如何从0到1设计一个消息队列中间件

题目标签

学习时长：20分钟

题目难度：中等

知识点标签：消息队列、通讯协议、AMQP、Kafka、JMS

题目描述

消息队列作为系统解耦，流量控制的利器，成为分布式系统核心组件之一。

如果你对消息队列背后的实现原理关注不多，其实了解消息队列背后的实现非常重要。

不仅知其然还要知其所以然，这才是一个优秀的工程师需要具备的特征。

今天，我们就一起来探讨设计一个消息队列背后的技术。

1. 消息队列整体设计思路

主要是设计一个整体的消息被消费的数据流。

这里会涉及到：消息生产Producer、Broker(消息服务端)、消息消费者Consumer。

1.Producer(消息生产者)：发送消息到Broker。

2.Broker(服务端)：Broker这个概念主要来自于Apache的ActiveMQ，特指消息队列的服务端。

主要功能就是：把消息从发送端传送到接收端，这里会涉及到消息的存储、消息通讯机制等。

3.Consumer(消息消费者)：从消息队列接收消息，consumer回复消费确认。

2. Broker(消息队列服务端)设计重点

1）消息的转储：在更合适的时间点投递，或者通过一系列手段辅助消息最终能送达消费机。

2）规范一种范式和通用的模式，以满足解耦、最终一致性、错峰等需求。

3）其实简单理解就是一个消息转发器，把一次RPC做成两次RPC，发送者把消息投递到broker，broker再将消息转发一手到接收端。

总结起来就是两次RPC加一次转储，如果要做消费确认，则是三次RPC。

为了实现上述消息队列的基础功能：

- 消息的传输
- 存储
- 消费

就需要涉及到如下三个方面的设计：

- 通信协议
- 存储选择
- 消费关系维护

3.通讯协议

消息Message:既是信息的载体，消息发送者需要知道如何构造消息，消息接收者需要知道如何解析消息，它们需要按照一种统一的格式描述消息，这种统一的格式称之为消息协议。

传统的通信协议标准有XMPP和AMQP协议等，现在更多的消息队列从性能的角度出发使用自己设计实现的通信协议。

1.JMS

JMS（Java MessageService）实际上是指JMS API。JMS是由Sun公司早期提出的消息标准，旨在为java应用提供统一的消息操作，包括创建消息、发送消息、接收消息等。

JMS通常包含如下一些角色：

Elements	Notes
JMS provider	实现了JMS接口的消息中间件，如ActiveMQ
JMS client	生产或者消费消息的应用
JMS producer/publisher	JMS消息生产者
JMS consumer/subscriber	JMS消息消费者
JMS message	消息，在各个JMS client传输的对象；
JMS queue	Provider存放等待被消费的消息的地方
JMS topic	一种提供多个订阅者消费消息的一种机制；在MQ中常常被提到，topic模式。

JMS提供了两种消息模型：

- 点对点
- 以及publish-subscribe（发布订阅）模型。

当采用点对点模型时，消息将发送到一个队列，该队列的消息只能被一个消费者消费。

而采用发布订阅模型时，消息可以被多个消费者消费。

在发布订阅模型中，生产者和消费者完全独立，不需要感知对方的存在。

2.AMQP

AMQP是 Advanced Message Queuing Protocol，即高级消息队列协议。

AMQP不是一个具体的消息队列实现，而 是一个标准化的消息中间件协议。

目标是让不同语言，不同系统的应用互相通信，并提供一个简单统一的模型和编程接口。 目前主流的ActiveMQ和RabbitMQ都支持AMQP协议。

AMQP是一种协议，更准确的说是一种binary wire-level protocol（链接协议）。这是其和JMS的本质差别，AMQP不从API层进行限定，而是直接定义网络交换的数据格式。

JMS和AMQP比较

JMS: 只允许基于JAVA实现的消息平台的之间进行通信

AMQP: AMQP允许多种技术同时进行协议通信

3.Kafka的通信协议

Kafka的Producer、Broker和Consumer之间采用的是一套自行设计的基于TCP层的协议。Kafka的这套协议完全是为了Kafka自身的业务需求而定制的。

4. 存储选型

对于分布式系统，存储的选择有以下几种

- 内存
- 本地文件系统
- 分布式文件系统
- nosql
- DB

从速度上内存显然是最快的，对于允许消息丢失，消息堆积能力要求不高的场景(例如日志)，内存会是比较好的选择。

DB则是最简单的实现可靠存储的方案，很适合用在可靠性要求很高，最终一致性的场景(例如交易消息)，对于不需要100%保证数据完整性的场景，要求性能和消息堆积的场景，hbase也是一个很好的选择。

理论上，从速度来看，文件系统>分布式KV（持久化）>分布式文件系统>数据库，而可靠性却截然相反。

还是要从支持的业务场景出发作出最合理的选择，如果你们的消息队列是用来支持支付/交易等对可靠性要求非常高，但对性能和量的要求没有这么高，而且没有时间精力专门做文件存储系统的研究，DB是最好的选择。

对于不需要100%保证数据完整性的场景，要求性能和消息堆积的场景，hbase也是一个很好的选择，典型的比如 kafka的消息落地可以使用hadoop。

消费关系处理

现在我们的消息队列初步具备了转储消息的能力。

下面一个重要的事情就是解析发送接收关系，进行正确的消息投递了。

市面上的消息队列定义了一堆让人晕头转向的名词，如JMS 规范中的Topic/Queue，Kafka里面的Topic/Partition/ConsumerGroup，RabbitMQ里面的Exchange等等。

抛开现象看本质，无外乎是**单播与广播**的区别。

所谓单播，就是点到点；而广播，是一点对多点。

为了实现广播功能，我们必须要维护消费关系，通常消息队列本身不维护消费订阅关系，可以利用zookeeper等成熟的系统维护消费关系，在消费关系发生变化时下发通知。

5. 消息队列需要支持高级特性

除了上述的消息队列基本功能以外，消息队列在某些特殊的场景还需要支持事务，消息重试等功能。

- 消息的顺序
- 投递可靠性保证
- 消息持久化
- 支持不同消息模型
- 多实例集群功能
- 事务特性等