消息队列 MQ

用户指南

用户指南

特色消息类型

定时消息和延时消息

本文档主要介绍 MQ 定时消息和延时消息的概念、适用场景以及使用过程中的注意事项。

概念介绍

- 定时消息：Producer 将消息发送到 MQ 服务端，但并不期望这条消息立马投递，而是推迟到在当前时间点之后的某一个时间投递到 Consumer 进行消费，该消息即定时消息。
- 延时消息：Producer 将消息发送到 MQ 服务端，但并不期望这条消息立马投递，而是延迟一定时间后才投递到 Consumer 进行消费，该消息即延时消息。

定时消息与延迟消息在代码配置上存在一些差异，但是最终达到的效果相同：消息在发送到 MQ 服务端后并不会立马投递，而是根据消息中的属性延迟固定时间后才投递给消费者。

适用场景

定时/延时消息适用于如下一些场景：

- 消息生产和消费有时间窗口要求：比如在电商交易中超时未支付关闭订单的场景，在订单创建时会发送一条 MQ 延时消息，这条消息将会在30分钟以后投递给消费者，消费者收到此消息后需要判断对应的订单是否已完成支付。如支付未完成，则关闭订单，如已完成支付则忽略。
- 通过消息触发一些定时任务，比如在某一固定时间点向用户发送提醒消息。

使用方式

定时消息、延时消息的使用在代码编写上存在略微的区别：

- 发送**定时消息**需要明确指定消息发送时间点之后的某一时间点作为消息投递的时间点。
- 发送**延时消息**时需要设定一个延时时间长度，消息将从当前发送时间点开始延迟固定时间之后才开始

投递。

注意事项

1. 定时/延时消息 msg.setStartDeliverTime 的参数需要设置成当前时间戳之后的某个时刻（单位毫秒），如果被设置成当前时间戳之前的某个时刻，消息将立刻投递给消费者。
2. 定时/延时消息 msg.setStartDeliverTime 的参数可设置40天内的任何时刻（单位毫秒），超过40天消息发送将失败。
3. StartDeliverTime 是服务端开始向消费端投递的时间。如果消费者当前有消息堆积，那么定时、延时消息会排在堆积消息后面，将不能严格按照配置的时间进行投递。
4. 由于客户端和服务端可能存在时间差，定时消息/延时消息的投递也可能与客户端设置的时间存在偏差。
5. 设置定时、延时消息的投递时间后，依然受 3 天的消息保存时长限制。 例如，设置定时消息 5 天后才能被消费，如果第 5 天后一直没被消费，那么这条消息将在第8天被删除。
6. 除 TCP 协议接入的 Java 语言支持延时消息，其他方式都不支持延时消息。

示例代码

发送定时消息和延时消息的示例代码请参考以下文档：

TCP 协议接入：

Java

- 发送定时消息
- 发送延时消息

C++

- 发送定时消息

.NET

- 发送定时消息

HTTP 协议接入：

- 发送定时消息

顺序消息

本文主要介绍 MQ 顺序的概念、适用场景以及使用过程中的注意事项。

SDK 支持

请使用 Java SDK 1.2.7 及以上版本。C++/.NET SDK 即将支持。

示例代码请参考以下文档：

- TCP Java 收发送顺序消息
- HTTP 发送顺序消息

概念介绍

顺序消息（FIFO 消息）是 MQ 提供的一种严格按照顺序进行发布和消费的消息类型。顺序消息由两个部分组成：顺序发布和顺序消费。

- **顺序发布**：对于指定的一个 Topic，客户端将按照一定的先后顺序进行发送消息。
- **顺序消费**：对于指定的一个 Topic，按照一定的先后顺序进行接收消息，即先发送的消息一定会先被客户端接收到。

顺序消息类型分为两种：全局顺序和分区顺序。以下分别对 2 种顺序消息类型进行介绍。

全局顺序

对于指定的一个 Topic，所有消息按照严格的先入先出（FIFO）的顺序进行发布和消费。

适用场景：

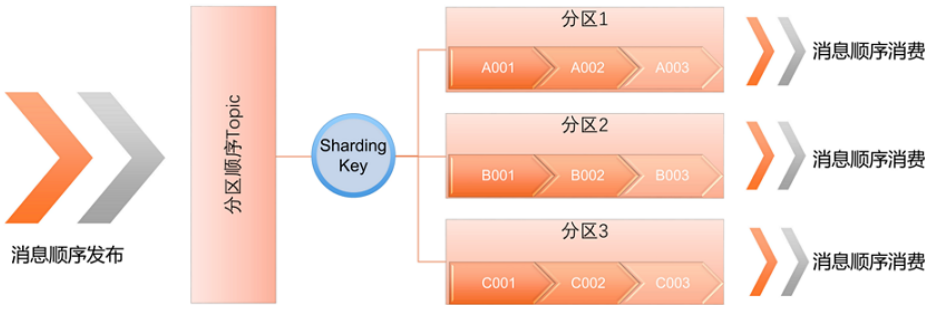
- 性能要求不高，所有的消息严格按照 FIFO 原则进行消息发布和消费的场景。

举例说明：

- 【例一】证券处理中，以人民币兑换美元为 Topic，在价格相同的情况下，先出价者优先处理，则可以通过全局顺序的方式按照 FIFO 的方式进行发布和订阅。

分区顺序

对于指定的一个 Topic，所有消息根据 sharding key 进行区块分区。同一个分区内的消息按照严格的 FIFO 顺序进行发布和消费。Sharding key 是顺序消息中用来区分不同分区的关键字段，和普通消息的 Key 是完全不同的概念。



适用场景：

- 性能要求高，以 sharding key 作为分区字段，在同一个区块中严格的按照 FIFO 原则进行消息发布和消费的场景。

举例说明：

【例一】用户注册需要发送发验证码，以用户 ID 作为 sharding key， 那么同一个用户发送的消息都会按照先后顺序来发布和订阅。

【例二】电商的订单创建，以订单 ID 作为 sharding key，那么同一个订单相关的创建订单消息、订单支付消息、订单退款消息、订单物流消息都会按照先后顺序来发布和订阅。

阿里巴巴集团内部电商系统均使用此种分区顺序消息，既保证业务的顺序，同时又能保证业务的高性能。

全局顺序与分区顺序对比

在控制台创建顺序消息使用的 Topic，各种类型 Topic 对比如下。

消息类型对比

Topic 类型	支持事务消息	支持定时消息	性能
无序消息	是	是	最高
分区顺序	否	否	高
全局顺序	否	否	一般

发送方式对比

消息类型	支持可靠同步发送	支持可靠异步发送	支持 Oneway 发送
无序消息	是	是	是
分区顺序	是	否	否
全局顺序	是	否	否

注意事项

- 顺序消息暂不支持广播模式。
- 同一个 Producer ID 或者 Consumer ID 只能对应一种类型的 Topic，即不能同时用于顺序消息和无序消息的收发。
- 顺序消息不支持异步发送方式，否则将无法严格保证顺序。
- 对于全局顺序消息，建议创建实例个数 >=2。同时运行多个实例的作用是为了防止工作实例意外退出时，业务中断。当工作实例退出时，其他实例可以立即接手工作，不会导致业务中断，实际同时工作的只会有一个实例。
- 对于分区顺序，建议创建实例个数 >=2 并且 <= 分区总数，当工作实例退出时，其他实例可以立即接手工作并重新进行负载均衡，不会导致业务中断，每个实例平均分配分区数。

事务消息

MQ 事务消息

本文档主要介绍 MQ 事务的概念、适用场景以及使用过程中的注意事项。

概念介绍

- 事务消息：MQ 提供类似 X/Open XA 的分布事务功能，通过 MQ 事务消息能达到分布式事务的最终一致。
- 半消息：暂不能投递的消息，发送方已经将消息成功发送到了 MQ 服务端，但是服务端未收到生产者对该消息的二次确认，此时该消息被标记成“暂不能投递”状态，处于该种状态下的消息即半消息。
- 消息回查：由于网络闪断、生产者应用重启等原因，导致某条事务消息的二次确认丢失，MQ 服务端通过扫描发现某条消息长期处于“半消息”时，需要主动向消息生产者询问该消息的最终状态（Commit 或是 Rollback），该过程即消息回查。

适用场景

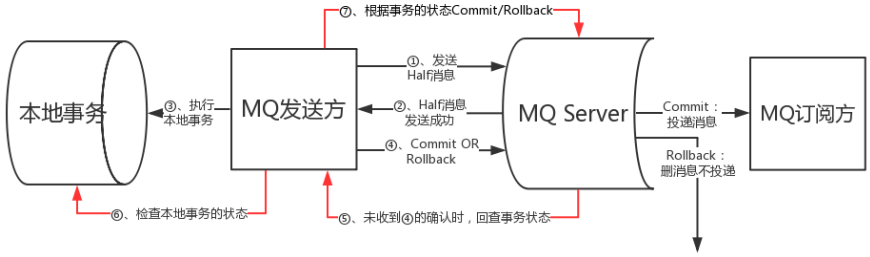
MQ 事务消息适用于如下场景：

- 帮助用户实现类似 X/Open XA 的分布事务功能，通过 MQ 事务消息能达到分布式事务的最终一致。

使用方式

交互流程

MQ 事务消息交互流程如下所示：



其中：

1. 发送方向 MQ 服务端发送消息；
2. MQ Server 将消息持久化成功之后，向发送方 ACK 确认消息已经发送成功，此时消息为半消息。
3. 发送方开始执行本地事务逻辑。
4. 发送方根据本地事务执行结果向 MQ Server 提交二次确认（Commit 或是 Rollback），MQ Server 收到 Commit 状态则将半消息标记为可投递，订阅方最终将收到该消息；MQ Server 收到 Rollback 状态则删除半消息，订阅方将不会接受该消息。
5. 在断网或者是应用重启的特殊情况下，上述步骤4提交的二次确认最终未到达 MQ Server，经过固定时间后 MQ Server 将对该消息发起消息回查。
6. 发送方收到消息回查后，需要检查对应消息的本地事务执行的最终结果。
7. 发送方根据检查得到的本地事务的最终状态再次提交二次确认，MQ Server 仍按照步骤4对半消息进行操作。

事务消息发送对应步骤1、2、3、4，事务消息回查对应步骤5、6、7。

示例代码

- TCP Java 发送事务消息
- TCP C++ 发送事务消息
- TCP .NET 发送事务消息

注意事项

事务消息的 Producer ID 不能与其他类型消息的 Producer ID 共用。与其他类型的消息不同，事务消息有回查机制，回查时 MQ Server 会根据 Producer ID 去查询客户端。

通过 ONSFactory.createTransactionProducer 创建事务消息的 Producer 时必须指定 LocalTransactionChecker 的实现类，处理异常情况下事务消息的回查。

事务消息发送完成本地事务后，可在 execute 方法中返回如下三种状态：

- TransactionStatus.CommitTransaction 提交事务，允许订阅方消费该消息。
- TransactionStatus.RollbackTransaction 回滚事务，消息将被丢弃不允许消费。

- TransactionStatus.Unknow 暂时无法判断状态，期待固定时间以后 MQ Server 向发送方进行消息回查。

可通过如下方式给每条消息设定第一次消息回查的最快时间：

```
Message message = new Message();
// 在消息属性中添加第一次消息回查的最快时间，单位秒，如下设置实际第一次回查时间为 120 ~ 125 秒之间
message.putUserProperties(PropertyKeyConst.CheckImmunityTimeInSeconds,"120");
// 以上方式只确定事务消息的第一次回查的最快时间，实际回查时间向后浮动0~5秒；如第一次回查后事务仍未提交，后续每隔5秒回查一次。
```

控制台使用指南

消息查询

MQ 控制台提供几种消息查询方式用于问题排查。

适用场景

MQ 提供了三种消息查询的方式，分别是按 Messag ID，Message Key 以及 Topic 查询。

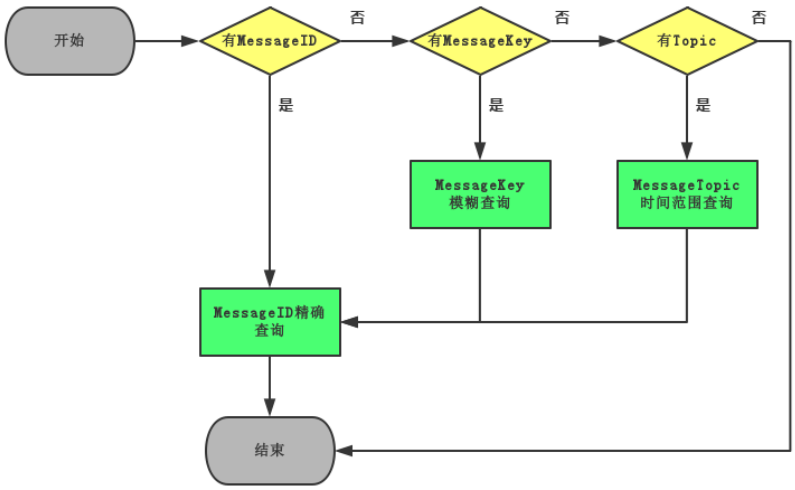
注意： HTTP 消息目前只支持按 Topic 和 Message ID 两种查询方式。

三种方式的特点和对比如下。

查询方式	查询条件	查询类别	说明
按 Message ID 查询	Topic+Message ID	精确查询	根据 Message ID 可以精确定位任意一条消息，获取消息的属性
按 Message Key 查询	Topic+Message Key	模糊查询	根据 Message Key 可以匹配到包含指定 Key 的最近 64 条消息,建议消息生产方为每条消息设置尽可能唯一的 Key，以确保相同的 Key 的消息不会超过 64 条，否则消息会漏查
按 Topic 查询	Topic+时间段	范围查询	根据 Topic 和时间范围，批量获取符合条件

			的所有消息，查询量大，不易匹配

建议查询过程：



按 Message ID 查询

按 Message ID 查询消息属于精确查询，用户输入 Message ID 信息即可精确查询到任意一条消息。因此，为了尽可能精确地查询，建议发送方调用 Producer 发送消息时，在发送消息成功后将 Message ID 信息打印到日志中，方便问题排查。

以 Java SDK 为例，获取 Message ID 的方法如下：

```
SendResult sendResult = producer.send(msg);
String msgId = sendResult.getMessageId();
```

查询结果说明

查询到消息后，您可以在消息详情页面看到每条消息的基本属性，消息体下载链接以及消息的消费状态。

其中，投递状态是 MQ 根据各个 Consumer ID 的消费进度计算出的结果，投递状态的信息参考以下表格。

注意：投递状态仅仅是根据消费进度估算的结果，如果需要详细的消费信息，请使用消息轨迹工具查询。消息轨迹可以展示该条消息的完整链路，具体请参见查询消息轨迹。

投递状态表

投递状态	可能的原因
已订阅，并且消息已被消费	该Consumer ID已经正常消费过这条消息
已订阅，但消息被过滤表达式过滤,请查看 Tag	该消息的 Tag 不符合消费方的订阅关系，消息被

	过滤，可以在控制台中的**消费者管理>消费者状态**查询菜单查询订阅方的订阅关系
已订阅，但消息未被消费	该Consumer ID订阅了该消息，但还未消费，有可能是消费过慢，或者消费出现异常导致阻塞
已订阅，但是Consumer ID当前不在线	该Consumer ID订阅了该消息，但是不在线，请检查消费者端应用不在线的原因
未知异常	出现未收录异常，可以进工单系统提工单解决

消费验证

MQ 提供了消费验证功能，该功能可以将指定消息推送给指定的在线客户端，以检测客户端消费该消息的逻辑和结果是否符合预期。

注意：

- 消费验证功能仅仅是用于验证客户端的消费逻辑是否正常，并不会影响正常的收消息流程，因此消息的消费状态等信息在点击消费验证后并不会改变。
- 采用 HTTP 协议接入时，由于消息发送是短连接方式，因此不支持 HTTP 消息的消费验证功能。

按 Message Key 查询

按 Message Key 查询消息的原理是，MQ 根据用户设置的 Message Key 信息建立消息的索引信息，当用户输入 Key 进行查询时，根据该索引即可匹配相关的消息返回。

注意：

- 按 Message Key 查询的条件是用户设置的 Message Key 属性。
- 按 Message Key 查询仅仅返回符合条件的最近的 64 条消息，因此建议业务方设置 Key 时尽可能唯一，并具有业务区分度。

设置 Message Key 的方法如下：

```
Message msg = new Message("Topic","*","Hello MQ".getBytes());
/**
 * 对每条消息设置其检索的 Key，该 Key 值设置代表消息的业务关键属性，请尽可能全局唯一。
 * 以方便您在无法正常收到消息情况下，可通过 MQ 控制台查询消息。不设置也不会影响消息正常收发
 */
msg.setKey("TestKey"+System.currentTimeMillis());
```

按 Topic 查询

按 Topic 查询一般用在前两种查询条件都无法获得的情况下，根据 Topic 和消息的发送时间范围，批量获取该时间范围内的所有消息，然后再找到关心的数据。

注意：

- 按 Topic 查询属于范围查询，获取 Topic 下符合时间条件的所有消息，消息量大，建议尽量缩短查询区间。
- 按 Topic 查询数据量大，采用分页展示。

消息轨迹

消息轨迹简介

本文档介绍 MQ 消息轨迹的基本原理、使用场景、以及使用案例。

目前 MQ 支持 TCP 和 HTTP 协议下的消息轨迹查询。

基本原理

定义：消息轨迹指的是一条消息从生产方发出到消费方消费处理，整个过程中的各个相关节点的时间地点等数据汇聚而成的完整链路信息。

原理： MQ 系统中，一条消息的完整链路包含生产方、服务方、消费方三个角色，每个角色处理消息的过程中都会在轨迹链路中增加相关的信息，将这些信息汇聚即可获取任意消息当前的状态，从而为生产环境中的问题排查提供强有力的数据支持。

消息轨迹的数据包含：

生产方信息	消费方信息	服务方信息
生产客户端信息	消费客户端信息	消息存储位置信息
发送时间	投递时间，投递轮次	消息存储时间
发送成功与否	消费成功与否	消息本身的属性
发送耗时	消费耗时	-

消息轨迹查询规则：

消息类型	可以查询的时间	查询说明
普通消息	消息发送后	消息发送之后有发送轨迹，没消费前提示没有消费。消费后会展示拉取和消费。
顺序消息	消息发送后	消息发送之后有发送轨迹，没消费前提示没有消费。消费后会展示拉取和消费。

定时/延时消息	当前时间到达消息指定消费的时间	当前时间没有到达指定消费的时间，轨迹可以查询到，但是消息查询不到。
事务消息	消息发送后	消息发送之后有发送轨迹。事务未提交之前，轨迹可以查询到，但是消息查询不到。

使用场景

在生产环境的消息收发不符合预期时可以使用消息轨迹工具排查问题。通过消息的属性（Message ID、Message Key、Topic）搜索相关的消息轨迹，找到消息的实际收发状态，帮助诊断问题。

示例：

假设业务方根据业务日志里的信息判断某条消息一直没有没有收到，则可以参考以下步骤，利用消息轨迹来排查 MQ 问题。

收集怀疑的消息的信息，Message ID，Message Key，Topic 以及大概的发送时间范围。

进入 MQ 控制台，根据已有的信息创建查询任务，查询相关的消息的轨迹。

查看结果并分析判断原因。如果轨迹显示尚未消费，则可以去消费者管理页面查询，确认是否有堆积导致消息尚未消费。

如果发现已经消费，请根据消费端的信息，找到对应的客户端机器和时间，登录查看相关日志。

查询消息轨迹

消息轨迹的使用对于业务方不会增加额外的接入成本，仅仅需要确保客户端 SDK 版本支持该特性。正常收发消息后以消息的相关属性在 MQ 控制台上查询即可。

1. 接入条件

消息轨迹功能目前支持 Java 客户端（1.2.7 版本及以上），C++，.NET 以及 HTTP 客户端。详细的客户端信息，请参考控制台给出的升级提示，获取最新的 SDK。

2. 创建轨迹查询任务

在 MQ 控制台左侧菜单栏选择**消息查询**>**消息轨迹**，选择所在的区域，并单击右上角**创建查询任务**按钮。

消息轨迹查询功能支持三种查询方式，请按照对应方式输入查询条件，创建查询任务。关于三种查询方式的具体说明如下：

- 按 Message ID 查询：需要输入消息的唯一 Message ID，Topic 名称以及消息的大致发送时间。
- 按 Message Key 查询：需要输入消息的 Message Key 和 Topic 以及大致发送时间，适用于没有记录 MessageID，但记录了 Message Key 的场景。
- 按 Topic 查询：仅仅输入 Topic 和时间段，批量查询，适用于没有上述 Message ID 和 Message Key，而且消息量比较小的场景。

注意：

1. 查询时，尽可能设置最为精确的时间区间，以便缩小查询范围，提高速度。
2. 按 MsgID 查询属于精确查询，速度快，精确匹配，推荐用户使用。
3. 按 MsgKey 查询属于模糊查询，仅适用于业务方没有记录 MessageID 但是设置了 MessageKey，同时 MessageKey 具有区分度的情况，MessageKey 查询最多查询 1000 条轨迹。
4. 按 Topic 分段查询属于范围查询，不推荐使用，因为时间范围内消息很多，不具备区分度。

3. 管理查询任务

创建查询任务后，会生成一个查询任务，MQ 后台会异步执行，并将任务状态反馈到**查询管理列表**页面。查询结束时，**任务状态**会显示**查询完成**，否则显示**查询中**。

您可以根据任务的状态可以选择查看轨迹，或者删除查询任务。

4. 查看轨迹

完成查询后，单击右侧操作选项里的**详情**按钮查看轨迹。如果发现没有结果，请参考弹窗链接，排查原因。

如果查询到轨迹信息，可以看到轨迹的简要信息，主要是消息本身的属性以及接收状态的统计。

5. 查看轨迹链路图

单击**查看轨迹**按钮即可查看完整的链路图。

消息链路图包含4个部分：

- 生产者信息
- Topic 信息
- 消费者信息
- 详情信息

各个字段区域均可以通过鼠标悬停的方式获取详细信息。对于 Message Key 和 Topic 查询方式，如果匹配到多条轨迹，可以进行上下翻页，查看比对轨迹数据。

消息轨迹名词解释

消息轨迹查询页面中涉及到的名词概念列表如下。

相关概念	含义（TCP 场景）	含义(HTTP 场景)
发送时间	记录消息从发送端发送时的客户端时间戳	同 TCP
发送耗时	记录发送端调用 send 方法发送消息的毫秒耗时	同 TCP
Region	记录消息存储的 Region 信息，或者消费方机器所在的 Region 信息	同 TCP
消费耗时	记录消息推送到客户端之后执行 consumeMessage 方法的耗时	从 HTTP 客户端执行 GET 拉取消息到执行 DELETE 操作的时间间隔
投递时间	记录客户端执行 consumeMessage 方法开始消费消息时的时间戳	HTTP 客户端执行 GET 拉取消息的时间戳

消息轨迹状态说明

消息轨迹查询页面中涉及到的状态列表如下。

相关概念	含义（TCP 场景）	含义(HTTP 场景)
发送成功	消息发送成功，服务端已经存储成功	同 TCP
发送失败	消息发送失败，服务端没有存储消息，需要重试	同 TCP
消息定时中	该消息是定时或者延时消息，且尚未到达投递时间	同 TCP

事务未提交	该消息是事务消息，且尚未提交状态	无
事务回滚	该消息是事务消息，并且已经回滚	无
全部成功	该消息所有投递都已成功消费	同 TCP
部分成功	该消息投递中存在消费失败并重试成功的情况	同 TCP
尚未消费	该消息尚未投递给任何消费方	HTTP 客户端尚未拉取消息
消费未返回	消费消息的方法尚未返回，或者被中断，导致本次消费结果未传回服务端	HTTP 客户端执行 GET 拉取消息后，没有调用 DELETE 方法返回消费结果
消费失败	消费消息的方法主动返回失败标志，或者是消费方法抛异常	无

如果对消息轨迹的查询结果有疑问，请查看《常见问题》中的**消息轨迹**一节。

重置消费位点

如果想跳过堆积的消息，即不想消费这部分消息，或者只想消费某个时间点之后的消息（这些消息不论之前是否消费过），您可以使用重置消息的消费位点。具体操作步骤如下：

1. 在 MQ 控制台左侧菜单栏单击**消费者管理**。
2. 找到需要重置消费位点的 Topic，在其右侧操作选项里单击**功能**>**重置消费位点**。
3. 在弹出的对话框中选择**清除所有累积消息**或**按时间段进行消费位点重置**。关于这两个选项的具体说明如下：
   - **清除所有累积消息**: 选择该选项会将当前堆积（未被消费）的消息将被全部清除。
     注意: 对于程序返回 "reconsumeLater"，即走重试流程的这部分消息来说，清除无效。
   - **按时间段进行消费位点重置**：点击该选项后会出现时间信息和时间点选择的控件，具体说明如下：
     - 最小时间：指的是现有的最早的那条消息的发送时间（消息保存期为3天）。
     - 最大时间：指的是最近发来的那条消息的时间。
     - 时间点选择：在最小时间和最大时间之间的时间范围内选择一个时间点，则这个时间点之后发送的消息才会被消费。
4. 单击**确认**执行消费位点重置。

说明：目前不支持指定 Message ID，Message Key 和 Tag 来对消息的消费位点进行重置。

资源报表

资源报表是 MQ 提供的消息发送和消费的统计数据查询功能，用户通过资源报表可以对某个 Topic 在一段时间范围内的发送总量或者 TPS 进行查询，也可以对一个 Consumer ID 消费某个 Topic 的总量或者 TPS 进行查询。

注意：MQ 资源报表暂时不支持 HTTP 消费数据的统计查询。

查询消息生产

消息生产查询用于对某个 Topic 在一个时间段内的消息发送总量或者 TPS 进行查询。

具体操作步骤如下：

在 MQ 控制台左侧菜单栏选择**资源报表**>**消息生产**。

选择您要查询的 Topic 所在的域。

在消息消费页面输入相应信息，单击搜索。

选项说明：

- **采集类型**：分为 TPS 和总量，总量提供该周期内 Topic 的发送总量，TPS 提供发送平均 TPS。
- **采集周期**：包括 1 分钟、10 分钟后、30 分钟、1 小时，采集周期决定了数据采集的时间间隔，周期越短，采集点越密集，消息消费数据越详细。
- **时间范围**：MQ 最多可以提供最近三天之内的消息消费查询。

查询结果以图表的形式返回。

查询消息消费

消息消费查询用于对一个 Consumer ID 消费某个 Topic 的总量或者 TPS 进行查询。

具体操作步骤如下：

在 MQ 控制台左侧菜单栏选择**资源报表**>**消息消费**。

选择您要查询的 Consumer ID 所在的域。

在消息消费页面输入相应信息，单击搜索。

选项说明：

- **采集类型**：分为 TPS 和总量，总量提供该周期内消费该 Topic 数据总量，TPS 提供消费该 Topic 的平均 TPS。
- **采集周期**：包括 1 分钟、10 分钟后、30 分钟、1 小时，采集周期决定了数据采集的时间间隔，周期越短，采集点越密集，消息消费数据越详细。
- **时间范围**：MQ 最多可以提供最近三天之内的消息消费查询。

查询结果以图表的形式返回。

监控报警

MQ 提供了对用户所订阅 Topic 的消费状态进行监控并且发送短信报警的功能，帮助您实时掌握消息消费状态并及时处理消费异常等情况。

使用监控报警功能，首先需要在 MQ 上通过 Consumer ID 对 Topic 进行订阅，然后对相关监控项进行设置。当消费者在线，并且消息堆积数或者延迟时间超过阈值之后，MQ 会对报警接收人发送短信进行报警。

新建监控项

用户可以通过新建监控功能对一个 Consumer ID 的消费状态进行监控：

登录 MQ 控制台，在左侧菜单栏选择**监控报警**。

选择您要的 Topic 所在的域。

在监控项管理页面，单击右上角**新增监控项**。

在**新增监控项**对话框输入相关信息，单击确定。新建的监控项会出现在**监控项管理**页面。

选项说明：

- **Consumer ID**：选择要监控的 Consumer ID。
- **Topic**：该 Consumer ID 对应的 Topic。
- **报警阈值**：是指消费堆积的阈值，堆积报警阈值的范围是 1 到 100，000，000。如果您选择的 Consumer ID 在消费对应的 Topic 时产生了消息堆积，并且堆积超过了报警阈值，MQ 就会给报警接收人发送短信通知。

- **消费延迟阈值**：是指该 Consumer ID 最后一次消费该 Topic 消息的时间和该 Topic 最近发送时间之间的差值，消费延迟阈值的最小值是 1 分钟。
- **报警时间**：范围精确到分钟，最大范围为 00:00-23:59，用户只会在设置的接收时间范围内才会收到报警短信。
- **报警频率**：根据需要选择 5 分钟、15 分钟或者 30 分钟。
- **报警接收人**：填写接收人的昵称和手机号码，昵称范围为 32 个字。

管理监控项

在监控项管理页面，您可以对已经建好的监控项进行编辑、禁用、删除等操作。对于禁用的监控项，也可以在操作栏单击**启用**重新启用。

RAM 访问控制

概述

RAM (Resource Access Management) 是阿里云为客户提供的集中式访问控制服务，其核心功能主要包括用户身份与授权管理，应用场景可以覆盖企业子账号与分权管理、针对移动 APP 的临时授权管理，和不同组织之间的资源互操作与授权管理。更多关于 RAM 的介绍可前往 RAM 访问控制 查看。

消息队列已全面对接 RAM 系统，通过 RAM 您可以对消息队列的 Topic 资源进行安全访问控制，包括 Topic 资源的创建、授权、发布以及订阅等。

RAM 相关术语

为了方便您更好地使用消息队列的访问控制功能，本文提供了以下几个关键术语的介绍。更多关于 RAM 的术语解释，请参考 RAM 相关术语。

云账户（主账号）

云账户是阿里云资源归属、资源使用计量计费的基本主体。当用户开始使用阿里云服务时，首先需要注册一个云账户。云账户为其名下所拥有的资源付费，并对其名下所有资源拥有完全权限。默认情况下，资源只能被属

主（ResourceOwner）所访问，任何其他用户访问都需要获得属主的显式授权。所以从权限管理的角度来看，云账户就是操作系统的 root 或 Administrator，所以我们有时称它为 根账户 或 主账户（主账号）。

RAM 用户（子账号）

RAM 允许在一个云账户下创建多个 RAM 用户（可以对应企业内的员工、系统或应用程序）。RAM 用户不拥有资源，没有独立的计量计费，这些用户由所属云账户统一控制和付费。RAM 用户是归属于云账户，只能在所属云账户的空间下可见，而不是独立的云账户。RAM 用户必须在获得云账户的授权后才能登录控制台或使用 API 操作云账户下的资源。

资源（Resource）

资源是云服务呈现给用户与之交互的对象实体的一种抽象。对于消息队列而言，Topic 是其唯一资源。

每个资源有一个全局的阿里云资源名称（Aliyun Resource Name, ARN）。格式如下：

acs:<service-name>:<region>:<account-id>:<resource-relative-id>

该格式的各组成部分代表的含义如下：

- acs: Alibaba Cloud Service 的首字母缩写，表示阿里云的公有云平台；
- service-name: 服务名称，消息队列的服务名称为 mq；
- region: 区域信息，也可使用通配符 “*” 号来代替。
  注意：目前 MQ 授权策略不支持 region，此处必须设置为”*“。
- account-id: 阿里云账号 ID，例如 1234567890123456；
- resource-relative-id: Topic 资源，填写具体的名称即可。

示例： 名称为 acs:mq:*:1234567890123456:TopicA 的资源表示的含义是：

- 消息队列 MQ 资源；
- 资源的 owner 是 1234567890123456；
- 资源名称是 TopicA。

操作（Action）

操作定义了资源相关的权限控制。消息队列 MQ 定义了三种操作：

操作	含义
PUB	发布权限，包括在 MQ 控制台上创建生产者以及通过 SDK 进行消息发送的权限。
SUB	订阅权限，包括在 MQ 控制台上创建消费者以及通过 SDK 进行消息订阅的权限。
*	包含了 PUB 和 SUB 的操作。

资源授权

消息队列支持云账户（主账号）将 Topic 资源授权给 RAM 用户（子账号），有权限的 RAM 用户可在消息队列控制台上进行资源的管理，以及通过 SDK 进行消息的发布与订阅。

授权策略

消息队列目前支持三种授权策略，您可以按照以下步骤查看：

1. 登录RAM 控制台，点击进入**策略管理**>**系统授权策略**。
2. 在**策略名或备注**的搜索框中输入“MQ”，单击**搜索**。您便可以看到目前消息队列支持的三种授权策略。

关于这三种授权策略的具体说明如下：

AliyunMQFullAccess，消息队列的管理权限，被授予该权限的 RAM 用户除了具有主账户所有资源的所有操作权限外，还可以代替主账户在消息队列的控制台上进行资源的管理，比如创建和删除 Topic 等，需要注意的是该 RAM 用户创建的资源所有者为主账户。

AliyunMQPubOnlyAccess，消息队列的发布权限，被授予该权限的 RAM 用户具有主账户所有资源的发布权限，包括 MQ 控制台上创建生产者以及通过 SDK 进行消息发送的权限。

AliyunMQSubOnlyAccess，消息队列的订阅权限，被授予该权限的 RAM 用户具有主账号所有资源的订阅权限，包括 MQ 控制台上创建消费者以及通过 SDK 进行消息订阅的权限。

建议:

您可通过组合 AliyunMQPubOnlyAccess、AliyunMQSubOnlyAccess 两种策略，将主账户所有资源的发布和订阅权限授予给 RAM 用户。

与 AliyunMQFullAccess 策略相比，这种组合策略并不会将消息队列的管理权限授予 RAM 用户，也就是说组合策略没有 Topic 资源的创建、删除等管理操作权限。

创建自定义策略

通常情况下，消息队列提供的三种系统授权策略已经能满足大部分业务需求，如果您需要有更细粒度的授权需求，那么可以通过创建自定义策略来进行访问控制。

可参考创建自定义授权策略，按文中步骤创建自定义策略。

以下是一个自定义策略的示例：

```
{
"Version": "1",
"Statement": [
{
```

```
"Action": "mq:PUB",
"Resource": [
"acs:mq:*:*:TopicA",
"acs:mq:*:*:TopicB"
],
"Effect": "Allow"
}
]
}
```

该示例表示：

- 资源名称： TopicA、TopicB；
- 操作权限：发布权限，包括创建生产者以及通过 SDK 进行消息订阅的权限。

给 RAM 用户授权

首先，请参考RAM 用户创建了解如何创建 RAM 用户，以及参考RAM 用户授权了解授权相关的基本操作以及用户组授权的概念。

授权步骤

主账户给 RAM 用户授权的具体步骤如下：

1. 使用主账户登录RAM 控制台。
2. 点击左侧导航栏中的**用户管理**。
3. 找到需要授权的用户（可通过**登录名**进行搜索），并点击其右侧操作栏目下的**授权**按钮进入**编辑个人授权策略**页面。
4. 添加需要的授权策略（可按关键词进行搜索），并点击**确认**。
   - 从左侧**可选授权策略**下选择需要的策略，点击右向箭头（即授权）将其添加到**已选授权策略**中。
   - 反之，通过左向箭头可将右侧**已选授权策略**取消。

RAM 用户权限校验

RAM 用户可登录消息队列控制台进行检验。登录消息队列控制台后，在**Topic 管理**页面可以看到被授权的所有 Topic 资源。

登录步骤如下：

1. 登录RAM 控制台。
2. 在 RAM 控制台左侧的产品列表中找到**消息队列** ，点击即跳转到消息队列控制台。或者直接点击 MQ 控制台进入。

子账号使用指南

子账号（RAM 用户）可登录消息队列控制台查看被授权的 Topic 资源以及创建生产者和消费者，最后可通过 SDK 进行消息的发布和订阅，具体步骤如下：

1. 登录RAM 控制台。
2. 在 RAM 控制台左侧的产品列表中找到**消息队列** ，点击即可跳转到消息队列控制台。或者在 RAM 子账号登录态下直接点击MQ 控制台 进入。
3. 点击左侧导航栏中的**Topic 管理**，找到被授权的 Topic，点击右侧操作栏目中的**创建生产者**创建 Producer ID， 或点击**创建消费者**创建 Consumer ID。
   注意：RAM 子账号创建的 Producer ID、Consumer ID 相互隔离，不可与其它 RAM 子账号混用，亦不可与主账号混用。
4. 使用创建好的 Producer ID 和 CID ，通过 SDK 进行消息的收发。

注意：

RAM 子账号登录控制台后，需要给被授权的 Topic 分别创建 Producer ID 和 Consumer ID, 不可以直接使用主账号的 Producer ID 和 Consumer ID。

通过 SDK 进行消息收发前，需要先使用 RAM 子账号的 AccessKey 和 SecrectKey 进行身份认证。具体 AccessKey 信息请参见创建 RAM 用户中的《为用户创建 AK》部分。

HTTP 下线通告

通告：感谢大家对消息队列 HTTP 协议的支持。考虑到与消息服务（MNS）功能上的重复性，避免客户在选型上产生困惑，消息队列 HTTP 协议将于 2018.04.01 起逐步下线。因此，我们建议您从今起：

- HTTP 协议相关需求可前往消息服务 MNS；
- MQ TCP 多语言客户端相关需求可前往消息队列 Kafka，目前已支持包括 Java、C++、GO、NodeJs、PHP、Python、logStash、filebeat 以及 springcloud 等；
- 如有其它疑问，您可以通过提交工单 进行反馈。