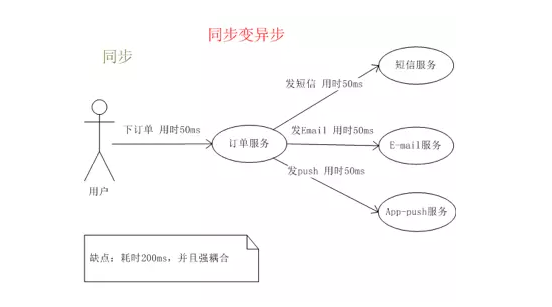


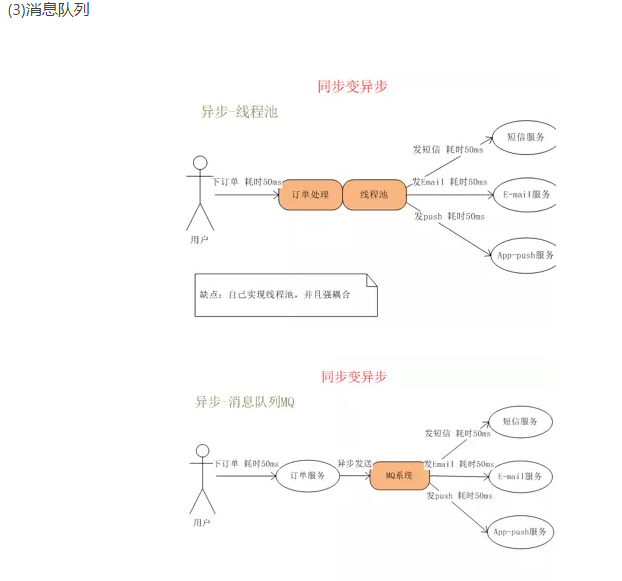
路由：根据那个关键字，发送到那个服务上去。根据路由规则。

生产者发送消息。

作用：

异步





引入消息队列后，把发送邮件,短信不是必须的业务逻辑异步处理

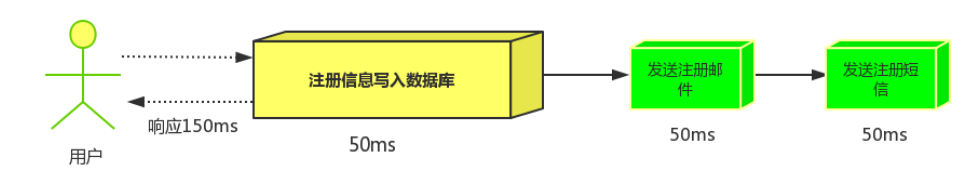
由此可以看出,引入消息队列后，用户的响应时间就等于写入数据库的时间+写入消息队列的时间(可以忽略不计),引入消息队列后处理后,响应时间是串行的3倍,是并行的2倍。

**应用场景**

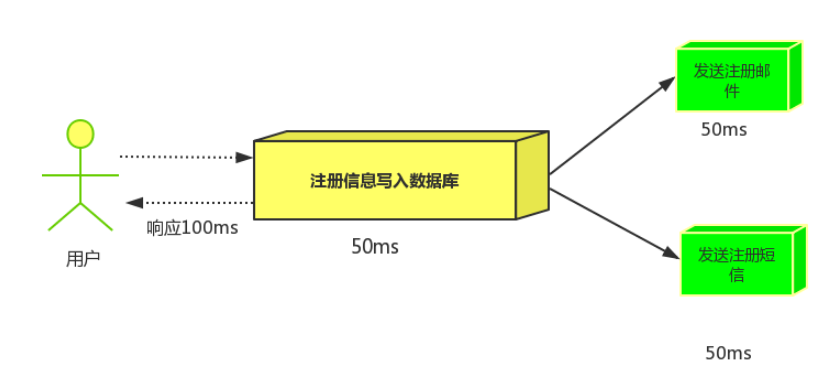
#### 1: 异步处理

场景说明：用户注册后，需要发注册邮件和注册短信,传统的做法有两种1.串行的方式;2.并行的方式

(1)串行方式:将注册信息写入数据库后,发送注册邮件,再发送注册短信,以上三个任务全部完成后才返回给客户端。 这有一个问题是,邮件,短信并不是必须的,它只是一个通知,而这种做法让客户端等待没有必要等待的东西.



(2)并行方式:将注册信息写入数据库后,发送邮件的同时,发送短信,以上三个任务完成后,返回给客户端,并行的方式能提高处理的时间。



假设三个业务节点分别使用50ms,串行方式使用时间150ms,并行使用时间100ms。虽然并性已经提高的处理时间,但是,前面说过,邮件和短信对我正常的使用网站没有任何影响，客户端没有必要等着其发送完成才显示注册成功,英爱是写入数据库后就返回.

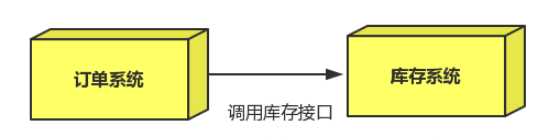
(3)消息队列

引入消息队列后，把发送邮件,短信不是必须的业务逻辑异步处理

由此可以看出,引入消息队列后，用户的响应时间就等于写入数据库的时间+写入消息队列的时间(可以忽略不计),引入消息队列后处理后,响应时间是串行的3倍,是并行的2倍。

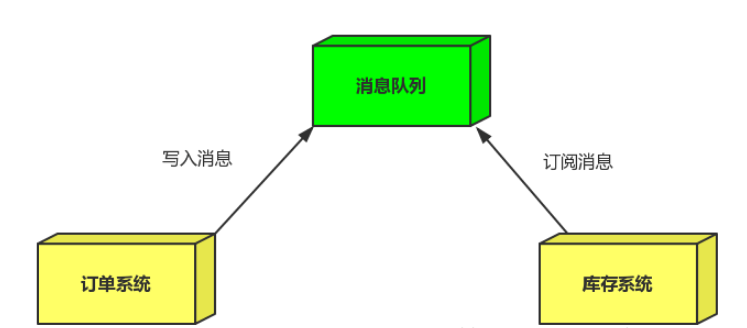
#### 2 应用解耦

场景：双11是购物狂节,用户下单后,订单系统需要通知库存系统,传统的做法就是订单系统调用库存系统的接口.



这种做法有一个缺点:

* 当库存系统出现故障时,订单就会失败。
* 订单系统和库存系统高耦合.
* 引入消息队列



* 订单系统:用户下单后,订单系统完成持久化处理,将消息写入消息队列,返回用户订单下单成功。
* 库存系统:订阅下单的消息,获取下单消息,进行库操作。
* 就算库存系统出现故障,消息队列也能保证消息的可靠投递,不会导致消息丢失.

#### 3. 流量削峰

流量削峰一般在秒杀活动中应用广泛

场景:秒杀活动，一般会因为流量过大，导致应用挂掉,为了解决这个问题，一般在应用前端加入消息队列。

作用:

1.可以控制活动人数，超过此一定阀值的订单直接丢弃(我为什么秒杀一次都没有成功过呢^^)

2.可以缓解短时间的高流量压垮应用(应用程序按自己的最大处理能力获取订单)



1.用户的请求,服务器收到之后,首先写入消息队列,加入消息队列长度超过最大值,则直接抛弃用户请求或跳转到错误页面.

2.秒杀业务根据消息队列中的请求信息，再做后续处理.

消息中间件主要有以下一些主要的应用场景，分别如下：

5.1业务解耦

由于系统业务流程较多，如果一次操作所有的流程，耗时较长。通过消息中间件平台，将消息发送给各个消费者，使得各个消费者仅需要处理自己的业务逻辑，而不需要等待其他消费者的状态，从而实现消息的异步处理，提高系统对消息的处理性能。

5.2削峰填谷

一般来说，消息中间件相当于一个消息缓冲池。当上游的系统吞吐量高于下游第三方消费者的吞吐量的时候，如果直接推送给第三方，会使得第三方由于来不及处理，使得系统崩溃。如果采用消息中间件，上游的峰值数据可以在消息中间件平台得到缓冲，而第三方则按照自身的消费能力去处理消息队列，达到均衡峰谷数据量的作用。

5.3广播通知

系统中一个状态，需要通知多个第三方，借用消息中间件平台，实现高效、高可用的系统广播。

5.4 日志分析

日志分析是大规模分布式系统不可避免的一个环节，用于快速定位系统问题，高效运维，实现高可靠的系统。一般来说，日志分布在若干台机器上，借助于消息中间件，收集多台机器上的日志数据，统一到日志分析平台作处理。

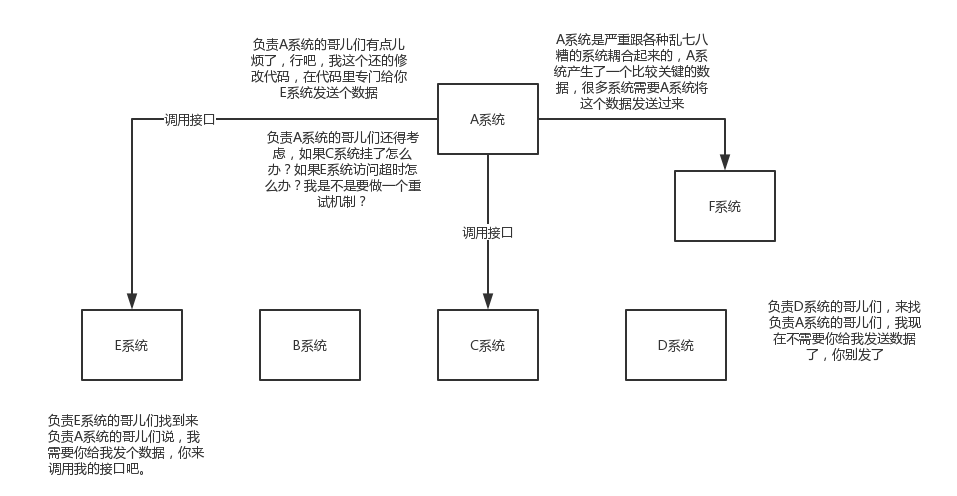
为什么使用消息队列？

其实这个话题也是面试官经常问询的问题，问问你消息队列都有哪些使用场景，然后你项目里具体是什么场景，说说你在这个场景里用消息队列是什么

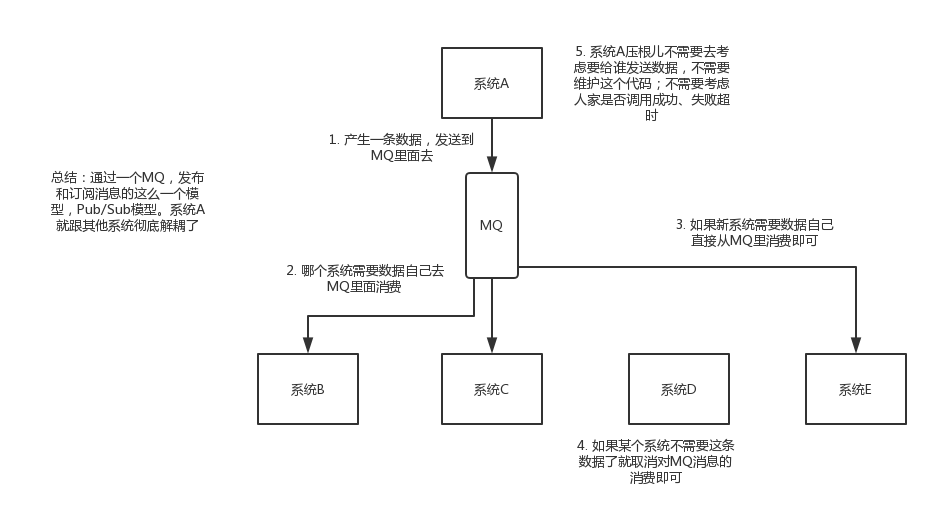
期望的一个回答是说，你们公司有个什么业务场景，这个业务场景有个什么技术挑战，如果不用MQ可能会很麻烦，但是你现在用了MQ之后带给了你很多的好处

现在你可以下想想你如何回答上述问题，想不起来？  好吧我这里先介绍几个常见使用场景，提醒下。。。

解耦：现场画个图来说明一下，

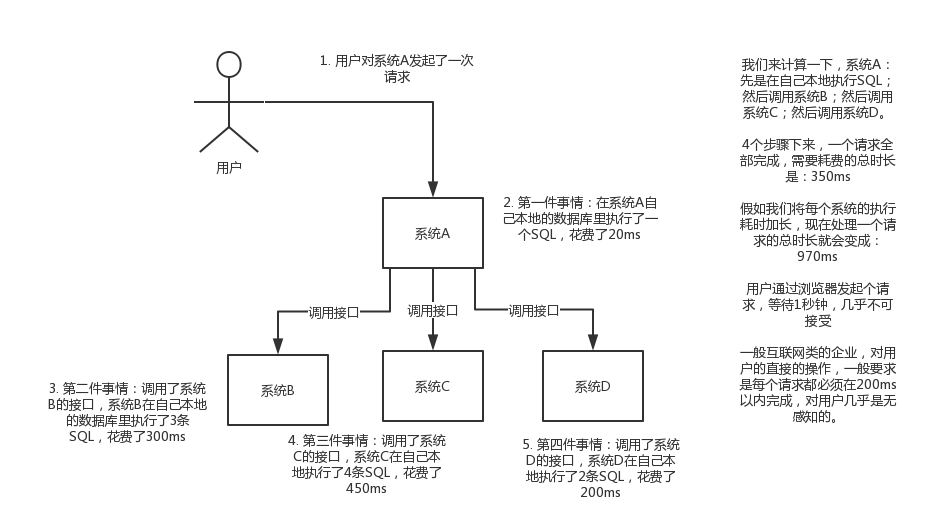


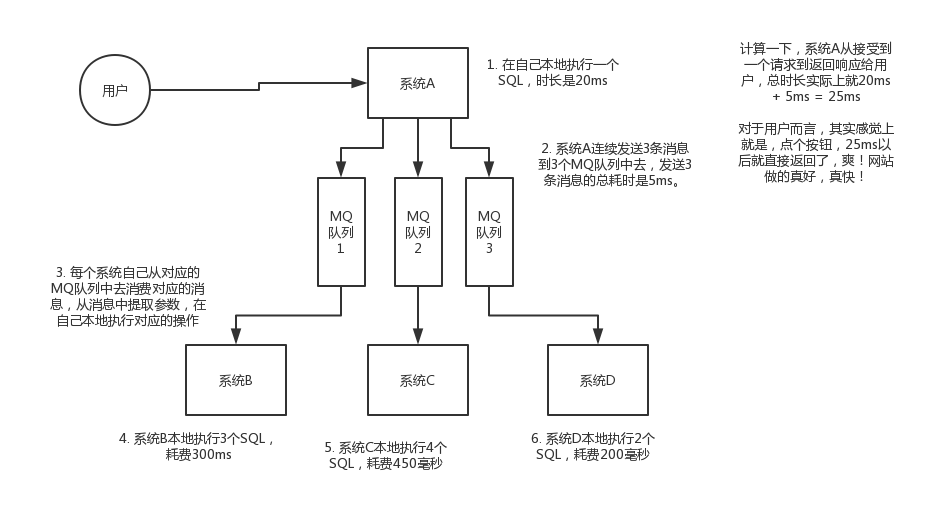
A系统发送个数据到BCD三个系统，接口调用发送，那如果E系统也要这个数据呢？那如果C系统现在不需要了呢？现在A系统又要发送第二种数据了呢？A系统负责人濒临崩溃中。。。再来点更加崩溃的事儿，A系统要时时刻刻考虑BCDE四个系统如果挂了咋办？我要不要重发？我要不要把消息存起来？头发都白了啊。。。



这是你需要去考虑一下你负责的系统中是否有类似的场景，就是一个系统或者一个模块，调用了多个系统或者模块，互相之间的调用很复杂，维护起来很麻烦。但是其实这个调用是不需要直接同步调用接口的，如果用MQ给他异步化解耦，也是可以的，你就需要去考虑在你的项目里（做过微服务项目的同学这里是不是考虑下  消息总线 搭配Rabbitmq 做解耦 用于广播配置文件的更改或者服务间的通讯？），是不是可以运用这个MQ去进行系统的解耦。在简历中体现出来这块东西，用MQ作解耦。

异步：现场画个图来说明一下，

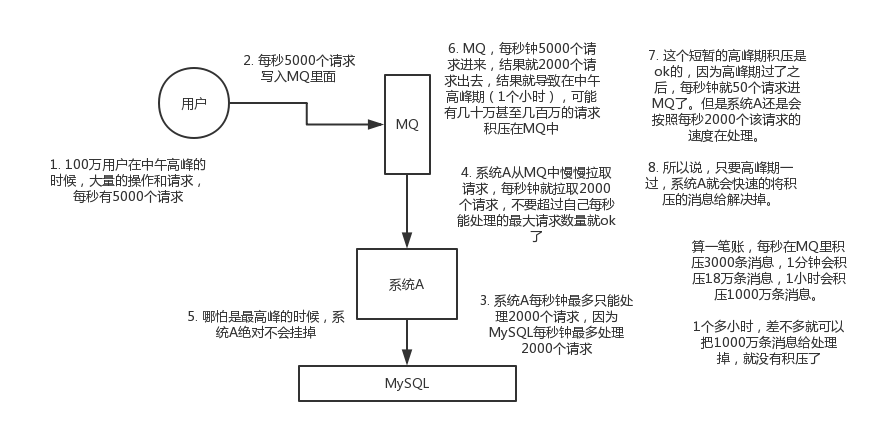


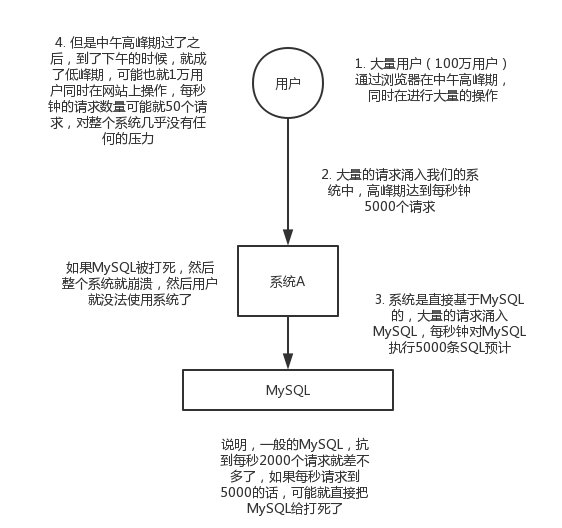


A系统接收一个请求，需要在自己本地写库，还需要在BCD三个系统写库，自己本地写库要3ms，BCD三个系统分别写库要300ms、450ms、200ms。最终请求总延时是3 + 300 + 450 + 200 = 953ms，接近1s，用户感觉搞个什么东西，慢死了慢死了。

更改为 异步后当消息发送到消息队列  自行让对应系统进行消费即可  所以给用户的体验为20 + 5 = 25ms  ,快 好快！

削峰：每天0点到11点，A系统风平浪静，每秒并发请求数量就100个。结果每次一到11点~1点，每秒并发请求数量突然会暴增到1万条。但是系统最大的处理能力就只能是每秒钟处理1000个请求啊。。。尴尬了，系统会死。。。





* 持久订阅

订阅关系建立后（关系保存在消息中间件中），不管消费者（也就是订阅者）是否在线消息都不会消失。

* 非持久订阅

建立一种类似长连接关系式的订阅模式，订阅者为了接收消息必须保持一直连接的状态，如果断开连接则丢失消息。

推和拉的使用场景：

推：

缓存数据同步更新

往应用推数据

发送消息：

Void basicPublish(){

}



拉：

1：把消息中间件当成可靠的消息暂存池。

2：定时进行消息投递，比如用户秒杀访问，进行系统性能压力测试。



核心，看时间，看实时性的要求。

ActiveMQ

AvtiveMQ是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ是一个完全支持JMS1.1和J2EE1.4规范的JMS Provider实现，尽管JMS规范出台已经是很久的事情了，但是JMS是在当今的J2EE应用中间仍然扮演着特殊的地位。

ActiveMQ特性

多种语言和协议编写客户端。语言：

java,C,C++,C#,Ruby,Perl,Python,PHP。应用协议：OpenWire，Stomp REST， WS Notification， XMPP，AMQP

完全支持JMS1.1和J2EE1.4规范（持久化，XA消息，事务）

虚拟主题、组合目的、镜像队列。

RabbitMQ

RabbitMQ是一个开源的AMQP实现，服务器端用Erlang语言编写。用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。

RabbitMQ特性

支持多种客户端，如：Python、Ruby、.NET、Java、JMS、C、PHP、ActionScript等

AMQP的完整实现（vhost、Exchange、Binding、Routing Key等）

事务支持/发布确认

消息持久化

Kafka

Kafka是一种高吞吐量的分布式发布订阅消息系统，是一个分布式的、分区的、可靠的分布式日志存储服务。它通过一种独一无二的设计提供了一个消息系统的功能。

Kafka特性

通过O(1)的磁盘数据结构提供消息的持久化，这种结构对于即使数以TB消息存储也能保持长时间的稳定性能。

高吞吐量：即使是非常普通的硬件Kafka也可以支持每秒数百万的消息。

Partition、Consumer Group

一般的业务系统要引入MQ，最早大家都用ActiveMQ，但是现在确实大家用的不多了，没经过大规模吞吐量场景的验证，社区也不是很活跃，所以大家还是算了吧，我个人不推荐用这个了；

后来大家开始用RabbitMQ，但是确实erlang语言阻止了大量的java工程师去深入研究和掌控他，对公司而言，几乎处于不可控的状态，但是确实人是开源的，比较稳定的支持，活跃度也高；

不过现在确实越来越多的公司，会去用RocketMQ，确实很不错，但是我提醒一下自己想好社区万一突然黄掉的风险，对自己公司技术实力有绝对自信的，我推荐用RocketMQ，否则回去老老实实用RabbitMQ吧，人是活跃开源社区，绝对不会黄

所以中小型公司，技术实力较为一般，技术挑战不是特别高，用RabbitMQ是不错的选择（我们项目也正在使用这个^\_^）；大型公司，基础架构研发实力较强，用RocketMQ是很好的选择

如果是大数据领域的实时计算、日志采集等场景，用Kafka是业内标准的，绝对没问题，社区活跃度很高，绝对不会黄，何况几乎是全世界这个领域的事实性规范