软件复用第二次讨论内容

复用产品方案

```
1352875 黄安娜
```

讨论内容

参考业界架构，讨论程序的扩展

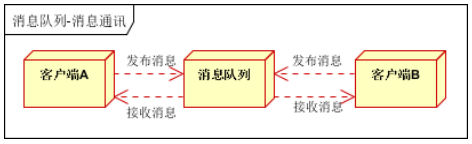
> - 分布式：高可用性，高吞吐量
> - 数据存储，分区，一致性，缓存
> - 负载均衡
> - 系统监控
> - ID分配
> - 通信可靠高效
> - 协议
> - 消息队列
> - 垃圾消息过滤
> - 安全

分析

该软件复用项目是“聊天室”的原型，但是为了满足真正的“聊天室“的要求，不仅要实现相应的功能，更要考虑相关的性能问题，本次讨论课就是要参考业界的架构，来分析讨论上述问题。

引言

如果使用标准socket进行传递的话，扩展性会很低，也就是说，随着数据量的扩大，单点会成为瓶颈，而且要确保较高的可用性，异步、缓存、重传等等都是需要考虑的问题。因此，业界常用的解决方案是使用消息队列而不是标准socket。



这里我找到了多种消息队列产品可供选择，这些消息队列各有优缺点，我首先比较一下常用的消息队列，接着选取阿里巴巴的消息中间件产品RocketMQ这一3PP产品就分布式、数据存储、负载均衡等上面十个方面进行具体的讨论分析。

常用消息队列的比较

- RabbitMQ

  RabbitMQ是使用Erlang编写的一个开源的消息队列，本身支持很多的协议：AMQP，XMPP, SMTP, STOMP，也正因如此，它非常重量级，更适合于企业级的开发。同时实现了Broker构架，这意味着消息在发送给客户端时先在中心队列排队。对路由，**负载均衡**或者数据持久化都有很好的支持。

- Redis

  Redis是一个基于Key-Value对的NoSQL数据库，开发维护很活跃。虽然它是一个Key-Value数据库存储系统，但它本身支持MQ功能，所以完全可以当做一个轻量级的队列服务来使用。对于RabbitMQ和Redis的入队和出队操作，各执行100万次，每10万次记录一次执行时间。测试数据分为128Bytes、512Bytes、1K和10K四个不同大小的数据。实验表明：入队时，当数据比较小时Redis的性能要高于RabbitMQ，而如果数据大小超过了10K，Redis则慢的无法忍受；出队时，无论数据大小，Redis都表现出非常好的性能，而RabbitMQ的出队性能则远低于Redis。

- ZeroMQ

ZeroMQ号称最快的消息队列系统，尤其针对**大吞吐量**的需求场景。ZMQ能够实现RabbitMQ不擅长的高级/复杂的队列，但是开发人员需要自己组合多种技术框架，技术上的复杂度是对这MQ能够应用成功的挑战。ZeroMQ具有一个独特的非中间件的模式，你不需要安装和运行一个消息服务器或中间件，因为你的应用程序将扮演了这个服务角色。你只需要简单的引用ZeroMQ程序库，可以使用NuGet安装，然后你就可以愉快的在应用程序之间发送消息了。但是ZeroMQ仅提供非持久性的队列，也就是说如果宕机，数据将会丢失。其中，Twitter的Storm 0.9.0以前的版本中默认使用ZeroMQ作为数据流的传输（Storm从0.9版本开始同时支持ZeroMQ和Netty作为传输模块）。

- ActiveMQ

  ActiveMQ是Apache下的一个子项目。 类似于ZeroMQ，它能够以代理人和点对点的技术实现队列。同时类似于RabbitMQ，它少量代码就可以高效地实现高级应用场景。

- Kafka/Jafka

  Kafka是Apache下的一个子项目，是一个高性能跨语言分布式发布/订阅消息队列系统，而Jafka是在Kafka之上孵化而来的，即Kafka的一个升级版。具有以下特性：快速持久化，可以在O(1)的系统开销下进行消息持久化；**高吞吐**，在一台普通的服务器上既可以达到10W/s的吞吐速率；**完全的分布式系统**，Broker、Producer、Consumer都原生自动支持分布式，**自动实现负载均衡**；支持Hadoop数据并行加载，对于像Hadoop的一样的日志数据和离线分析系统，但又要求实时处理的限制，这是一个可行的解决方案。Kafka通过Hadoop的并行加载机制来统一了在线和离线的消息处理。Apache Kafka相对于ActiveMQ是一个非常轻量级的消息系统，除了性能非常好之外，还是一个工作良好的分布式系统。

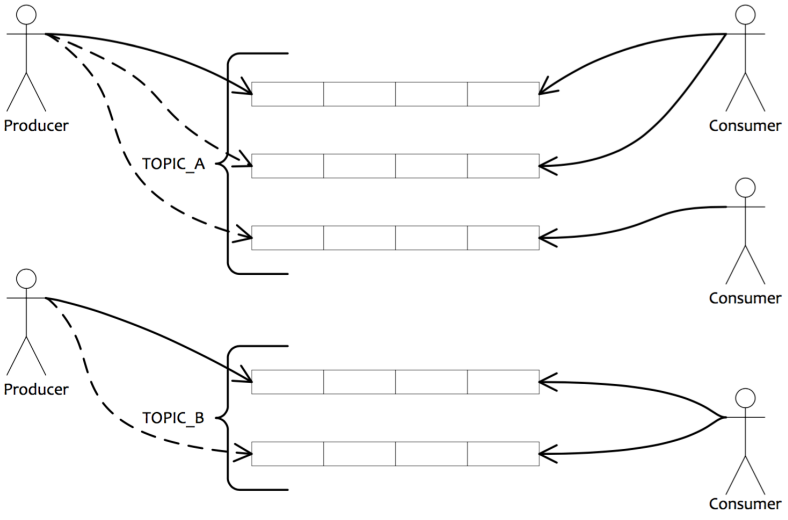
- RocketMQ

  淘宝内部的交易系统使用了淘宝自主研发的Notify消息中间件，使用Mysql作为消息存储媒介，可完全水平扩容，为了进一步降低成本，进一步优化存储部分，2011年初，Linkin开源了Kafka这个优秀的消息中间件，淘宝中间件团队在对Kafka做过充分Review之后，Kafka无限消息堆积，高效的持久化速度吸引了我们，但是同时发现这个消息系统主要定位于日志传输，对于使用在淘宝交易、订单、充值等场景下还有诸多特性不满足，为此我们重新用Java语言编写了RocketMQ，定位于非日志的可靠消息传输（日志场景也OK），目前RocketMQ在阿里集团被广泛应用在订单，交易，充值，流计算，消息推送，日志流式处理，binglog分发等场景。

讨论分析--RocketMQ

RocketMQ是什么

RocketMQ是一个队列模型的消息中间件,具有高性能、高可靠、高实时、分布式特点。



术语

- Producer

  消息生产者,负责产生消息,一般由业务系统负责产生消息

- Consumer

消息消费者,负责消费消息,一般是后台系统负责异步消费。

- Push Consumer

  Consumer 的一种,应用通常向 Consumer 对象注册一个 Listener 接口,一旦收到消息,Consumer 对象立 刻回调 Listener 接口方法。

- Pull Consumer

  Consumer 的一种,应用通常主动调用 Consumer 的拉消息方法从 Broker 拉消息,主动权由应用控制。

- Producer Group

  一类 Producer 的集合名称,这类 Producer 通常发送一类消息,且发送逻辑一致。

- Broker

  消息中转角色,负责存储消息,转发消息,一般也称为 Server。在 JMS 规范中称为 Provider。

- Message Queue

  在 RocketMQ 中,所有消息队列都是持久化,长度无限的数据结构,所谓长度无限是指队列中的每个存储 单元都是定长,访问其中的存储单元使用 Offset 来访问,offset 为 java long 类型,64 位,理论上在 100 年内不会溢出,所以认为是长度无限,另外队列中只保存最近几天的数据,之前的数据会按照过期时间来删除。

也可以认为 Message Queue 是一个长度无限的数组,offset 就是下标。

分布式

分布式系统

Producer、Consumer、队列都可以分布式。

分布式事务

已知的几个分布式事务规范,如 XA,JTA 等。其中 XA 规范被各大数据库厂商广泛支持,如 Oracle,Mysql 等。 其中 XA 的 TM 实现佼佼者如 Oracle Tuxedo,在金融、电信等领域被广泛应用。

分布式事务涉及到两阶段提交问题,在数据存储方面必然需要 KV 存储的支持,因为第二阶段的提交回滚需要修改消息状态,一定涉及到根据 Key 去查找 Message 的动作。RocketMQ 在第二阶段绕过了根据 Key 去查找 Message 的问题,采用第一阶段发送 Prepared 消息时,拿到了消息的 Offset,第二阶段通过 Offset 去访问消息, 并修改状态,Offset 就是数据的地址。

RocketMQ 这种实现事务方式,没有通过 KV 存储做,而是通过 Offset 方式,存在一个显著缺陷,即通过 Offset 更改数据,会令系统的脏页过多,需要特别关注。

高性能

1、同一网络内，消息传输RT在10毫秒之内，性能测试下，网卡可被打满

2、公有云默认单Topic 发送消息为每秒5000条，最高可申请扩展至10W以上 支持大量消息并发发送，超过5万个队列，性能依然卓越

3、支持消息海量堆积，单Topic可堆积100+亿条消息

4、单条消息默认最大支持256K，北京为4M

数据存储

消息中间件通常采用的几种持久化方式:

(1). 持久化到数据库,例如 Mysql。

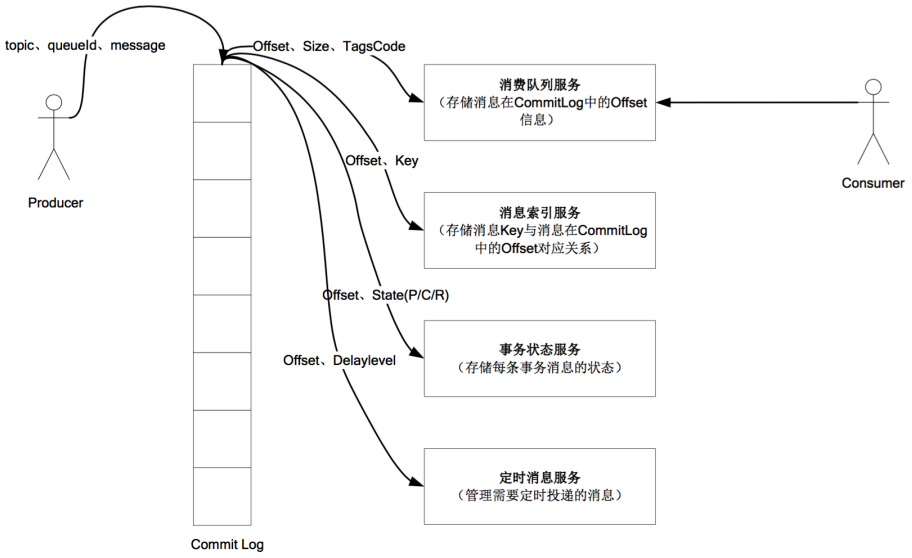
(2). 持久化到KV存储,例如levelDB、伯克利DB等KV存储系统。

(3). 文件记录形式持久化,例如 Kafka,RocketMQ

(4). 对内存数据做一个持久化镜像,例如 beanstalkd,VisiNotify

(1)、(2)、(3)三种持久化方式都具有将内存队列 Buffer 进行扩展的能力,(4)只是一个内存的镜像,作用是当 Broker 挂掉重启后仍然能将之前内存的数据恢复出来。 JMS 与 CORBA Notification 规范没有明确说明如何持久化,但是持久化部分的性能直接决定了整个消息中间件 的性能。 RocketMQ 参考了 Kafka 的持久化方式,充分利用 Linux 文件系统内存 cache 来提高性能。

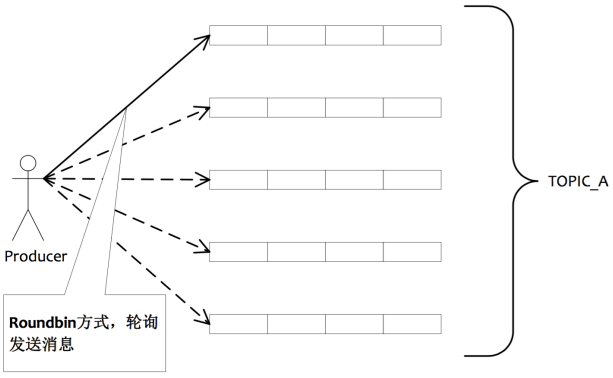
- 文件系统 RocketMQ 选择 Linux Ext4 文件系统,原因如下: Ext4 文件系统删除 1G 大小的文件通常耗时小于 50ms,而 Ext3 文件系统耗时约 1s 左右,且删除文件时,磁盘 IO 压力极大,会导致 IO 写入超时。
- 数据存储结构



如上图所示， - 所有数据单独存储到一个 Commit Log，完全顺序写，随机读 - 对最终用户展现的队列实际只存储消息在 Commit Log 的位置信息

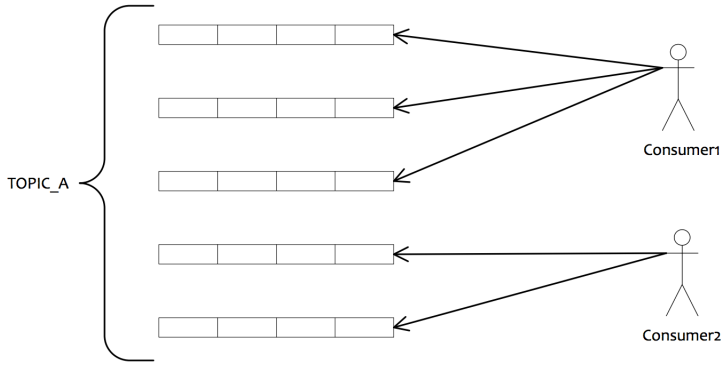
负载均衡(Rebalance)

- 发送消息负载均衡



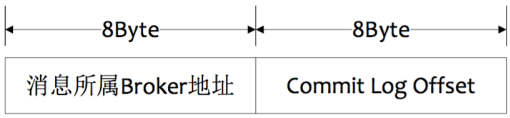
如图所示，5 个队列可以部署在一台机器上，也可以分别部署在 5 台不同的机器上，发送消息通过轮询队列的方式发送，每个队列接收平均的消息量。通过增加机器，可以水平扩展队列容量。另外也可以自定义方式选择发往哪个队列。

- 订阅消息负载均衡



如上图所示，如果有 5 个队列，2 个 consumer，那么第一个 Consumer 消费 3 个队列，第二 consumer 消费 2 个队列。这样即可达到平均消费的目的，可以水平扩展 Consumer 来提高消费能力。但是 Consumer 数量要小于等于队列数量，如果 Consumer 超过队列数量，那么多余的 Consumer 将不能消费消息。

ID分配



MsgId总共16个字节，包含消息储存主机地址，消息Commit Log Offset。从MsgId中解析出Broker的地址和Commit Log 偏移地址，然后按照存储格式所在位置消息buffer解析成一个完整消息

通信可靠高效

- 影响消息可靠性的几种情况:

(1). Broker 正常关闭

(2). Broker 异常 Crash

(3). OS Crash

(4). 机器掉电,但是能立即恢复供电情况。

(5). 机器无法开机(可能是cpu、主板、内存等关键设备损坏)

(6). 磁盘设备损坏。

```
(1)、(2)、(3)、(4)四种情况都属于硬件资源可立即恢复情况,RocketMQ 在这四种情况下能保证消息不丢,或者丢失少量数据(依赖刷盘方式是同步还是异步)。

(5)、(6)属于单点故障,且无法恢复,一旦发生,在此单点上的消息全部丢失。RocketMQ 在这两种情况下,通过异步复制,可保证 99%的消息不丢,但是仍然会有极少量的消息可能丢失。通过同步双写技术可以完全避免单点, 同步双写势必会影响性能,适合对消息可靠性要求极高的场合,例如与交易相关的应用。

RocketMQ 从 3.0 版本开始支持同步双写。
```

- At least once

是指每个消息必须投递一次 RocketMQ Consumer 先 pull 消息到本地,消费完成后,才向服务器返回 ack,如果没有消费一定不会 ack 消息, 所以 RocketMQ 可以很好的支持此特性。由于这一特性，RocketMQ能够保证通信的可靠性。

协议

RocketMQ 通信组件使用了 Netty-4.0.9.Final,在之上做了简单的协议封装。

1. length: 4个字节整数，等于 2、3、4 字段的长度总和
2. header length: 4个字节整数，等于 3 的长度
3. 使用 json 序列化数据
4. 应用自定义二进制序列化数据

Header格式：

```
json { "code": 0, "language": "JAVA", "version": 0, "opaque": 0, "flag": 1, "remark": "hello, I am res
```

Header 字段名	类型	Request	Response
code	整数	请求操作代码，请求接收方根据不同的代码做不同的操作	应答结果代码，0 表示成功，非 0 表示各种错误代码
language	字符串	请求发起方实现语言，默认JAVA	应答接收方实现语言
version	整数	请求发起方程序版本	应答接收方程序版本
opaque	整数	请求发起方在同一连接上不同的请求标识代码，多线程连接复用使用	应答方不做修改，直接返回
flag	整数	通信层的标志位	通信层的标志位
remark	字符串	传输自定义文本信息	错误详细描述信息
extFields	HashMap<String,String>	请求自定义字段	应答自定义字段

消息队列

在 RocketMQ 中,所有消息队列都是持久化,长度无限的数据结构,所谓长度无限是指队列中的每个存储单元都是定长,访问其中的存储单元使用 Offset 来访问,offset 为 java long 类型,64 位,理论上在 100 年内不会溢出,所以认为是长度无限,另外队列中只保存最近几天的数据,之前的数据会按照过期时间来 删除。

也可以认为 Message Queue 是一个长度无限的数组,offset 就是下标。

消息过滤

- Broker 端消息过滤 在 Broker 中,按照 Consumer 的要求做过滤,优点是减少了对于 Consumer 无用消息的网络传输。 缺点是增加了 Broker 的负担,实现相对复杂。 (1) 淘宝 Notify 支持多种过滤方式,包含直接按照消息类型过滤,灵活的语法表达式过滤,几乎可以满足最苛刻的过滤需求。 (2) 淘宝 RocketMQ 支持按照简单的 Message Tag 过滤,也支持按照 Message Header、body 进行过滤。 (3) CORBA Notification 规范中也支持灵活的语法表达式过滤。
- Consumer 端消息过滤 这种过滤方式可由应用完全自定义实现,但是缺点是很多无用的消息要传输到 Consumer端。
- 服务器消息过滤

RocketMQ 的消息过滤方式有别于其他消息中间件，是在订阅时，再做过滤，先来看下 Consume Queue 的存储 结构。

(1)在Broker端进行 Message Tag 比对，先遍历 Consume Queue，如果存储的 Message Tag 与订阅的 Message Tag不符合，则跳过，继续比对下一个，符合则传输给 Consumer。注意：Message Tag 是字符串形式，Consume Queue 中存储的是其对应的 hashcode，比对时也是比对 hashcode。 (2)Consumer 收到过滤后的消息后，同样也要执行在 Broker 端的操作，但是比对的是真实的

Message Tag 字 符串，而不是 Hashcode。

- **垃圾消息过滤**一般步骤
    1. 首先采集一定数量的信息,建立相应的垃圾信息集和合法信息集
    2. 信息要进行预处理,信息预处理的过程也就是对信息进行空间向量化的过程
    3. 利用相应的分类算法对已知的垃圾信息样本集进行训练,统计相应数据,获得相应参数和阈值,构建分类器
    4. 利用获得的分类器,对未知信息进行过滤

安全

- 信息加密 常用的加密算法有MD5、TEA等
- 增强可靠性

参考资料

- RocketMQ开源: https://github.com/alibaba/RocketMQ/
- RocketMQ的原理与实践: http://www.jianshu.com/p/453c6e7ff81c#
- RocketMQ原理简介: http://alibaba.github.io/RocketMQ-docs/document/design/RocketMQ_design.pdf
- 阿里中间件团队博客: http://jm.taobao.org/2016/03/24/rmq-vs-kafka/#more
- 基于支持向量机的垃圾信息过滤方法: http://journal.bit.edu.cn/zr/ch/reader/create_pdf.aspx?file_no=20131013
- Kafka深度解析:http://www.jasongj.com/2015/01/02/Kafka%E6%B7%B1%E5%BA%A6%E8%A7%A3%E6%9E%90/