# SOFAStack

## 消息队列 SOFAMQ

## 技术白皮书

产品版本：AntStack Plus 1.11.0

文档版本：20220930

蚂蚁集团
ANT GROUP

# 法律声明

## 蚂蚁集团版权所有©2022，并保留一切权利。



## 商标声明



## 免责声明

# 通用约定

| 格式 | 说明 | 样例 |
| --- | --- | --- |
| 危险 | 该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 危险<br>重置操作将丢失用户配置数据。 |
| 警告 | 该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 警告<br>重启操作将导致业务中断，恢复业务时间约十分钟。 |
| 注意 | 用于警示信息、补充说明等，是用户必须了解的内容。 | 注意<br>权重设置为0，该服务器不会再接受新请求。 |
| 说明 | 用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。 | 说明<br>您也可以通过按Ctrl+A选中全部文件。 |
| > | 多级菜单递进。 | 单击**设置**> **网络**> **设置网络类型**。 |
| **粗体** | 表示按键、菜单、页面名称等UI元素。 | 在**结果确认**页面，单击**确定**。 |
| Courier字体 | 命令或代码。 | 执行 `cd /d C:/window` 命令，进入Windows系统文件夹。 |
| *斜体* | 表示参数、变量。 | `bae log list --instanceid`<br>*Instance_ID* |
| [] 或者 [a\|b] | 表示可选项，至多选择一个。 | `ipconfig [-all\|-t]` |
| {} 或者 {a\|b} | 表示必选项，至多选择一个。 | `switch {active\|stand}` |

# 目录

# 1.概述

SOFAStack 消息队列（SOFAStack MQ，简称 SOFAMQ）是基于 Apache RocketMQ 构建的分布式消息中间件，并与金融分布式架构 SOFAStack 深度集成，为分布式应用系统提供异步解耦和削峰填谷的能力，支持事务消息、顺序消息、定时消息等多种消息类型，并具备高可靠、高吞吐、低延时等金融级特性。

## 应用场景

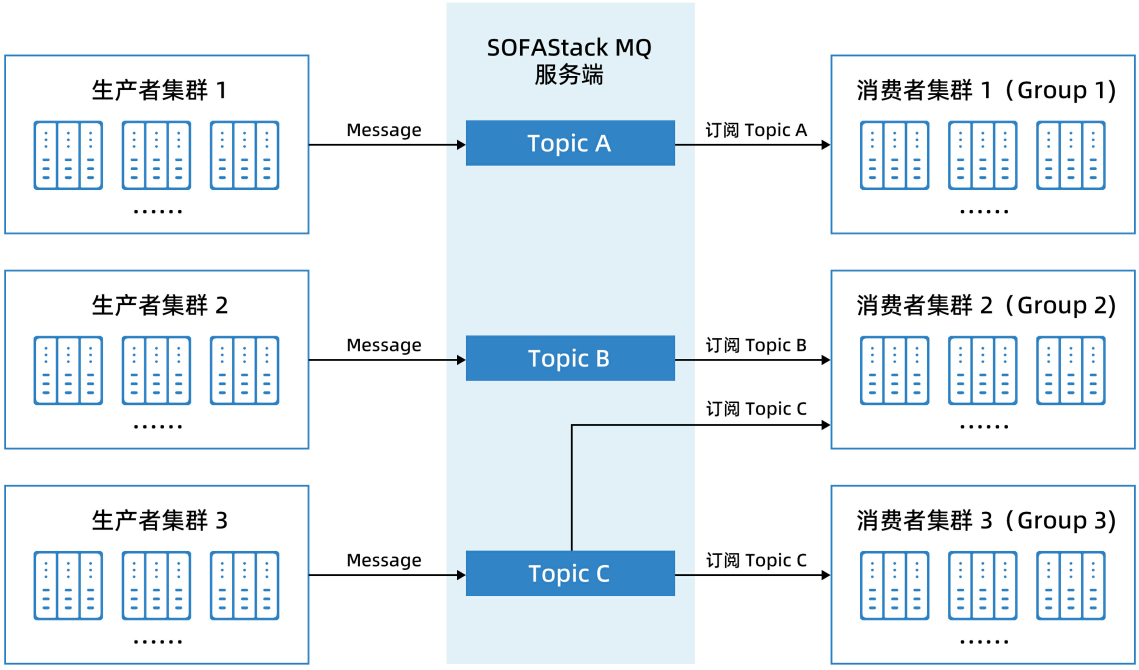
- 异步解耦消息队列的生产消费模型可以解耦上下游业务系统，并支持下游多个消费者对同一消息进行消费和处理。以金融场景为例，支付中心作为支付宝主站最核心的系统，每笔支付数据的产生会引起几百个下游业务系统的关注，包括账户中心、用户中心、权益中心、流计算分析等，整体业务系统庞大而且复杂，在应用强耦合的情况下，任一应用故障都将可能对业务带来影响。通过消息队列进行异步通信和应用解耦，可以很好的提升业务的连续性。
- 削峰填谷应用分布式改造后，不同应用能承载的性能情况往往不一致，在诸如双 11、店庆、秒杀等大型活动时，将会带来较高的流量脉冲，部分系统可能导致系统超负荷甚至崩溃，影响用户体验，消息队列可提供强大的抗积压能力，实现削峰填谷，生产方生产消息后，消费方可以按照系统自身的承受能力进行消息的消费。
- 顺序收发指的是消息消费者按照消息发送的顺序进行消费，保证 FIFO。金融场景里需要保证顺序的应用场景非常多，例如证券交易过程中的时间优先原则，交易系统中的订单创建、支付、退款等流程等等。
- 分布式事务一致性应用解耦往往带来多个应用之间的事务一致性的问题。例如支付转账成功后，需要生成账单，更新用户积分等，此时通过消息队列的分布式事务处理功能，既可以实现系统之间的解耦，又可以保证最终的数据一致性。

## 核心概念

- Topic：消息主题，一级消息类型，生产者向其发送消息。
- 生产者：也称为消息发布者，负责生产并发送消息至 Topic。
- 消费者：也称为消息订阅者，负责从 Topic 接收并消费消息。
- 消息：生产者向 Topic 发送并最终传送给消费者的数据和（可选）属性的组合。
- 消息属性：生产者可以为消息定义的属性，包含 Message Key 和 Tag。
- Group：一类生产者或消费者，这类生产者或消费者通常生产或消费同一类消息，且消息发布或订阅的逻辑一致。

## 消息收发模型

消息队列支持发布/订阅模型，消息生产者应用创建 Topic 并将消息发送到 Topic。消费者应用创建对 Topic 的订阅以便从其接收消息。通信可以是一对多（扇出）、多对一（扇入）和多对多。




**生产者集群**

用来表示发送消息应用，一个生产者集群下包含多个生产者实例，可以是多台机器，也可以是一台机器的多个进程，或者一个进程的多个生产者对象。

一个生产者集群可以发送多个 Topic 消息。发送分布式事务消息时，如果生产者中途意外宕机，Broker 会主动回调生产者集群的任意一台机器来确认事务状态。

**消费者集群**

用来表示消费消息应用，一个消费者集群下包含多个消费者实例，可以是多台机器，也可以是多个进程，或者是一个进程的多个消费者对象。

一个消费者集群下的多个消费者以均摊方式消费消息。如果设置的是广播方式，那么这个消费者集群下的每个实例都消费全量数据。

一个消费者集群对应一个 Group ID，一个 Group ID 可以订阅多个 Topic，如上图中的 Group 2 所示。Group 和 Topic 的订阅关系可以通过直接在程序中设置。

# 2.产品优势

SOFAStack 消息队列的产品优势如下：

- **功能齐全**
  - 支持多种消息类型：普通消息、定时消息、分区顺序消息、事务消息
  - 支持多种消费模式：Pub/Sub 模式、Tag 过滤
  - 支持 TCP Java SDK
- **运维体系便捷**
  - 支持多维度消息查询
  - 支持全链路消息轨迹
- **性能优越**
  - 海量消息堆积能力
  - 毫秒级端到端延迟
  - 千万级高并发处理能力
- **服务可靠**
  - 服务高可用
  - 数据高可靠
  - 支持消息重投机制

# 3.产品架构

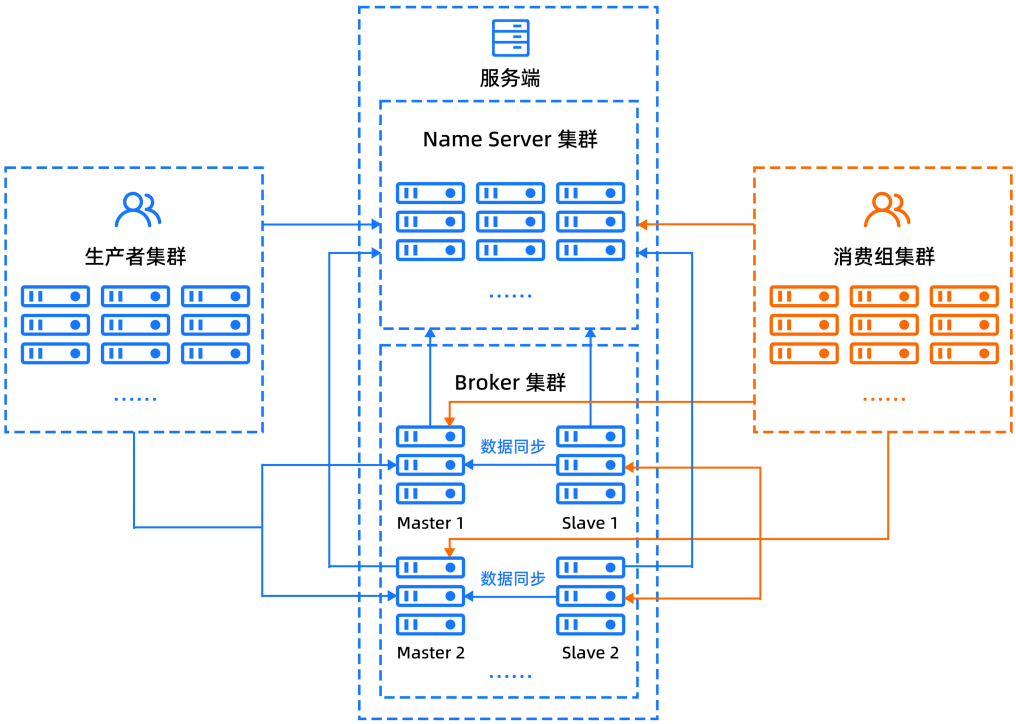
## 产品架构

消息队列在任何一个环境都是可扩展的，生产者必须是一个集群，消息服务器必须是一个集群，消费者也同样。集群级别的高可用，是消息队列跟其他的消息服务器的主要区别，消息生产者发送一条消息到消息服务器，消息服务器会随机的选择一个消费者，只要这个消费者消费成功就认为是成功了。

> 说明
>
> 白皮书中所提及的消息队列的服务端或者服务器包含 Name Server、Broker 等。服务端不等同于 Broker。

## 系统架构

系统部署架构如下图所示。




图中所涉及到的概念如下所述：

- **NameServer**：是一个几乎无状态节点，可集群部署，在消息队列中提供命名服务，该组件内置 zookeeper 用来负责 Broker 的选主和集群管理。
- **Broker**：消息中转角色，负责存储和转发消息。分为 Master Broker 和 Slave Broker，一个 Master Broker 可以对应多个 Slave Broker，但是一个 Slave Broker 只能对应一个 Master Broker。Broker 启动后需要完成一次将自己注册至 Name Server 的操作；随后每隔 30s 定期向 Name Server 上报 Topic 路由信息。
- **生产者 Producer**：与 Name Server 集群中的其中一个节点（随机）建立长连接（Keep-alive），定期从 Name Server 读取 Topic 路由信息，并与提供 Topic 服务的 Master Broker 建立长连接，且定时向 Master Broker 发送心跳。

- **消费者 Consumer**：与 Name Server 集群中的其中一个节点（随机）建立长连接，定期从 Name Server 拉取 Topic 路由信息，并向提供 Topic 服务的 Master Broker、Slave Broker 建立长连接，且定时向 Master Broker、Slave Broker 发送心跳。Consumer 既可以从 Master Broker 订阅消息，也可以从 Slave Broker 订阅消息，订阅规则由 Broker 配置决定。

SOFAMQ 还包含如下两个未在图中展示的组件，主要功能如下：

- **Console**：SOFAMQ 的图形化管理控制台。
- **Router**：LDC 场景下，根据配置的 Router 规则，进行逻辑单元之间消息转发，使 SOFAMQ 支持 LDC 能力的组件。

## 网络架构

## Broker 集群部署

- 为保证服务的可用性，消息队列支持多节点集群化部署。
- 支持在多个机房部署。
- 支持集群化部署方式，提高部分节点不可用时的容灾能力。
- 消息重发机制。当某个节点服务不可用时，迅速切换到其他节点，提高集群的服务能力。

## Broker 主备部署

- 为保证数据的可靠性，消息队列支持一主多备部署方式。主备复制模式包括同步复制和异步复制。
- SOFAMQ Broker 同时支持主备间自动切换。一旦主机出现不可用时，根据主备切换策略进行切换，迅速恢复服务。

# 4.性能指标

不同机器配置下的性能不同，即对于不同的 POD 规格，系统吞吐量（TPS）会有所不同。详细的性能数据如下表所示：

| POD | 存储磁盘 | TPS（1KB 消息） |
| --- | --- | --- |
| 4C8G（一主一备） | 500GB SSD | 5000 |
| 16C60G（一主一备） | 2TB SSD | 20000 |

> **说明**
>
> 一组机器要求至少包括 2 POD 一主一备，如有需要，可以在此基础上以机器组的维度进行扩容。

# 5.功能原理

## 5.1. 消息发送

TCP 协议支持三种消息发送方式：可靠同步发送、可靠异步发送、单向（Oneway）发送。本文介绍了每种发送方式的原理、适用场景以及三种发送方式的异同。

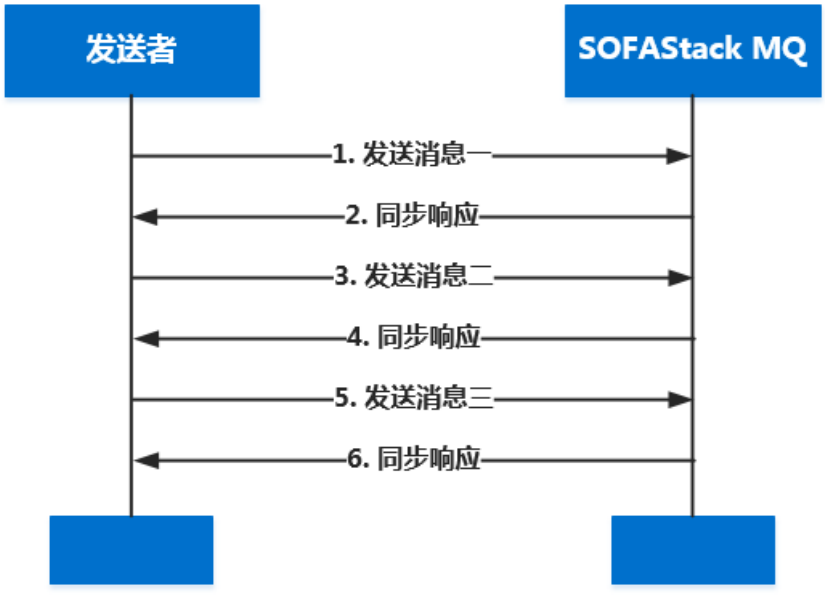
### 三种发送方式的对比

三者的特点和主要区别如下：

| 发送方式 | 发送TPS | 发送结果反馈 | 可靠性 | 适用场景 |
| --- | --- | --- | --- | --- |
| 可靠同步发送 | 快 | 有 | 不丢失 | 此种方式应用场景非常广泛，例如重要通知邮件、报名短信通知、营销短信系统等。 |
| 可靠异步发送 | 快 | 有 | 不丢失 | 异步发送一般用于链路耗时较长，对响应时间较为敏感的业务场景，例如用户视频上传后通知启动转码服务，转码完成后通知推送转码结果等。 |
| 单向（ONEWAY）发送 | 最快 | 无 | 可能丢失 | 适用于某些耗时非常短，但对可靠性要求并不高的场景，例如日志收集。 |

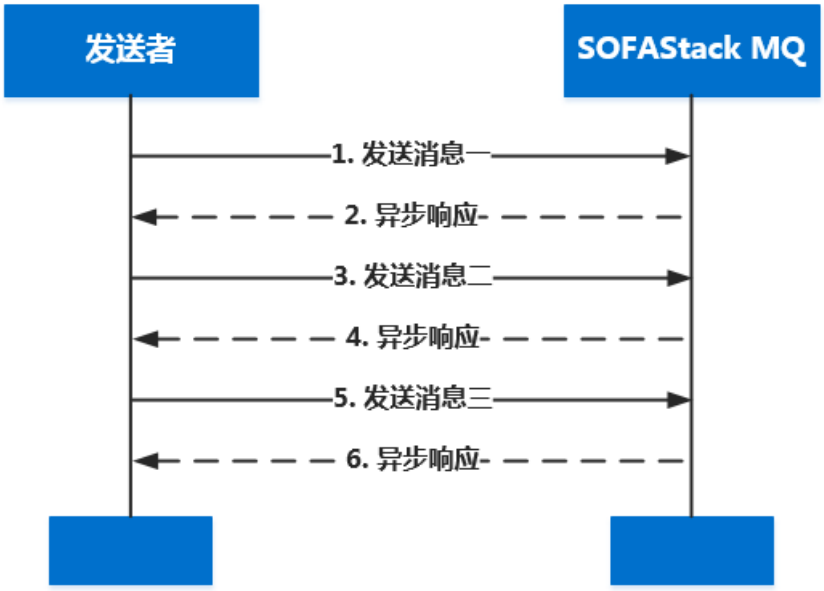
### 可靠同步发送

同步发送是指消息发送方发出数据后，会在收到接收方发回响应之后才发下一个数据包的通讯方式。



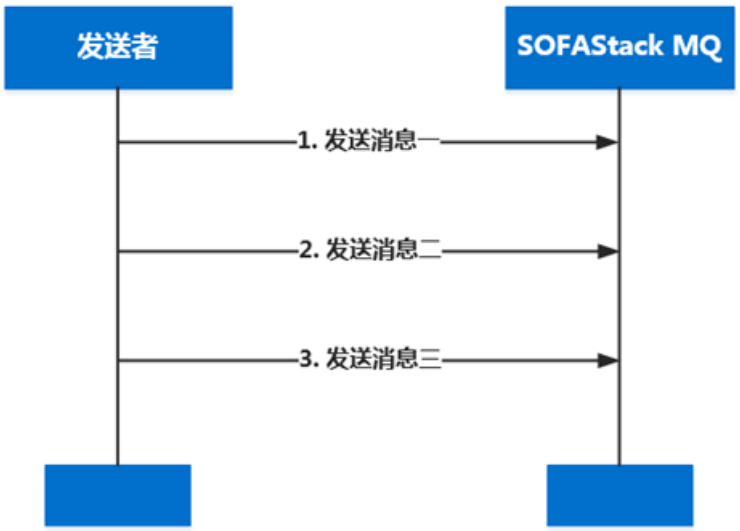

### 可靠异步发送

异步发送是指发送方发出一条消息后，不等服务端返回响应，接着发送下一条消息的通讯方式。消息队列的异步发送，需要用户实现异步发送回调接口（SendCallback）。消息发送方在发送了一条消息后，无需优化等待服务端响应即可发送第二条消息。发送方通过回调接口接收服务端响应，并处理响应结果。




### 单向（ONEWAY）发送

发送方只负责发送消息，不等待服务端返回响应且没有回调函数触发，即只发送请求不等待应答。此方式发送消息的过程耗时非常短，一般在微秒级别。



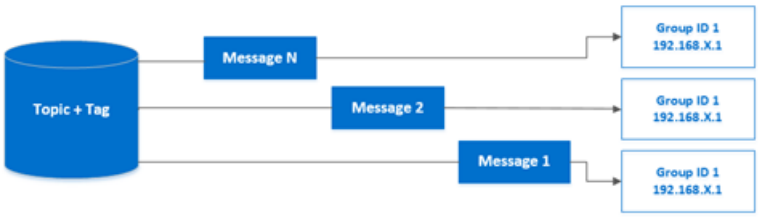

# 5.2. 消息订阅

SOFAStack MQ 是基于发布订阅模型的消息系统。在 SOFAStack MQ 消息系统中，消息的订阅方通过订阅关注 Topic 以获取并消费消息。

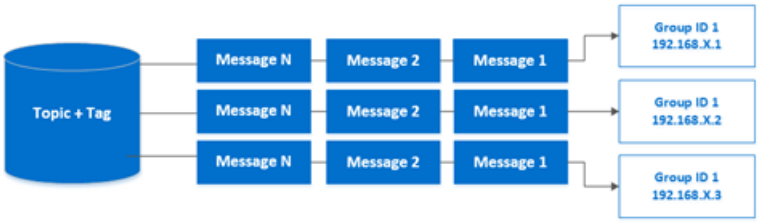
由于订阅方应用一般是分布式系统，以集群方式部署有多台机器，因此 SOFAStack MQ 约定以下概念。

- **集群**：SOFAStack MQ 约定使用相同 Group ID 的订阅者属于同一个集群，同一个集群下的订阅者费逻辑必须完全一致（包括 Tag 的使用），这些订阅者在逻辑上可以认为是一个消费节点。

- **集群消费**：当使用集群消费模式时，SOFAStack MQ 认为任意一条消息只需要被集群内的任意一个消费者处理即可。




- **广播消费**：当使用广播消费模式时，SOFAStack MQ 会将每条消息推送给集群内所有注册过的客户端，保证消息至少被每台机器消费一次。




# 5.3. 事务消息

## 概念介绍

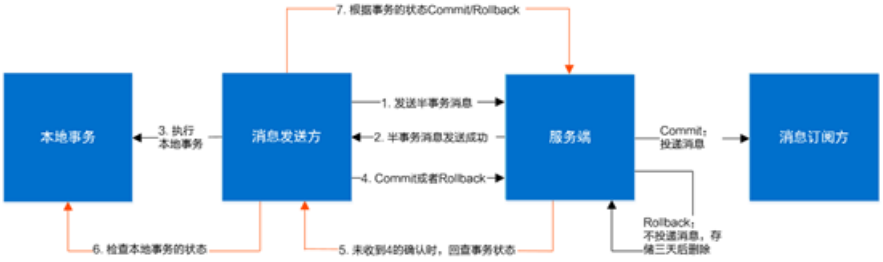
- **事务消息**：消息队列提供类似 X/Open XA 的分布式事务功能，通过消息队列事务消息能达到分布式事务的最终一致。
- **半事务消息**：暂不能投递的消息，发送方已经成功地将消息发送到了消息队列服务端，但是服务端未收到生产者对该消息的二次确认，此时该消息被标记成“暂不能投递”状态，处于该种状态下的消息即半事务消息。
- **消息回查**：由于网络闪断、生产者应用重启等原因，导致某条事务消息的二次确认丢失，消息队列服务端通过扫描发现某条消息长期处于“半事务消息”时，需要主动向消息生产者询问该消息的最终状态（Commit 或是 Rollback），该询问过程即消息回查。

## 适用场景

事务消息的适用场景示例如下：

在转账过程中，比如从支付宝转账到余额宝，两个系统之间的数据需要保持最终一致性，这时可以通过事务消息进行处理。支付宝扣款执行前，发送一条半事务消息，扣款事务执行成功后，将消息状态更新为 Commit，余额宝系统订阅消息队列的扣款消息，做相应的存款业务处理。

## 交互流程



事务消息发送步骤如下：

1. 发送方将半事务消息发送至消息队列服务端。
2. 消息队列服务端将消息持久化成功之后，向发送方返回 Ack 确认消息已经发送成功，此时消息为半事务消息。
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服务端提交二次确认（Commit 或是 Rollback），服务端收到 Commit 状态则将半事务消息标记为可投递，订阅方最终将收到该消息；服务端收到 Rollback 状态则删除半事务消息，订阅方将不会接受该消息。

事务消息回查步骤如下：

1. 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服务端，经过固定时间后服务端将对该消息发起消息回查。
2. 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。
3. 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，服务端仍按照步骤 4 对半事务消息进行操作。

# 5.4. 定时消息

## 概念介绍

- **定时消息**：Producer 将消息发送到消息队列服务端，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 将消息发送到消息队列服务端，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。

## 适用场景

定时消息和延时消息适用于以下场景：

消息生产和消费有时间窗口要求：比如在蚂蚁森林场景中，相关低碳行为触发时，会发送一条定时消息，在第二天 7 点定时投递给消费者，产生绿色能量；或者发送一条延时消息，24 小时后产生绿色能量。

## 使用方法

定时消息和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。

# 5.5. 顺序消息

顺序消息（FIFO 消息）是消息队列提供的一种严格按照顺序来发布和消费的消息。顺序发布和顺序消费是指对于指定的一个 Topic，生产者按照一定的先后顺序发布消息；消费者按照既定的先后顺序订阅消息，即先发布的消息一定会先被客户端接收到。

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。

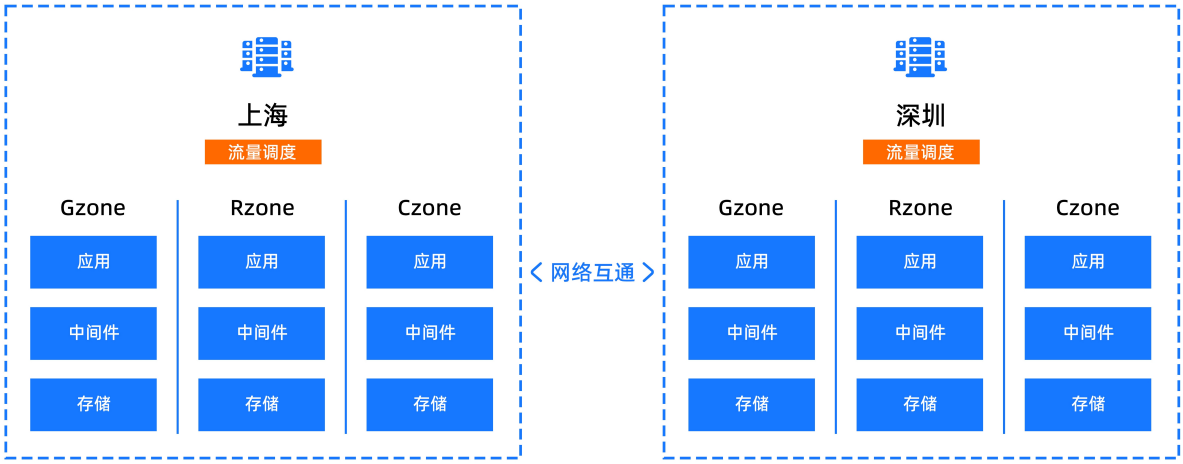
## 适用场景

适用于性能要求高，以 Sharding Key 作为分区字段，在同一个区块中严格地按照 FIFO 原则进行消息发布和消费的场景。

顺序消息的适用场景示例如下：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发布的先后顺序来消费。

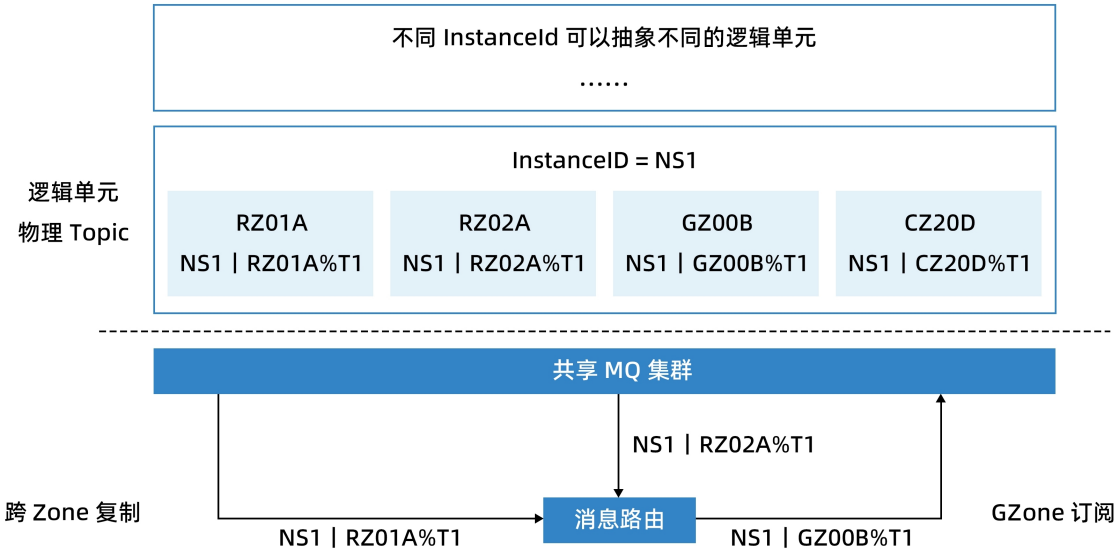
# 5.6. 单元化功能

SOFAStack MQ 的部署物理单元以城市维度部署。用户自行划分逻辑单元，共享使用 SOFAStack MQ 集群，优点是机器成本比较低，适合体量适中的用户。




## 实现原理

核心原理是通过消息路由（Router）组件实现消息复制。




### Topic

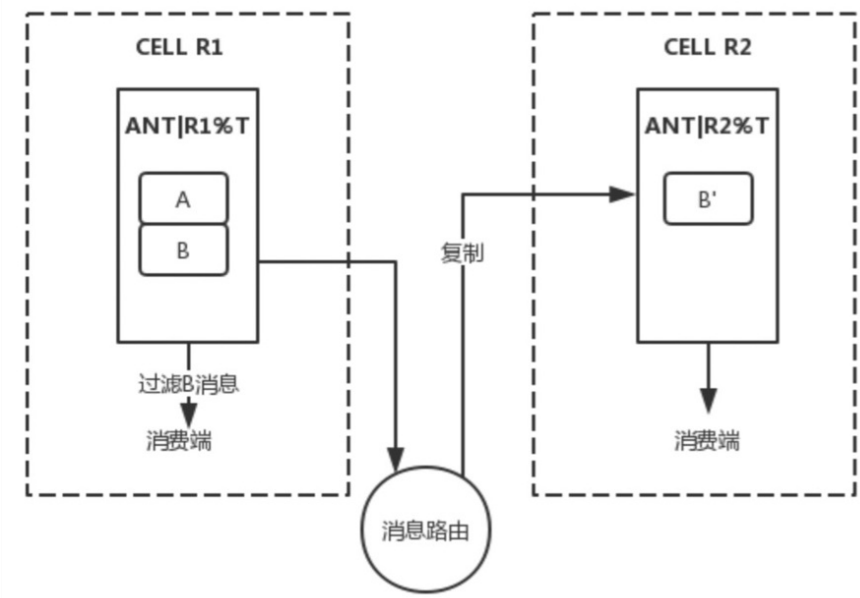
通过用户逻辑 Topic 和实际物理 Topic 的映射来实现逻辑单元消息隔离，每个逻辑 Zone 发送和消费都是各自的 Topic，数据是互相隔离的。

例如，InstanceId= ANT 应用在逻辑单元 CELL1 发送 Topic 为 T1 的消息，客户端实际发送的时候会将逻辑 Topic 转换成物理 Topic `ANT|CELL1%T1` 。这样不同逻辑单元的应用，虽然在用户侧感知到的是同一个 Topic，实际上不同逻辑单元的物理 Topic 是不一样的。

### Group

通过用户逻辑 Group 和实际物理 Group 的映射来实现逻辑单元消费的隔离，每个逻辑 Zone 发送和消费都是各自的 Group，事务回查、订阅关系和消费进度是互相隔离的。

例如，InstanceId= ANT 应用在逻辑单元 CELL1 通过 GroupG1 消费 Topic 为 T1 的消息，客户端实际消费的时候会将逻辑 Group 转换成物理 Group `ANT|CELL1%G1` ，最终就是以 Group `ANT|CELL1%G1` 来消费 Topic `ANT|CELL1%T1` 。




如上图示应用，对于在 R1 的应用，因为 B 不属于 R1，需要过滤掉 B 的账单消息。

对于在 R2 的应用，因为 B 的账单消息在 R1，如果要被 R2 的应用消费到的话，需要通过消息路由组件将 B 的账单消息从 R1 复制到 R2。

从物理上来说，即将消息从 R1 对应的 Topic 复制到 R2 对应的 Topic 。

## LDC 实现原理

### 发送方实现

发送端发送消息时，客户端将逻辑 Topic 替换成物理 Topic `$NS|$CELL%$TOPIC` ，发送消息会自动根据 zonemng 设置 __UNIT.ORIUNIT 、__UNIT.TARGET_G、__UNIT.TARGET_C。如果设置了 cellUid，还会设置 __UNIT.TARGET_R。

### Local 订阅

使用 `$NS|$CELL%$GROUP` 订阅 `$NS|$CELL%$TOPIC` 消费 `$NS|$CELL%$TOPIC` 里的数据，只消费 `__UNIT.ORIUNIT ==` 当前 CELL 的消息。

### RZone 订阅

客户端只在 RZone 启动对应的订阅关系，使用 `$NS|$CELL%$GROUP` 订阅 `$NS|$CELL%$TOPIC`，只消费 `__UNIT.TARGET_R ==` 当前 CELL 的消息。

### GZone 订阅

客户端只在 GZone 启动对应的订阅关系，使用 `$NS|$CELL%$GROUP` 订阅 `$NS|$CELL%$TOPIC`，只消费 `__UNIT.TARGET_G ==` 当前 CELL 的消息。

### CZone 订阅

客户端只在 CZone 启动对应的订阅关系，使用 `$NS|$CELL%$GROUP` 来订阅 `$NS|$CELL%$TOPIC`，只消费 `__UNIT.TARGET_C ==` 当前 CELL 的消息。

# 6.附录：基础术语

本文主要对 SOFAStack 消息队列涉及的专有名词及术语进行定义和解析，方便您更好地理解相关概念并使用消息队列。

| 中文 | 英文 | 释义 |
| --- | --- | --- |
| 消息主题 | Topic | 消息主题，一级消息类型，通过 Topic 对消息进行分类。 |
| 消息 | Message | 消息队列中信息传递的载体。 |
| Message ID | Message ID | 消息的全局唯一标识，由消息队列系统自动生成，唯一标识某条消息。 |
| Message Key | Message Key | 消息的业务标识，由消息生产者（Producer）设置，唯一标识某个业务逻辑。 |
| 消息标签 | Tag | 消息标签，二级消息类型，用来进一步区分某个 Topic 下的消息分类。 |
| 消息生产者 | Producer | 消息生产者，也称为消息发布者，负责生产并发送消息。 |
| Producer 实例 | Producer instance | Producer 的一个对象实例，不同的 Producer 实例可以运行在不同进程内或者不同机器上。Producer 实例线程安全，可在同一进程内多线程之间共享。 |
| 消息消费者 | Consumer | 消息消费者，也称为消息订阅者，负责接收并消费消息。 |
| Consumer 实例 | Consumer instance | Consumer 的一个对象实例，不同的 Consumer 实例可以运行在不同进程内或者不同机器上。一个 Consumer 实例内配置线程池消费消息。 |
| Group | Group | 一类 Producer 或 Consumer，这类 Producer 或 Consumer 通常生产或消费同一类消息，且消息发布或订阅的逻辑一致。 |
| Group ID | Group ID | Group 的标识。 |

| 中文 | 英文 | 释义 |
| --- | --- | --- |
| 队列 | Queue | 每个 Topic 下会由一到多个队列来存储消息。 |
| 集群消费 | Clustering consumption | 一个 Group ID 所标识的所有 Consumer 平均分摊消费消息。例如某个 Topic 有 9 条消息，一个 Group ID 有 3 个 Consumer 实例，那么在集群消费模式下每个实例平均分摊，只消费其中的 3 条消息。 |
| 广播消费 | Broadcasting consumption | 一个 Group ID 所标识的所有 Consumer 都会各自消费某条消息一次。例如某个 Topic 有 9 条消息，一个 Group ID 有 3 个 Consumer 实例，那么在广播消费模式下每个实例都会各自消费 9 条消息。 |
| 定时消息 | Scheduled message | Producer 将消息发送到消息队列服务端，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 | Delayed message | Producer 将消息发送到消息队列服务端，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。 |
| 事务消息 | Transactional message | 消息队列提供类似 X/Open XA 的分布事务功能，通过消息队列的事务消息能达到分布式事务的最终一致。 |
| 顺序消息 | Ordered message | 消息队列提供的一种按照顺序进行发布和消费的消息类型，分为全局顺序消息和分区顺序消息，当前仅支持分区顺序消息。 |
| 分区顺序消息 | Partitionally ordered message | 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 Message Key 是完全不同的概念。 |
| 消息堆积 | Message accumulation | Producer 已经将消息发送到消息队列的服务端，但由于 Consumer 消费能力有限，未能在短时间内将所有消息正确消费掉，此时在消息队列的服务端保存着未被消费的消息，该状态即消息堆积。 |
| 消息过滤 | Message filtering | Consumer 可以根据消息标签（Tag）对消息进行过滤，确保 Consumer 最终只接收被过滤后的消息类型。消息过滤在消息队列的服务端完成。 |

| 中文 | 英文 | 释义 |
| --- | --- | --- |
| 消息轨迹 | Message trace | 在一条消息从 Producer 发出到 Consumer 消费处理过程中，由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。通过消息轨迹，您能清晰定位消息从 Producer 发出，经由消息队列服务端，投递给 Consumer 的完整链路，方便定位排查问题。 |
| 重置消费位点 | Reset consumption offset | 以时间轴为坐标，在消息持久化存储的时间范围内（默认 3 天），重新设置 Consumer 对已订阅的 Topic 的消费进度，设置完成后 Consumer 将接收设定时间点之后由 Producer 发送到消息队列服务端的消息。 |