消息队列

# 消息队列的使用场景

主要解决应用耦合,异步消息,流量削锋

实现高可用,高性能,可伸缩和最终一致性

常用activeMq,rabbitMQ, ZeroMQ,kafka,MetaMQ,RocketMQ

四个场景:异步处理,应用解耦,流量削锋,消息通讯

**异步处理**:用户注册后,需要完成发送邮件和注册短信,串行方式和并行方式

串行方式:注册信息写入数据库,发送邮件,再发送注册短信.三种完成后,返回个客户端.

并行方式:将注册信息写入数据库后,发送邮件的通知,发送短信.

三者完成后,返回客户端,并行能够提高处理的时间写入.

写入消息队列的时间很短,基本可以忽略.

**应用解耦:**

用户添加订单后,订单需要通知库存系统

传统做法:订单系统调用库存系统接口

缺点:如果库存系统无法访问,择订单减库存失败,从而导致订单失败.

订单系统与库存系统耦合.

消息队列:

用户下单后,订单完成持久化处理,将消息写入消息队列,返回用户订单下单成功,

库存系统,订阅下单的信息,采用推拉的方式,获取下单信息,库存系统根据下单信息,进行库存操作,

**流量削锋:**

一般在秒杀或团抢活动中广泛使用,

应用场景,秒杀活动,一般会因为流量过大,导致流量暴增,应用前端加入消息队列,可以控制活动的人数,可以缓解短时间内高流量压垮引用,用户请求写入到消息队列,如果消息队列长度超过最大长度,直接抛弃用户请求,或跳转到错误页面,秒杀业务根据消息队列中的请求,做后续处理.

日志处理:日志采集客户端,负责日志数据采集,定时写入kafka队列,kafaka消息队列负责日志数据的接受存储转发,日志处理应用,订阅并消费kafka队列中的日志数据.

**消息通讯:**

消息队列一般都内置了高效的通信机制,可以进行纯的通信.

**消息中间件:**

应用将主干逻辑处理完成后,写入消息队列,消息发送是否成功可以开启消息的确认模式.(消息队列接受成功状态后,应用再返回,保障消息的完整性)

扩展流程(发短信,配送处理)订阅队列消息,采用推或拉的方式,获取消息并处理.

消息将应用解耦的同时,带来了数据一致性的问题,可以采用最终一致性方式解决,比如主数据写入数据库,扩展应用根据消息队列,并结合数据库方式实现基于消息队列的后续处理.

**日志收集系统**

包含zookeeper注册中心,日志收集客户端,kafka集群,storm集群

zookeeper注册中心,提出负载均衡和地址查找服务,

日志收集客户端,用于采集应用系统的日志,并将数据推送到kafka队列

kafka集群:接受路由,存储,转发等消息处理

Storm集群:采用辣的方式消费消息队列中的数据.

**JMS消息服务**

是消息服务的标准规范,允许应用程序组件基于java创建发送接收读取消息,使分布式通信耦合度降低,消息服务更加可靠,以及异步性.

消息模型:P2P,publish/subcribe

**p2p:**消息队列,发送者,消费者,每个消息都被发送到特定的队列,接受者从队列中获取消息,队列保留消息,直到他们被消费或超时.

每个消息只有一个消费者,一旦被消费就不存在,

发送者和接受者之间在时间上没有依赖性.

接受者须在成功接收消息后向队列应答成功.

如果希望发送的每个消息都被成功处理,使用p2p模式.

**Pub/Sub模式:**

包含topic,发布者,订阅者

多个发布者将消息发送到Topic,系统将这些消息传递给多个订阅者。

每个消息可以有多个消费者，

发布真和订阅者有时间上的依赖性，针对某个主题非订阅者，他必须创建一个订阅者后，才能消费发布者的消息。

为了消费消息订阅者，订阅者必须保持运行的状态。

为了缓和严格的时间相关性，允许订阅者创建一个可持久化的订阅，订阅没有被运行，也能收到发布者的消息。

**消息消费:**

消息的生产和消费都是异步的,同步消费消息和一部消费消息.

同步:订阅者或接受者通过receive接收消息,在接收消息之前一直阻塞.

异步:可以注册为一个消息监听器,系统自动调用监听器的onMessage方法.

JNDI:标准的java命名系统接口

**jms编程模型**

连接工厂connectionFactory,Quene工厂和Topic工厂.

Destination:消息发送目标或者消费消息来源.主要是某个队列或者主题(topic)

connection:客户端和jms系统间建立连接,可以产生一个或者多个session,也有两种类型Quene和Topic

session:是操作信息的接口,可以通过session创建生产者,消费者,消息.session提供了事物的功能.当使用session发送接收多个消息,可以将动作放到一个事务中,分为QueneSession和TopicSession.

生产者(调用send/publish)

消费者(createReceiver/createSubscriber()) 创建持久化的订阅者.

messageListener消息监听器onMessage方法.

**常用的消息队列**

**ActiveMQ**

1.多语言多协议

2.支持持久化,XA消息,事务

3.支持spring

4.支持常见服务器

5.支持多种传输协议

6.支持JDBC和journal提供的高速的持久化

7.保证了几群的高性能,点对点

8.支持Ajax

9.支持与Axis(???)的整合

10.很容易调用内嵌JMS provider,进行测试

**RabbitMQ**

1.erlang开发,需要erlang的编译环境

2.是AMQP的标准实现

3.支持多种客户端支持ajax,持久化

4.用于在分布式系统中存储转发消息,易用,高扩展,高可用.

5.broker:消息队列服务器实体,

6.exchange:消息交换机,指定消息按照规则,路由到那个队列

7.Quene:消息队列载体,每个消息都会被投放到一个或者多个队列

8.binding:绑定,把交换机和队列载体按路由规则绑定在一起

9.Routing key :路由关键字,交换机根据关键字进行消息传递

10.vhost:虚拟主机.一个broker中可以开设多个vhost,用作不同用户的权限分离

11.producer:生产者

12.consumer:消费者

13.channel:消息通道,在客户端的每个连接里,可建立多个channel,每个channel代表一个回话任务

**客户端连接到 队列服务器,打开一个channel消息通道,声明一个exchange消息交换机(根据规则路由队列),声明一个消息队列,使用routing key 在交换机和队列之间建立好绑定关系.消息投递到exchange交换机,交换机得到消息后,根据消息的key和已设置的binding,进行消息路由,将消息投递到一个或者多个队列.**

**ZeroMQ:**

史上最快的消息队列,类似于socket的一系列接口,普通的socket是端对端的关系(1:1),ZMQ是多对多的关系,点对点需要显式的创建连接,销毁连接,选择协议,处理错误,ZMQ忽略这些,用于node与node的通信,node可以是主机或者进程.

特点:高性能非持久化,跨平台,多语言,单独部署,或者集群,可作为socket通讯库使用,

与RMQ相比,ZMQ更像一个底层的网络通讯库,,在socketapi上做了一层封装,将网络通讯,进程通讯和线程通讯,抽象为统一的接口.

zMQ采用无锁的cas队列算法,进行批量处理的算法优化,,多核的线程绑定,无需切换cpu.

**Kafka:**

是一个高吞吐量的分布式发布订阅消息系统,可以处理消费者规模的网站中的所有动作流数据.-这些数据通常是由吞吐量的要求,通过处理日志和日志聚合来解决,

Kafka的目的是通过Hadoop的并行加载机制,来统一线上和离线的消息处理,,通过集群机来实现实时的消费.

特性:通过磁盘数据持久化,保证存储长时间的稳定性能.

高吞吐量;非常普通的硬件也能支持每秒百万级的消息.

支持集群消费消息;

支持Hadoop并行加载数据.

broker:kafka包含的服务器,一个或者多个.

topic:每个发送的消息都有一个类别,不同topic分开存储,但读取时,不关心数据存与何处.

parition:物理上的概念,每个topic包含多个,parition

producer:发布者

consumer:消费者

consumer group

一般应用在大数据日志处理,或对实时性,可靠性要求较低的场景.

# 消息的重发补偿解决思路

1.那些情况需要重发

消费者处理完一条消息,没有回复,需要重发,如果采用非事务持久化消息,没有回复不会重发,宕机后,会重发,但消息不会有重发标记,不知道宕机,不知道消息被发送过.

队列设置了预读参数,在接收第一条消息时,宕机了,预读数据量的消息都将被重发.

Session是事务的,只要有一条消息没有确认,出现宕机所有消息重发.

在客户端上有一套AMQ环境.判断各种消息的状况.

重发策略的配置

碰撞躲避因子,0.15 ,躲避高并发的重发带来的问题,,延迟发送时间默认是1000ms,躲避后幅度在850-1150之间.第二次的默认基数时间就不是1000了,

最大重发次数:6次,不限重发次数,设置为-1;

重发最大拖延时间;默认为-1,表示没有最大拖延时间

第一次重发的拖延时间基础,默认1000,

是否采用避免碰撞参数,

是否使用成倍拖延,默认false,成倍的倍率

# 消息的幂等性解决思路

幂等性:用户对于统一操作的发起的一次请求或者多次请求的结果是一致的,不会因为多次点击出现副作用.

单应用系统,将数据操作放于事务中,发生错误就会滚.但在此方位还肯能出现错误.

增删改查四个操作,主要是增加和修改

1.单次支付请求.也就是直接支付,不需要操作数据库,发起异步请求创建一个唯一的ticketId,用一次就作废

具体步骤:

异步请求获取门票,

调用支付传入门票,

根据id判断此次操作是否存在,存在表示操作已经执行,直接返回结果,不存在,支付付款,保存结果.

返回结果到客户端,如果失败,再次发送请求.

分布式环境各个服务相互调用,

查看订单的支付状态,如果已经支付,直接返回结果,如果未支付,则支付扣款并保存流水,返回支付结果。

MQ消息必达，架构上有两个核心设计点，

消息落地，消息超时重传，确认

发送端，服务端，固化存储，接收端。

为保证消息的可达性，超时重传，确认机制，可能导致消息总线或者业务方，收到重复的消息。

上半场:

对每条消息生成一个inner-msg-id,作为去重和幂等的依据,特点全局唯一,MQ生成,具备业务无关性,对消息发送和消息街搜房屏蔽.

下半场:业务消息体中,有一个BIZ-id;

对于同一个业务场景,全局唯一,

由业务消息发送方生成,业务相关,对MQ透明,

有业务消息方负责判重保证幂等.

# 消息的堆积解决思路

调用方法获取队列中的消息堆积量,消息放入前可调用此方法..然后判断是否立即放入.

在没有消费者的情况下,生产者持续向mq发送消息,是的消息在mq中大量堆积,发送速率不受影响,但当有新的消费者连接上mq时,生产速率答复降低.

# 如何保证消息的有序性

一般来说，消息队列都会保证queue当中的消息的顺序。然而如果有多个consumer同时消费同一个queue，那么这时就不能保证的消息的顺序性。

针对单个消费者,通过某种算法,将需要保持先后顺序的消息放在同一个消息队列中,(kafka中是partition,rabbitMQ中是Quene),然后只用一个消费者去消费.

针对多个消费者,保证入队有序就行,出队以后的顺序交给消费者自己保证.

# 如何保证消息队列的高可用

以rocketMQ为例,他的集群模式就有master模式,多master多slave异步复制模式,多master多slave同步双写模式.主从