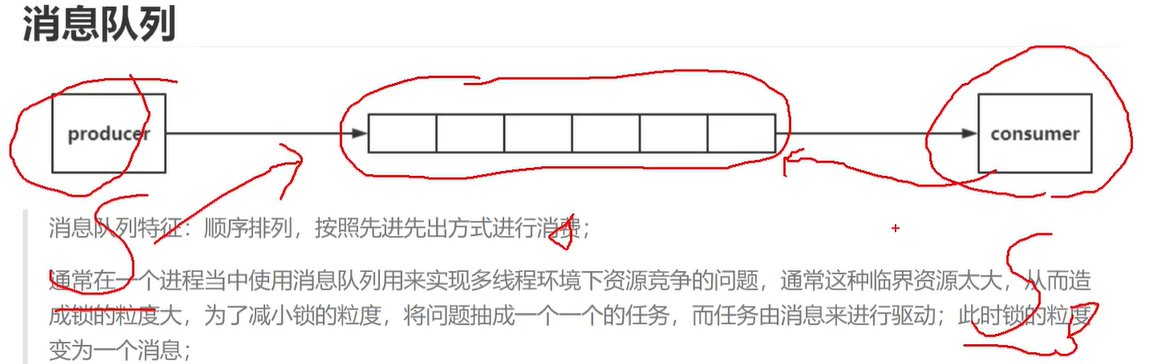
分布式消息队列kafka



# 消息队列：关键在于异步和解耦



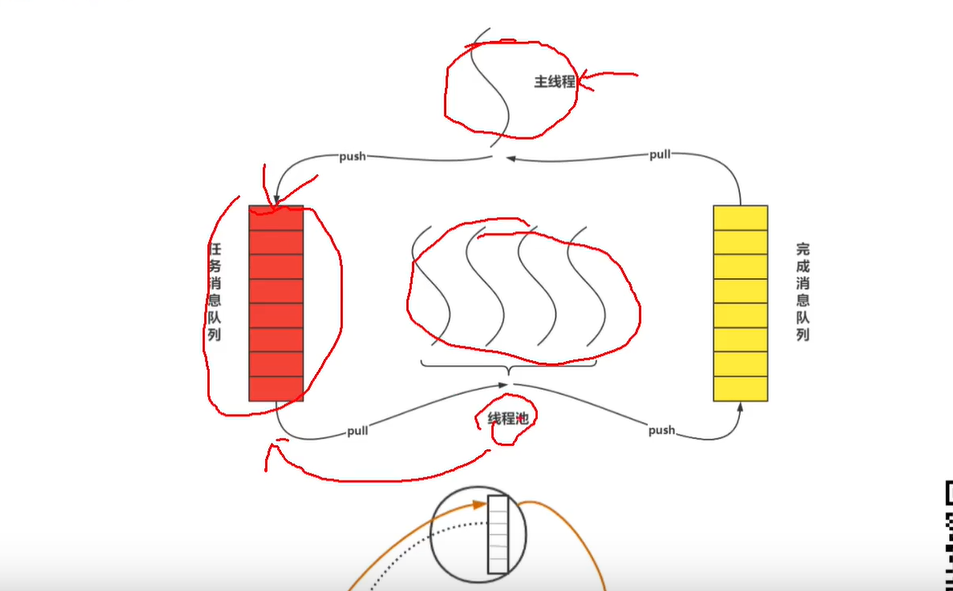
这里可以把producer和consumer看成线程

## Engiens

消息队列使任务可以异步执行，出现问题也不会阻塞主线程，只会阻塞线程池中的线程。

主线程往任务队列中push任务，线程池从任务队列中取消息，执行对应任务（消费消息）。

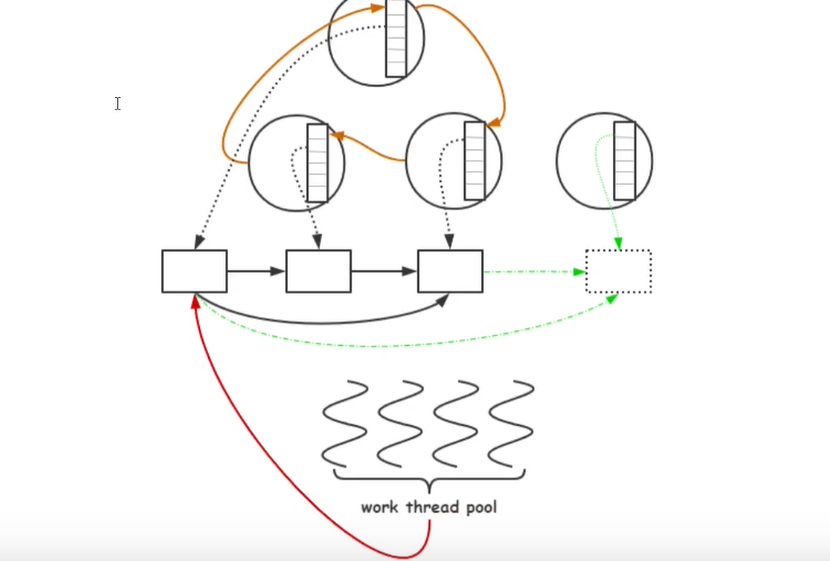
线程池把任务执行完后，会把消息push到一个完成消息队列。主线程再从这个完成消息队列中去除消息，进行消费



消息队列还起到了解耦的作用：解耦到其他线程去处理。处理完毕后，主线程再从完成消息队列中取出，相当于迂回的解决问题

## Actors模型：抽象轻量级进程。　Actors是一个轻量级的对象，通过发送消息实现交互。每个Actors在同一时间处理最多一个消息，可以发送消息给其他Actors

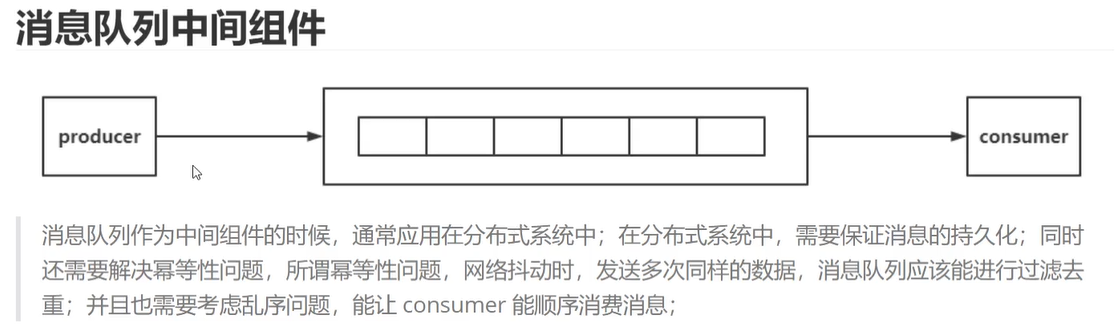
<https://www.jdon.com/actors.html>



由线程池去消费具体actor中的消息，来实现具体actor的运行

# 消息队列中间组件：分布式系统中

Producer 和consumer 都变成了进程



**这种情况，三者属于不同进程->**

这样的话，如果消息队列中间组件挂掉了，所以它需要具备下面的特性：

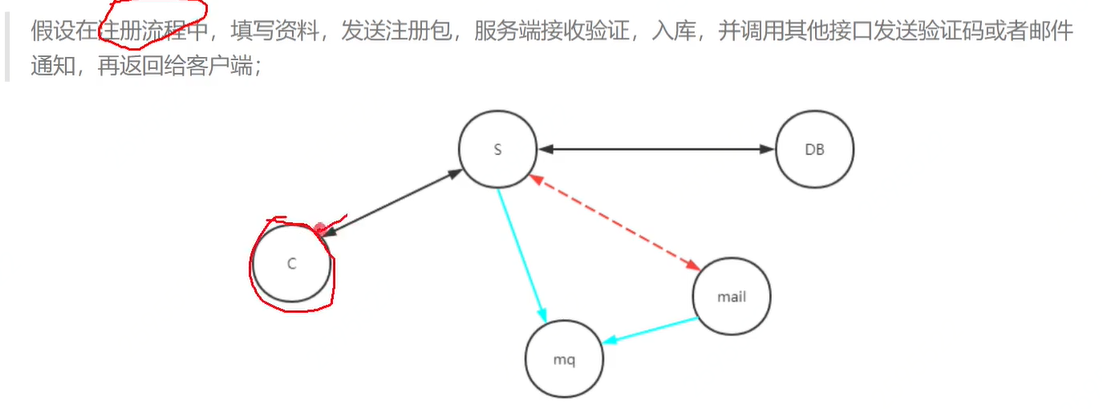
## 它需要具备高可用性：需要一个新的消息队列，让producer可以沿着新的消息队列去生产消息，而consumer也可以通过新的去消费

## 可靠性，持久化：重启的话，可以拉取已经生产的消息

## 可扩展性，高吞吐量：如果一个或多个producer往一个消息队列中发消息，我们希望这一个消息队列可以扩展成多个消息队列，这样可以提升系统吞吐量，以及承载能力

## 典型应用：

### 异步处理：



用户发注册请求给服务器(C->S)，服务器用这个请求操作DB(S->DB,DB->S)，把注册信息填到数据库当中，通过邮件验证(S->mail)，再回发给client(mail->S->C)。

第三方（邮件或短信，回发给服务器，再给client）

1. 流程必要性：不必要（完全可以C to S to DB, DB to S to C就可以了）
2. 流程紧迫性：不紧迫。

我们可以异步请求mail，流程通过后，再来异步回到mail->S->C。这时再通知用户填充短信消息，再登录流程

所以这里我们给他加一个消息队列MQ，S到mq（提起短信验证的消息），mail从mq中取出验证消息，再生成验证码，再到S到C。

### 系统解耦

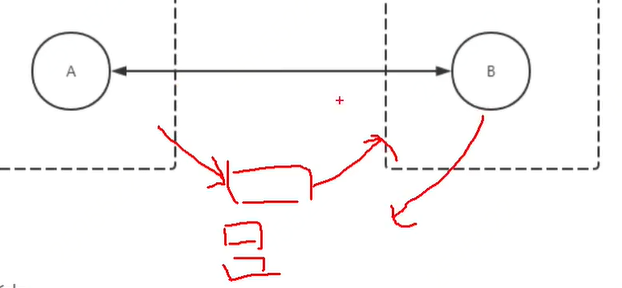


原本的处理方式如下三种：

1. Socket（协议，制定交互流程）
2. restful api（**公开数据**，http请求，A请求B的数据，数据安全？）
3. rpc（A远程调用B的方法：耦合性增加，需要知道具体函数名和参数）

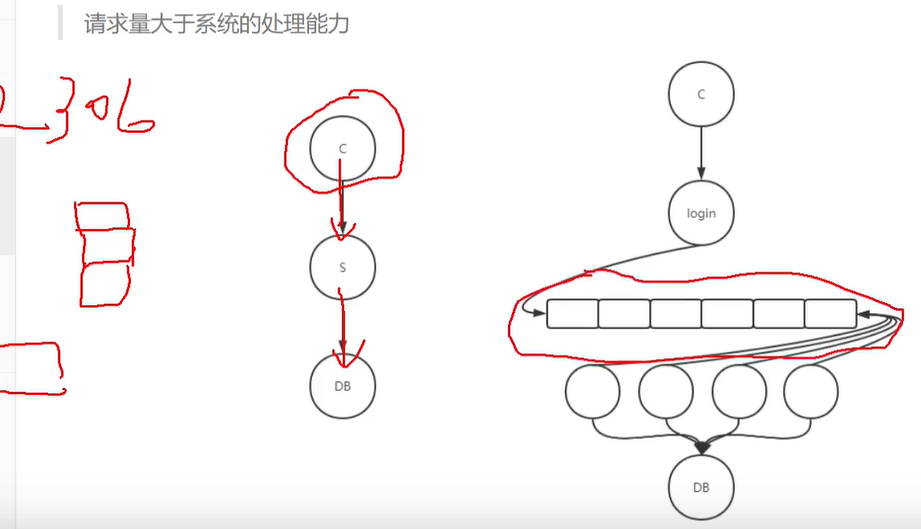
例子：北京团队和深圳团对，内部交互数据：先否认restful api。再否认rpc，因为沟通成本太大。

我们采用消息队列的方式：不关注具体实现，只关注需要的数据



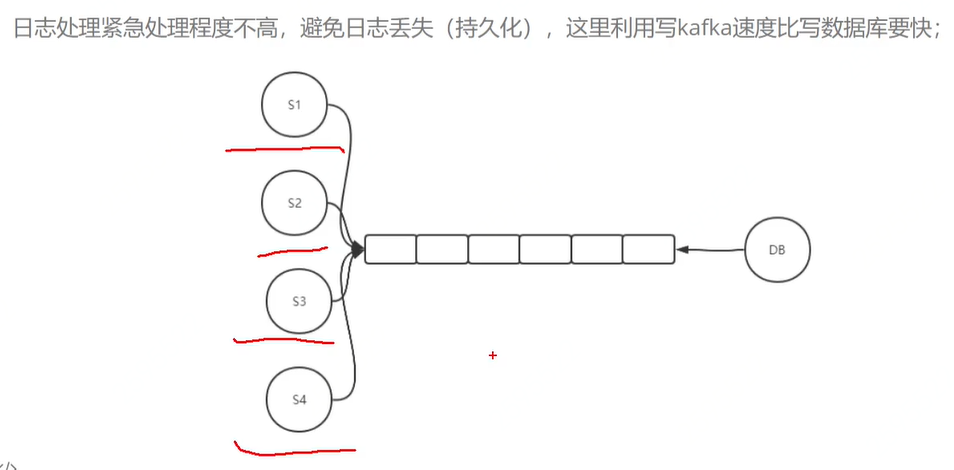
### 流量削峰

比如12306买火车票或者滴滴打车排队



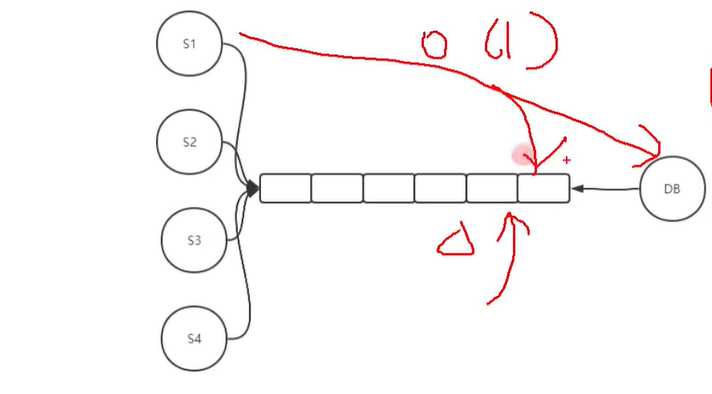
用户登录了，然后登录请求，把用户的登录请求塞到消息队列，下方的消费者从消息队列中取出信息，根据DB的处理能力，进行消费

### 日志处理（持久化，保证数据一定到达DB）



如果写到磁盘数据库的话操作的是B+ tree，时间复杂度Hlog2n，而写消息队列是O（1）.

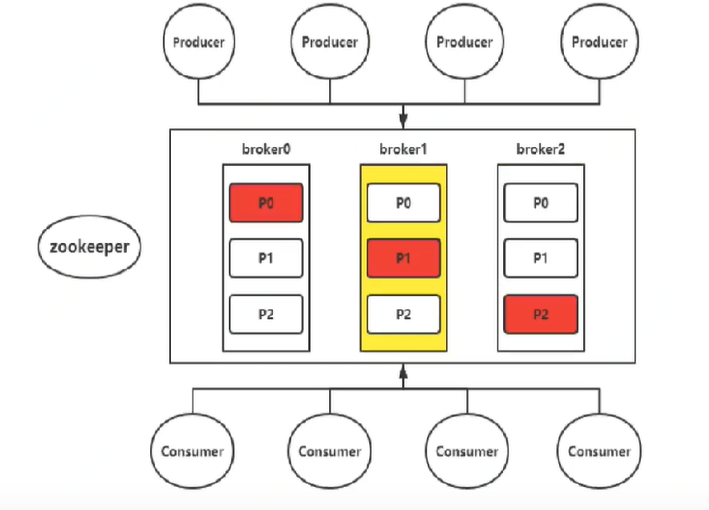
消息队列会记住上次的位置



## 总结：梳理流程，是否必要，是否紧急。非必要不紧急的用消息队列处理->提升系统吞吐量

# Kafka：

## 体系结构



### 主题（逻辑概念）：一类消息 类似于表

### 存储：



#### broker 一个kafka进程（或者节点）->会有 主broker(controller)

#### Partition:分区。一个主题会拆分成多个分区

Leader partition， replication partition