目录

[基础知识篇 1](#_Toc2178214)

[消息队列的概念 1](#_Toc2178215)

[概念模型 2](#_Toc2178216)

[1. 异步 2](#_Toc2178217)

[2 应用解耦 3](#_Toc2178218)

[3 流量削峰 3](#_Toc2178219)

[4. 日志处理 4](#_Toc2178220)

[5. 消息通讯 4](#_Toc2178221)

[消息模型 5](#_Toc2178222)

[1. PTP模型 5](#_Toc2178223)

[2. Sub/Pub模型 6](#_Toc2178224)

[4. JMS 编程模型 6](#_Toc2178225)

[实现队列需要注意的地方 8](#_Toc2178226)

[队列持久化 8](#_Toc2178227)

[消息顺序，消费顺序 8](#_Toc2178228)

[存在优先级 8](#_Toc2178229)

[事务 8](#_Toc2178230)

[深入篇 8](#_Toc2178231)

[ActiveMQ，Kafaka，RocketMQ 比较 8](#_Toc2178232)

[ActiveMQ 8](#_Toc2178233)

[Kafaka 8](#_Toc2178234)

[RocketMQ 8](#_Toc2178235)

# 基础知识篇

# 消息队列的概念

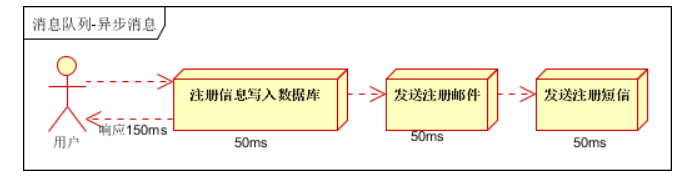
用来一个实现inter-process-communication,或者说是实现了inter-threads-communication within in a process.

强调的是用异步解耦的方式用于应用的解耦，异步消息，流量削峰的问题，最终实现分布式高可用，高性能，高一致性的架构。

## 概念模型

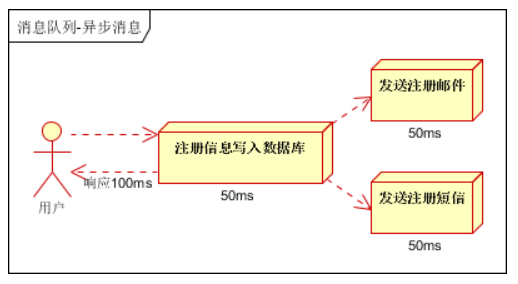
### 1. 异步

（1）在一般的**串行**模型中（常用的为邮件发送结构）：



这个模型总共响应时间为 3\*50ms=150ms

（2）通过**异步方式（并行）**解耦：

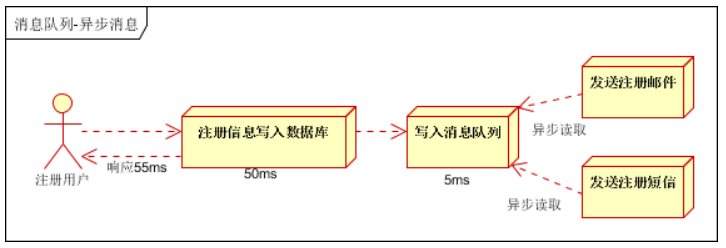


响应时间即刻降为2\*50 =100ms

因为CPU在单位时间内处理的请求数是一定的，假设CPU1秒内吞吐量是100次。则串行方式1秒内CPU可处理的请求量是7次（1000/150）。并行方式处理的请求量是10次（1000/100）（并行可以略高效与串行）

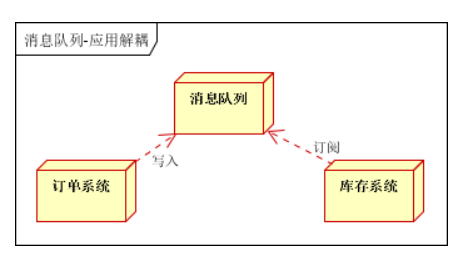
小结：如以上案例描述，传统的方式系统的性能（**并发量**，**吞吐量**，**响应时间**）会有瓶颈。如何解决这个问题呢？

（3）通过引入**消息队列**进行模型升级：



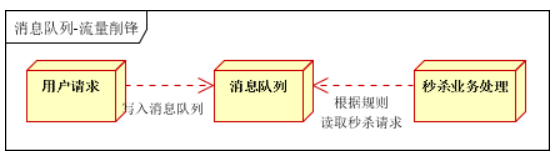
从图中就可以看出，注册这个**服务**对于整个应用而言是仅仅于在用户提交信息后将信息**写入数据库**后，继续将队列标识信息**写入消息队列**。整个过程 50ms+5ms=55ms

### 2 应用解耦



这是一个MQ典型的pub/sub模式， 订单系统在用户下单成功后， 对MQ进行publish。 而库存系统对于消息队列中的内容进行订阅抽取后进行操作。看的出来消息队列可以很成功的把两个系统进行解耦。Kiss(keep it simple and stupid )让每个系统的部分独立，逻辑清晰。

### 3 **流量削峰**



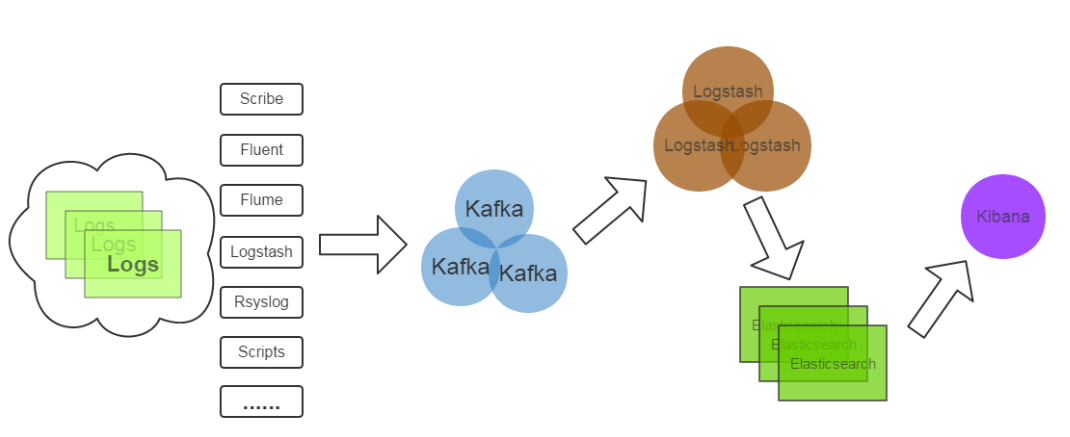
大量用户在面对秒杀系统的时候都有一个明显的特点，卡点，准时塞入大量请求，这个时候首先可以肯定的是串行式的系统必然爆炸。当然还有很多别的优化，在这里只针对MQ。

引入MQ，客户端或者浏览器只需要完成写入队列的这一步。后端业务系统只需要挨个取出相应处理。

**\***但我有个疑惑， 在流量削峰中的秒杀系统， 把用户请求写入队列， 系统取出队列挨个处理， 这个不就是很像代理模式吗，使用了代理模式，我们建立了一个代理类， 同样是用队列的结构去代理请求。 请问这两个的异同处在哪呢

### 4. 日志处理

以新浪日志处理架构为例，Kafka应用。



目标是解决大量日志传输问题。

Kafka： 接受日志的队列

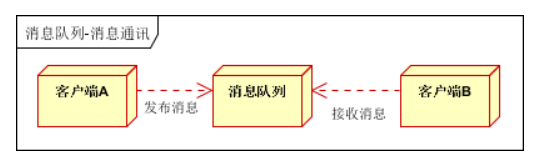
Logstash: 日志解析，JSON格式化传输给下一层

Elasticsearch: 实时日志分析服务的核心技术，一个schemaless，实时的数据存储服务，通过index组织数据，兼具强大的搜索和统计功能

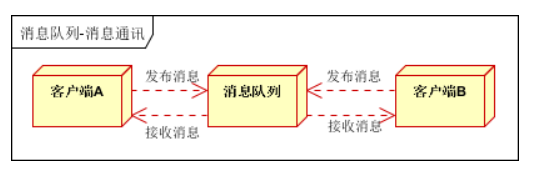
Kibana：基于Elasticsearch的数据可视化组件，超强的数据可视化能力是众多公司选择ELK stack的重要原因

### 5. 消息通讯

P2P模型： 两端使用同一队列

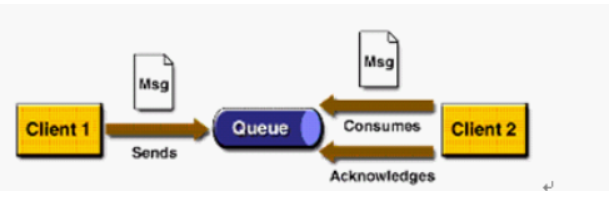


Pub/Sub模型： 中间连接为topic ， 双方订阅topic



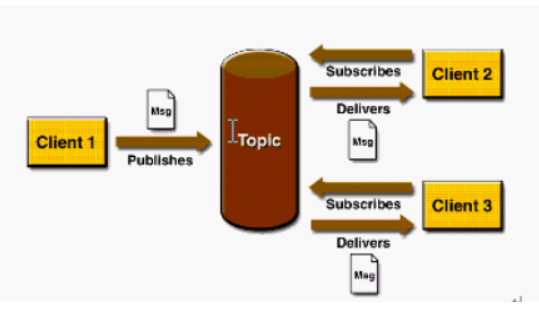
## 消息模型

### PTP模型



2P模式包含三个角色：**消息队列（Queue）**，**发送者(Sender)**，**接收者(Receiver)**。每个消息都被发送到**一个特定**的队列，接收者从队列中获取消息。队列保留着消息，**直到**他们被消费或超时。P2P的特点:每个消息只有一个**消费者**（Consumer）(即一旦被消费，消息就不再在消息队列中)发送者和接收者之间在时间上没有依赖性，也就是说当发送者发送了消息之后，不管接收者有没有正在运行，它不会影响到消息被发送到队列。接收者在成功接收消息之后需向队列应答成功 。如果希望发送的每个消息都会被成功处理的话，那么需要P2P模式。

### Sub/Pub模型

1. 

包含三个角色**主题（Topic）**，**发布者（Publisher）**，**订阅者（Subscriber）** 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。Pub/Sub的特点：每个消息可以有**多个**消费者**发布者和订阅者之间有时间上的依赖性**。针对某个主题（Topic）的订阅者，它必须创建一个订阅者之后，才能消费发布者的消息为了消费消息，订阅者必须保持运行的状态为了缓和这样严格的时间相关性，JMS允许订阅者创建一个**可持久化**的订阅。这样，即使订阅者没有被激活（运行），它也能接收到发布者的消息。如果希望发送的消息可以不被做任何处理、或者只被一个消息者处理、或者可以被多个消费者处理的话，那么可以采用Pub/Sub模型。

### 3. JMS 编程模型

(1) ConnectionFactory创建Connection对象的工厂，针对两种不同的jms消息模型，分别有QueueConnectionFactory和TopicConnectionFactory两种。可以通过JNDI来查找ConnectionFactory对象。

(2) DestinationDestination的意思是消息生产者的消息发送目标或者说消息消费者的消息来源。对于消息生产者来说，它的Destination是某个队列（Queue）或某个主题（Topic）;对于消息消费者来说，它的Destination也是某个队列或主题（即消息来源）。所以，Destination实际上就是两种类型的对象：Queue、Topic可以通过JNDI来查找Destination。(3) ConnectionConnection表示在客户端和JMS系统之间建立的链接（对TCP/IP socket的包装）。Connection可以产生一个或多个Session。跟ConnectionFactory一样，Connection也有两种类型：QueueConnection和TopicConnection。(4) SessionSession是操作消息的接口。可以通过session创建生产者、消费者、消息等。Session提供了事务的功能。当需要使用session发送/接收多个消息时，可以将这些发送/接收动作放到一个事务中。同样，也分QueueSession和TopicSession。(5) 消息的生产者消息生产者由Session创建，并用于将消息发送到Destination。同样，消息生产者分两种类型：QueueSender和TopicPublisher。可以调用消息生产者的方法（send或publish方法）发送消息。(6) 消息消费者消息消费者由Session创建，用于接收被发送到Destination的消息。两种类型：QueueReceiver和TopicSubscriber。可分别通过session的createReceiver(Queue)或createSubscriber(Topic)来创建。当然，也可以session的creatDurableSubscriber方法来创建持久化的订阅者。(7) MessageListener消息监听器。如果注册了消息监听器，一旦消息到达，将自动调用监听器的onMessage方法。EJB中的MDB（Message-Driven Bean）就是一种MessageListener。

# 实现队列需要注意的地方

## 队列持久化

消息持久性的原理很简单，就是在发送者将消息发送出去后，消息中心首先将消息存储到本地数据文件、内存数据库或者远程数据库等，然后试图将消息发送给接收者，发送成功则将消息从存储中删除，失败则继续尝试。消息中心启动以后首先要检查制定的存储位置，如果有未发送成功的消息，则需要把消息发送出去。

持久化部分，我认为关注的点应该在存储快速， 抓取速度在O（1）的水平线上，O（1）的时间就意味着桶，或者是key-value的方式。

以ActiveMQ为例，持久化的方式有:

1. 基于**文件形式**，写入文件，异步刷入磁盘（AMQ）
2. 基于**文件的本地数据库储存形式**，虽然没有AMQ的速度快，但是它具有强扩展性，恢复的时间比AMQ短(KahaDB) {相对Kahadb 还有一个levelDB，不同的是kaha基于传统b-树结构对日志数据的提前写，leveldb是基于索引}
3. 基于**数据库**，消息直接储存数据库（JDBC-MYSQL…..）

## 消息顺序，消费顺序

一 顺序消息MessageQueueSelector中用类似于hash的方法，将需要一致并且有某种共性的消息落入同一个队列。

二 消息重复

消费端处理消息的业务逻辑保持幂等性。（所有相同消息处理的结果都一样）保证每条消息都有唯一编号且保证消息处理成功与去重表的日志同时出现。（查阅日志判断是否被处理过）

三 事务消息

大事务 = 小事务 + 异步( 分割处理， 减少并行性， 增大吞吐量)

# 深入篇

# ActiveMQ，Kafaka，RocketMQ 比较

# ActiveMQ

# Kafaka

# RocketMQ