neoremind

搜索文章...

消息队列技术点梳理（思维导图版）

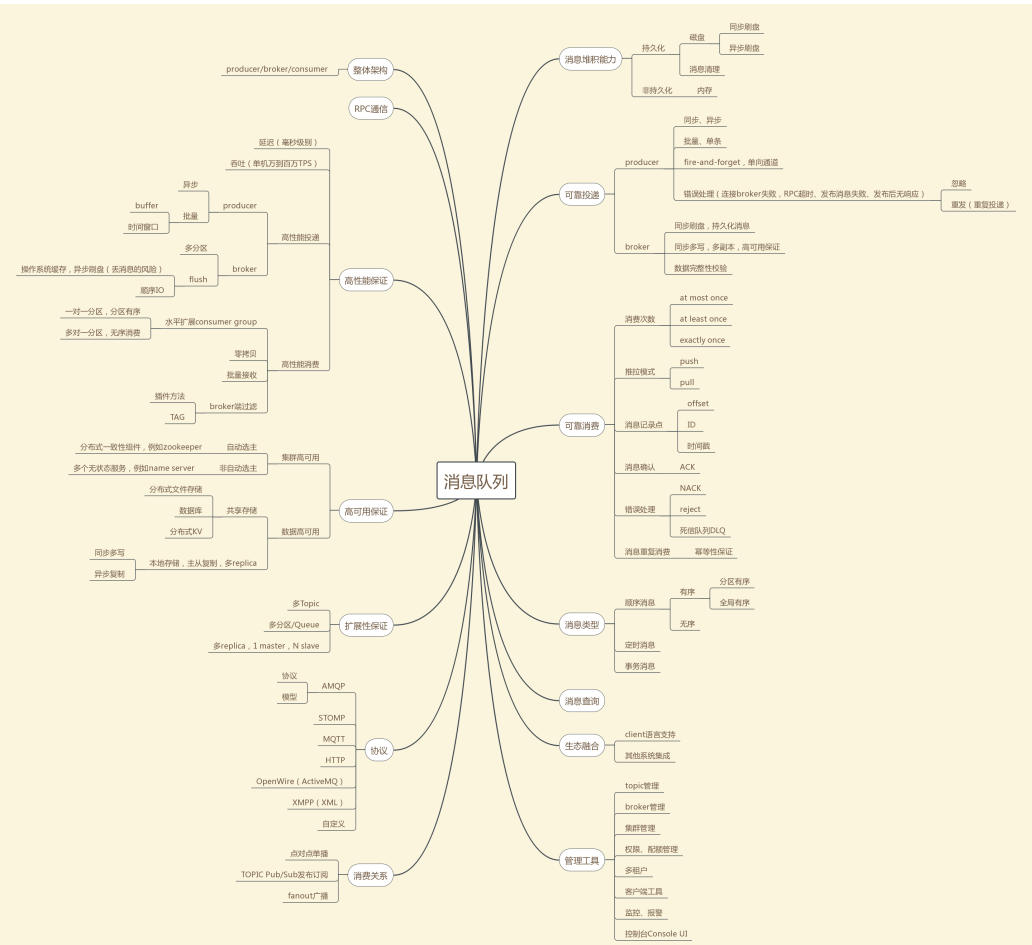
三月 18, 2018

消息队列作为服务/应用之间的通信中间件，可以起到业务耦合、广播消息、保证最终一致性以及错峰流控（克服短板瓶颈）等作用。本文不打算详细深入讲解消息队列，而是体系化的梳理消息队列可能涉及的技术点，起到提纲挈领的作用，构造一个宏观的概念，使用思维导图梳理。

再介绍之前，先简短比较下RPC和消息队列。RPC大多属于请求-应答模式，也包括越来越多响应式范式，对于需要点对点交互、强事务保证和延迟敏感的服务/应用之间的通信，RPC是优于消息队列的。那么消息队列（下文也简称MQ，即Message Queueu）可以看做是一种异步RPC，把一次RPC变为两次，进行内容转存，再在合适的时机投递出去。消息队列中间件往往是一个分布式系统，内部组件间的通信仍然会用到RPC。

目前开源界用的比较多的选型包括，ActiveMQ、RabbitMQ、Kafka、阿里巴巴的Notify、MetaQ、RocketMQ。下文的技术点梳理也是学习借鉴了这些开源组件，然后萃取出一些通用技术点。

关于消息队列的体系化认知，见下方的思维导图。



1. 整体架构

一般分为producer，broker，consumer三者。

2. RPC通信

详细参考《体系化认识RPC》。

3. 高性能保证

主要考虑MQ的延迟和吞吐。

高性能投递方面，分为producer和broker考虑。producer可以同步变异步、单条变批量保证发送端高性能，批量发送的触发条件可以分为buffer满或者时间窗口到了。broker可以进行多topic划分，再多分区/queue来进行分治（Divide and Conquer）策略，加大并行度，分散投递压力。另外broker对于需要持久化的消息，可以使用顺序IO，page cache，异

步刷盘等技术提高性能，但是异步刷盘在掉电的情况下，可能会丢失数据，可以结合下面的高可用方案，在数据严格不丢和高性能吞吐之间做折中。

高性能消费，即consumer和broker通信，进行推、拉消息。使用consumer group水平扩展消费能力，需要按照业务场景使用分区有序或者无序消费。零拷贝技术节省broker端用户态到内核态的数据拷贝，直接从page cache发送到网络，从而最大化发送性能。consumer批量pull，broker批量push。broker端还可以做消息过滤，可通过tag或者插件实现。

4. 高可用保证

主要针对broker而言。

集群高可用，producer通过broker投递消息，所以必然有且仅有一个broker主负责“写”，选主策略分为自动选主和非主动选择，自动选主使用分布一致性组件完成，例如Kafka使用zookeeper，非自动选主，例如RocketMQ依赖多个无状态的name server。

数据高可用，针对broker持久化积压消息场景。可借助分布式存储完成，但是往往性能上是个短板，所以大多数主流产品都进行本地IO顺序写，进行主从备份，多副本拷贝保证可用性，例如RocketMQ分为同步双写和异步复制，前者像HDFS一样，写完多个副本再返回producer成功，有一定性能损失，但不大，后者最大化性能，但是当主挂的时候，数据有丢失风险。

同样，MQ集群也需要考虑跨机房高可用（非“异地多活”），broker的写高可用，要考虑最小化MTTR，同时不阻塞consumer消费。

5. 扩展性保证

采用分治（Divide and Conquer）策略，加大投递和消费的并行度，多个topic、多个分区/queue、多个副本、多个slave或者镜像。

6. 协议

producer、consumer和broker通信的协议，包括AMQP、STOMP、MQTT、HTTP、OpenWire（ActiveMQ）、XMPP、自定义等等。

AMQP是比较全面和复杂的一个协议，包括协议本身以及模型（broker、exchange、routing key等概念），目前RabbitMQ是AMQP消息队列最有名的开源实现，有非常多语言已经支持基于AMQP协议与消息队列通信，同时还可以通过插件支持STOMP、MQTT等协议接入。Kafka、RocketMQ均使用自定义的协议。

7. 消费关系

包括三种
1) 点对点，也就是P2P，FIFO的队列，可以看做单播。
2) Topic模式，Pub/Sub发布订阅。
3) fanout广播模式。

8. 消息堆积能力

持久化消息，如果存储在本地磁盘，可以使用同步刷盘和异步刷盘两种策略。磁盘不能无限堆积，会有清理策略，例如Kafka、RocketMQ都按照时间、数据量进行retention。

非持久化，仅放在内存，消费者处理完可选择删除掉。

9. 可靠投递

对于producer，从API和I/O层面可使用同步、异步，对于吞吐层面可使用单条、批量。fire-and-forget模式，类似UDP，尽管发送即可。针对可能发生的错误，例如连接broker失败，RPC超时、发布消息失败、发布后无响应，可选择忽略或者重发，所以往往重复投递的情况不可避免。

对于broker，如果要保证数据100%不丢，是可能的，但是需要牺牲下性能和吞吐，使用同步多写、多副本策略+同步刷盘持久化消息，可以严格保证不丢。另外，broker对于写入消息的payload，也会做完整性校验，例如CRC等。

10. 可靠消费

消费次数，包括at most once、at least once、exactly once，其中前两个比较好做到，最后的exactly once需要streaming consumer系统和broker端协作完成，例如storm的trident和flink。

推拉模式，push or pull。推模式最小化投递延迟，但是没有考虑consumer的承载能力，拉一般是轮询接收broker的数据，按照consumer自己的能力消费。

消费记录点，一般每个消息都有一个offset、ID或者时间戳，consumer可以按照这个offset来进行定点消费以及消息重放。

消息确认，consumer消费完成ACK回调broker或者集群高可用中间件（zk）通知消费进度。

错误处理，对于消费失败的情况，可以回复NACK，要求重发/requeue消息，当错误超多一定阈值时候，放到死信队列中。

消息重复消费，这和消费次数有关系，consumer在某些时候需要做到幂等性，保证重复消费不会引起业务异常。

11. 消息类型

顺序消息，有序的话，分为分区有序或者全局有序，前者可以按照某个业务ID取模，在发送端发到不同的分区/queue即可，后者往往需要单个队列才可以满足。无序消费则可最大化吞吐。

定时消息，事务消息，例如RocketMQ均支持。

12. 消息查询

目前RocketMQ支持消息根据msgId查询。

13. 生态融合

客户端语言的丰富性，与其他系统的集成度，例如Kafka和大数据技术栈融合很紧密，Spark、Storm、Flink、Kylin都有对应的connector。

14. 管理工具

分布式系统的管理是提高生产效率的必备保障，一个好的系统，如果周边工具不完善，对于使用者会很不友好，推广也会有困难。

对于消息队列，可以从topic管理、broker管理、集群管理、权限/配额管理、多租户、客户端工具、监控、报警、控制台Console UI来全方位进行

治理。

总结

由于笔者经验所限，已尽可能广泛且全面的梳理，日后随着认识的深入，会不断的更新材料，也欢迎读者指出问题，欢迎交流。

转载时请注明转自neoremind.com。

6 Comments on this Post.

开发者头条 2018/03/21 ·

感谢分享！已推荐到《开发者头条》：
https://toutiao.io/posts/0f2ptm 欢迎点赞支持！
使用开发者头条 App 搜索 9199 即可订阅《Java技术路》

薛定谔的汪 2018/03/21 ·

很赞！

neo 2018/03/22 ·

谢谢支持！

zenk 2018/03/23 ·

赞，好文章
请教博主，你的UML图是用什么画的，感谢

neo 2018/03/26 ·

processon.com

zenk 2018/03/28 ·

感谢，博主

Leave a Comment.

Enter your Name...

Enter your Email Address...

Enter your Website...

Submit Comment