据、加密数据或图像。
- 自定义类型 – 您可以将自定义类型标签附加到支持的数据类型（字符串、数字、二进制）以创建自定义数据类型。此功能类似于编程语言中的 *type traits*。例如，如果您的应用程序需要知道在消息中发送的数字类型，则您可以创建类似于下面这样的自定义类型：*Number.byte*、*Number.short*、*Number.int* 和 *Number.float*。使用二进制数据类型的另一个示例是使用 *Binary.gif* 和 *Binary.png* 在一个消息或批量消息中区分不同的图像文件类型。附加数据是可选的并且对于 Amazon SQS 不透明，这意味着 Amazon SQS 不会解释、验证或使用附加数据。自定义类型扩展在允许的字符方面具有与消息正文相同的限制。

通过 AWS 管理控制台使用消息属性

您可以使用 AWS 管理控制台配置消息属性。在 Amazon SQS 控制台中，选中一个队列，单击 Queue Actions (队列操作) 下拉列表，然后选择 Send a Message (发送消息)。控制台需要用户输入 Base-64 编码的值以便发送二进制类型。

Send a Message to MyQueue

Message Body

Message Attributes

Name

Type

String

(Custom Type: Optional)

Value

Enter a string value.

Add Attribute

What is Message Attribute?

Name	Type	Values
------	------	--------

Cancel

Send Message

在 Message Attributes (消息属性) 选项卡上，为消息属性输入名称，选择类型，然后输入值。(可选) 您还可以将自定义信息附加到类型。例如，以下屏幕显示选择了 *Number* (数字) 类型，并添加了 *byte* (字节) 以进行自定义。有关支持的数据类型的自定义数据的更多信息，请参阅[消息属性数据类型和验证](#) (p. 37) 部分。

Send a Message to MyQueue

Message Body

Message Attributes

Name

MyMessageAttributeName

Type

Number

byte

Value

24

Enter a numerical value.

Add Attribute

What is Message Attribute?

Name	Type	Values
------	------	--------

Cancel

Send Message

要添加属性，请单击 Add Attribute (添加属性)。属性信息随后会出现在 Name (名称)、Type (类型) 和 Values (值) 列表中。

Send a Message to MyQueue

Message Body

Message Attributes

Name

Type

String

(Custom Type: Optional)

Value

Enter a string value.

Add Attribute

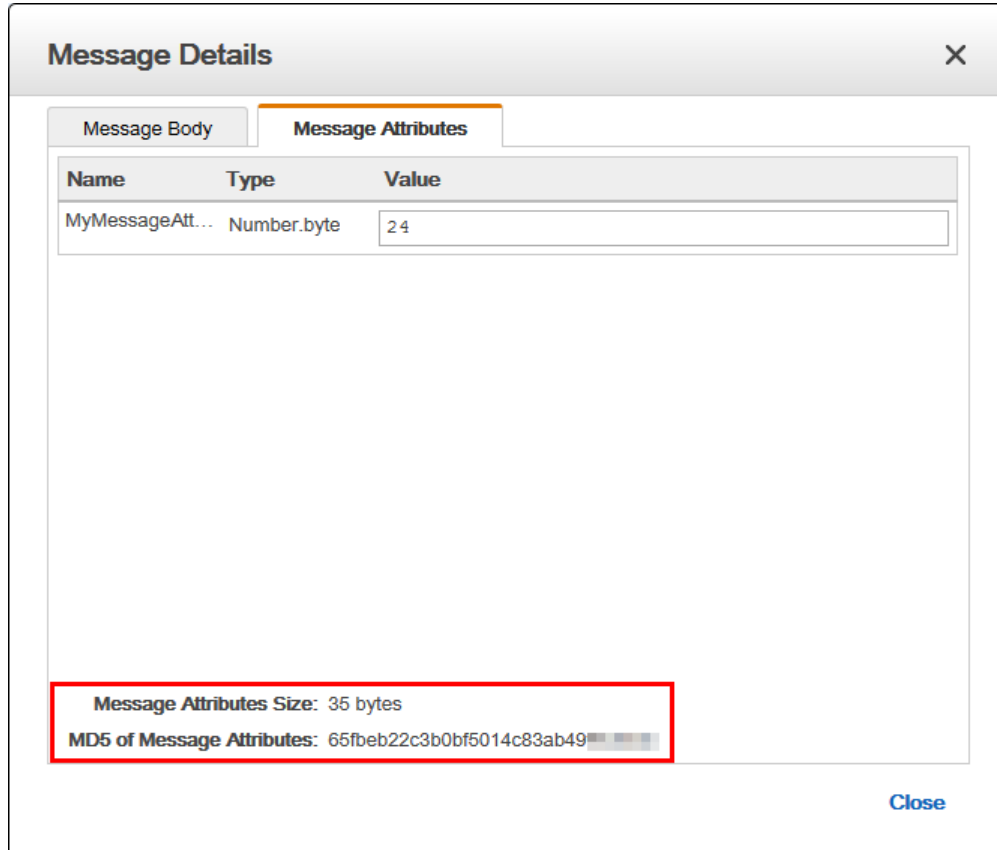
What is Message Attribute?

Name	Type	Values
MyMessage...	Number.byte	24

Cancel

Send Message

您还可以使用控制台查看有关收到的消息的消息属性的信息。在控制台中，选择一个队列，单击 Queue Actions (队列操作) 下拉列表，然后选择 View/Delete Messages (查看/删除消息)。在消息列表中，单击 Message Details (消息详细信息) 以查看信息。例如，您可以查看消息属性大小和 MD5 消息摘要。



利用 AWS 软件开发工具包来使用消息属性

[AWS 软件开发工具包](#)提供了多种语言的 API，以便将消息属性用于 Amazon SQS。此部分包括一些 Java 示例，用于演示如何使用消息属性。这些示例可以与 [适用于 Java 的开发工具包](#) 中的 `SimpleQueueServiceSample.java` 示例集成。最新的 [适用于 Java 的开发工具包](#) 会自动计算 `MessageBody` 和 `MessageAttributes` 校验和并将其与 Amazon SQS 返回的数据进行比较。有关 [适用于 Java 的开发工具包](#) 的更多信息，请参阅[适用于 Java 的 AWS 软件开发工具包入门](#)。

以下三个 Java 示例演示如何使用 `MessageAttributeValue` 方法设置消息属性的 `String`、`Number` 和 `Binary` 参数：

字符串

```
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("attributeName", new MessageAttributeValue().withData
Type("String").withStringValue("string-value-attribute-value"));
```

数字

```
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("attributeName", new MessageAttributeValue().withData
```

```
Type("Number").withStringValue("230.000000000000000001"));
```

二进制

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

以下三个示例演示如何将可选的自定义类型用于消息属性：

字符串 – 自定义

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

数字 – 自定义

```
// NOTE Because the type is a number, the result in the receive message call  
// will be 123456.  
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();  
messageAttributes.put("AccountId", new MessageAttributeValue().withDataType("Num  
ber.AccountId").withStringValue("000123456"));
```

二进制 – 自定义

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

要使用前面的消息属性示例之一发送消息，您的代码应类似于以下内容：

```
SendMessageRequest request = new SendMessageRequest();  
request.withMessageBody("A test message body.");  
request.withQueueUrl("MyQueueUrlStringHere");  
request.withMessageAttributes(messageAttributes);  
sqs.sendMessage(request);
```

通过 Amazon SQS 查询 API 使用消息属性

要使用查询 API 指定消息属性，请调用 [SendMessage](#)、[SendMessageBatch](#) 或 [ReceiveMessage](#) 操作。

此示例的查询 API 请求类似于以下内容：

您如何构造 AUTHPARAMS 取决于您如何签署您的 API 请求。有关签名版本 4 中 AUTHPARAMS 的信息，请转至 [已签名的签名版本 4 请求示例](#)。

```
POST http://sqs.us-east-1.amazonaws.com/123456789012/MyQueue
...
?Action=SendMessage
&MessageBody=This+is+a+test+message
&MessageAttribute.1.Name=test_attribute_name_1
&MessageAttribute.1.Value.StringValue=test_attribute_value_1
&MessageAttribute.1.Value.DataType=String
&MessageAttribute.2.Name=test_attribute_name_2
&MessageAttribute.2.Value.StringValue=test_attribute_value_2
&MessageAttribute.2.Value.DataType=String
&Version=2012-11-05
&Expires=2014-05-05T22%3A52%3A43PST
&AUTHPARAMS
```



Note

队列名称和队列 URL 区分大小写。

查询 API 响应应类似于以下内容：

```
HTTP/1.1 200 OK
...
<SendMessageResponse>
  <SendMessageResult>
    <MD5OfMessageBody>
      fafb00f5732ab283681e124bf8747ed1
    </MD5OfMessageBody>
    <MD5OfMessageAttributes>
      3ae8f24a165a8cedc005670c81a27295
    </MD5OfMessageAttributes>
    <MessageId>
      5fea7756-0ea4-451a-a703-a558b933e274
    </MessageId>
  </SendMessageResult>
  <ResponseMetadata>
    <RequestId>
      27daac76-34dd-47df-bd01-1f6e873584a0
    </RequestId>
  </ResponseMetadata>
</SendMessageResponse>
```

使用 [SendMessageBatch](#) 时，需要为批处理中的每个消息指定消息属性。

此示例的查询 API 请求类似于以下内容：

```
POST http://sqs.us-east-1.amazonaws.com/123456789012/MyQueue
...
?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
&SendMessageBatchRequestEntry.2.MessageAttribute.1.Name=test_attribute_name_1
&SendMessageBatchRequestEntry.2.MessageAttribute.1.Value.StringValue=test_at
tribute_value_1
&SendMessageBatchRequestEntry.2.MessageAttribute.1.Value.DataType=String
&Version=2012-11-05
&Expires=2014-05-05T22%3A52%3A43PST
&AUTHPARAMS
```

查询 API 响应应类似于以下内容：

```
HTTP/1.1 200 OK
...
<SendMessageBatchResponse>
<SendMessageBatchResult>
  <SendMessageBatchResultEntry>
    <Id>test_msg_001</Id>
    <MessageId>0a5231c7-8bff-4955-be2e-8dc7c50a25fa</MessageId>
    <MD5OfMessageBody>0e024d309850c78cba5eabbef7cae71</MD5OfMessageBody>
  </SendMessageBatchResultEntry>
  <SendMessageBatchResultEntry>
    <Id>test_msg_002</Id>
    <MessageId>15eeled3-87e7-40c1-bdaa-2e49968ea7e9</MessageId>
    <MD5OfMessageBody>7fb8146a82f95e0af155278f406862c2</MD5OfMessageBody>
    <MD5OfMessageAttributes>295c5fa15a51aae6884d1d7c1d99ca50</MD5OfMessageAt
tributes>
  </SendMessageBatchResultEntry>
</SendMessageBatchResult>
<ResponseMetadata>
  <RequestId>calad5d0-8271-408b-8d0f-1351bf547e74</RequestId>
</ResponseMetadata>
</SendMessageBatchResponse>
```

MD5 消息摘要计算

如果您要为 Amazon SQS 消息属性计算 MD5 消息摘要，但使用的是不支持 Amazon SQS 消息属性 MD5 消息摘要的查询 API 或 AWS 软件开发工具包之一，则必须使用有关算法的以下信息计算消息属性的 MD5 消息摘要。



Note

当前 AWS SDK for Java 对于 Amazon SQS 消息属性支持 MD5 消息摘要。这可在 `MessageMD5ChecksumHandler` 类中获得。如果您使用 适用于 Java 的开发工具包，则无需使用以下信息。

为 Amazon SQS 消息属性计算 MD5 消息摘要的算法的宏观步骤是：

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（名称、类型和值）进行编码并存入缓冲区。
3. 计算整个缓冲区的消息摘要。

对单个 Amazon SQS 消息属性编码：

1. 对名称编码（名称长度 [4 字节] + 名称的 UTF-8 字节）。
2. 对类型编码（类型长度 [4 字节] + 类型的 UTF-8 字节）。
3. 对值的传输类型（字符串或二进制）编码 [1 个字节]。
 - a. 对于字符串 传输类型，编码为 1。
 - b. 对于二进制 传输类型，编码为 2。

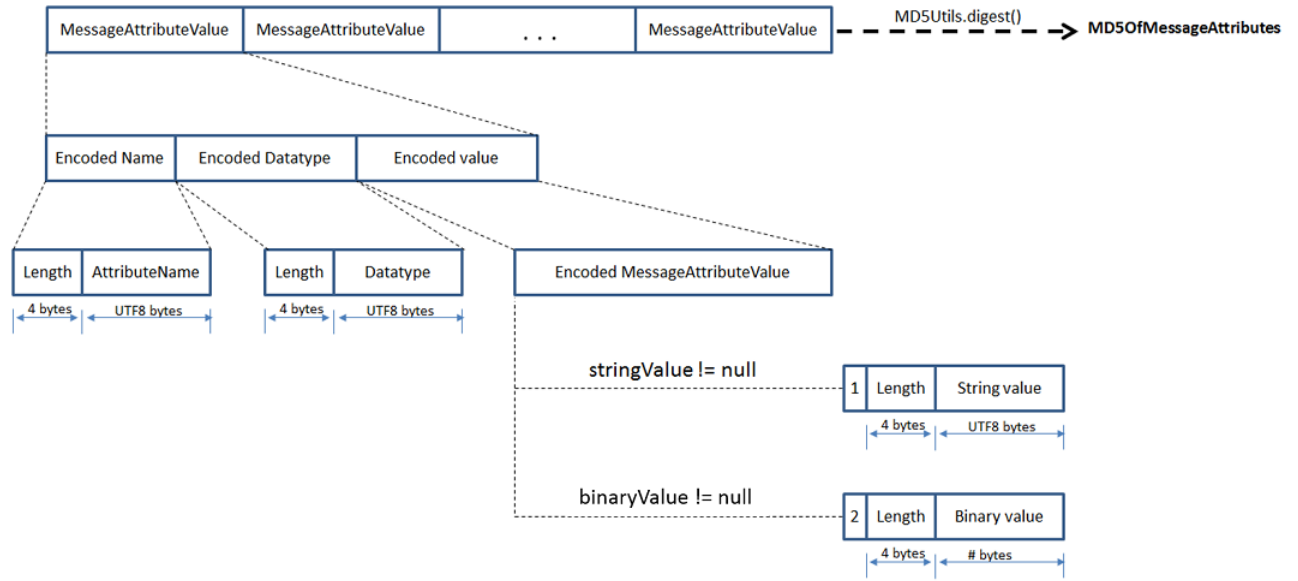


Note

字符串和数字逻辑数据类型使用字符串 传输类型。二进制逻辑数据类型使用二进制 传输类型。

4. 对属性值编码。
 - a. 对于字符串 传输类型，对属性值编码（长度 [4 字节] + 值的 UTF-8 字节）。
 - b. 对于二进制 传输类型，对属性值编码（长度 [4 字节] + 直接使用原始字节）。

下图演示单个消息属性的 MD5 消息摘要的编码：



Amazon SQS 长轮询

Abstract

介绍使用 Amazon SQS 长轮询消除空响应和假空响应以降低成本的好处。

主题

- [使用 AWS 管理控制台启用长轮询 \(p. 48\)](#)
- [使用查询 API 启用长轮询 \(p. 50\)](#)

Amazon SQS 的 API 版本 2012-11-05 提供了对长轮询的支持。

使用 Amazon SQS 来进行长轮询的一个好处是：在没有消息可返回以答复发送到 Amazon SQS 队列的 `ReceiveMessage` 请求时，可以减少空响应数量。在发送响应之前，长轮询允许 Amazon SQS 产品等到队列中的消息可用为止。因此，如果连接没有超时，则对 `ReceiveMessage` 请求的响应将至少包含一条可用的消息（如果有），并且最多包含 `ReceiveMessage` 调用中请求的最大数量的消息。

另一个好处是：有助于消除假的空响应（其中，消息在队列中可用，但不包含在响应中）。这种情况在 Amazon SQS 使用短（标准）轮询（默认行为）时会出现，此时，系统只会查询服务器的一个子集（基于加权随机分布），以查看是否有任何消息可包含在响应中。另一方面，如果启用长轮询，则 Amazon SQS 会查询所有服务器。

此外，减少空响应和假的空响应数量还有助于降低使用 Amazon SQS 的费用。

如果 `ReceiveMessage` 调用的 `WaitTimeSeconds` 参数设置为 0，则会出现短轮询。这种情况会在以下两种方式之一中出现：`ReceiveMessage` 调用将 `WaitTimeSeconds` 设置为 0，或者 `ReceiveMessage` 调用不设置 `WaitTimeSeconds` 并且队列属性 `ReceiveMessageWaitTimeSeconds` 为 0。



Note

为 `ReceiveMessage` 的 `WaitTimeSeconds` 参数设置的值（介于 1 到 20 之间）优先于为队列属性 `ReceiveMessageWaitTimeSeconds` 设置的任何值。

您可以用来在 Amazon SQS 中启用长轮询的不同 API 操作调用有三种：`ReceiveMessage`、`CreateQueue` 和 `SetQueueAttributes`。对于 `ReceiveMessage`，您可以配置 `WaitTimeSeconds` 参数；对于 `CreateQueue` 和 `SetQueueAttributes`，您可以配置 `ReceiveMessageWaitTimeSeconds` 属性。



Important

如果您决定对多个队列实施长轮询，则建议您对每个队列使用一个线程，而不是试图使用单一线程来轮询所有队列。如果对每个队列使用一个线程，则您的应用程序可以在各队列中的消息可用时处理这些消息，这与单一线程用于多个队列的情况相反，在后一种情况下，您的应用程序在等待（最长 20 秒）没有任何可用消息的队列时，可能无法处理其他队列中的可用消息。

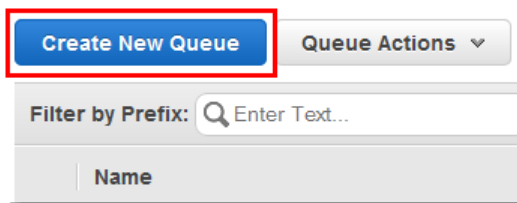
对于大多数使用长轮询的使用案例，您应将超时值设置为最大值（20 秒）。如果最大值（20 秒）不适用于您的应用程序，则您可以选择较短的长轮询超时，最短为 1 秒。如果您不是使用的 AWS SDK 访问 Amazon SQS，或者您已对 AWS SDK 进行了特别配置使其具有较短的超时，则可能需要修改您的 Amazon SQS 客户端，才能允许时间更长的请求或者使用较短的长轮询超时。

使用 AWS 管理控制台启用长轮询

通过设置大于 0 的“Receive Message Wait Time”值，您可以使用 AWS 管理控制台启用长轮询。

使用 AWS 管理控制台为新队列启用长轮询

1. 登录 AWS 管理控制台 并通过以下网址打开 Amazon SQS 控制台：<https://console.amazonaws.cn/sqs/>。
2. 单击“Create New Queue”。



3. 在“Create New Queue”对话框的“Queue Name”字段中，输入队列的名称（例如，MyQueue）。

Create New Queue [X]

Region ⓘ US West (Oregon)

Queue Name ⓘ MyQueue

Configure your new queue by setting queue attributes (optional).

Default Visibility Timeout ⓘ 30 seconds Value must be between 0 seconds and 12 hours.

Message Retention Period ⓘ 4 days Value must be between 1 minute and 14 days.

Maximum Message Size ⓘ 256 KB Value must be between 1 and 256 KB.

Delivery Delay ⓘ 0 seconds Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time ⓘ 0 seconds Value must be between 0 and 20 seconds.

Dead Letter Queue Settings

Use Redrive Policy ⓘ ☐

Dead Letter Queue ⓘ Value must be an existing queue name.

Maximum Receives ⓘ Value must be between 1 and 1000.

Cancel Create Queue

4. 为“Receive Message Wait Time”输入 1 到 20 秒的正整数值。
您可以保留剩余字段的默认值设置，或者输入新值。

Create New Queue [X]

Region ⓘ US West (Oregon)

Queue Name ⓘ MyQueue

Configure your new queue by setting queue attributes (optional).

Default Visibility Timeout ⓘ 30 seconds Value must be between 0 seconds and 12 hours.

Message Retention Period ⓘ 4 days Value must be between 1 minute and 14 days.

Maximum Message Size ⓘ 256 KB Value must be between 1 and 256 KB.

Delivery Delay ⓘ 0 seconds Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time ⓘ 10 seconds Value must be between 0 and 20 seconds.

Dead Letter Queue Settings

Use Redrive Policy ⓘ ☐

Dead Letter Queue ⓘ Value must be an existing queue name.

Maximum Receives ⓘ Value must be between 1 and 1000.

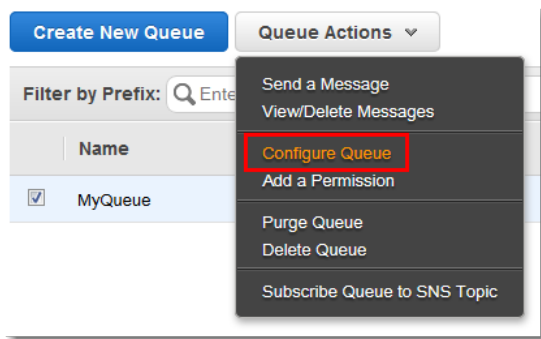
Cancel Create Queue

5. 单击“Create Queue”。

在现有队列突出显示时选择“Configure Queue”操作，您可以使用 AWS 管理控制台更改现有队列的“Receive Message Wait Time”设置。

为现有队列设置新的“Receive Message Wait Time”值

1. 在现有队列突出显示时选择“Configure Queue”操作。



2. 将 Receive Message Wait Time (接收消息等待时间) 的值更改为正整数。

A screenshot of the 'Configure MyQueue' dialog box in the AWS Management Console. The dialog has a title bar with a close button. It contains two sections: 'Queue Settings' and 'Dead Letter Queue Settings'. In the 'Queue Settings' section, there are four rows of settings, each with an information icon, a text input field, a unit dropdown, and a validation message. The 'Receive Message Wait Time' row is highlighted with a red rectangular box; its input field contains the number '5' and the unit is 'seconds'. The other settings are: 'Default Visibility Timeout' (30 seconds), 'Message Retention Period' (4 days), and 'Maximum Message Size' (256 KB). The 'Delivery Delay' setting is set to 0 seconds. The 'Dead Letter Queue Settings' section includes a 'Use Redrive Policy' checkbox (unchecked), a 'Dead Letter Queue' text field (empty), and a 'Maximum Receives' text field (empty). At the bottom right of the dialog are 'Cancel' and 'Save Changes' buttons.

3. 单击 Save Changes (保存更改)。

使用查询 API 启用长轮询

以下查询 API 示例通过调用 `ReceiveMessage` 操作并将 `WaitTimeSeconds` 参数设置为 10 秒来启用长轮询。有关更多信息，请转到《[Amazon Simple Queue Service API Reference](#)》中的“[ReceiveMessage](#)”。

您如何构造 AUTHPARAMS 取决于您如何签署您的 API 请求。有关签名版本 4 中 AUTHPARAMS 的信息，请转至 [已签名的签名版本 4 请求示例](#)。

```
http://sqs.us-east-1.amazonaws.com/123456789012/testQueue/  
?Action=ReceiveMessage  
&WaitTimeSeconds=10  
&MaxNumberOfMessages=5  
&VisibilityTimeout=15  
&AttributeName=All;  
&Version=2012-11-05  
&Expires=2013-10-25T22%3A52%3A43PST  
&AUTHPARAMS
```

以下示例显示了启用长轮询的另一种选择。其中，`SetQueueAttributes` 操作的 `ReceiveMessageWaitTimeSeconds` 属性设置为 20 秒。有关更多信息，请转到《*Amazon Simple Queue Service API Reference*》中的“[SetQueueAttributes](#)”。

```
http://sqs.us-east-1.amazonaws.com/123456789012/testQueue/  
?Action=SetQueueAttributes  
&Attribute.Name=ReceiveMessageWaitTimeSeconds  
&Attribute.Value=20  
&Version=2012-11-05  
&Expires=2013-10-25T22%3A52%3A43PST  
&AUTHPARAMS
```


Amazon SQS 延迟队列

Abstract

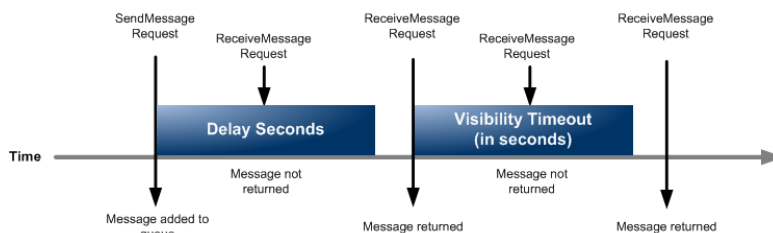
使用 Amazon SQS 延迟队列可将队列中新消息的传递操作推迟特定的秒数。

主题

- [使用 AWS 管理控制台创建延迟队列 \(p. 53\)](#)
- [使用查询 API 创建延迟队列 \(p. 55\)](#)

延迟队列允许您将队列中新消息的传递操作推迟特定的秒数。如果您创建延迟队列，则发送到该队列的任何消息在延迟期间对使用者都不可见。通过将 *DelaySeconds* 属性设置为介于 0 和 900 (15 分钟) 之间的任何值，您可以使用 *CreateQueue* 创建延迟队列。此外，通过使用 *SetQueueAttributes* 来设置现有队列的 *DelaySeconds* 属性，您还可以将该队列转变为延迟队列。

延迟队列类似于可见性超时，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。延迟队列和可见性超时之间的区别在于：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，消息只有在从队列取回后才是隐藏的。下图说明了延迟队列和可见性超时之间的关系。



Note

每个队列处于飞行状态的消息数限制为 120 000。消息的飞行状态是指该消息已由使用组件从队列接收，但尚未从队列中删除。如果您达到 120 000 的限制，将会收到一条来自 Amazon SQS 的“OverLimit”错误消息。为避免达到限制，您应该在处理消息后将其从队列删除。您还可以增加用来处理消息的队列数量。

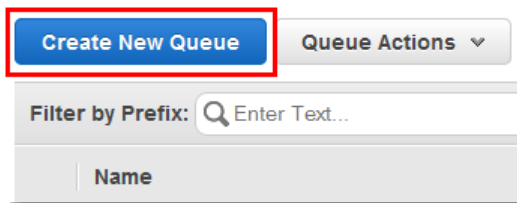
要设置各条消息（而不是整个队列）的延迟秒数，请使用消息定时器。如果您发送一条带有消息定时器的消息，则 Amazon SQS 会使用消息定时器的延迟秒数值（而不是延迟队列的延迟秒数值）。有关更多信息，请参阅 [Amazon SQS 消息定时器 \(p. 57\)](#)。

使用 AWS 管理控制台创建延迟队列

通过设置大于 0 的“Delivery Delay”值，您可以使用 AWS 管理控制台创建延迟队列。

使用 AWS 管理控制台创建延迟队列

1. 登录 AWS 管理控制台 并通过以下网址打开 Amazon SQS 控制台：<https://console.amazonaws.cn/sqs/>。
2. 单击“Create New Queue”。



3. 在 Create New Queue 对话框的 Queue Name 字段中，输入队列的名称（例如，MyQueue）。

A screenshot of the 'Create New Queue' dialog box. The dialog has a title bar with 'Create New Queue' and a close button. Below the title bar, there is a 'Region' dropdown set to 'US West (Oregon)'. The 'Queue Name' field is highlighted with a red box and contains the text 'MyQueue'. Below this, there is a section titled 'Configure your new queue by setting queue attributes (optional)'. This section contains several input fields: 'Default Visibility Timeout' (30 seconds), 'Message Retention Period' (4 days), 'Maximum Message Size' (256 KB), 'Delivery Delay' (0 seconds), and 'Receive Message Wait Time' (0 seconds). Each field has a help icon and a validation message. Below this section is a 'Dead Letter Queue Settings' section with three fields: 'Use Redrive Policy' (checkbox), 'Dead Letter Queue' (text field), and 'Maximum Receives' (text field). At the bottom right of the dialog are 'Cancel' and 'Create Queue' buttons.

4. 为“交付延迟”属性输入正整数值（例如，30）。
您可以保留剩余字段的默认值设置，或者输入新值。

Create New Queue [X]

Region ⓘ US West (Oregon)

Queue Name ⓘ MyQueue

Configure your new queue by setting queue attributes (optional).

Default Visibility Timeout ⓘ 30 seconds Value must be between 0 seconds and 12 hours.

Message Retention Period ⓘ 4 days Value must be between 1 minute and 14 days.

Maximum Message Size ⓘ 256 KB Value must be between 1 and 256 KB.

Delivery Delay ⓘ 30 seconds Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time ⓘ 0 seconds Value must be between 0 and 20 seconds.

Dead Letter Queue Settings

Use Redrive Policy ⓘ ☐

Dead Letter Queue ⓘ Value must be an existing queue name.

Maximum Receives ⓘ Value must be between 1 and 1000.

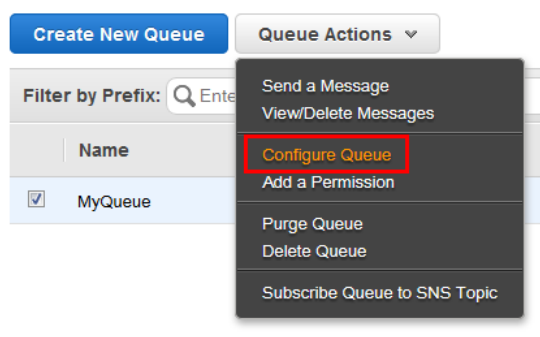
Cancel Create Queue

5. 单击“Create Queue”。

在现有队列突出显示时选择“Configure Queue”操作，您可以使用 AWS 管理控制台更改现有队列的“Delivery Delay”设置。

为现有队列设置新的传递延迟值

1. 在现有队列突出显示时选择“Configure Queue”操作。



2. 将“Delivery Delay”的值更改为正整数值。

Configure MyQueue

Queue Settings

Default Visibility Timeout *i* 30 seconds Value must be between 0 seconds and 12 hours.

Message Retention Period *i* 4 days Value must be between 1 minute and 14 days.

Maximum Message Size *i* 256 KB Value must be between 1 and 256 KB.

Delivery Delay *i* 0 seconds Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time *i* 0 seconds Value must be between 0 and 20 seconds.

Dead Letter Queue Settings

Use Redrive Policy *i* ☐

Dead Letter Queue *i* Value must be an existing queue name.

Maximum Receives *i* Value must be between 1 and 1000.

Cancel Save Changes

3. 单击 Save Changes (保存更改)。

使用查询 API 创建延迟队列

以下查询 API 示例会调用 `CreateQueue` 操作来创建延迟队列，该延迟队列会在每条消息进入队列后的前 45 秒对使用者隐藏该消息。

您如何构造 AUTHPARAMS 取决于您如何签署您的 API 请求。有关签名版本 4 中 AUTHPARAMS 的信息，请转至[已签名的签名版本 4 请求示例](#)。

```
http://sqs.us-east-1.amazonaws.com/  
?Action=CreateQueue  
&QueueName=testQueue  
&Attribute.1.Name=DelaySeconds  
&Attribute.1.Value=45  
&Version=2012-11-05  
&Expires=2015-12-20T22%3A52%3A43PST  
&AUTHPARAMS
```



Note

队列名称和队列 URL 区分大小写。

此外，通过将 `DelaySeconds` 属性从默认值 0 更改为小于或等于 900 的正整数值，您还可以将现有队列更改为延迟队列。以下示例会调用 `SetQueueAttributes` 将名为 `testQueue` 的队列的 `DelaySeconds` 属性设置为 45 秒。

```
http://sqs.us-east-1.amazonaws.com/123456789012/testQueue/  
?Action=SetQueueAttributes  
&Attribute.Name=DelaySeconds  
&Attribute.Value=45  
&Version=2012-11-05  
&Expires=2015-12-20T22%3A52%3A43PST  
&AUTHPARAMS
```

Amazon SQS 消息定时器

Abstract

使用 Amazon SQS 消息定时器指定添加到队列的消息的初始不可见时段。

主题

- [使用控制台创建消息定时器 \(p. 57\)](#)
- [使用查询 API 创建消息定时器 \(p. 61\)](#)

Amazon SQS 消息定时器允许您为要添加到队列的消息指定初始的不可见时段。例如，如果您发送一条消息并将 `DelaySeconds` 参数设置为 45，则使用者在该消息进入队列后的前 45 秒将看不到该消息。`DelaySeconds` 的默认值为 0。



Note

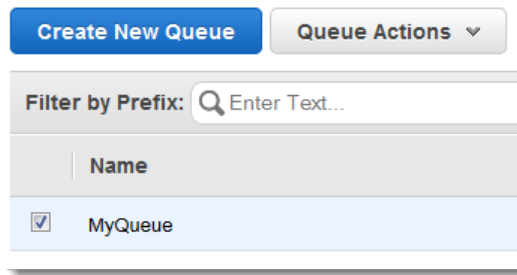
每个队列处于飞行状态的消息数限制为 120 000。消息的飞行状态是指该消息已由使用组件从队列接收，但尚未从队列中删除。如果您达到 120 000 的限制，将会收到一条来自 Amazon SQS 的“OverLimit”错误消息。为避免达到限制，您应该在处理消息后将其从队列删除。您还可以增加用来处理消息的队列数量。

要设置适用于队列中所有消息的延迟时段，请使用延迟队列。有关更多信息，请参阅 [Amazon SQS 延迟队列 \(p. 52\)](#)。各条消息的消息定时器设置会覆盖适用于整个延迟队列的任何 `DelaySeconds` 值。

使用控制台创建消息定时器

使用 AWS 管理控制台发送带有消息定时器的消息

1. 登录 AWS 管理控制台 并通过以下网址打开 Amazon SQS 控制台：<https://console.amazonaws.cn/sqs/>。
2. 选择队列。

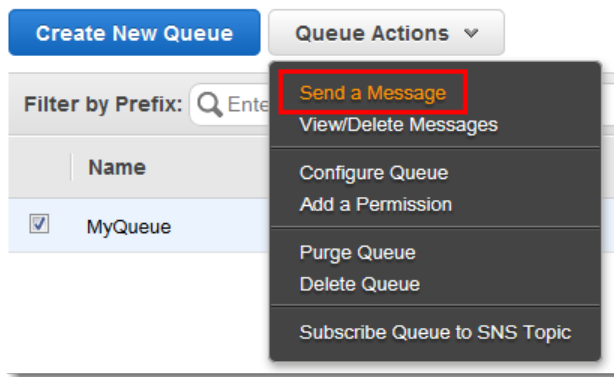


3. 从“Queue Actions”下拉列表选择“Send a Message”。



Note

仅在已选中队列的情况下，“Queue Actions”下拉列表才可用。



4. 在 Send a Message to MyQueue 对话框中，输入消息（例如，This is a test message with a message timer.）。

Send a Message to MyQueue ✕

Message Body | Message Attributes

Enter the text of a message you want to send.

This is a test message with a message timer.

☐ Delay delivery of this message by seconds ▾ (up to 15 minutes).

[Cancel](#) [Send Message](#)

5. 在 Delay delivery of this message by 文本框中，输入延迟值（例如，30）。

Send a Message to MyQueue ✕

Message Body | Message Attributes

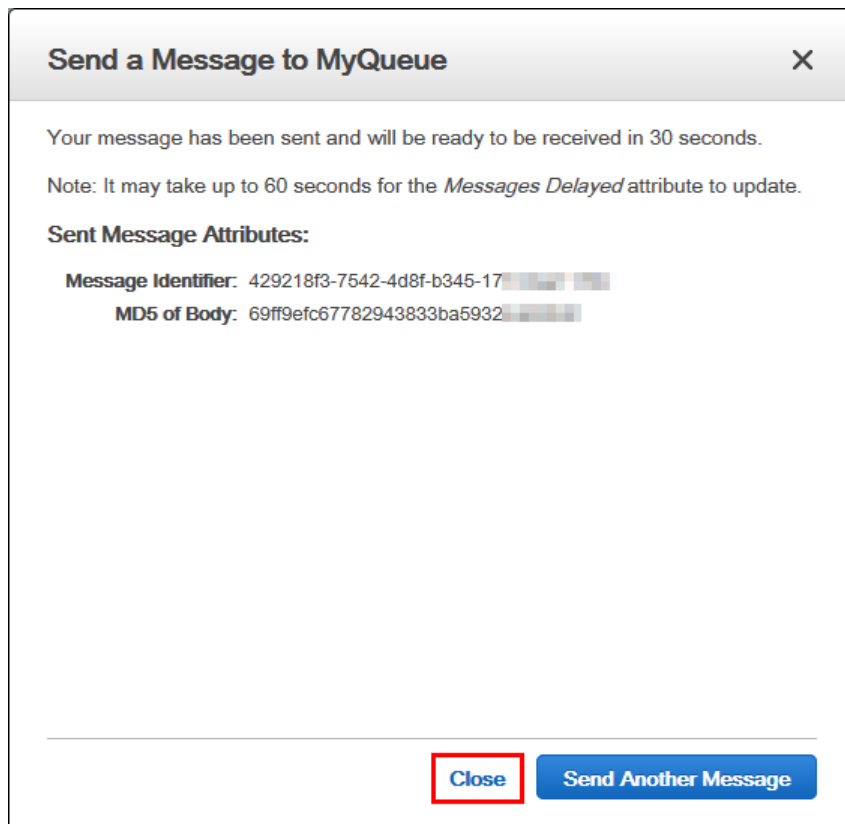
Enter the text of a message you want to send.

This is a test message with a message timer.

☒ Delay delivery of this message by seconds (up to 15 minutes).

Cancel **Send Message**

6. 单击“Send Message”。
7. 在 Send a Message to MyQueue (发送消息至 MyQueue) 确认框中，单击 Close (关闭)。



使用查询 API 创建消息定时器

以下查询 API 示例会为使用 `SendMessage` 发送的单一消息应用 45 秒的初始可见性延迟。

您如何构造 AUTHPARAMS 取决于您如何签署您的 API 请求。有关签名版本 4 中 AUTHPARAMS 的信息，请转至[已签名的签名版本 4 请求示例](#)。

```
http://sqs.us-east-1.amazonaws.com/123456789012/testQueue/  
?Action=SendMessage  
&MessageBody=This+is+a+test+message  
&Attribute.Name=DelaySeconds  
&Attribute.Value=45  
&Version=2012-11-05  
&Expires=2015-12-18T22%3A52%3A43PST  
&AUTHPARAMS
```



Note

队列名称和队列 URL 区分大小写。

此外，您最多还可以使用查询 API 的 `SendMessageBatch` 操作来发送十条带有消息定时器的消息。您可以为每条消息分配不同的 `DelaySeconds` 值，或者完全不分配任何值。如果您不为 `DelaySeconds` 设置值，则将消息添加到延迟队列时，消息仍然可能会延迟。有关延迟队列的更多信息，请参阅[Amazon SQS](#)

[延迟队列 \(p. 52\)](#)”。以下示例会使用 `SendMessageBatch` 发送三条消息：一条不带消息定时器的消息，以及两条具有不同的 `DelaySeconds` 值的消息。

```
http://sqs.us-east-1.amazonaws.com/123456789012/testQueue/  
?Action=SendMessageBatch  
&SendMessageBatchRequestEntry.1.Id=test_msg_no_message_timer  
&SendMessageBatchRequestEntry.1.MessageBody=test%20message%20body%201  
&SendMessageBatchRequestEntry.2.Id=test_msg_delay_45_seconds  
&SendMessageBatchRequestEntry.2.MessageBody=test%20message%20body%202  
&SendMessageBatchRequestEntry.2.DelaySeconds=45  
&SendMessageBatchRequestEntry.3.Id=test_msg_delay_2_minutes  
&SendMessageBatchRequestEntry.3.MessageBody=test%20message%20body%203  
&SendMessageBatchRequestEntry.3.DelaySeconds=120  
&Version=2012-11-05  
&Expires=2015-12-18T22%3A52%3A43PST  
&AUTHPARAMS
```

Amazon SQS 批处理 API 操作

Abstract

介绍您可以用于各种版本的 Amazon SQS 的批处理 API 操作。

主题

- [SendMessageBatch 的最大消息大小 \(p. 63\)](#)
- [客户端缓冲和请求批处理 \(p. 64\)](#)
- [通过水平扩展和批处理提高吞吐量 \(p. 66\)](#)

在 Amazon SQS 的 2009-02-01 API 版本中，只有一个操作 (`ReceiveMessage`) 支持批处理（例如，使用单一调用处理多条消息）。在 2011-10-01 API 版本和更高 API 版本中，Amazon SQS 添加了用于发送消息、删除消息以及更改消息可见性超时值的批处理功能。要同时最多发送十条消息，请使用 `SendMessageBatch` 操作。要使用一次 API 调用最多删除十条消息，请使用 `DeleteMessageBatch` 操作。要最多更改十条消息的可见性超时值，请使用 `ChangeMessageVisibilityBatch` 操作。

要使用新的批处理操作，您必须使用查询 API 或支持新批处理操作的软件开发工具包。请查看您的特定开发工具包的文档，以了解它是否支持新的 Amazon SQS 批处理操作。Amazon SQS 控制台当前不支持批处理 API 操作。

有关这三个批处理 API 操作的详细信息和示例，请转到《*Amazon Simple Queue Service API Reference*》：

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

SendMessageBatch 的最大消息大小

您可以使用 `SendMessageBatch` 发送一条长达 262 144 字节 (256 KB) 的消息。但是，您在 `SendMessageBatch` 的单个调用中发送的所有消息的总大小不能超过 262 144 字节 (256 KB)。

客户端缓冲和请求批处理

Abstract

启用客户端缓冲，从客户端进行的调用会先在其中缓冲，然后作为批处理请求发送到 Amazon SQS。

AWS SDK for Java (<http://www.amazonaws.cn/sdkforjava/>) 包含一个缓冲的异步客户端 `AmazonSQSBufferedAsyncClient`，用于访问 Amazon SQS。通过启用客户端缓冲（其中，从客户端发出的调用先进行缓冲，然后再作为批处理请求发送到 Amazon SQS），此新客户端允许您更轻松地对请求进行批处理。

客户端缓冲最多允许缓冲 10 个请求并将这些请求作为批处理请求发送，而不是分别发送各个请求。因此，随着发送到 Amazon SQS 的请求数量的减少，使用该服务的费用会降低。`AmazonSQSBufferedAsyncClient` 会缓冲同步和异步调用。此外，成批请求以及对长轮询的支持还可以帮助提高吞吐量（每秒传输的消息数量）。有关更多信息，请参阅 [Amazon SQS 长轮询 \(p. 47\)](#) 和 [通过水平扩展和批处理提高吞吐量 \(p. 66\)](#)。

从异步客户端 `AmazonSQSAsyncClient` 迁移到缓冲的异步客户端 `AmazonSQSBufferedAsyncClient` 只需要对现有的代码进行最少的更改。这是因为 `AmazonSQSBufferedAsyncClient` 会实施与 `AmazonSQSAsyncClient` 相同的接口。

AmazonSQSBufferedAsyncClient 入门

开始使用此部分中的示例代码之前，您必须先安装 AWS SDK for Java 并设置 AWS 证书。有关说明，请参阅 [AWS SDK for Java 入门](#)。

以下代码示例显示了如何基于 `AmazonSQSAsyncClient` 创建新的 `AmazonSQSBufferedAsyncClient`。

```
// Create the basic Amazon SQS async client
AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

创建了新的 `AmazonSQSBufferedAsyncClient` 后，您可以像调用 `AmazonSQSAsyncClient` 那样调用该客户端，如以下代码示例所示。

```
CreateQueueRequest createRequest = new CreateQueueRequest().withQueueName("MyTestQueue");

CreateQueueResult res = bufferedSqs.createQueue(createRequest);

SendMessageRequest request = new SendMessageRequest();
String body = "test message_" + System.currentTimeMillis();
request.setRequestBody( body );
request.setQueueUrl(res.getQueueUrl());

SendMessageResult sendResult = bufferedSqs.sendMessage(request);

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

高级配置

AmazonSQSBufferedAsyncClient 预配置了适用于大多数使用案例的设置。如果您想自己配置它，则可以使用 `QueueBufferConfig` 类进行配置。只需使用所需的设置创建一个 `QueueBufferConfig` 实例，并将该实例提供给 `AmazonSQSBufferedAsyncClient` 构造函数，如下示例代码所示。

```
// Create the basic Amazon SQS async client
AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

QueueBufferConfig config = new QueueBufferConfig()
    .withMaxInflightReceiveBatches(5)
    .withMaxDoneReceiveBatches(15);

// Create the buffered client
AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync, config);
```

可用于配置 `QueueBufferConfig` 的参数如下：

- `longPoll` – 如果此参数设置为 `true`，则 `AmazonBufferedAsyncClient` 会在取回消息时尝试使用长轮询。默认值为 `true`。
- `longPollWaitTimeoutSeconds` – 在返回空接收结果前，接收消息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）。如果禁用长轮询，则此设置不起作用。此设置的默认值为 20 秒。
- `maxBatchOpenMs` – 传出调用等待其他同类调用进行批处理的最长时间（以毫秒为单位）。该设置越长，则执行等量工作所需的批处理就越少。当然，该设置越长，则批处理中的首次调用必须等待的时间就越长。如果此参数设置为零，则提交的请求不会等待其他请求，从而有效地禁用了批处理。此设置的默认值为 200 毫秒。
- `maxBatchSize` – 将在单一批处理请求中一起进行批处理的最大消息数量。该设置越大，则执行等量请求所需的批处理就越少。此设置的默认值为每个批处理 10 个请求，这也是 Amazon SQS 当前允许的最大批处理大小。
- `maxBatchSizeBytes` – 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）。默认值为 256KB，这也是 Amazon SQS 当前允许的最大消息和批处理大小。
- `maxDoneReceiveBatches` – `AmazonBufferedAsyncClient` 预取并存储在客户端上的最大接收批处理数量。该设置越大，则不必调用 Amazon SQS 服务器即可满足的接收请求就越多。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，这意味着消息的可见性超时将会过期。如果此参数设置为零，则消息的所有预取操作都会禁用，并且系统仅按需取回消息。默认值为 10 个批处理。
- `maxInflightOutboundBatches` – 可以同时处理的最大活跃出站批处理数量。该设置越大，则可以发送出站批处理的速度就越快（受限与其他限制，例如 CPU 或带宽）。该设置越大，则 `AmazonSQSBufferedAsyncClient` 占用的线程就越多。默认值为 5 个批处理。
- `maxInflightReceiveBatches` – 可以同时处理的最大活跃接收批处理数量。该设置越大，则可以接收的消息就越多（受限与其他限制，例如 CPU 或带宽、命中）。尽管如此，但是该设置越大，则 `AmazonSQSBufferedAsyncClient` 占用的线程就越多。如果此参数设置为 0，则消息的所有预取操作都会禁用，并且系统仅按需取回消息。默认值为 10 个批处理。

- `visibilityTimeoutSeconds` – 如果此参数设置为非零正值，则此可见性超时会覆盖从中取回消息的队列上设置的可见性超时。零秒的可见性超时不受支持。默认值为 -1，这意味着使用默认队列设置。

通过水平扩展和批处理提高吞吐量

作者：Marc Levy，2012 年 7 月

Abstract

使用您可以自己尝试的简单示例，通过水平扩展和批处理提高 Amazon SQS 中的吞吐量。

Amazon SQS 队列可以传递极高的吞吐量（每秒数千条消息）。达到此吞吐量的关键是水平扩展消息的创建者和使用者。此外，您还可以使用 Amazon SQS API 中的批处理操作来一次发送、接收或删除多达 10 条消息。在结合水平扩展后，批处理可以达到给定的吞吐量，而线程、连接和请求的数量却比单独的消息请求所需的数量更少。此外，由于 Amazon SQS 按请求（而不是消息）收费，因此，批处理还可以大幅降低费用。

本附录详细讨论了水平扩展和批处理，然后介绍了您自己可以尝试的简单示例。此外，本附录还简要讨论了您可以通过使用 CloudWatch 来监控的 Amazon SQS 吞吐量指标。

水平扩展

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从单一线程达到的吞吐量。例如，如果从基于 Amazon Elastic Compute Cloud (Amazon EC2) 的客户端到同一地区内的 Amazon SQS 的延迟时间平均为大约 20ms，则通过单一连接从单一线程达到的最大吞吐量平均为每秒 50 次操作。

水平扩展意味着增加消息创建者（提出 `SendMessage` 请求）和使用者（提出 `ReceiveMessage` 和 `DeleteMessage` 请求）的数量，以提高整个队列的吞吐量。您可以通过增加客户端上的线程数量以及/或者添加客户端来进行水平扩展。添加更多客户端后，基本上应该能实现队列吞吐量的线性增长。例如，如果将客户端数量加倍，则您将获得两倍的吞吐量。



Important

进行水平扩展时，您需要确保您使用的 Amazon SQS 具有足够的连接或线程，以支持将要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 的 `AmazonSQSClient` 类的实例最多会维持 50 个与 Amazon SQS 的连接。要创建其他并发创建者和使用者，您需要调整该限制。例如，在 [AWS SDK for Java](#) 中，您可以使用下面一行代码来调整 `AmazonSQSClient` 对象上允许的最大创建者和使用者线程数量：

```
AmazonSQS sqsClient = new AmazonSQSClient(credentials,
                                             new ClientConfiguration().withMaxConnections(producerCount + consumerCount));
```

此外，对于适用于 Java 的开发工具包 异步客户端 `AmazonSQSAsyncClient`，您还需要确保有足够的线程可用。有关更多信息，请参阅您所使用的开发工具包库的文档。

批处理

Amazon SQS API 中的批处理操作（`SendMessageBatch` 和 `DeleteMessageBatch`）于 2011 年 10 月 (WSDL 2011-10-01) 推出，这些操作可一次处理最多十条消息，进一步优化吞吐量。由于 `ReceiveMessageBatch` 一次可以处理十条消息，因此，API 中没有操作。

批处理的基本理念是：在与服务的每次往返操作中执行更多工作（例如，使用单一 `SendMessageBatch` 请求发送多条消息），以及在批处理请求的多条消息中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。Amazon SQS 按请求收费，因此，如果由更少的请求来处理等量的消息，则费用可以大幅降低。此外，更少的线程和连接可以降低客户端资源使用率，并且可以通过使用更小或更少的主机执行相同的工作来降低客户端费用。

不过，批处理确实会让应用程序变得有点复杂。例如，应用程序必须先积累消息，然后才能发送消息，并且有时候必须花费较长的时间来等待响应，但是批处理在以下情况下可能很有效：

- 您的应用程序正在短时间内生成大量消息，因此，延迟时间决不会很长。
- 消息使用者从队列中自行获取消息，这与典型的消息创建者相反，后者需要发送消息来响应它们无法控制的事件。



Important

即使批处理中单独的消息失败了，批处理请求（`SendMessageBatch` 或 `DeleteMessageBatch`）也可能会成功。在提出批处理请求后，您应始终检查各条消息是否失败了，并在必要时重试。

示例

本节所示的示例会实施简单的创建者-使用者模式。完整示例可在以下位置作为免费下载获取：<https://s3.amazonaws.com/cloudformation-examples/sqs-producer-consumer-sample.tar>。本节后面的部分描述了各个模板部署的资源。

`/tmp/sqs-producer-consumer-sample/src` 中的配置实例提供了示例代码。配置运行的命令行位于 `/tmp/sqs-producer-consumer-sample/command.log` 中。

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。示例包括提出单一操作请求的创建者和使用者，以及提出批处理请求的其他创建者和使用者。

在程序中，每个创建者线程都会发送消息，直到主线程停止创建者线程为止。`producedCount` 对象会跟踪所有创建者线程生成的消息数量。处理错误的操作很简单：如果存在错误，则程序会退出 `run()` 方法。默认情况下，对于因暂时性错误而失败的请求，`AmazonSQSClient` 会重试三次，因此，此类错误很少会出现。必要时，可以配置重试计数，以减少系统引发的异常数量。消息创建者上的 `run()` 方法实施如下：

```
try {
    while (!stop.get()) {
        sqsClient.sendMessage(new SendMessageRequest(queueUrl, theMessage));
        producedCount.incrementAndGet();
    }
} catch (AmazonClientException e) {
    // By default AmazonSQSClient retries calls 3 times before failing,
    // so when this rare condition occurs, simply stop.
    log.error("Producer: " + e.getMessage());
    System.exit(1);
}
```

批处理创建者几乎相同。一个显著的区别是需要重试失败的各个批处理条目：

```
SendMessageBatchResult batchResult = sqsClient.sendMessageBatch(batchRequest);

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));
}
```

使用者的 run() 方法如下：

```
while (!stop.get()) {
    result = sqsClient.receiveMessage(new ReceiveMessageRequest(queueUrl));

    if (!result.getMessages().isEmpty()) {
        m = result.getMessages().get(0);
        sqsClient.deleteMessage(new DeleteMessageRequest(queueUrl,
                                                            m.getReceiptHandle()));

        consumedCount.incrementAndGet();
    }
}
```

每个使用者线程都会接收和删除消息，直到主线程停止使用者线程为止。consumedCount 对象会跟踪所有使用者线程使用的消息数量，并且计数会定期记录。批处理使用者与此相似，不同之处是一次最多接收十条消息，并且它使用的是 [DeleteMessageBatch](#)，而不是 [DeleteMessage](#)。

运行示例

您可以使用提供的 AWS CloudFormation 模板以下列三种不同配置运行示例代码：单一主机和单一操作请求、两台主机和单一操作请求、一台主机和批处理请求。



Important

完整示例位于单一 .tar 文件中。本节后面的部分描述了各个模板部署的资源。

/tmp/sqs-producer-consumer-sample/src 中的配置实例提供了示例代码。配置运行的命令行位于 /tmp/sqs-producer-consumer-sample/command.log 中。

示例设置了默认持续时间（20 分钟），以提供容量指标的三个或四个 5 分钟 CloudWatch 数据点。每次运行的 Amazon EC2 费用将是 m1.large 实例费用。Amazon SQS 费用基于每个示例的 API 调用率而发生变化，并且该调用率应介于大约 38 000 次 API 调用/分（针对批处理示例）和 380 000 次 API 调用/分（针对 2 台主机的单一 API 示例）之间。例如，在单一主机上运行一次单一 API 示例应花费大约 1 实例小时的 m1.large（大型标准按需实例，自 2012 年 7 月起为 0.32 USD），而对于持续时间为默认的 20 分钟的 Amazon SQS 操作，则应花费大约 20 分钟 x 190 000 次 API 调用/分 x 1 USD/1 000 000 次 API 调用 = 3.80 USD（自 2012 年 7 月起的定价，请检查当前定价）。

如果您要在除 US East (N. Virginia) 区域以外的区域部署 AWS CloudFormation 堆栈，请在 AWS CloudFormation 控制台的“区域”框中单击所需的区域。

运行示例

1. 单击下面与您要启动的堆栈相对应的链接：

- [单一操作 API，一台主机](#)：SQS_Sample_Base_Producer_Consumer.template 示例模板使用 Amazon SQS API 请求的单一操作形式：SendMessage、ReceiveMessage 和 DeleteMessage。单一 m1.large Amazon EC2 实例会启动 16 个创建者线程和 32 个使用者线程。

要查看该模板，请转到 https://s3.amazonaws.com/cloudformation-templates-us-east-1/SQS_Sample_Base_Producer_Consumer.template

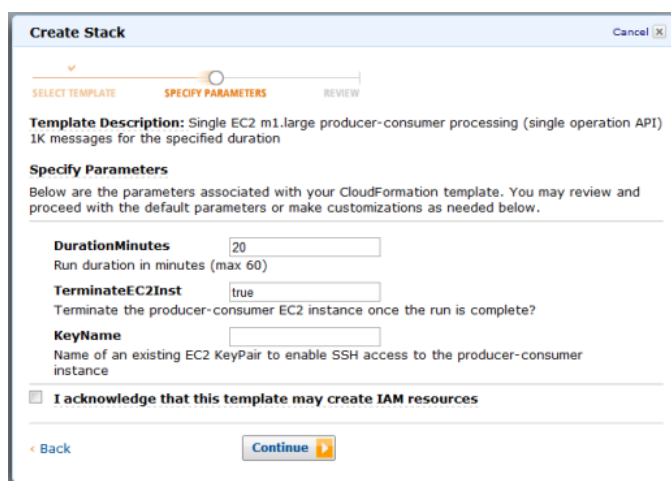
- **单一操作 API，两台主机**：SQS_Sample_Base_Producer_Consumer_x2.template 示例模板使用 Amazon SQS API 请求的单一操作形式，但它使用的是两个（而不是一个）m1.large Amazon EC2 实例，每个实例都有 16 个创建者线程和 32 个使用者线程，总共有 32 个创建者和 64 个使用者。这说明了 Amazon SQS 在吞吐量与更大的创建者和使用者数量按比例提高时的弹性。

要查看该模板，请转到 https://s3.amazonaws.com/cloudformation-templates-us-east-1/SQS_Sample_Base_Producer_Consumer_x2.template

- **批处理 API，一台主机**：SQS_Sample_Batch_Producer_Consumer.template 示例模板在具有 12 个创建者线程和 20 个使用者线程的单一 m1.large Amazon EC2 实例上使用 Amazon SQS API 请求的批处理形式。

要查看该模板，请转到 https://s3.amazonaws.com/cloudformation-templates-us-east-1/SQS_Sample_Batch_Producer_Consumer.template

2. 如果系统提示您，请登录到 AWS 管理控制台。
3. 在“Create Stack”向导的“Select Template”页面上，单击“Continue”。
4. 在“Specify Parameters”页面上，指定程序应运行的时长以及您是否希望在运行完成后自动终止 Amazon EC2 实例，并且提供 Amazon EC2 密钥对，以便您可以访问运行该示例的实例。示例如下：



5. 选中“I acknowledge that this template may create IAM resources”复选框。所有模板都会创建 AWS Identity and Access Management (IAM) 用户，以便创建者-使用者程序可以访问队列。
6. 根据需要设定好所有设置后，单击“Continue”。
7. 在“Review”页面上，检查设置。如果这些设置与您期望的设置相符，请单击“Continue”。否则，请单击“Back”并进行必要的更改。
8. 在向导的最后页面上，单击“Close”。堆栈部署可能需要花费几分钟时间。

要了解堆栈部署的进度，请在 AWS CloudFormation 控制台中单击示例堆栈。在下窗格中，单击“Events”选项卡。创建堆栈后，不到 5 分钟时间，示例就会开始运行。示例开始运行后，您可以在 Amazon SQS 控制台中查看队列。

要监控队列活动，您可以执行以下操作：

- 访问客户端实例，然后打开其针对迄今为止生成和使用的消息计数的输出日志文件 (/tmp/sqs-producer-consumer-sample/output.log)。此计数每秒更新一次。
- 在 [Amazon SQS 控制台](#) 中，观察“Message Available”和“Messages in Flight”数字的变化。

此外，在启动队列后最多延迟 15 分钟，您就可以在 CloudWatch 中监控队列，如本主题后面所述。

虽然模板和示例有安全措施来防止过度使用资源，但是在运行完示例后，您最好删除您的 AWS CloudFormation 堆栈。要执行此操作，请在 [Amazon SQS 控制台](#) 中单击要删除的堆栈，然后单击“Delete Stack”。删除所有资源后，CloudWatch 指标将全部下降为零。

监控示例运行的容量指标

Amazon SQS 会针对发送、接收和删除的消息自动生成容量指标。您可以通过 [CloudWatch 控制台](#) 访问这些指标和其他指标。队列启动后，这些指标最多可能需要花费 15 分钟才可用。要管理搜索结果集，请单击“Search”，然后选中与您要监控的队列和指标相对应的复选框。

以下是针对这三个示例连续运行的 NumberOfMessagesSent 指标。您的结果可能会有所不同，但这些结果在本质上应该是相似的：



- NumberOfMessagesReceived 和 NumberOfMessagesDeleted 指标显示了相同的模式，但在此图中省略了这两个指标，以减少凌乱。
- 第一个示例（单一 m1.large 上的单一操作 API）在 5 分钟内传递了大约 210 000 条消息（每秒传递了大约 700 条消息），并且接收和删除操作的吞吐量相同。
- 第二个示例（两个 m1.large 实例上的单一操作 API）传递了大约相当于该吞吐量两倍的吞吐量：在 5 分钟内传递了大约 440 000 条消息（每秒传递了大约 1 450 条消息），并且接收和删除操作的吞吐量相同。
- 最后一个示例（单一 m1.large 上的批处理 API）在 5 分钟内传递了 800 000 多条消息（每秒传递了大约 2 500 条消息），并且接收和删除的消息的吞吐量相同。如果批处理大小为 10，则系统会使用少得多的请求（并因此以更低费用）来处理这些消息。

提出 API 请求

Abstract

对 Amazon SQS 发出请求 (使用)。这些主题会帮助您熟悉界面之间的基本区别、请求的组件、对请求进行身份验证的方法，以及响应的内容

主题

- [终端节点 \(p. 72\)](#)
- [提出查询请求 \(p. 73\)](#)
- [提出 SOAP 请求 \(p. 75\)](#)
- [请求身份验证 \(p. 76\)](#)
- [响应 \(p. 84\)](#)
- [共享队列 \(p. 86\)](#)
- [编程语言 \(p. 89\)](#)

本节描述了如何向 Amazon SQS 提出请求。这些主题会帮助您熟悉界面之间的基本区别、请求的组件、对请求进行身份验证的方法，以及响应的内容。

此外，我们还提供了开发工具包，这些开发工具包可让您从您的首选编程语言访问 Amazon SQS。这些开发工具包包含自动处理诸如以下任务的功能：

- 使用密码对服务请求签名
- 重试请求
- 处理错误响应

有关可用的开发工具包的列表，请转到[“适用于 Amazon Web Services 的工具”](#)

终端节点

Abstract

介绍终端节点。每个 Amazon SQS 终端节点完全独立。

有关此产品地区和终端节点的信息，请访问《Amazon Web Services General Reference》中的“[地区和终端节点](#)”部分。

例如，要在欧洲创建队列，您可能会生成类似以下请求的查询请求：

您如何构造 AUTHPARAMS 取决于您如何签署您的 API 请求。有关签名版本 4 中 AUTHPARAMS 的信息，请转至[已签名的签名版本 4 请求示例](#)。

```
http://sqs.eu-west-1.amazonaws.com/  
?Action=CreateQueue  
&DefaultVisibilityTimeout=40  
&QueueName=testQueue  
&Version=2012-11-05  
&AUTHPARAMS
```

每个 Amazon SQS 终端节点完全独立。例如，如果您有两个名为“MyQueue”的队列，一个在 sqs.us-east-1.amazonaws.com 中，一个在 sqs.eu-west-1.amazonaws.com 中，则它们是完全独立的，不共享任何数据。



Note

队列名称和队列 URL 区分大小写。

提出查询请求

Abstract

使用 GET 或 POST 方法发出调用服务操作的查询请求。

主题

- [GET 请求的结构 \(p. 73\)](#)
- [POST 请求的结构 \(p. 74\)](#)
- [相关主题 \(p. 75\)](#)

Amazon SQS 支持用于调用服务操作的查询请求。查询请求是使用 GET 或 POST 方法的简单 HTTP 或 HTTPS 请求。查询请求必须包含 *Action* 参数，以指示要执行的操作。响应是符合某种模式的 XML 文档。

GET 请求的结构

本指南将 Amazon SQS GET 请求显示为可直接在浏览器中使用的 URL。URL 由以下要素构成：

- 终端节点 – 请求对其执行操作的资源（对于 Amazon SQS 的情况，终端节点是队列）
- 操作 – 您希望对终端节点执行的操作；例如：发送消息
- 参数 – 任何请求参数

以下是向 Amazon SQS 队列发送消息的示例 GET 请求。

您如何构造 AUTHPARAMS 取决于您如何签署您的 API 请求。有关签名版本 4 中 AUTHPARAMS 的信息，请转至 [已签名的签名版本 4 请求示例](#)。

```
http://sqs.us-east-1.amazonaws.com/123456789012/queue1?Action=SendMessage&MessageBody=Your%20Message%20Text&Version=2012-11-05&AUTHPARAMS
```



Important

由于 GET 请求是 URL，因此，您必须对参数值进行 URL 编码。例如，在前面的示例请求中，*MessageBody* 参数的值实际为 *Your Message Text*。但是，URL 中不允许使用空格，空格在 URL 中编码为“%20”。示例的其余部分没有进行 URL 编码，更易于您阅读。



Note

队列名称和队列 URL 区分大小写。

为了使 GET 示例更易于阅读，本指南采用了以下解析格式来显示这些示例。

```
http://sqs.us-east-1.amazonaws.com/123456789012/queue1
?Action=SendMessage
&MessageBody=Your%20Message%20Text
&Version=2012-11-05
&Expires=2011-10-15T12:00:00Z
&AUTHPARAMS
```



Note

在本指南显示的示例查询请求中，我们使用了假的 AWS 访问密钥 ID 和假的签名，各项均附加了 `EXAMPLE`。我们这样做是为了表明，您不应根据示例中显示的请求参数来指望示例中的签名准确无误。针对创建查询请求签名的说明中有一个例外。那里的示例显示了基于我们指定的特定 AWS 访问密钥 ID 以及示例中请求参数的真实签名（有关更多信息，请参阅[“查询请求身份验证 \(p. 83\)”](#)）。

在 Amazon SQS 中，除 `MessageBody` 以外的所有参数始终具有不带空格的值。您为 `SendMessage` 请求中的 `MessageBody` 提供的值可以有空格。在本指南中，带有包含空格的 `MessageBody` 的任何示例 `SendMessage` 查询请求在显示时，其空格在 URL 中都进行了编码（编码为 `%20`）。为了清楚起见，URL 的其余部分没有使用 URL 编码格式显示。

第一行代表请求的终端节点。这是该请求所影响的资源。前面的示例对队列执行操作，因此，请求的终端节点是队列的标识符，称为 *队列 URL*。有关队列 URL 的更多详细信息，请参阅[“队列 URL \(p. 5\)”](#)。

终端节点后是一个问号 (`?`)，将终端节点与参数隔开。参数之间用 `&` 号隔开。

`Action` 参数表示要执行的操作（有关操作的列表，请参阅《Amazon SQS API Reference》中的[“API 操作”](#)）。有关所有查询请求中的其他通用参数的列表，请参阅 Amazon SQS API 参考中的[通用参数](#)。

POST 请求的结构

Amazon SQS 也接受 POST 请求。使用 POST 请求，您可以在 HTTP 请求正文中以表单发送查询参数，如下步骤所述。

您如何构造 `AUTHPARAMS` 取决于您如何签署您的 API 请求。有关签名版本 4 中 `AUTHPARAMS` 的信息，请转至[已签名的签名版本 4 请求示例](#)。

创建 POST 请求

1. 将查询参数名称和值汇编到表单中。

这意味着，您可以像对待 GET 请求那样，将参数和值放在一起（使用 `&` 号分隔每个名称-值对）。以下示例显示了带有换行符的 `SendMessage` 请求；我们在本指南中使用这些换行符是为了使信息更易于阅读。

```
Action=SendMessage
&MessageBody=Your Message Text
&Version=2012-11-05
&Expires=2011-10-15T12:00:00Z
&AUTHPARAMS
```

2. 根据 HTML 规范的“表单提交”一节，对表单进行表单 URL 编码（有关更多信息，请转到 http://www.w3.org/MarkUp/html-spec/html-spec_toc.html#SEC8.2.1）。

```
Action=SendMessage
&MessageBody=Your+Message+Text
&Version=2012-11-05
&Expires=2011-10-15T12%3A00%3A00Z
&AUTHPARAMS
```

3. 向表单添加请求签名（有关更多信息，请参阅[“查询请求身份验证 \(p. 83\)”](#)）。

```
Action=SendMessage
&MessageBody=Your+Message+Text
&Version=2012-11-05
&Expires=2011-10-15T12%3A00%3A00Z
&AUTHPARAMS
```

4. 提供结果表单作为 POST 请求的正文。
5. 包含 Content-Type HTTP 标头，并将值设置为 application/x-www-form-urlencoded。

以下示例显示了最终的 POST 请求。

```
POST /queue1 HTTP/1.1
Host: sqs.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded

Action=SendMessage
&MessageBody=Your+Message+Text
&Version=2012-11-05
&Expires=2011-10-15T12%3A00%3A00Z
&AUTHPARAMS
```

除了 Content-Type 以外，Amazon SQS 不要求在请求中使用其他 HTTP 标头。您提供的身份验证签名是您在发送 GET 请求时提供的相同签名（有关签名的信息，请参阅[“查询请求身份验证 \(p. 83\)”](#)）。



Note

根据您的 HTTP 客户端所使用的 HTTP 版本的需要，该客户端通常会向 HTTP 请求添加其他项目。我们没有在本指南的示例中包含这些额外的项目。

相关主题

- [查询请求身份验证 \(p. 83\)](#)
- [响应 \(p. 84\)](#)

提出 SOAP 请求

Abstract

Amazon SQS 不再支持发出 SOAP 请求。



Important

自 2011 年 8 月 8 日起，Amazon SQS 不再支持 SOAP 请求。

请求身份验证

Abstract

介绍 Amazon SQS 如何对请求进行身份验证，AWS 账户和 ID 如何支持身份验证以及如何创建 HMAC-SHA1 签名。

Topics

- [什么是身份验证？ \(p. 76\)](#)
- [您的 AWS 账户 \(p. 78\)](#)
- [您的访问密钥 \(p. 78\)](#)
- [HMAC-SHA 签名 \(p. 78\)](#)
- [查询请求身份验证 \(p. 83\)](#)

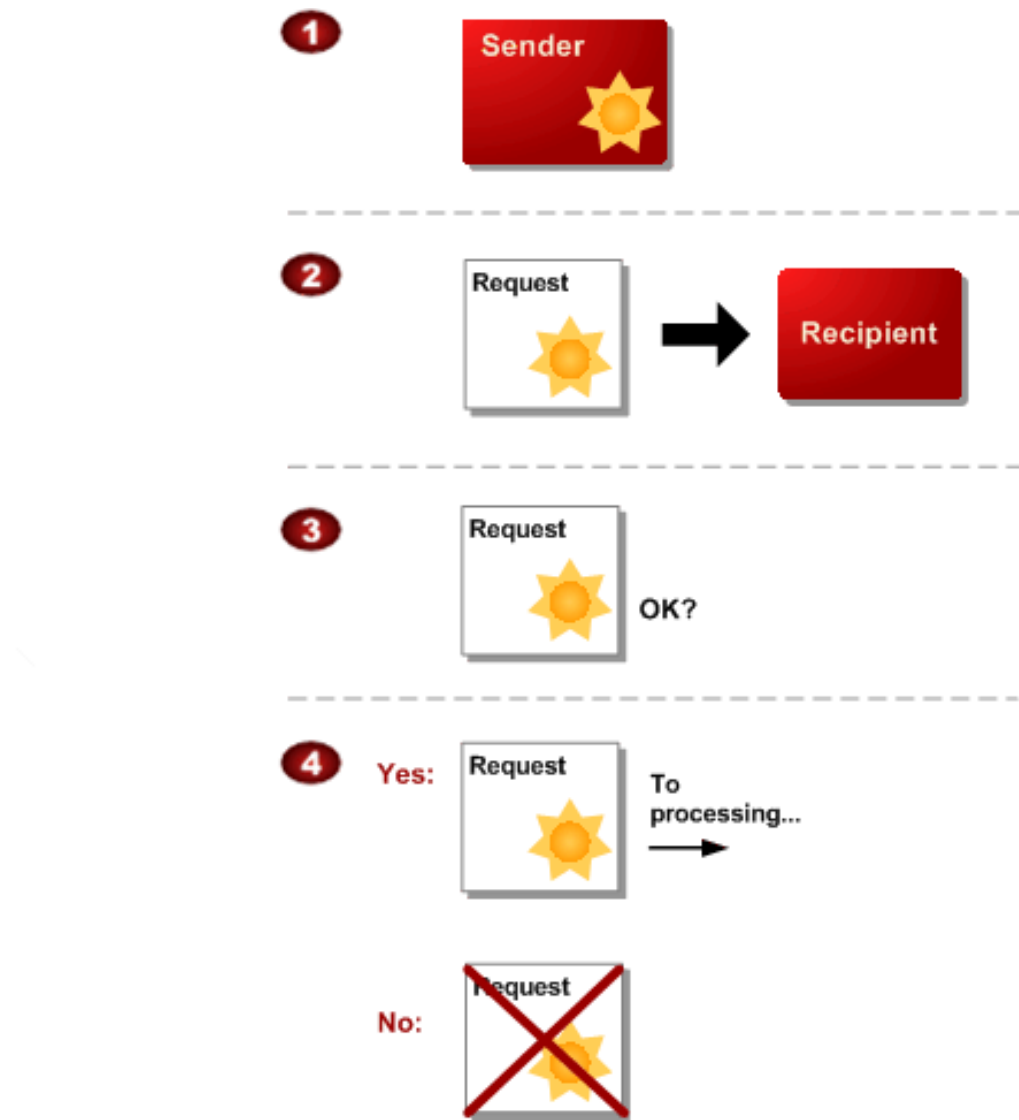
本部分中的主题介绍了 Amazon SQS 如何对您的请求进行身份验证。在本节中，您可以了解身份验证的基础知识、您的 AWS 账户和访问密钥如何用于支持身份验证，以及如何创建 HMAC-SHA1 签名。此外，本节还介绍了查询请求的请求身份验证要求。

什么是身份验证？

Abstract

将身份验证定义为一个用于识别和验证请求发送者的过程，并图示简化的身份验证过程。

身份验证是一个用于识别并验证发送请求的用户的过程。下图显示了简化版的身份验证过程。



身份验证的一般过程

1	发件人获取必要的证书。
2	发件人向收件人发送带有证书的请求。
3	收件人使用证书来验证发件人是否确实发送了该请求。
4	如果是，则收件人会处理该请求。否则，收件人会拒绝该请求并相应地作出响应。

在身份验证过程中，AWS 会同时验证发件人的标识以及发件人是否已注册使用 AWS 提供的服务。如果任一测试失败，则该请求不会得到进一步处理。

有关身份验证的进一步讨论，请转到 techencyclopedia.com 的 [身份验证](#) 条目。有关与身份验证相关的常用行业术语的定义，请转到“[RSA 实验室术语表](#)”。

后续几部分介绍了 Amazon SQS 如何实施身份验证来保护您的数据。

您的 AWS 账户

Abstract

创建 AWS 账户以访问 AWS 提供的任何 Web 服务。

您必须首先在 <http://www.amazonaws.cn> 上创建 AWS 账户，然后才能访问 AWS 提供的任何 Web 服务。AWS 账户只是可以使用 AWS 产品的 Amazon.com 账户；您可以在创建 AWS 账户时使用现有的 Amazon.com 账户登录名和密码。

或者，您可以通过使用新的登录名和密码来创建一个启用了 AWS 的新 Amazon.com 账户。您提供作为账户登录名的电子邮件地址必须是有效的。系统会要求您提供信用卡或其他付款方式，以支付您使用的任何 AWS 产品的费用。

您可以从您的 AWS 账户查看 AWS 账户活动并查看使用率报告。

更多信息，请参阅 Amazon Simple Queue Service 入门指南 中的 [创建 AWS 账户](#)。

相关主题

- [您的访问密钥 \(p. 78\)](#)

您的访问密钥

Abstract

介绍用于 API 访问的访问密钥 ID 和秘密密钥。

For API access, you need an access key ID and secret access key. Use IAM user access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account. For more information about creating access keys, see [How Do I Get Security Credentials?](#) in the *AWS General Reference*.

相关主题

- [HMAC-SHA 签名 \(p. 78\)](#)
- [查询请求身份验证 \(p. 83\)](#)

HMAC-SHA 签名

Abstract

介绍 Amazon SQS 如何使用 HMAC-SHA 签名对查询请求进行身份验证。

Topics

- [所需的身份验证信息 \(p. 79\)](#)
- [基本身份验证过程 \(p. 80\)](#)
- [关于待签字符串 \(p. 81\)](#)
- [关于时间戳 \(p. 81\)](#)
- [Base64 编码的 Java 示例代码 \(p. 82\)](#)
- [计算 HMAC-SHA1 签名的 Java 示例代码 \(p. 82\)](#)

本部分中的主题介绍了 Amazon SQS 如何使用 HMAC-SHA 签名对查询请求进行身份验证。

所需的身份验证信息

Abstract

列出您必须提供才能对请求进行身份验证的必需信息。

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

- AWS 访问密钥 ID – 您的 AWS 账户由您的访问密钥 ID 标识，AWS 使用该 ID 来查找您的私有访问密钥。
- 签名 – 每个请求都必须包含一个有效的 HMAC-SHA 请求签名，否则请求会被拒绝。
您通过使用您的私有访问密钥（该密钥是只有您和 AWS 才知道的共享秘密）来计算请求签名。
- 日期 – 每个请求必须包含请求的时间戳。您可以为请求提供过期日期和时间来代替时间戳或者作为时间戳的补充。

相关主题

- [您的访问密钥 \(p. 78\)](#)

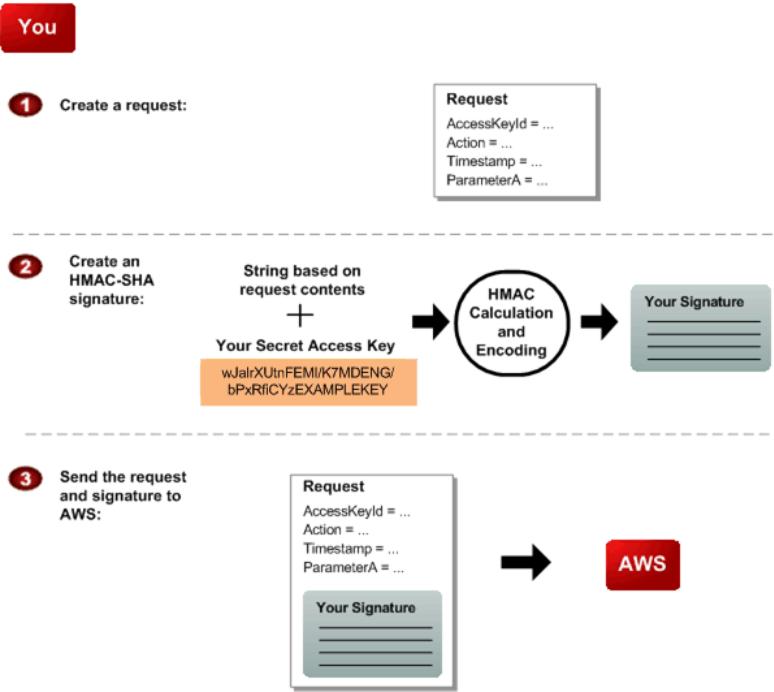
基本身份验证过程

Abstract

分步介绍使用 HMAC-SHA 请求签名对 AWS 请求进行身份验证所需的基本身份验证过程。

以下是使用 HMAC-SHA 请求签名对发送到 AWS 的请求进行身份验证所需执行的一系列任务。假设您已经创建了 AWS 账户，并且创建了访问密钥 ID 和私有访问密钥。有关 AWS 账户、访问密钥 ID 和私有访问密钥的更多信息，请参阅“您的 AWS 账户 (p. 78)”和“您的访问密钥 (p. 78)”。

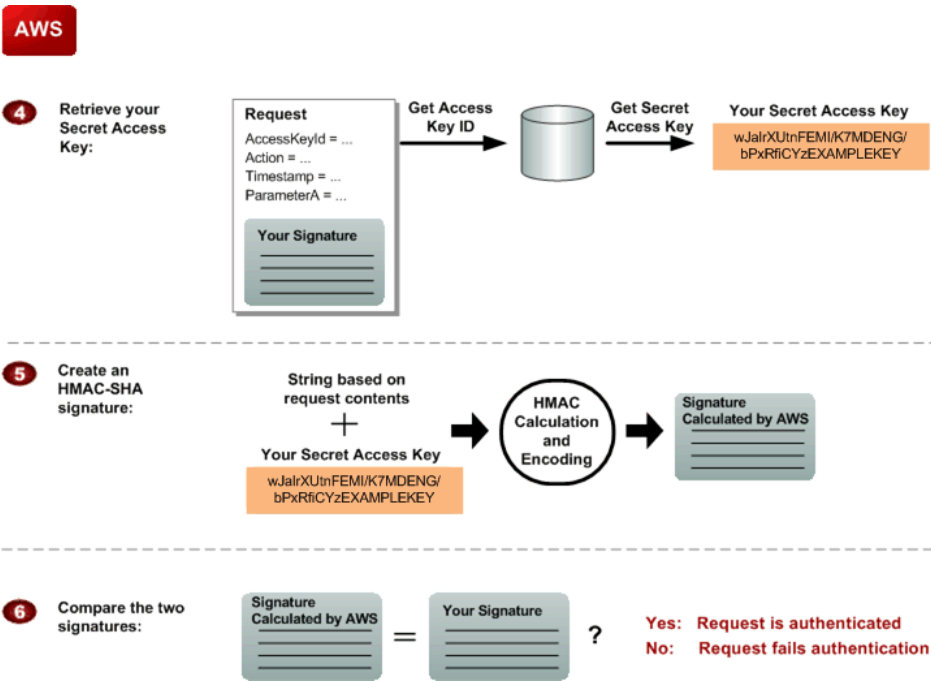
您执行前三个任务。



身份验证过程：您执行的任务

1	您构造发送到 AWS 的请求。
2	您使用您的私有访问密钥来计算密钥式哈希消息身份验证代码 (HMAC-SHA) 签名 (有关 HMAC 的信息，请转到 http://www.faqs.org/rfcs/rfc2104.html)。
3	您在请求中包含签名和您的访问密钥 ID，然后将请求发送到 AWS。

AWS 执行后三个任务。



身份验证过程：AWS 执行的任务

4	AWS 使用访问密钥 ID 来查找私有访问密钥。
5	通过使用与计算请求中发送的签名相同的算法，AWS 会从请求数据和私有访问密钥中生成一个签名。
6	如果由 AWS 生成的签名与您在请求中发送的签名相匹配，将认为请求是真实的。如果比较签名这一操作失败，那么请求将被丢弃，同时 AWS 将返回一份错误响应。

关于待签字符串

Abstract

每个 AWS 请求都包含使用您的私有访问密钥计算的 HMAC-SHA 请求签名。

您发送的每个 AWS 请求都必须包含一个使用您的私有访问密钥计算的 HMAC-SHA 请求签名。“[查询请求身份验证 \(p. 83\)](#)”中介绍了详细信息。

关于时间戳

Abstract

每个 AWS 请求都包含时间戳或过期时间。

您在请求中使用的时间戳（或过期时间）必须是 `dateTime` 数据元，并且带有完整的日期和时分秒（有关更多信息，请转到 <http://www.w3.org/TR/xmlschema-2/#dateTime>）。例如：2007-01-31T23:59:59Z。我们建议您以协调世界时（格林威治标准时间）时区提供时间戳（虽然这并不是必须的）。

如果您指定了时间戳（而不是过期时间），则请求会在时间戳过后 15 分钟自动过期（换句话说，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）。确保您的服务器时间设置正确。



Important

如果您使用的是 .NET，则不能发送过于特定的时间戳，因为 AWS 和 .NET 对如何确定额外时间精度有不同的解释。要避免过于特定的时间戳，请手动构造精度不超过毫秒的 `dateTime` 数据元。

Base64 编码的 Java 示例代码

Abstract

提供一个 Java 示例代码，用于演示如何对请求签名执行 base64 编码。

请求签名必须采用 Base64 编码。以下 Java 示例代码显示了如何执行 Base64 编码。

```
package amazon.webservices.common;
/**
 * This class defines common routines for encoding data in AWS requests.
 */
public class Encoding {
    /**
     * Performs base64-encoding of input bytes.
     *
     * @param rawData * Array of bytes to be encoded.
     * @return * The base64 encoded string representation of rawData.
     */
    public static String EncodeBase64(byte[] rawData) {
        return Base64.encodeBytes(rawData);
    }
}
```

计算 HMAC-SHA1 签名的 Java 示例代码

Abstract

提供一个 Java 示例代码，用于演示如何计算 HMAC 请求签名。

以下 Java 代码示例显示了如何计算 HMAC 请求签名。

```
package amazon.webservices.common;

import java.security.SignatureException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

/**
 * This class defines common routines for generating
 * authentication signatures for AWS requests.
 */
public class Signature {
    private static final String HMAC_SHA1_ALGORITHM = "HmacSHA1";
}
```

```
/**
 * Computes RFC 2104-compliant HMAC signature.
 * * @param data
 * The data to be signed.
 * @param key
 * The signing key.
 * @return
 * The Base64-encoded RFC 2104-compliant HMAC signature.
 * @throws
 * java.security.SignatureException when signature generation fails
 */
public static String calculateRFC2104HMAC(String data, String key)
throws java.security.SignatureException
{
    String result;
    try {

        // get an hmac_sha1 key from the raw key bytes
        SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), HMAC_SHA1_ALGORITHM);

        // get an hmac_sha1 Mac instance and initialize with the signing key
        Mac mac = Mac.getInstance(HMAC_SHA1_ALGORITHM);
        mac.init(signingKey);

        // compute the hmac on input data bytes
        byte[] rawHmac = mac.doFinal(data.getBytes());

        // base64-encode the hmac
        result = Encoding.EncodeBase64(rawHmac);

    } catch (Exception e) {
        throw new SignatureException("Failed to generate HMAC : " + e.getMessage());
    }
    return result;
}
```

查询请求身份验证

Abstract

通过包含签名或以编程方式调用 Amazon SQS API 公开的功能，通过 HTTP/HTTPS 提交查询请求。

以编程方式调用 Amazon SQS API 提供的功能时，所有发送到 Amazon SQS 的调用都必须签名。如果您使用 [AWS 开发工具包](#)，则开发工具包会为您处理签名过程，因此您不必手动完成这些任务。另一方面，如果您通过 HTTP/HTTPS 提交查询请求，则必须在每个查询请求中包含签名。

Amazon SQS 支持签名版本 4。签名版本 4 提供比先前版本更出色的安全性和性能。如果您正在创建使用 Amazon SQS 的新应用程序，则应使用签名版本 4。

有关如何使用签名版本 4 创建签名的信息，请参阅《AWS General Reference》中的[“Signature Version 4 Signing Process”](#)。

响应

Abstract

在针对操作请求的响应中，返回包含请求结果的 XML 数据结构。

主题

- [成功响应的结构 \(p. 84\)](#)
- [错误响应的结构 \(p. 84\)](#)
- [相关主题 \(p. 85\)](#)

在响应操作请求时，Amazon SQS 会返回包含请求结果的 XML 数据结构。此数据符合 Amazon SQS 架构。有关更多信息，请参阅《Amazon SQS API Reference》中的“[WSDL 位置和 API 版本](#)”。

成功响应的结构

如果请求成功了，则主要响应元素会以该操作命名，但会附加上“Response”。例如，CreateQueueResponse 是针对成功的 CreateQueue 请求返回的响应元素。此元素包含以下子元素：

- ResponseMetadata，它包含 RequestId 子元素
- 包含特定于操作的结果的可选元素；例如，CreateQueueResponse 元素包含名为 CreateQueueResult 的元素

XML 架构描述了针对每个 SQS 操作的 Amazon SQS 响应消息。

以下是成功响应的示例。

```
<CreateQueueResponse
  xmlns=http://sqs.us-east-1.amazonaws.com/doc/2012-11-05/
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:type=CreateQueueResponse>
  <CreateQueueResult>
    <QueueUrl>
      http://sqs.us-east-1.amazonaws.com/770098461991/queue2
    </QueueUrl>
  </CreateQueueResult>
  <ResponseMetadata>
    <RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
  </ResponseMetadata>
</CreateQueueResponse>
```

错误响应的结构

如果请求不成功，则无论调用的操作如何，主要响应元素都名为 ErrorResponse。此元素包含一个 Error 元素和一个 RequestId 元素。每个 Error 都包含：

- 一个 Type 元素，该元素标识错误是收件人错误还是发件人错误
- 一个 Code 元素，该元素标识所发生错误的类型
- 一个 Message 元素，该元素以人类可读的格式来描述错误条件
- 一个 Detail 元素，该元素可能会提供有关错误的其他详细信息或者可能为空

以下是错误响应的示例。

```
<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>
```

相关主题

- [提出查询请求 \(p. 73\)](#)

共享队列

Abstract

通过使用 Amazon SQS 在访问控制策略中授予权限，将队列与其他用户共享。

主题

- [共享队列的简单 API \(p. 86\)](#)
- [共享队列的高级 API \(p. 86\)](#)
- [了解资源级权限 \(p. 86\)](#)
- [授予对队列的匿名访问权限 \(p. 87\)](#)

Amazon SQS 包含共享您队列的方法，以便其他人可以通过使用访问控制策略中设置的权限来使用这些队列。权限使另一个人能够以某种特定的方式来使用您的队列。策略是包含您授予的权限的实际文档。

Amazon SQS 提供了两种设置策略的方法：简单 API 和高级 API。在简单 API 中，Amazon SQS 会为您生成访问控制策略。在高级 API 中，您可以创建访问控制策略。

共享队列的简单 API

用于共享队列的简单 API 有两个操作：

- [AddPermission](#)
- [RemovePermission](#)

通过简单 API，Amazon SQS 会基于您在 [AddPermission](#) 操作中包含的信息，使用所需语言编写策略。但是，Amazon SQS 生成的策略范围有限。您可以向委托人授予权限，但是无法指定限制。

共享队列的高级 API

通过高级 API，您可以自己使用访问策略语言直接编写策略，然后通过 [SetQueueAttributes](#) 操作上传策略。高级 API 允许您拒绝访问或应用更精细的访问限制（例如，基于时间或基于 IP 地址）。

如果您选择编写自己的策略，则需要了解策略的构造方式。有关策略的完整参考信息，请参阅“[使用 Access Policy Language \(p. 95\)](#)”。有关策略示例，请参阅“[Amazon SQS 策略示例 \(p. 109\)](#)”。

了解资源级权限

权限是您向委托人（接收权限的用户）授予的访问类型。您可以为每个权限提供一个用于识别该权限的标签。如果您将来要删除该权限，则可以使用该标签来识别该权限。如果您要查看队列中有哪些权限，请使用 [GetQueueAttributes](#) 操作。Amazon SQS 会返回整个策略（包含所有权限）。

Amazon SQS 支持下表所示的权限类型。

许可	描述
*	此权限类型可以向委托人授予对共享队列执行以下操作的权限：更改消息的可见性、删除消息、获取队列的属性、获取队列的 URL、接收消息和发送消息。

许可	描述
ChangeMessageVisibility	此权限类型可以授予延长或终止指定消息读取锁定超时的权限。ChangeMessageVisibilityBatch 继承了与 ChangeMessageVisibility 关联的权限。有关可见性超时的更多信息，请参阅“ 可见性超时 (p. 7) ”。有关更多信息，请参阅 ChangeMessageVisibility 操作。
DeleteMessage	此权限类型可以授予从队列中删除消息的权限。DeleteMessageBatch 继承了与 DeleteMessage 关联的权限。有关更多信息，请参阅 DeleteMessage 操作。
GetQueueAttributes	此权限类型可以授予获取所有队列属性（只能由队列拥有者访问的策略除外）的权限。有关更多信息，请参阅 GetQueueAttributes 操作。
GetQueueUrl	此权限类型可以授予获取队列的 URL 的权限。有关更多信息，请参阅 GetQueueUrl 操作。
ReceiveMessage	此权限类型可以授予接收队列中消息的权限。有关更多信息，请参阅 ReceiveMessage 操作。
SendMessage	此权限类型可以授予向队列发送消息的权限。SendMessageBatch 继承了与 SendMessage 关联的权限。有关更多信息，请参阅 SendMessage 操作。



Note

如果为 SendMessage、DeleteMessage 或 ChangeMessageVisibility 设置权限，则系统也会为这些操作的相应批处理版本（SendMessageBatch、DeleteMessageBatch 和 ChangeMessageVisibilityBatch）设置权限。不允许显式对 SendMessageBatch、DeleteMessageBatch 和 ChangeMessageVisibilityBatch 设置权限。

Amazon SQS 将各种不同权限类型的权限视为单独的权限，即使 * 包含其他权限类型提供的访问权限，也是如此。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，该请求不会执行任何操作，因为委托人之前不具有显式 SendMessage 权限。

如果您要删除 * 并且只为委托人留下 ReceiveMessage 权限，请先添加 ReceiveMessage 权限，然后删除 * 权限。



Tip

您可以为每个权限提供一个用于识别该权限的标签。如果您将来要删除该权限，则可以使用该标签来识别该权限。



Note

如果您要查看队列中有哪些权限，请使用 [GetQueueAttributes](#) 操作。将返回整个策略（包含所有权限）。

授予对队列的匿名访问权限

您可以向匿名用户授予共享队列访问权限。此类访问权限无需任何签名或访问密钥 ID。

要允许匿名访问，您必须编写自己的策略，并将 `Principal` 设置为 `*`。有关编写您自己的策略的信息，请参阅“[使用Access Policy Language \(p. 95\)](#)”。



Caution

请记住，队列所有者负责承担与队列相关的所有费用。因此，您可能希望以其他某种方式（例如，按时间或 IP 地址）限制匿名访问。

编程语言

Abstract

列出可以用于 Amazon SQS 的编程语言、库、示例代码和其他资源。

对于那些喜欢使用特定语言的 API 而不喜欢使用 Amazon SQS 的查询 API 构建应用程序的软件开发人员，AWS 为他们提供了库、示例代码、教程和其他资源。这些库提供基本功能（不包括在 Amazon SQS 的查询 API 中），例如请求身份验证、请求重试和错误处理，让您可以更轻松地开始使用。现已推出适用以下语言的库和资源：

- [转到](#)
- [Java](#)
- [JavaScript](#)
- [PHP](#)
- [Python](#)
- [Ruby](#)
- [Windows 和 .NET](#)

有关移动应用程序开发，请参阅：

- [适用于 Android 的 AWS SDK](#)
- [AWS SDK for iOS](#)
- [适用于 Unity 的 AWS 软件开发工具包](#)



Note

此外，还有可用于与 Amazon SQS 交互的命令行工具：

- AWS Command Line Interface (AWS CLI)。有关更多信息，请参阅[使用 AWS Command Line Interface 进行设置](#)和 [AWS CLI 参考的 Amazon SQS 部分](#)。
- 适用于 Windows PowerShell 的 AWS 工具。有关更多信息，请参阅[使用 适用于 Windows PowerShell 的 AWS 工具 进行设置](#)和 [适用于 Windows PowerShell 的 AWS 工具 Cmdlet 参考的 Amazon Simple Queue Service 部分](#)。

使用 Amazon SQS 死信队列

Abstract

向死信队列（定义为其他队列可以将其作为目标发送无法成功处理的消息的队列）发送消息。

Amazon SQS 现在为死信队列提供支持。死信队列是其他（源）队列将其作为目标发送因为某种原因而无法成功处理的消息的队列。使用死信队列的主要好处是能够放弃和隔离未成功处理的消息。您随后可以分析发送到死信队列的任何消息，以尝试和确定它们未成功处理的原因。

要指定死信队列，您可以使用 AWS 管理控制台或查询 API。必须为将向死信队列发送消息的每个（源）队列执行此操作。可以让多个（源）队列将一个死信队列作为目标。



Important

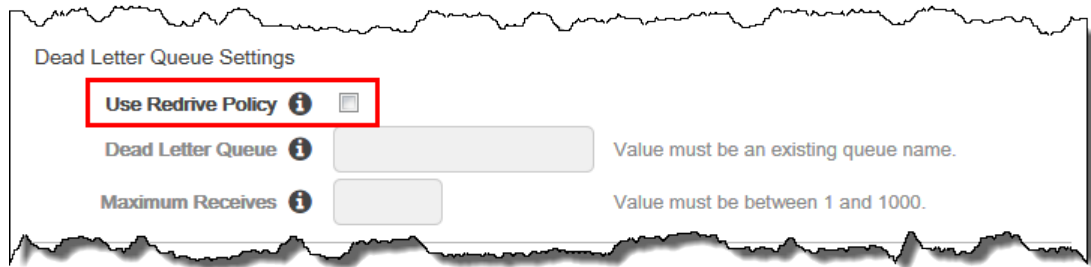
您必须使用相同 AWS 账户创建将用作死信队列的队列，以及将向该死信队列发送消息的其他（源）队列。死信队列还必须与使用死信队列的其他队列位于相同的区域中。例如，如果在 US East (N. Virginia) 区域中创建了一个队列，并且要对该队列使用死信队列，则两个队列必须都位于 US East (N. Virginia) 区域中。

主题

- [使用 AWS 管理控制台设置死信队列 \(p. 90\)](#)
- [通过 Amazon SQS API 使用死信队列 \(p. 92\)](#)
- [问题：使用 Amazon SQS 控制台查看消息可能会导致消息移至死信队列 \(p. 93\)](#)

使用 AWS 管理控制台设置死信队列

您可以使用 AWS 管理控制台配置死信队列功能，以便将出于某种原因而无法成功处理的消息发送到指定死信队列。实现此目的的方式是首先为现有和新创建的队列选中源队列的 Use Redrive Policy (使用重新驱动策略) 复选框。



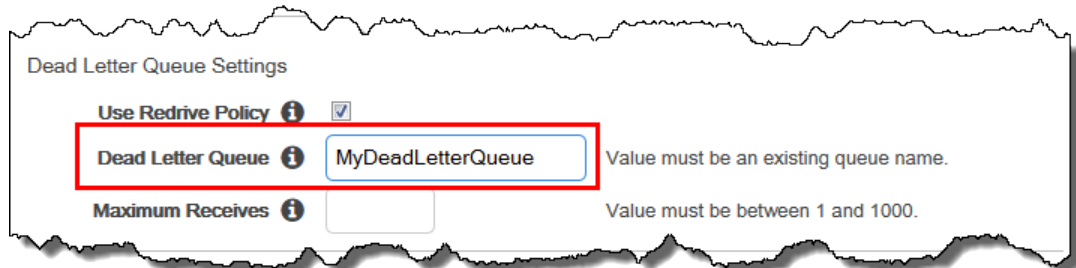
Dead Letter Queue Settings

Use Redrive Policy ⓘ ☒

Dead Letter Queue ⓘ Value must be an existing queue name.

Maximum Receives ⓘ Value must be between 1 and 1000.

其次，输入将从源队列向其发送消息的队列的名称。



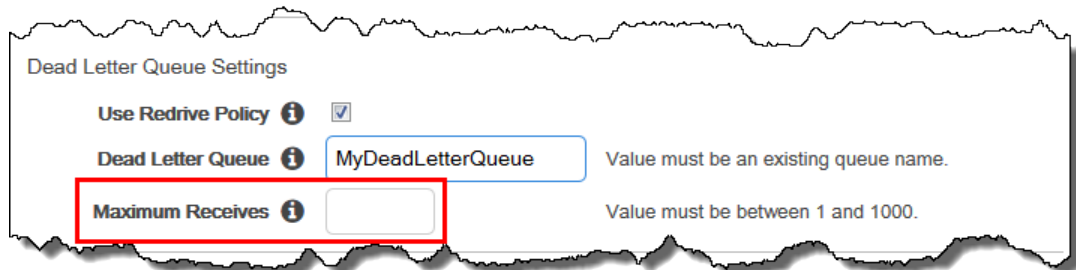
Dead Letter Queue Settings

Use Redrive Policy ⓘ ☒

Dead Letter Queue ⓘ Value must be an existing queue name.

Maximum Receives ⓘ Value must be between 1 and 1000.

然后将 Maximum Receives (最大接收数) 设置为介于 1 到 1000 之间的值。



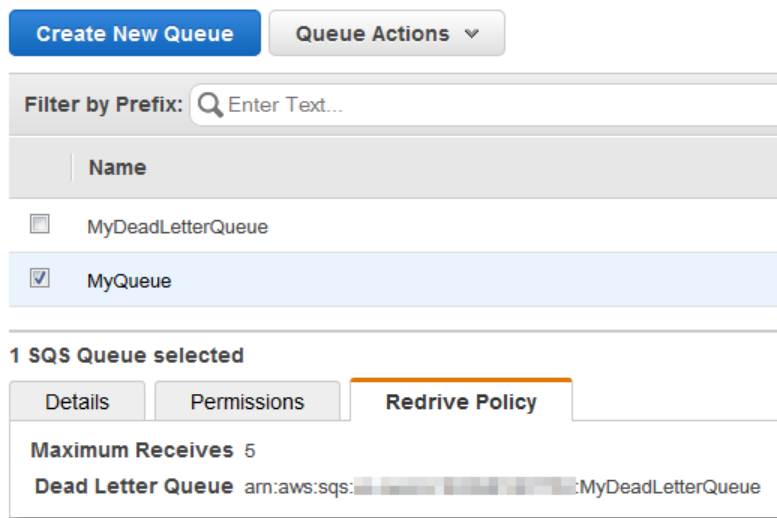
Dead Letter Queue Settings

Use Redrive Policy ⓘ ☒

Dead Letter Queue ⓘ Value must be an existing queue name.

Maximum Receives ⓘ Value must be between 1 and 1000.

下图显示 *MyQueue* 队列，该队列配置了 Redrive Policy，以将消息发送到 *MyDeadLetterQueue* 队列。



通过 Amazon SQS API 使用死信队列

要使用查询 API 指定死信队列，请调用 `CreateQueue` 或 `SetQueueAttributes` 操作并为 `RedrivePolicy` 队列属性设置 `maxReceiveCount` 和 `deadLetterTargetArn` 参数。

您可以将 `maxReceiveCount` 设置为介于 1 到 1000 之间的值。`deadLetterTargetArn` 值是将接收死信消息的队列的 Amazon 资源名称 (ARN)。

以下 Java 示例演示如何使用 `SetQueueAttributes` 为 `RedrivePolicy` 队列属性设置 `maxReceiveCount` 和 `deadLetterTargetArn` 参数。此示例基于 AWS SDK for Java 中的 `SimpleQueueServiceSample.java` 示例。

首先，设置一个字符串，其中包含 JSON 格式的参数和 `RedrivePolicy` 队列属性的值：

```
String redrivePolicy = "{\"maxReceiveCount\": \"5\", \"deadLetterTargetArn\": \"arn:aws:sqs:us-east-1:123456789012:MyDeadLetterQueue\"}";
```

接下来，`SetQueueAttributesRequest` 用于设置 `RedrivePolicy` 队列属性：

```
SetQueueAttributesRequest queueAttributes = new SetQueueAttributesRequest();
Map<String,String> attributes = new HashMap<String,String>();
attributes.put("RedrivePolicy", redrivePolicy);
queueAttributes.setAttributes(attributes);
queueAttributes.setQueueUrl(myQueueUrl);
sqs.setQueueAttributes(queueAttributes);
```

此示例的 API 查询请求应类似于以下内容：

```
http://sqs.us-east-1.amazonaws.com/123456789012/MySourceQueue
```

```
?Action=SetQueueAttributes
&Attribute.1.Value=%7B%22maxReceiveCount%22%3A%225%22%2C+%22deadLetterTarget
Arn%22%3A%22arn%3Aaws%3Asqs%3Aus-east-1%3A123456789012%3AMyDeadLetterQueue%22%7D
&Version=2012-11-05
&Attribute.1.Name=RedrivePolicy
```



Note

队列名称和队列 URL 区分大小写。

API 查询响应应类似于以下内容：

```
<SetQueueAttributesResponse xmlns="http://queue.amazonaws.com/doc/2012-11-05/">

  <ResponseMetadata>
    <RequestId>40945605-b328-53b5-aed4-1cc24a7240e8</RequestId>
  </ResponseMetadata>
</SetQueueAttributesResponse>
```

问题：使用 Amazon SQS 控制台查看消息可能会导致消息移至死信队列

如果您在 Amazon SQS 控制台中将消息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。

发生此行为是因为，Amazon SQS 将对按照相应队列的重新驱动策略在 Amazon SQS 控制台中查看消息计数。

要调整此行为，可增加相应队列的重新驱动策略的 Maximum Receives 设置，或不在 Amazon SQS 控制台中查看相应队列的消息。

重现此行为：

1. 在 Amazon SQS 控制台中，创建名 2 个分别名为 MyQueueA 和 MyQueueB 的队列。
2. 指定 MyQueueA 的设置：在队列列表中，选择（选中）MyQueueA。然后依次选择 Queue Actions 和 Configure Queue。
3. 对于 Dead Letter Queue Settings，选择（选中）Use Redrive Policy。
4. 对于 Dead Letter Queue，键入 **MyQueueB**。（这可将 MyQueueB 设置为 MyQueueA 的死信队列。）
5. 对于 Maximum Receives，键入 2。（这意味着，在查看消息 2 次后，该消息将发送到死信队列（MyQueueB）。）然后选择 Save Changes。
6. 将一些消息发送到 MyQueueA：在队列列表中仍选择（选中）MyQueueA 的情况下，依次选择 Queue Actions 和 Send a Message。
7. 在已选定 Message Body 选项卡的情况下，键入一些消息文本，然后选择 Send Message。
8. 选择 Send Another Message，然后重复步骤 7 以向 MyQueueA 发送另一条消息。然后选择 Close。
9. 查看发送到 MyQueueA 的消息：在队列列表中仍选择（选中）MyQueueA 的情况下，依次选择 Queue Actions 和 View/Delete Messages。
10. 选择 Start Polling for Messages。将显示步骤 7 和步骤 8 中的消息。

11. 等待 1 分钟，然后再次选择 Start Polling for Messages。将再次显示步骤 7 和步骤 8 中的消息。（请注意，两条消息的 Receive Count 现为 2。这与步骤 5 中指定的 Maximum Receives 值相等。）
12. 等待 1 分钟，然后再次选择 Start Polling for Messages。此时将不再显示步骤 7 和步骤 8 中的消息。（这是因为，您已查看消息 2 次，该次数等于步骤 5 中指定的 Maximum Receives 值。）选择 Close。
13. 查看 MyQueueA 的死信队列中的消息：在队列列表中，取消选择（取消选中）MyQueueA。然后选择（选中）MyQueueB，并依次选择 Queue Actions 和 View/Delete Messages。
14. 选择 Start Polling for Messages。将显示步骤 7 和步骤 8 中的消息。（这是因为 MyQueueB 是 MyQueueA 的死信队列。）

使用 Access Policy Language

Abstract

使用访问策略语言编写您自己的访问控制策略，以用于 Amazon SQS

主题

- [概述 \(p. 96\)](#)
- [Amazon SQS 策略示例 \(p. 109\)](#)
- [Amazon SQS 策略的特别信息 \(p. 113\)](#)

本部分内容面向希望编写自己的访问控制策略的 Amazon SQS 用户。如果您希望仅基于 AWS 账户 ID 和基本权限（例如，SendMessage、ReceiveMessage）来允许访问，则不需要编写自己的策略。在这种情况下，您可以仅使用 Amazon SQS AddPermission 操作。如果您希望基于更精细的条件（例如，请求到达的时间或请求者的 IP 地址）来显式拒绝访问或允许访问，则需要编写自己的策略并使用 Amazon SQS SetQueueAttributes 操作将其上传到 AWS 系统。



Note

要编写您自己的策略，您必须熟悉 JSON。有关更多信息，请转到 <http://json.org>。

本节的主要部分包括您需要了解的基本概念、编写策略的方法，以及 AWS 用来评估策略并决定是否向请求者授予资源访问权限的逻辑。虽然本节中的大多数信息都是服务兼容的，但是其中也有一些您需要了解的特定于 SQS 的详细信息。有关更多信息，请参阅 [Amazon SQS 策略的特别信息 \(p. 113\)](#)。

概述

Abstract

介绍使用访问策略语言编写策略所需要了解的基本概念。

主题

- [何时使用访问控制 \(p. 96\)](#)
- [主要概念 \(p. 96\)](#)
- [架构概述 \(p. 99\)](#)
- [使用 Access Policy Language \(p. 100\)](#)
- [评估逻辑 \(p. 102\)](#)
- [关于访问控制的基本使用案例 \(p. 106\)](#)

本部分介绍了要使用access policy language编写策略需要了解的基本概念。本部分还介绍了访问控制与access policy language一起使用的一般过程以及如何评估策略。

何时使用访问控制

Abstract

介绍有关何时使用访问控制的典型使用案例。

对于如何授权或拒绝资源访问，您有很大的灵活性。然而，普通的使用案例都相当简单。

- 您欲授予另一 AWS 账户特定访问类型，以访问您的队列（例如 SendMessage）。有关更多信息，请参阅 [使用案例 1 \(p. 106\)](#)。
- 您欲授权另一 AWS 账户可在指定时段内访问您的队列。有关更多信息，请参阅 [使用案例 2 \(p. 106\)](#)。
- 您欲授权另一 AWS 账户仅在从 Amazon EC2 实例发出请求的情况下访问您的队列。有关更多信息，请参阅 [使用案例 3 \(p. 107\)](#)。
- 您欲拒绝另一 AWS 账户访问您的队列。有关更多信息，请参阅 [使用案例 4 \(p. 107\)](#)。

主要概念

Abstract

列出使用访问策略语言编写策略的主要概念。按逻辑顺序提供。

以下章节介绍了您需要了解的概念，以便使用 access policy language。它们都按逻辑顺序介绍，您需要了解的第一项术语在清单最前面。

许可

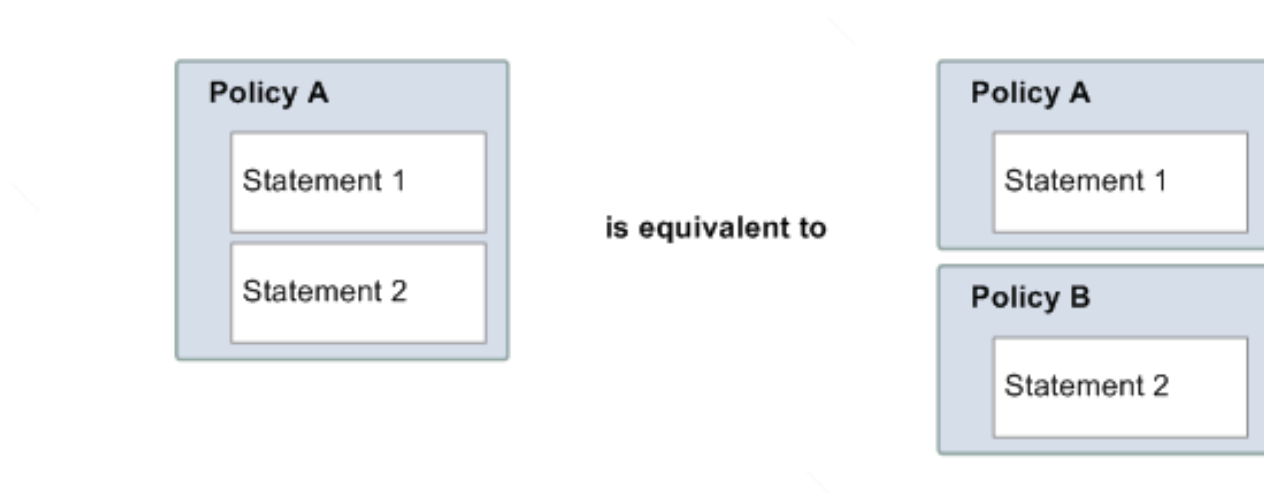
权限是指允许或不允许访问某种特殊资源的概念。权限基本遵循这种形式：“如果 D 适用时，那么 A 被允许或未被允许向 C 执行 B”。例如，只要 Jane (A) 在 2009 年 5 月 30 日午夜之前请求接收消息 (D)，她就有权从 John 的 Amazon SQS 队列 (C) 接收消息 (B)。无论 Jane 何时向 Amazon SQS 发送使用 John 的队列的请求，该产品都会检查她是否拥有权限，同时还会检查发送的请求是否满足 John 在权限中设定的条件。

语句

声明是指 access policy language 中编写的关于单一许可的正式描述。您通常编写的语句是一个更大的容器式文档，被称为 **策略**（见下一概念），的一部分。

策略

策略 是一个使用 access policy language 编写的文档，用作存放一个或多个语句的容器。例如，一份策略中可包含两份声明：一份为 Jane 可使用 John 的队列的声明，另一份为 Bob 不得使用 John 的队列的声明。如下图所示，一个等效情境中将包含两份策略，一份包含 Jane 可使用 John 的队列的声明，另一份包含 Bob 不得使用 John 的队列的声明。



AWS 产品使用声明（无论是包含在单一策略，或是多份策略中）中的信息实施访问控制（例如 Amazon SQS），以确定是否授权请求访问资源者访问资源。我们经常将 **策略** 与 **声明** 这两个词互换，因为二者一般表示相同的概念（一个实体代表一个许可）。

发布者

发布者是指编写策略以授予资源权限的人。发布者（按定义）通常指资源所有者。AWS 不允许 AWS 服务用户为他们不拥有的资源创建策略。如果 John 是资源所有者，那么当他提交他编写的策略为该资源授予权限时，AWS 会对 John 的身份进行认证。

委托人

委托人是指在策略中获取权限的个人和多个人。委托人是“如果 D 适用的情况下，那么 A 可以对 C 执行 B”语句中的 A。在一个策略中，您可将委托人设置为“任何人”（例如，如您可指定一个通配符代表所有人。您这样操作，例如，如果您不想根据请求者的实际身份限制访问，那么您可以根据其他的识别特征，例如请求者的 IP 地址。

操作

操作是委托人可以执行的活动。操作是“如果 D 适用的情况下，那么 A 可以对 C 执行 B”语句中的 B。通常情况下，该操作只是向 AWS 提出请求的操作。例如，Jane 使用 `Action=ReceiveMessage` 向 Amazon SQS 发送请求。在一个策略中您可指定一个或多个操作。

资源

资源是委托人请求访问的数据元。在表述“在满足 D 的情况下，A 拥有对 C 执行 B 的许可”中，C 即指资源。

条件与密钥

条件是所有有关权限的限制条件和具体内容。条件是“如果 D 适用的情况下，那么 A 可以对 C 执行 B”语句中的 D。说明条件的策略部分可能是整个部分最详细且最复杂的内容。普通条件与以下项目相关：

- 日期和时间（例如：请求必须在指定日期前到达）
- IP 地址（例如，请求者的 IP 地址必须是某个特定 CIDR 范围的一部分）

一个密钥是设置访问限制指定的特性。例如，访问日期和时间。

您需使用条件和密钥一起明确说明限制。下列示例可帮助您以最简单的方式了解如何实际实施限制：若您要在 2010 年 5 月 30 日之前限制访问，则使用名为 `DateLessThan` 的条件。您使用的密钥名为 `AWS:CurrentTime`，并将其值设置为 `2010-05-30T00:00:00Z`。AWS 确定您能使用的条件和密钥。此外，AWS 产品本身（例如，Amazon SQS）也可能会定义特定于服务的密钥。有关可用密钥的更多信息，请参见[Amazon SQS 密钥](#) (p. 119)。

请求者

请求者指向一个 AWS 服务发出请求并要求访问某个特定资源的人。请求者向 AWS 发送的请求基本表述为：“在满足 D 的情况下，您是否允许我对 C 执行 B？”

评估

评估是指 AWS 服务根据可适用策略决定拒绝或接收一个输入请求的过程。有关评估逻辑的信息，请参见[评估逻辑](#) (p. 102)。

效果

效果是指在评估期间您希望一个策略语句返回的结果。当您在策略中编写语句时，您需指定该值，可能值为拒绝和允许。

例如，您可编写一个策略，并声明拒绝所有来自南极洲地区的请求（效果=如果请求是使用南极洲配置的 IP 地址发出，会拒绝）。同样地，您可以编写一个策略，并声明允许所有并非来自南极洲地区的请求（效果=如果请求不是来自南极洲地区，会允许）。虽然这两个语句看似执行相同的操作，但是在 access policy language 逻辑上，它们是不同的。有关更多信息，请参阅[评估逻辑](#) (p. 102)。

尽管仅可指定两个效果值（“允许”或“拒绝”），但在执行策略评估时，却可能产生三种不同的结果，即：默认拒绝、允许或显式拒绝。有关更多详细，参见以下概念和[评估逻辑](#) (p. 102)。

默认拒绝

默认拒绝是从一个策略中没有允许或显式拒绝的默认结果。

允许

假设任何声明的条件已满足，效果=允许的声明会产生允许。例如：若在 2010 年 4 月 30 日下午 1:00 之前收到请求，则请求将获得允许。“允许”可置换所有“默认拒绝”，但无法置换“显式拒绝”。

显式拒绝

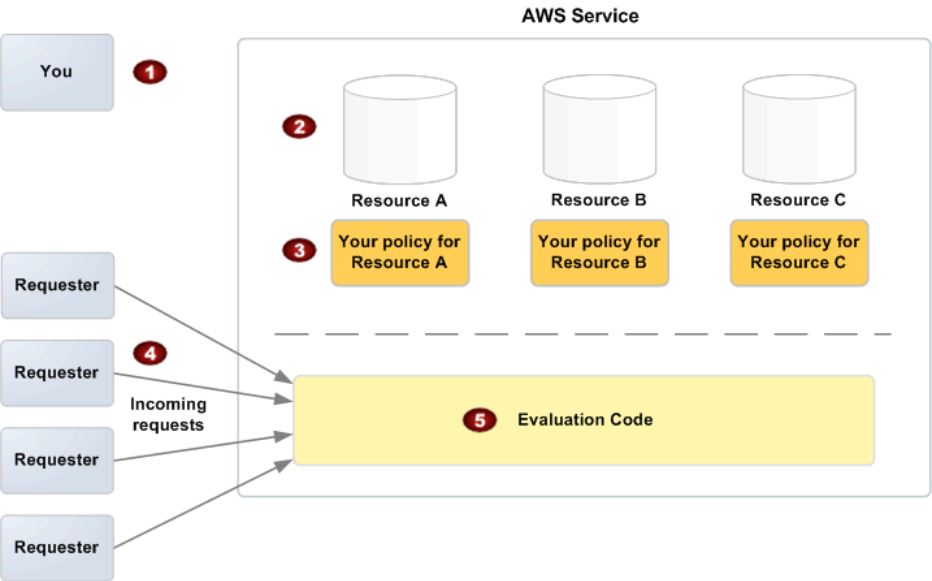
假设任何声明的条件已满足，效果=拒绝的语句会产生显式拒绝。例如：拒绝来自南极洲的所有请求。不管其他什么策略会允许，所有来自南极洲地区的请求都会被拒绝。

架构概述

Abstract

介绍为资源提供访问控制所需的主要组件的架构和设计。

下列图和表格介绍了为您提供资源访问控制的相互作用的主要组成部分。



1	您，资源所有者。
2	您的资源（包含在 AWS 产品内，例如，Amazon SQS 队列）。
3	您的策略。 通常情况下，每个资源拥有一个策略，虽然您可以有多个。AWS 服务本身提供一个您用来上传和管理您的策略 API。
4	请求者和他们向 AWS 服务传入的请求。
5	access policy language 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。有关产品如何作出决定的信息，请参见 评估逻辑 (p. 102) 。

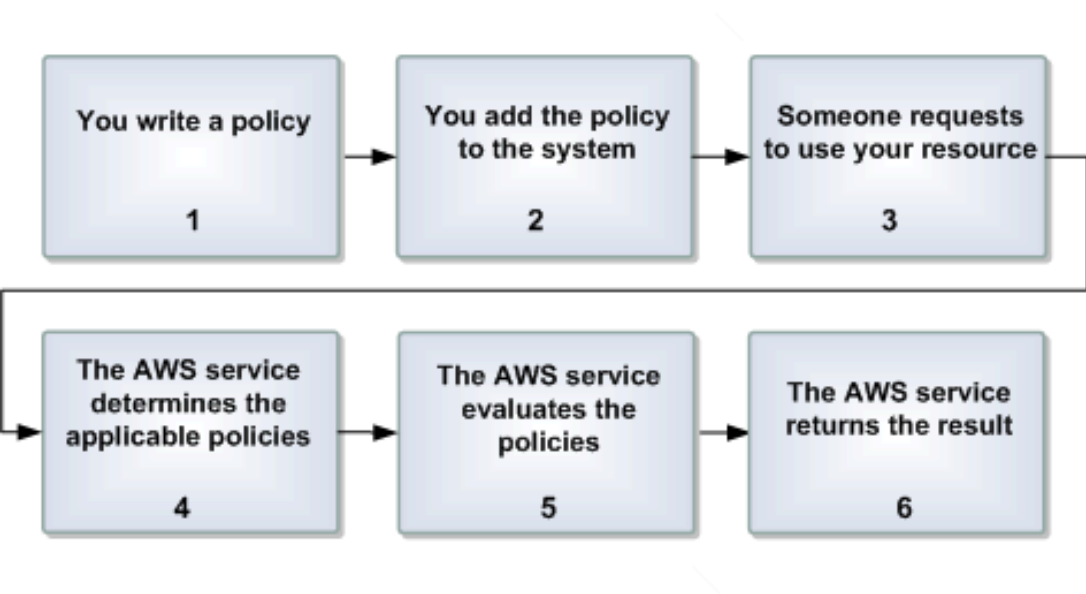
有关各组成部分如何协作的一般流程，请参见 [使用 Access Policy Language \(p. 100\)](#)。

使用 Access Policy Language

Abstract

介绍访问控制如何与访问策略语言结合使用的一般过程和流程。

下面的图表介绍了访问控制与 access policy language 协作的一般过程。



将访问控制与Access Policy Language一起使用的过程

1	为您的资源编写一个策略。 例如，您编写策略来为 Amazon SQS 队列指定权限。
2	上传您的策略至 AWS。 AWS 服务自身提供一个您用来上传您的策略的 API。例如，您使用 Amazon SQS <code>SetQueueAttributes</code> 操作来为特定的 Amazon SQS 队列上传策略。
3	某人向您发出使用您的资源的请求。 例如，某用户向 Amazon SQS 发送使用您的一个队列的请求。
4	AWS 服务决定哪些策略能适用于该请求。 例如，Amazon SQS 将查看所有可用的 Amazon SQS 策略，并确定哪些策略适用（基于资源是什么、请求者是谁等）。
5	AWS 服务将对这些策略进行评估。 例如，Amazon SQS 将对策略进行评估，确定是否允许请求者使用您的队列。有关决策逻辑的信息，请参见 评估逻辑 (p. 102)。
6	AWS 服务或许会拒绝请求，或许会继续处理这个请求。 例如，根据策略评估结果，服务将返回一个“访问被拒绝”的错误信息给请求者，或继续处理该请求。

相关主题

- [架构概述 \(p. 99\)](#)

评估逻辑

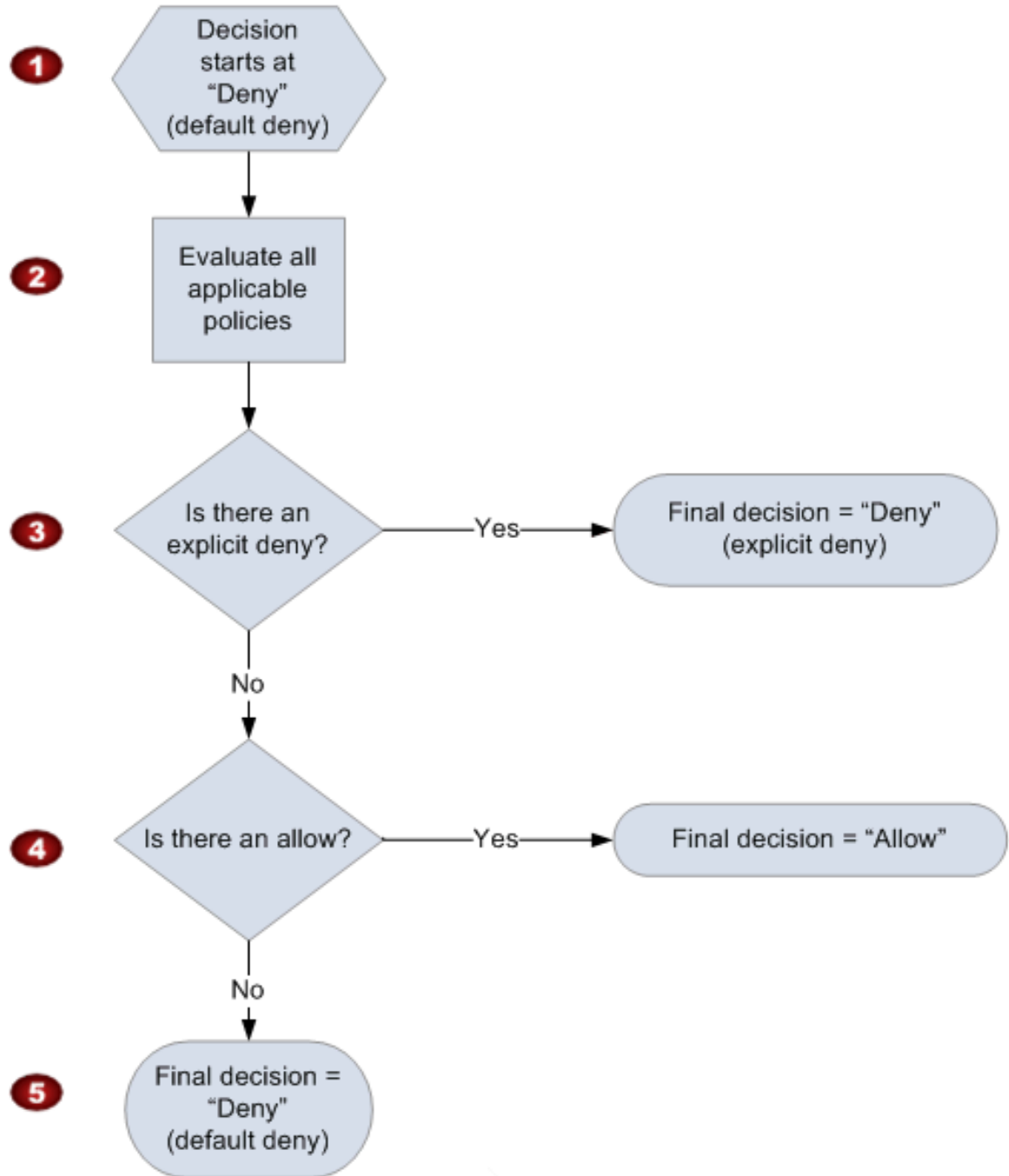
Abstract

介绍编写访问策略的评估逻辑的流程。

评估时的目标是决定应允许还是拒绝除您（资源所有者）以外的某个人发送的给定请求。评估逻辑遵循多个基本规则：

- 在默认情况下，除了您，任何人提出使用您资源的请求均会被拒绝。
- 一个允许可以超控任何其他默认拒绝
- 一个显式拒绝可以超控任何允许
- 策略评估的顺序不重要

以下流程图和讨论更加详细地描述了如何做出决定。



1	决定开始是一个默认拒绝。
2	然后执行代码将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不重要。

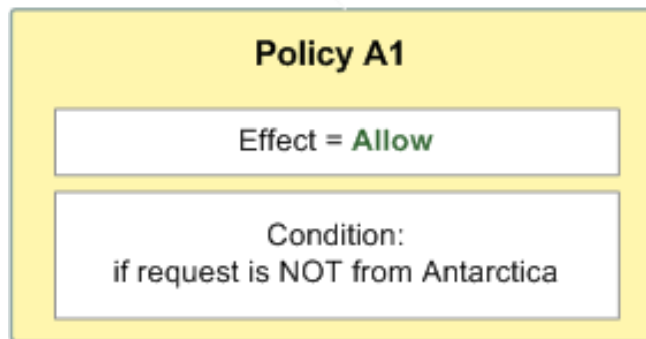
3	在所有这些策略中，执行代码将寻找一个能适用于请求的显式拒绝指令。 即使仅找到一处，执行代码也会发回“拒绝”的决定，并结束处理流程（此为“显式拒绝”；有关更多信息，请参见 显式拒绝 (p. 99) ）。
4	如果没有找到显式拒绝，那么执行代码将寻找适用于请求的任何“允许”指令。 如果它还是找到了一个，那么执行代码将返回一个“允许”决定，且整个过程完成（服务将继续处理该请求）。
5	如果没有找到允许，那么最终的决定将是“拒绝”（因为没有显式拒绝或允许，所以这将被视为是一个默认拒绝（有关更多信息，请参见 默认拒绝 (p. 98) ））。

显式拒绝和默认拒绝的相互作用

如果策略不直接适用于请求，那么策略将产生一个默认拒绝。例如，如果用户请求使用 Amazon SQS，但适用于该用户的唯一策略表示该用户可使用 Amazon DynamoDB，则该策略将导致“默认拒绝”的结果。

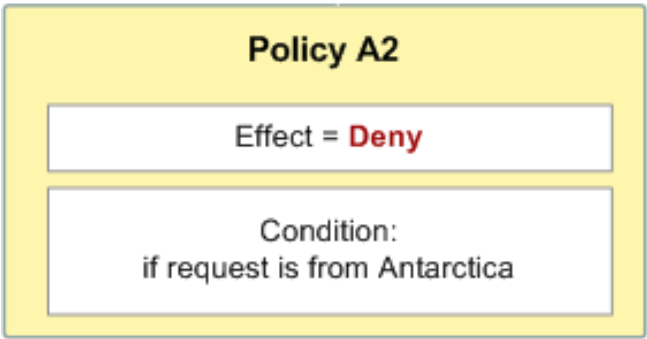
如果一个语句中的某个条件未被满足，那么策略将产生一个默认拒绝。如果声明中的所有条件都满足，那么根据策略中的效果元素的值，策略或许会产生允许，或许会产生显式拒绝。如果一个条件未被满足，策略没有指定如何处理，那么在那种情况下默认值将产生一个默认拒绝。

例如，假设您想要阻止来自南极洲地区的请求进入。只要请求不是来自于南极洲地区，您编写的策略（称作策略 A1）将允许接受请求。下列示意图说明了该策略。



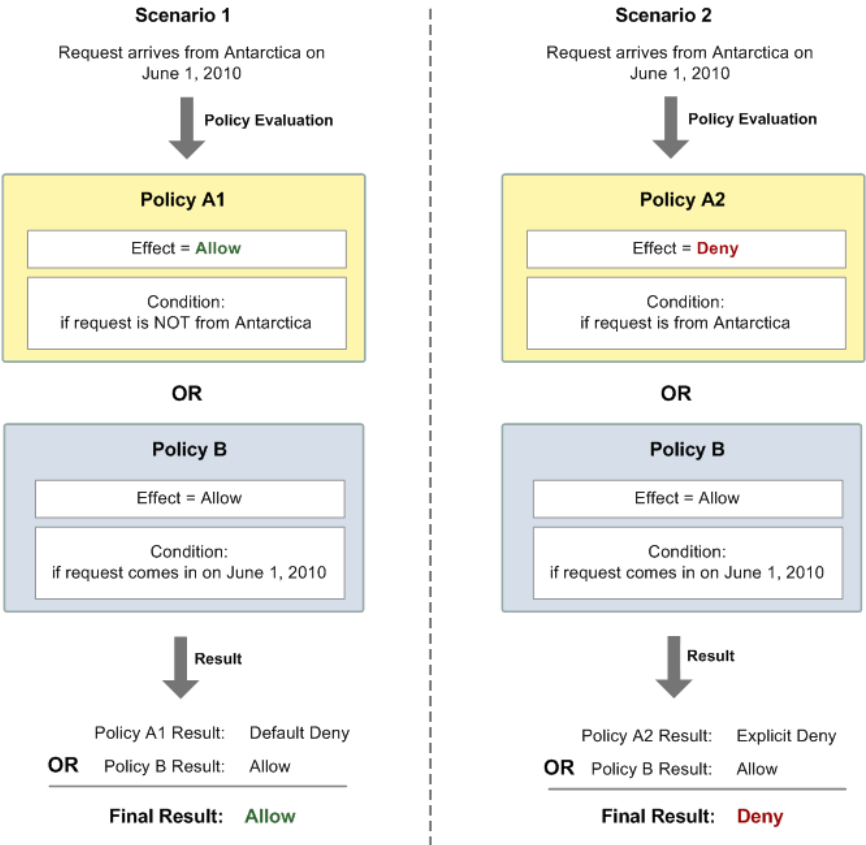
如果某人从美国发出请求，那么条件已经满足（该请求不是来自南极洲）。因此，该请求将被允许。但是，如果某人从南极洲地区发出请求，那么条件未满足，因此策略结果将是默认拒绝。

您可通过按照下列示意图重新编写策略（称作策略 A2）将结果转变为一个显式拒绝。此时，如果请求是来自南极洲地区，那么策略将明确拒绝该请求。



如果某人从南极洲发出请求，那么条件已经满足，策略的结果将是一个显式拒绝。

默认拒绝和显式拒绝的区别很重要，因为一个默认拒绝能被一个允许超控，但是显式拒绝就不能。例如，如果说如果请求是在 2010 年 6 月 1 日到达，那么将有另一个策略允许这些请求。那么，与限制从南极洲访问的策略相结合，该策略将如何对总体结果产生影响？当将按日期要求设置的策略与上述策略 A1 和 A2 相结合时，我们将对比综合结果。方案 1 是将策略 A1 与策略 B 相结合，方案 2 是将策略 A2 与策略 B 相结合。以下图表和讨论显示了如果于 2010 年 6 月 1 日从南极洲区域发出请求输入时的结果。



在方案 1 中，策略 A1 将返回一个默认拒绝，如本节之前所描述的那样。Policy B 返回“允许”结果，因为该策略（依照定义）允许在 2010 年 6 月 1 日发送请求。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。

在方案 2 中，策略 B2 返回了一个显式拒绝，如本节之前所描述的那样。此外，策略 B 返回了一个允许。从策略 A2 发出的显式拒绝将超控从策略 B 发出的允许，因此该请求会被拒绝。

关于访问控制的基本使用案例

Abstract

为访问控制提供典型使用案例。

本节介绍了几个访问控制典型使用案例。

使用案例 1

假设您在 Amazon SQS 系统中拥有一个队列集。在最简单的情况下，您可能希望授予一个或多个 AWS 账户特定访问类型，以访问某一队列（例如 `SendMessage`、`ReceiveMessage`）。

通过 Amazon SQS API 操作 `AddPermission` 即可轻松做到这一点。此操作需要几个输入参数，并且会在 Amazon SQS 系统中自动针对该队列创建一个策略。对于该使用案例而言，因为 Amazon SQS 可为您自动创建策略，因此，您无需阅读本附录，亦无需了解如何自行编写策略。

下列示例显示策略授予 AWS 账户 ID 1111-2222-3333 许可，以从您的队列（名为 `queue2`）发送和接收请求。在此示例中，您的 AWS 账户 ID 为 4444-5555-6666。

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

使用案例 2

在这一使用案例中，您希望仅在指定时段内允许一个或多个 AWS 账户访问您的队列。

您需要了解如何针对队列自行编写策略，因为当授权某人访问您的队列时，Amazon SQS `AddPermission` 操作禁止您执行指定时限的操作。在这一案例中，您需要自行编写策略，然后利用 `SetQueueAttributes` 操作将其上传至 AWS 系统。该操作会有效地将您的策略设置为队列属性。

下列示例与使用案例 1 相同，但有一点不同的是其包括一个“在 2009 年 6 月 30 日中午之前 (UTC) 限制访问”的条件。

```
{
  "Version": "2012-10-17",
  "Id": "UseCase2",
  "Statement" : [
    {
      "Sid": "1",
```

```
    "Effect": "Allow",
    "Principal" : {
        "AWS": "111122223333"
    },
    "Action": ["sqs:SendMessage", "sqs:ReceiveMessage"],
    "Resource": "arn:aws:sqs:us-east-1:444455556666:queue2",
    "Condition" : {
        "DateLessThan" : {
            "AWS:CurrentTime": "2009-06-30T12:00Z"
        }
    }
}
]
```

使用案例 3

在这一使用案例中，您希望仅在由 *Amazon EC2* 实例发出请求时 允许访问您的队列。

再次重申，您需要了解如何自行编写策略，因为当授权访问您的队列时，Amazon SQS `AddPermission` 操作禁止您执行指定 IP 地址限制的操作。

下列示例以使用案例 2 中的示例为基础，同时还包括一个“限制 IP 范围 10.52.176.0/24 的访问”的条件。因此，在此示例中，AWS 账户 1111-2222-3333 只有在 2009 年 6 月 30 日中午之前从地址范围为 10.52.176.0/24 的 IP 地址发出向或从 queue2 发送或接收消息的请求，请求才能得到允许。

```
{
  "Version": "2012-10-17",
  "Id": "UseCase3",
  "Statement" : [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Principal" : {
        "AWS": "111122223333"
      },
      "Action": ["sqs:SendMessage", "sqs:ReceiveMessage"],
      "Resource": "arn:aws:sqs:us-east-1:444455556666:queue2",
      "Condition" : {
        "DateLessThan" : {
          "AWS:CurrentTime": "2009-06-30T12:00Z"
        },
        "IpAddress" : {
          "AWS:SourceIp": "10.52.176.0/24"
        }
      }
    }
  ]
}
```

使用案例 4

在这一使用案例中，您希望 *拒绝* 某一特定 AWS 账户访问您的队列。

再次重申，您需要了解如何自行编写策略，因为 Amazon SQS `AddPermission` 操作禁止您执行 *拒绝* 访问队列的操作，而您只能授予访问权限。

下列示例与第一个使用案例 (#1) 相同，但有一点不同的是其*拒绝*访问指定的 AWS 账户。

```
{
  "Version": "2012-10-17",
  "Id": "UseCase4",
  "Statement" : [
    {
      "Sid": "1",
      "Effect": "Deny",
      "Principal" : {
        "AWS": "111122223333"
      },
      "Action": [ "sqs:SendMessage", "sqs:ReceiveMessage" ],
      "Resource": "arn:aws:sqs:us-east-1:444455556666:queue2",
    }
  ]
}
```

从这些使用案例中，您会发现当您希望执行基于特别条件的限制访问或完全拒绝某人的访问时，需要阅读本附录，并了解如何自行编写策略。您还会发现，策略本身并不复杂，访问策略语言也很简单直接。

Amazon SQS 策略示例

Abstract

为 Amazon SQS 常用案例提供示例策略。

本部分显示了 Amazon SQS 常用案例的示例策略。

以下示例策略向 AWS 账号为 111122223333 的开发人员授予对 US East (N. Virginia) 区域中名为 444455556666/queue1 的队列的 SendMessage 权限。

```
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [
    {
      "Sid": "Queue1_SendMessage",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111122223333"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:444455556666:queue1"
    }
  ]
}
```

以下策略示例向 AWS 账户号为 111122223333 的开发人员授予 SendMessage 和 ReceiveMessage 权限，允许上述开发人员在名为 444455556666/queue1 的队列中进行上述操作。

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

以下示例策略向两个不同的开发人员 (AWS 账号分别为 111122223333 和 444455556666) 授予对 US East (N. Virginia) 区域中名为 123456789012/queue1 的队列使用 Amazon SQS 允许共享访问的所有操作的权限。

```
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [
    {
      "Sid": "Queue1_AllActions",
      "Effect": "Allow",

```

```
    "Principal": {
      "AWS": [ "111122223333", "444455556666" ]
    },
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:us-east-1:123456789012:queue1"
  }
}
```

以下示例策略向所有用户授予对名为 111122223333/queue1 的队列的 `ReceiveMessage` 权限。

```
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [
    {
      "Sid": "Queue1_AnonymousAccess_ReceiveMessage",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:ReceiveMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1"
    }
  ]
}
```

以下示例策略向所有用户授予对名为 111122223333/queue1 的队列的 `ReceiveMessage` 权限，但使用时间仅限于 2009 年 1 月 31 日正午 12:00 至下午 3:00 期间。

```
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [
    {
      "Sid": "Queue1_AnonymousAccess_ReceiveMessage_TimeLimit",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:ReceiveMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition": {
        "DateGreaterThan": {
          "aws:CurrentTime": "2009-01-31T12:00Z"
        },
        "DateLessThan": {
          "aws:CurrentTime": "2009-01-31T15:00Z"
        }
      }
    }
  ]
}
```

以下示例策略向所有用户授予对名为 111122223333/queue1 的队列使用可以共享的所有可能的 Amazon SQS 操作的权限，但条件是请求必须来自于 192.168.143.0/24 范围。

```
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [
    {

```

```
        "Sid": "Queue1_AnonymousAccess_AllActions_WhitelistIP",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "sqs:*",
        "Resource": "arn:aws:sqs:*:111122223333:queue1",
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": "192.168.143.0/24"
            }
        }
    }
}
```

以下策略示例具有两项陈述：

- 一组语句是向 192.168.143.0/24 范围（192.168.143.188 除外）内的所有用户授予对名为 111122223333/queue1 的队列使用 SendMessage 操作的权限。
- 一组语句是阻止 10.1.2.0/24 范围内的所有用户使用该队列。

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

以下示例策略实现了 Amazon 资源名称 (ARN) `arn:aws:sns:us-east-1:111122223333:test-topic` 指定的 Amazon Simple Notification Service 主题与名为 `arn:aws:sqs:us-east-1:111122223333:test-topic-queue` 的队列之间的连接。

```
{
  "Version": "2012-10-17",
  "Id": "SNStoSQS",
  "Statement":
  {
    "Sid": "rule1",
    "Effect": "Allow",
    "Principal": "*",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:us-east-1:111122223333:test-topic-queue",
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:sns:us-east-1:111122223333:test-topic"
      }
    }
  }
}
```

Amazon SQS 策略的特别信息

Abstract

列出对 Amazon SQS 实现访问控制策略时需要了解的特定信息。

以下列表提供了特定于访问控制的 Amazon SQS 实施的信息。

- Amazon SQS 允许您仅共享某些类型的权限（有关更多信息，请参阅[“了解资源级权限 \(p. 86\)”](#)）
- 每个策略都必须仅覆盖单一队列（在编写一个策略时，请勿包括覆盖不同队列的语句）
- 每个策略必须有一个唯一的策略 ID (`Id`)
- 策略中的每个语句必须有一个唯一的语句 ID (`sid`)
- 在您编写条件时，Amazon SQS 不会实施任何要使用的特殊密钥；可用的密钥只是常规 AWS 范围内的密钥。

以下表格列举了策略信息的最大限值。

名称	最大限值
字节	8 192
语句	20
委托人	50
条件	10

使用 AWS Identity and Access Management (IAM) 进行访问控制

Abstract

使用 Amazon SQS 策略或 IAM 策略控制对 Amazon SQS 资源的访问。

主题

- [Amazon SQS 策略的 IAM 相关功能 \(p. 115\)](#)
- [IAM 和 Amazon SQS 策略一起使用 \(p. 116\)](#)
- [Amazon SQS ARN \(p. 118\)](#)
- [Amazon SQS 操作 \(p. 119\)](#)
- [Amazon SQS 密钥 \(p. 119\)](#)
- [Amazon SQS 的 IAM 策略示例 \(p. 120\)](#)
- [使用临时安全证书 \(p. 121\)](#)

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) 策略的同一语言编写的策略。这意味着，使用 Amazon SQS 策略可以达到与使用 IAM 策略（例如，在 IAM 策略中使用变量）相同的效果。有关更多信息，请参阅《[使用 IAM](#)》指南中的“[策略变量](#)”部分。

使用 Amazon SQS 策略与使用 IAM 策略的主要区别在于：您可以使用 Amazon SQS 策略向另一个 AWS 账户授予访问您队列的权限，而使用 IAM 策略则不行。



Note

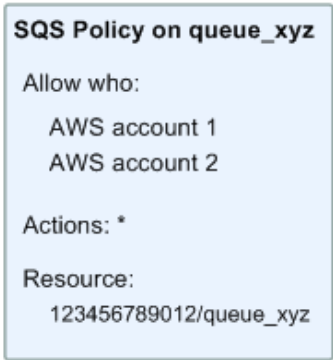
当您向其他 AWS 账户授予访问您 AWS 资源的权限时，请注意所有 AWS 账户都可以向其账户下的用户授予权限。这称为交叉账户访问。交叉账户访问能让你共享 AWS 资源，不需要管理新增的用户。有关使用交叉账户访问的信息，请参见“[IAM 用户指南](#)”中的“[Enabling Cross-Account Access](#)”。

本部分介绍了 Amazon SQS 策略系统如何与 IAM 搭配工作。

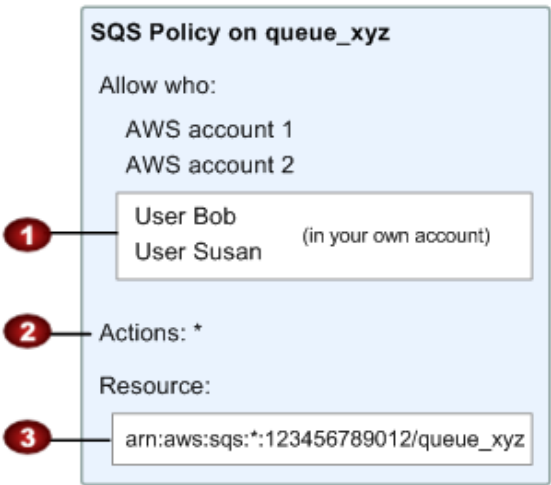
Amazon SQS 策略的 IAM 相关功能

您可以对队列使用 Amazon SQS 策略，以指定哪些 AWS 账户拥有访问该队列的权限。您可以指定访问类型和条件（例如，使用 `SendMessage` 和 `ReceiveMessage` 的权限；如果请求早于 2010 年 12 月 31 日）。您可以为其授予权限的特定操作是整个 Amazon SQS 操作列表的子集。如果您编写 Amazon SQS 策略并指定 * 以表示“所有 Amazon SQS 操作”，即表示该子集中的所有操作。

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 授予对队列使用任何允许的操作的权限。请注意，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 以及用户和 Amazon 资源名称 (ARN) 概念的推出，SQS 策略发生了一些变化。以下示意图和表格描述了这些变化。



1	除了指定哪些 AWS 账户拥有访问该队列的权限以外，您还可以指定您自己的 AWS 账户中哪些用户拥有访问该队列的权限。 这些用户不能位于另一个 AWS 账户中。
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 操作 (p. 119) ）。

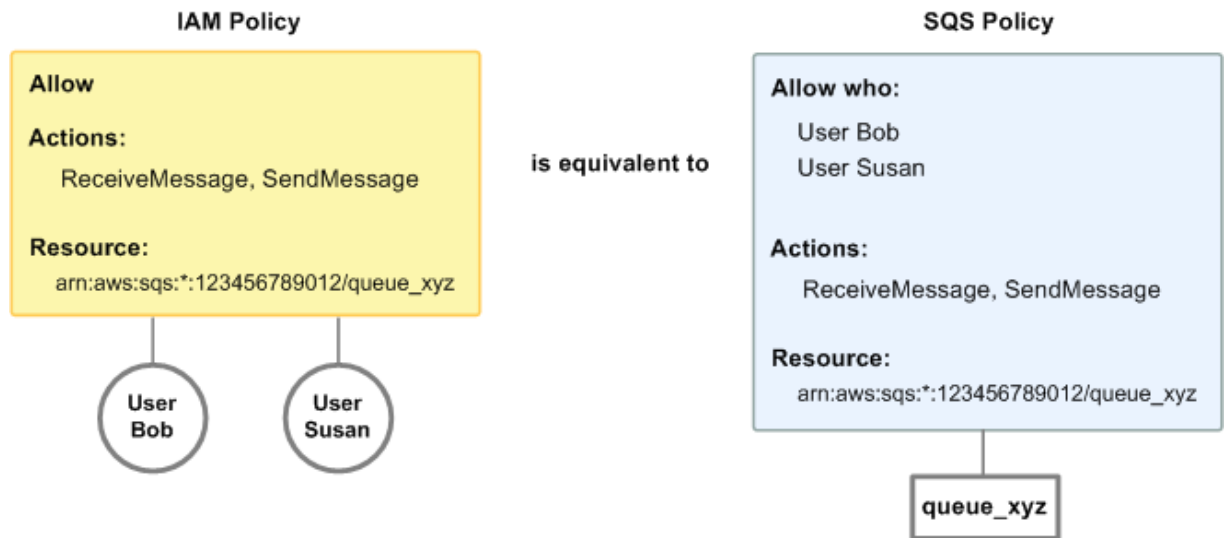
3	您可以使用 <i>Amazon 资源名称 (ARN)</i> 指定资源，这是您在 IAM 策略中必须指定资源的方法。有关 Amazon SQS 队列的 ARN 格式的信息，请参阅“ Amazon SQS ARN (p. 118) ”。您仍可以改用原始格式 (<account_ID>/<queue_name>)。
---	--

例如，根据上图中所示的 Amazon SQS 策略，拥有 AWS 账户 1 或 AWS 账户 2 的安全证书的任何人都可以访问 queue_xyz。此外，您自己的 AWS 账户（ID 为 123456789012）中的用户 Bob 和 Susan 也可以访问该队列。

在推出 IAM 之前，Amazon SQS 会自动向某个队列的创建者授予对该队列的完全控制权（例如，访问针对该队列的所有可能的 Amazon SQS 操作）。除非创建者使用的是 AWS 安全证书，否则上述情况将不会再出现。此外，任何有权创建队列的用户还必须有权使用其他 Amazon SQS 操作，这样才能对自己创建的队列执行任何操作。

IAM 和 Amazon SQS 策略一起使用

您向您的用户授予访问您的 Amazon SQS 资源的权限有两种方式：通过 Amazon SQS 策略系统或 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 策略允许对您 AWS 账户中名为 queue_xyz 的队列执行 Amazon SQS `ReceiveMessage` 和 `SendMessage` 操作，并且它附加到用户 Bob 和 Susan（这意味着，Bob 和 Susan 拥有该策略中所述的权限）。此外，Amazon SQS 策略还向 Bob 和 Susan 授予对同一队列访问 `ReceiveMessage` 和 `SendMessage` 的权限。



Note

前面的示例显示了不带任何条件的简单策略。您可以在上述任一策略中指定特定条件，并获得同样的结果。

IAM 和 Amazon SQS 策略之间有一个区别：Amazon SQS 策略系统允许您向其他 AWS 账户授予权限，而 IAM 则不允许。

将由您自己决定是否上述两种系统管理您的权限，您可以根据自身需求做出决定。以下示例展示这两种策略系统是如何共同运行的。

1

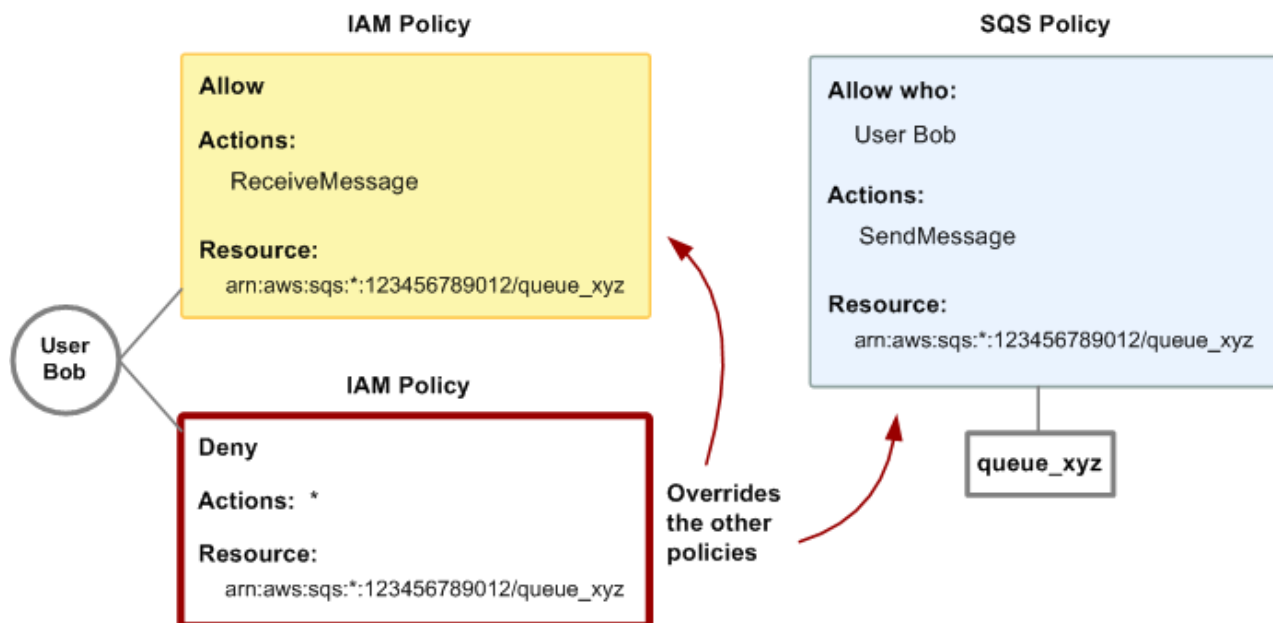
在本例中，Bob 同时拥有适用于他的 IAM 策略和 Amazon SQS 策略。IAM 策略向他授予对 queue_xyz 使用 `ReceiveMessage` 的权限，而 Amazon SQS 策略则向他授予对同一队列使用 `SendMessage` 的权限。下图阐明了这一概念。



如果 Bob 要发送从 queue_xyz 接收消息的请求，则 IAM 策略将允许该操作。如果 Bob 要发送向 queue_xyz 发送消息的请求，则 Amazon SQS 策略将允许该操作。

2

在本示例中，我们基于示例 1（其中，Bob 拥有两个适用于他的策略）来进行描述。假设 Bob 滥用他对 queue_xyz 的访问权限，因此，您希望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拒绝他对该队列访问所有操作的策略。此策略会覆盖其他两个策略，因为显式拒绝始终会覆盖允许（有关策略评估逻辑的更多信息，请参阅[“评估逻辑 \(p. 102\)”](#)）。下图阐明了这一概念。



或者，您可以向 Amazon SQS 策略中添加一条额外的语句，该语句拒绝 Bob 以任何方式访问该队列。此操作与添加拒绝他访问该队列的 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 的 IAM 策略示例 \(p. 120\)”](#)。有关编写 Amazon SQS 策略的更多信息，请转到《[Amazon Simple Queue Service 开发人员指南](#)》。

Amazon SQS ARN

对于 Amazon SQS，队列是您可以在策略中指定的唯一资源类型。以下是队列的 Amazon 资源名称 (ARN) 格式：

```
arn:aws:sqs:region:account_ID:queue_name
```

有关 ARN 的更多信息，请访问《[使用 IAM](#)》中的“[IAM ARN](#)”部分。

以下是 US East (N. Virginia) 区域中名为 my_queue 的队列的 ARN，它属于 AWS 账户 123456789012。

```
arn:aws:sqs:us-east-1:123456789012:my_queue
```

如果您在 Amazon SQS 支持的各个不同地区中均有一个名为 my_queue 的队列，则您可以使用以下 ARN 指定这些队列。

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

您可以在主题名称中使用 * 和 ? 通配符。例如，以下语句可以引用 Bob 创建的所有队列，这些队列的前缀为 bob_。

```
arn:aws:sqs:*:123456789012:bob_*
```

为了方便起见，Amazon SQS 有一个名为 `Arn` 的队列属性，其值为队列的 ARN。您可以通过调用 Amazon SQS `GetQueueAttributes` 操作来获取该值。

Amazon SQS 操作

您在策略中指定的所有 Amazon SQS 操作都必须以小写字符串 `sqs:` 作为前缀。例如，`sqs:CreateQueue`。

在推出 IAM 之前，您可以对队列使用 Amazon SQS 策略，以指定哪些 AWS 账户拥有访问该队列的权限。此外，您还可以指定访问类型（例如，`sqs:SendMessage`、`sqs:ReceiveMessage`，等等）。您可以为其授予权限的特定操作是整个 Amazon SQS 操作集的子集。如果您编写 Amazon SQS 策略并指定 `*` 以表示“所有 Amazon SQS 操作”，即表示该子集中的所有操作。该子集最初包括：

- `sqs:SendMessage`
- `sqs:ReceiveMessage`
- `sqs:ChangeMessageVisibility`
- `sqs>DeleteMessage`
- `sqs:GetQueueAttributes`（适用于除 `Policy` 以外的所有属性）
- `sqs:GetQueueUrl`

随着 IAM 的推出，该操作列表扩展到包括以下操作：

- `sqs:CreateQueue`
- `sqs>DeleteQueue`
- `sqs:ListQueues`

与向队列授予权限以及从队列删除权限相关的操作（`sqs:AddPermission` 和 `sqs:RemovePermission`）已保留，因此没有显示在前面的两个列表中。这意味着，AWS 账户中的用户不能使用这些操作。但是，AWS 账户可以使用这些操作。

Amazon SQS 密钥

Amazon SQS 实施了以下策略密钥，但没有实施其他策略密钥。

适用整个 AWS 范围的策略密钥

- `aws:CurrentTime`—检查日期/时间条件。
- `aws:EpochTime`—使用纪元日期或 UNIX 时间检查日期/时间条件。
- `aws:MultiFactorAuthAge`—使用 Multi-Factor Authentication (MFA) 查看多久之前下发了经 MFA 验证的发出请求之安全证书（以秒计）。与其他密钥不同，若 MFA 未使用，则表示该密钥不存在。
- `aws:principaltype`—要检查委托人类型（用户、账户、联合用户等）。
- `aws:SecureTransport`—检查请求是否使用 SSL 发送的。对于仅使用 SSL 的服务（如 Amazon RDS 和 Amazon Route 53），`aws:SecureTransport` 密钥没有意义。
- `aws:SourceArn`—使用来源的 Amazon 资源名称 (ARN) 查看请求的来源。（该值仅适用于某些产品。更多信息，请参阅 [Amazon Simple Queue Service 开发人员指南](#) 中“Element Descriptions”部分的 *Amazon Resource Name (ARN)*。）
- `aws:SourceIp`—检查请求者的 IP 地址。请注意，如果您使用 `aws:SourceIp`，并且请求是来自 Amazon EC2 实例，则会评估实例的公有 IP 地址。
- `aws:UserAgent`—检查发出请求的客户端应用程序。
- `aws:useruid`—检查请求者的用户 ID。

- `aws:username`—检查请求者的用户名称（如果可用）。



Note

密钥名称区分大小写。

Amazon SQS 的 IAM 策略示例

此部分演示了用于控制用户访问 Amazon SQS 的几个简单的 IAM 策略。



Note

未来，根据策略陈述的目标，Amazon SQS 可能会添加逻辑上包含在以下策略之一中的新操作。

1：允许用户创建和使用自己的队列

在本例中，我们为 Bob 创建了一个策略，该策略允许他访问所有 Amazon SQS 操作，但是仅限于针对名称以文本字符串 `bob_queue` 开头的队列。



Note

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

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

2：允许开发人员向共享测试队列写入消息

在本例中，我们为开发人员创建了一个组，并附加了一个策略，该策略允许该组使用 Amazon SQS `SendMessage` 操作，但是仅限于针对名为 `CompanyTestQueue` 的 AWS 账户队列。

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

3：允许管理人员获取队列的一般大小

在本例中，我们为管理人员创建了一个组，并附加了一个策略，该策略允许该组对所有 AWS 账户队列使用 Amazon SQS `GetQueueAttributes` 操作。

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

4：允许合作伙伴向特定队列发送消息

您可以使用 Amazon SQS 策略或 IAM 策略来执行此操作。如果合作伙伴拥有 AWS 账户，则使用 Amazon SQS 策略可能更容易。但是，合作伙伴公司中拥有 AWS 安全证书的任何人（而不只是特定用户）都可以向该队列发送消息。假设您希望仅向特定人员（或应用程序）授予访问权限，则需要像对待您自己公司内的用户那样对待合作伙伴，并使用 IAM 策略（而不是 Amazon SQS 策略）。

在本示例中，我们创建了一个名为 `WidgetCo` 的组（代表合作伙伴公司），接着为需要访问权限的合作伙伴公司特定人员（或应用程序）创建了一个用户，然后将该用户放入该组。

随后，我们附加了一个策略，该策略向该组授予对名为 `WidgetPartnerQueue` 的特定队列的 `SendMessage` 访问权限。

此外，我们还希望阻止 `WidgetCo` 组对队列执行其他任何操作，因此，我们添加了一条语句，该语句拒绝该组对 `WidgetPartnerQueue` 以外的任何队列执行 `SendMessage` 以外的任何 Amazon SQS 操作。只有在系统中的其他地方存在广泛策略（该策略向用户授予广泛访问 Amazon SQS 的权限）时，才需要执行此操作。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
  },
  {
    "Effect": "Deny",
    "NotAction": "sqs:SendMessage",
    "NotResource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
  }
]
```

使用临时安全证书

除了创建有其自己的安全证书的 IAM 用户之外，IAM 还可以让您向任何用户授予临时安全证书，从而使此用户可以访问您的 AWS 服务和资源。您可以管理有 AWS 账户的用户；这些用户是 IAM 用户。您还可以对您系统中没有 AWS 账户的用户进行管理；这些用户被称为联合用户。此外，“用户”还可以是您创建的能访问您的 AWS 资源的应用程序。

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

首先，使用 IAM 创建临时安全证书，该证书包含安全令牌、访问密钥 ID 和私有访问密钥。其次，准备要使用临时访问密钥 ID 和安全令牌进行签名的字符串。然后，使用临时私有访问密钥（而不是您自己的私

有访问密钥) 对您的查询 API 请求签名。最后, 在提交签名的查询 API 请求时, 别忘了使用临时访问密钥 ID (而不是您自己的访问密钥 ID), 并且包含安全令牌。有关 IAM 对临时安全证书的支持的更多信息, 请转到《使用 IAM》中的“[授予对您 AWS 资源的临时访问权限](#)”部分。

使用临时安全证书调用 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, 请转到《IAM 用户指南》中的“[创建临时安全证书, 为 IAM 用户启用访问](#)”部分。
IAM 会返回一个安全令牌、一个访问密钥 ID 和一个私有访问密钥。
2. 像通常那样准备查询, 但要使用临时访问密钥 ID (而不是您自己的访问密钥 ID), 并且包含安全令牌。使用临时私有访问密钥 (而不是您自己的私有访问密钥) 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。

您如何构造 AUTHPARAMS 取决于您如何签署您的 API 请求。有关签名版本 4 中 AUTHPARAMS 的信息, 请转至[已签名的签名版本 4 请求示例](#)。

```
http://sqs.us-east-1.amazonaws.com/  
?Action=CreateQueue  
&DefaultVisibilityTimeout=40  
&QueueName=testQueue  
&Attribute.1.Name=VisibilityTimeout  
&Attribute.1.Value=40  
&Version=2012-11-05  
&Expires=2015-12-18T22%3A52%3A43PST  
&SecurityToken=SecurityTokenValue  
&AWSAccessKeyId= Access Key ID provided by AWS Security Token Service  
&AUTHPARAMS
```

以下示例使用了临时安全证书来通过 SendMessageBatch 发送两条消息。

```
http://sqs.us-east-1.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  
&Version=2012-11-05  
&Expires=2015-12-18T22%3A52%3A43PST  
&SecurityToken=SecurityTokenValue  
&AWSAccessKeyId=Access Key ID provided by AWS Security Token Service  
&AUTHPARAMS
```

为队列订阅 Amazon SNS 主题

Abstract

通过使用 Amazon SQS 的 AWS 管理控制台 为 Amazon SQS 队列订阅 Amazon SNS 主题，实现流程简化。

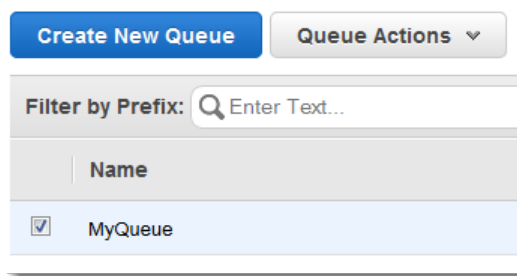
现在，您可以使用 Amazon SQS 的 AWS 管理控制台来为 Amazon SQS 队列订阅 Amazon SNS 主题，该控制台简化了该过程。例如，您可以从可用于选定队列的主题列表中进行选择。随后，Amazon SQS 会负责为队列订阅主题，并添加必要的权限。消息发布到主题后，Amazon SNS 会向订阅的队列发送 Amazon SQS 消息。有关 Amazon SNS 的更多信息，请参阅《[Amazon SNS 入门](#)》。有关 Amazon SQS 的更多信息，请参阅《[Amazon SQS 入门](#)》。

使用 AWS 管理控制台为队列订阅 Amazon SNS 主题

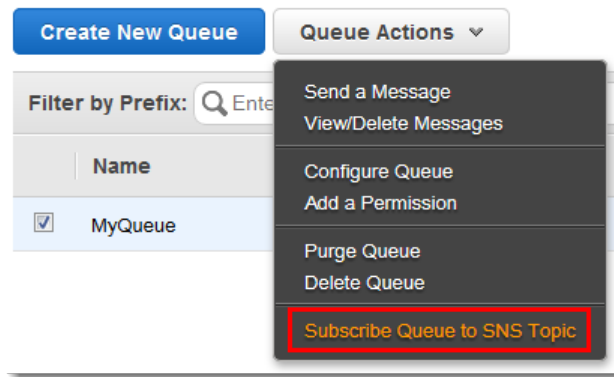
以下步骤假设您已经创建了队列和 Amazon SNS 主题。

使用 AWS 管理控制台为队列订阅 Amazon SNS 主题

1. 登录 AWS 管理控制台 并通过以下网址打开 Amazon SQS 控制台：<https://console.amazonaws.cn/sqs/>。
2. 选择要为其订阅 Amazon SNS 主题的队列。



3. 从“Queue Actions”下拉列表中选择“Subscribe Queue to SNS Topic”。

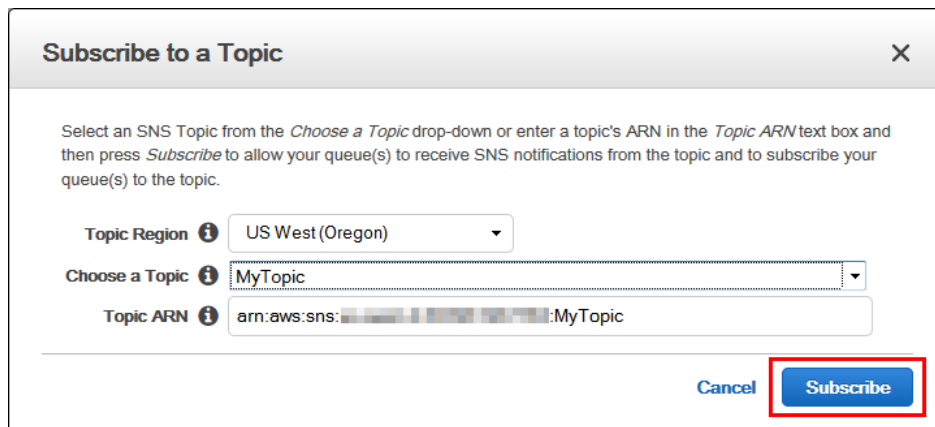


4. 从 Choose a Topic (选择一个主题) 下拉列表中选择要为队列订阅的 Amazon SNS 主题，然后单击 Subscribe (订阅)。

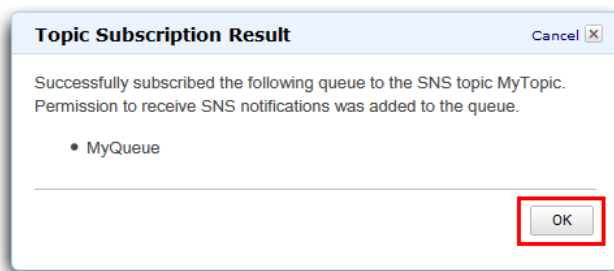


Note

您还可以在 Topic ARN: (主题 ARN:) 框中输入 Amazon SNS 主题的 ARN。如果您希望从某个 AWS 账户 (该账户不同于您用来创建该队列的账户) 为队列订阅 Amazon SNS 主题，则此操作非常有用。如果 Amazon SNS 主题未在 Choose a Topic (选择一个主题) 下拉列表中列出，则此操作也非常有用。



5. 在“Topic Subscription Result”对话框中，单击“OK”。



通过向主题发布消息，然后查看主题向队列发送的消息，您可以验证主题的队列订阅结果。有关详细步骤，请参阅[“通过向主题发布消息，然后阅读来自队列的消息，对其进行测试”](#)。

使用 CloudWatch 监控 Amazon SQS

Abstract

使用 CloudWatch 控制台、CloudWatch 自己的命令行界面或者以编程方式使用 CloudWatch API 来监控 Amazon SQS 的指标。

Amazon SQS 和 CloudWatch 相集成，因此，您可以使用 CloudWatch 来轻松收集、查看和分析 Amazon SQS 队列的指标。为 Amazon SQS 配置 CloudWatch 后，您可以更好地了解您 Amazon SQS 队列和应用程序的性能。例如，您可以监控 NumberOfEmptyReceives 指标，以确保您的应用程序不会耗费太多时间来轮询新消息。此外，您还可以设置一个报警器，使其在 Amazon SQS 指标（例如，NumberOfMessagesReceived）达到指定的阈值时，向您发送电子邮件通知。有关 Amazon SQS 发送到 CloudWatch 的所有指标的列表，请参阅“[Amazon SQS 指标 \(p. 126\)](#)”。

您使用 CloudWatch 为 Amazon SQS 队列配置的指标将每五分钟自动收集一次并推送到 CloudWatch。详细监控或 1 分钟指标目前不适用于 Amazon SQS。系统会对满足 CloudWatch 认为队列有效的准则的所有队列收集这些指标。从某个队列上次活动（即，任何 API 调用）以来最长六小时内，CloudWatch 会认为该队列有效。



Note

CloudWatch 中报告的 Amazon SQS 指标是不收费的；它们是作为 Amazon SQS 产品的一部分提供的。

访问 Amazon SQS 的 CloudWatch 指标

您可以使用 CloudWatch 控制台、CloudWatch 自己的命令行界面 (CLI) 或者以编程方式使用 CloudWatch API 来监控 Amazon SQS 的指标。下列程序告诉您使用不同的方式如何获得指标。

使用 CloudWatch 控制台查看指标

1. 登录 AWS 管理控制台 并通过以下网址打开 CloudWatch 控制台 <https://console.amazonaws.cn/cloudwatch/>。
2. 单击“View Metrics”。
3. 从“Viewing”下拉菜单中选择“SQS: Queue Metrics”，以显示每个队列的可用指标。
4. 单击“MetricName”列中的特定指标以查看更多详细信息，例如所收集数据的图表。

从 CloudWatch CLI 访问指标

- 调用 `mon-get-stats`。您可以在《[Amazon CloudWatch 开发人员指南](#)》中了解更多有关此函数以及其他与指标相关的函数的信息。

从 CloudWatch API 访问指标

- 调用 `GetMetricStatistics`。您可在 [Amazon CloudWatch API Reference](#) 了解更多有关这个或其他与指标相关功能的信息。

为 Amazon SQS 指标设置 CloudWatch 报警器

此外，CloudWatch 还允许您设置指标达到阈值时的警报。例如，您可能会为指标 `NumberOfEmptyReceives` 设置报警器，这样，如果该指标在采样周期内达到您指定的阈值数字，则系统会发送电子邮件通知以告知您该事件。

使用 CloudWatch 控制台设置警报

1. 登录 AWS 管理控制台 并通过以下网址打开 CloudWatch 控制台 <https://console.amazonaws.cn/cloudwatch/>。
2. 请点击“Alarms”，然后请点击“Create Alarm”按钮。这样会启动“Create Alarm Wizard”。
3. 滚动 Amazon SQS 指标以找到您要为其设置报警器的指标。选择指标创建一个报警器并点击“Continue”。
4. 填写指标的“Name”、“Description”、“Threshold”和“Time”值，然后单击“Continue”。
5. 按照报警器说明选择“Alarm”。当达到报警状态，您想 CloudWatch 发给您一封电子邮件，您可或者选择先前存在的 Amazon SNS 主题或者点击“Create New Email Topic”。如果您单击 Create New Email Topic (新建电子邮件主题)，则您可以为一个新的主题设置名称和电子邮件地址。此清单将会被保存下来并在将来报警器的下列框显示。单击继续。



Note

如果您使用 Create New Email Topic (新建电子邮件主题) 创建新的 Amazon SNS 主题，那么电子邮件地址在接收通知之前必须通过验证。当报警器进入报警状态时，才发送电子邮件。如果在电子邮件地址验证之前报警状态发生变化，那么他们不会收到通知。

6. 此时，“Create Alarm Wizard”会给您一次机会检查您即将创建的报警器。如果您需要进行任何更改，则可以使用右侧的“Edit”链接。感觉满意后，单击“Create Alarm”。

有关更多使用 CloudWatch 和报警器的信息，参阅“[CloudWatch Documentation](#)”。

Amazon SQS 指标

Amazon SQS 会向 CloudWatch 发送以下指标。

指标	说明
<code>NumberOfMessagesSent</code>	添加到队列的消息数量。 单位：计数 有效统计数据：Sum

指标	说明
SentMessageSize	添加到队列的消息大小。 单位：字节 有效统计数据：最低、最高、平均和计数
NumberOfMessagesReceived	调用 <code>ReceiveMessage</code> API 操作返回的消息数量。 单位：计数 有效统计数据：Sum
NumberOfEmptyReceives	未返回消息的 <code>ReceiveMessage</code> API 调用数量。 单位：计数 有效统计数据：Sum
NumberOfMessagesDeleted	从队列删除的消息数量。 单位：计数 有效统计数据：Sum
ApproximateNumberOfMessagesDelayed	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。 单位：计数 有效统计数据：Average
ApproximateNumberOfMessagesVisible	可从队列取回的消息数量。 单位：计数 有效统计数据：Average
ApproximateNumberOfMessagesNotVisible	“处于飞行状态”的消息数量。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，则消息被认为处于飞行状态。 单位：计数 有效统计数据：Average

使用 AWS CloudTrail 记录 Amazon SQS API 调用

Abstract

介绍如何使用 AWS CloudTrail 记录 Amazon SQS API 调用。

Amazon SQS 与 CloudTrail 集成，后者是一种服务，它在 AWS 账户中捕获 Amazon SQS 或代表 Amazon SQS 发出的 API 调用，并将日志文件提交到您指定的 Amazon S3 存储桶。CloudTrail 捕获从 Amazon SQS 控制台或从 Amazon SQS API 发出的 API 调用。通过使用 CloudTrail 收集的信息，您可以确定对 Amazon SQS 发出了什么请求、发出请求的源 IP 地址、何人发出的请求以及发出请求的时间等。要了解有关 CloudTrail 的更多信息，包括如何对其进行配置和启用，请参阅 [AWS CloudTrail User Guide](#)

CloudTrail 中的 Amazon SQS 信息

在您的 AWS 账户中启用 CloudTrail 日志记录后，将在日志文件中跟踪对 Amazon SQS 操作进行的 API 调用。Amazon SQS 记录与其他 AWS 服务记录一起写入日志文件中。CloudTrail 基于时间段和文件大小来确定何时创建新文件并向其写入内容。

支持以下操作：

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

每个日志条目都包含有关生成请求的人员的信息。日志中的用户身份信息有助于确定请求是由根或 IAM 用户证书发出，通过某个角色或联合用户的临时安全证书发出，还是由其他 AWS 服务发出。有关更多信息，请参阅 [CloudTrail 事件参考](#) 中的 `userIdentity` 字段。

日志文件可以在存储桶中存储任意长时间，不过您也可以定义 Amazon S3 生命周期规则以自动存档或删除日志文件。默认情况下，将使用 Amazon S3 服务器端加密 (SSE) 对日志文件进行加密。

如果需要针对日志文件传输快速采取措施，可选择让 CloudTrail 在传输新日志文件时发布 Amazon SNS 通知。有关更多信息，请参阅[配置 Amazon SNS 通知](#)。

您也可以将多个 AWS 区域和多个 AWS 账户的 Amazon SQS 日志文件聚合到单个 Amazon S3 存储桶中。更多信息，请参阅[将 CloudTrail 日志文件聚合到单个 Amazon S3 存储桶中](#)。

了解 Amazon SQS 日志文件条目

CloudTrail 日志文件可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。也即，它们不是公用 API 调用的有序堆栈跟踪。

AddPermission

以下示例显示 AddPermission 的一个 CloudTrail 日志条目：

```
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-07-16T00:44:19Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "AddPermission",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0",
      "requestParameters": {
        "actions": [
          "SendMessage"
        ],
        "awsAccountIds": [
          "123456789012"
        ],
        "label": "label",
        "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
      },
      "responseElements": null,
      "requestID": "334cccd-b9bb-50fa-abdb-80f274981d60",
      "eventID": "0552b000-09a3-47d6-a810-c5f9fd2534fe"
    }
  ]
}
```

CreateQueue

以下示例显示 CreateQueue 的一个 CloudTrail 日志条目：

```
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-07-16T00:42:42Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "CreateQueue",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
      "requestParameters": {
        "queueName": "hello1"
      },
      "responseElements": {
        "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
      },
      "requestID": "49ebdbb7-5cd3-5323-8a00-f1889011fee9",
      "eventID": "68f4e71c-4f2f-4625-8378-130ac89660b1"
    }
  ]
}
```

DeleteQueue

以下示例显示 DeleteQueue 的一个 CloudTrail 日志条目：

```
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-07-16T00:44:47Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "DeleteQueue",
      "awsRegion": "us-east-1",
```

```
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
    "requestParameters": {
      "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
    },
    "responseElements": null,
    "requestID": "e4c0cc05-4faa-51d5-aab2-803a8294388d",
    "eventID": "af1bb158-6443-4b4d-abfd-1b867280d964"
  }
]
}
```

RemovePermission

以下示例显示 RemovePermission 的一个 CloudTrail 日志条目：

```
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-07-16T00:44:36Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "RemovePermission",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
      "requestParameters": {
        "label": "label",
        "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
      },
      "responseElements": null,
      "requestID": "48178821-9c2b-5be0-88bf-c41e5118162a",
      "eventID": "fed8a623-3fe9-4e64-9543-586d9e500159"
    }
  ]
}
```

SetQueueAttributes

以下示例显示 SetQueueAttributes 的一个 CloudTrail 日志条目：

```
{
  "Records": [
    {
```



```
    "eventVersion": "1.01",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "EX_PRINCIPAL_ID",
      "arn": "arn:aws:iam::123456789012:user/Alice",
      "accountId": "123456789012",
      "accessKeyId": "EXAMPLE_KEY_ID",
      "userName": "Alice"
    },
    "eventTime": "2014-07-16T00:43:15Z",
    "eventSource": "sqs.amazonaws.com",
    "eventName": "SetQueueAttributes",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
    "requestParameters": {
      "attributes": {
        "VisibilityTimeout": "100"
      },
      "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
    },
    "responseElements": null,
    "requestID": "7f15d706-f3d7-5221-b9ca-9b393f349b79",
    "eventID": "8b6fb2dc-2661-49b1-b328-94317815088b"
  }
]
```

Amazon SQS 资源

Abstract

列出在您使用 Amazon SQS 时可为您提供帮助的相关资源。

下表列出了在您使用此服务时可为您提供帮助的相关资源。

资源	说明
Amazon Simple Queue Service 入门指南	该入门指南基于简单的使用案例，提供了该服务的快速教程。其中包括使用多种编程语言编写的示例和说明。
Amazon Simple Queue Service API Reference	API 参考提供了 WSDL 位置；API 操作、参数和数据类型的完整描述；该服务返回的错误列表。
Amazon SQS 发行说明	发行说明从高层面上概括介绍了当前发行版本的情况。特别说明了一些新功能、修复和已知问题。
Amazon SQS 产品信息	提供了 Amazon SQS 相关信息的主要 Web 页面。
开发论坛	由开发人员组成的社区形式的论坛，他们可以在这里讨论与 Amazon SQS 有关的技术问题。
AWS Premium Support 信息	提供 AWS Premium Support 相关信息的主要 Web 页面，这是一种一对一、快速响应的支持渠道，能帮助您在 AWS 基础设施服务上构建和运行应用程序。

文档历史记录

Abstract

查找 Amazon SQS 开发人员指南的修订日期、相关发行版和重要更改。

下表说明了自 *Amazon Simple Queue Service 开发人员指南* 上次发布以来对该文档所做的重要更改。

- API 版本 2012-11-05
- 最近文档更新时间：2015 年 12 月 7 日

更改	说明	修改日期
更新	更新后的 Amazon SQS 控制台屏幕截图。	2015 年 12 月 7 日
新功能	适用于 Java 的 Amazon SQS 扩展客户端库可让您使用 Amazon S3 管理 Amazon SQS 消息。有关更多信息，请参阅 <i>Amazon SQS 开发人员指南</i> 中的 使用 Amazon S3 管理 Amazon SQS 消息 。	2015 年 10 月 27 日
新功能	Amazon SQS 现在支持结合使用 JMS (Java Message Service) 与 Amazon SQS 队列。有关更多信息，请参阅 <i>Amazon SQS 开发人员指南</i> 中的 结合使用 JMS 与 Amazon SQS 。	2014 年 12 月 29 日
新功能	Amazon SQS 现在允许您使用 <code>PurgeQueue</code> API 删除队列中的消息。有关更多信息，请参阅 <i>Amazon SQS API 参考</i> 中的 PurgeQueue 。	2014 年 12 月 8 日
更新	有关访问密钥的更新信息。有关更多信息，请参阅 您的访问密钥 (p. 78) 。	2014 年 8 月 4 日
新功能	Amazon SQS 现在允许您使用 AWS CloudTrail 记录 API 调用。有关更多信息，请参阅 使用 AWS CloudTrail 记录 Amazon SQS API 调用 (p. 128) 。	2014 年 7 月 16 日
新功能	Amazon SQS 现在为消息属性提供支持。有关更多信息，请参阅 使用 Amazon SQS 消息属性 (p. 36) 。	2014 年 5 月 6 日
新功能	Amazon SQS 现在为死信队列提供支持。有关更多信息，请参阅 使用 Amazon SQS 死信队列 (p. 90) 。	2014 年 1 月 29 日

更改	说明	修改日期
新控制台功能	现在，您可以使用 Amazon SQS 的 AWS 管理控制台来为 Amazon SQS 队列订阅 Amazon SNS 主题，该控制台简化了该过程。有关更多信息，请参阅 为队列订阅 Amazon SNS 主题 (p. 123) 。	2012 年 11 月 21 日
新功能	Amazon SQS 的 2012-11-05 API 版本添加了对签名版本 4 (该版本提高了安全性和性能) 的支持。有关签名版本 4 的更多信息，请参阅 查询请求身份验证 (p. 83) 。	2012 年 11 月 5 日
新功能	AWS SDK for Java 现在包含一个缓冲的异步客户端 AmazonSQSBufferedAsyncClient，以便访问 Amazon SQS。通过启用客户端缓冲 (其中，从客户端发出的调用先进行缓冲，然后再作为批处理请求发送到 Amazon SQS)，此新客户端允许您更轻松地对请求进行批处理。有关客户端缓冲和请求批处理的更多信息，请参阅 客户端缓冲和请求批处理 (p. 64) 。	2012 年 11 月 5 日
新功能	Amazon SQS 的 2012-11-05 API 版本添加了长轮询支持。长轮询允许 Amazon SQS 等待指定的时间，直到消息可用，而不是在消息不可用时返回空响应。有关长轮询的更多信息，请参阅 “Amazon SQS 长轮询 (p. 47)” 。	2012 年 11 月 5 日