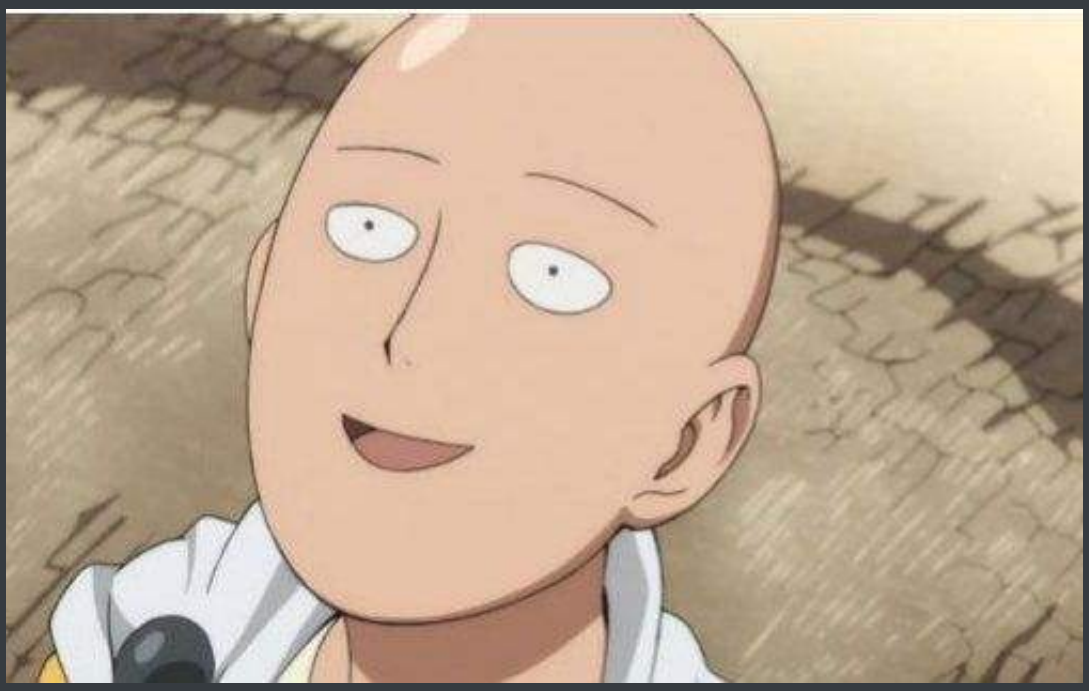
点赞再看，养成习惯，微信搜索【**三太子敖丙**】关注这个互联网苟且偷生的工具人。

面试开始

> 一个风度翩翩，穿着格子衬衣的中年男子，拿着一个满是划痕的mac向你走来，看着铮亮的头，心想着肯定是尼玛顶级架构师吧！但是我们看过暖男敖丙的系列，腹有诗书气自华，虚都不虚。



小伙子之前问了你这么多Redis的知识，你不仅对答如流，你还能把各自场景的解决方案，优缺点说得这么流畅，说你是不是看过敖丙写的《我们一起进大厂》系列呀？

惊！！！老师你怎么知道的，我看了他的系列根本停不下来啊。

呵呵，Redis没难住你，但是我问个新的技术栈我还怕难不住你？我问问你你项目中用过消息队列么？你为啥用消息队列？

噗此，这也叫问题？别人用了我能不用么？别人用了我就用了呗，我就是为了用而用。

你心里嘀咕就好了，千万别说出来哈，说出来了没拿到Offer别到时候就在那说，敖丙那个渣男教我说的！

面试官你好：我们公司本身的业务体量很小，所以直接**单机一把梭**啥都能搞定了，但是后面业务体量不断扩大，采用**微服务的设计思想**，**分布式的部署方式**，所以拆分了很多的服务，随着体量的增加以及业务场景越来越复杂了，很多场景单机的技术栈和中间件以及不够用了，而且对系统的友好性也下降了，最后做了很多技术选型的工作，我们决定引入**消息队列中间件**。

哦？你说到业务场景越来越复杂，你那说一下你都在什么场景用到了消息队列？

嗯，我从三个方面去说一下我使用的场景吧。

Tip：这三个场景也是消息队列的经典场景，大家基本上要烂熟于心那种，就是一说到消息队列你脑子就要想到**异步**、**削峰**、**解耦**，条件反射那种。

异步：

我们之前的场景里面有很多步骤都是在一个流程里面需要做完的，就比如说我的下单系统吧，本来我们业务简单，下单了付了钱就好了，流程就走完了。

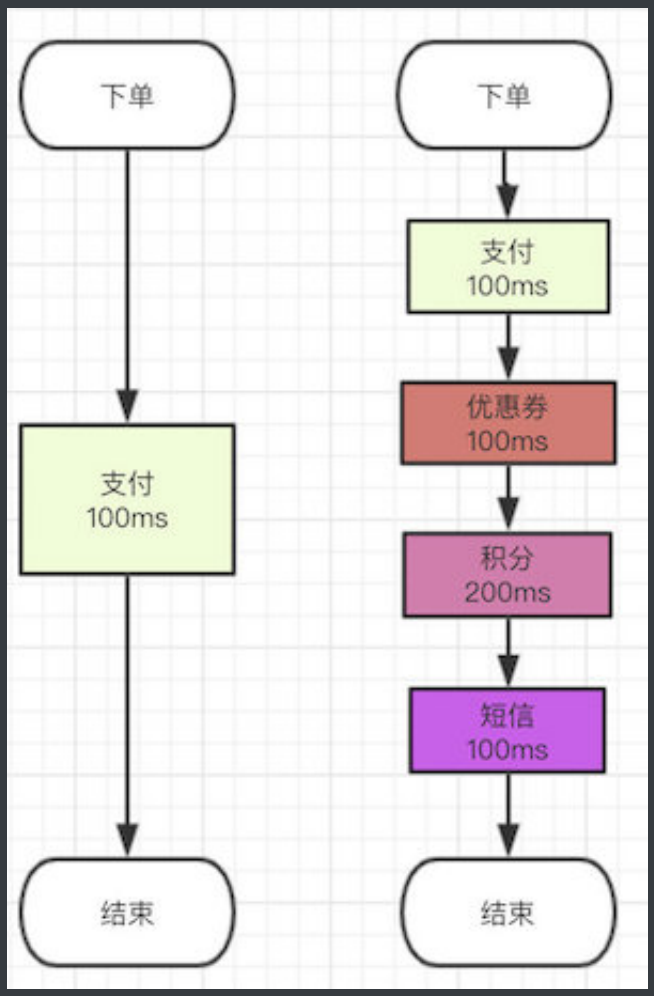
但是后面来了个产品经理，搞了个**优惠券系统**，OK问题不大，流程里面多100ms去扣减优惠券。

后来产品经理灵光一闪说我们可以搞个**积分系统**啊，也行吧，流程里面多了200ms去增减积分。

再后来后来隔壁的产品老王说：下单成功后我们要给用户发短信，也将就吧，100ms去发个短信。

再后来。。。（敖丙你有完没完！！！）

反正就流程有点像这样↓



你们可以看到这才加了三个，我可以**斩钉截铁**的告诉你真正的下单流程涉及的系统绝对在10个以上（主流电商），越大的越多。

这个链路这样下去，**时间长得一批**，用户发现我买个东西你特么要花几十秒，垃圾电商我不在你这里买了，不过要是都像**并夕夕**这么便宜，**真香**！

但是我们公司没有夕夕的那个经济实力啊，那只能优化系统了。

Tip：我之前在的电商老东家要求所有接口的**Rt（ResponseTime响应时间）**在200ms内，超出的全部优化，我现在所负责的系统QPS也是**9W+**就是抖动一下**网络集群都可能炸锅**那种，**RT**基本上都要求在50ms以内。

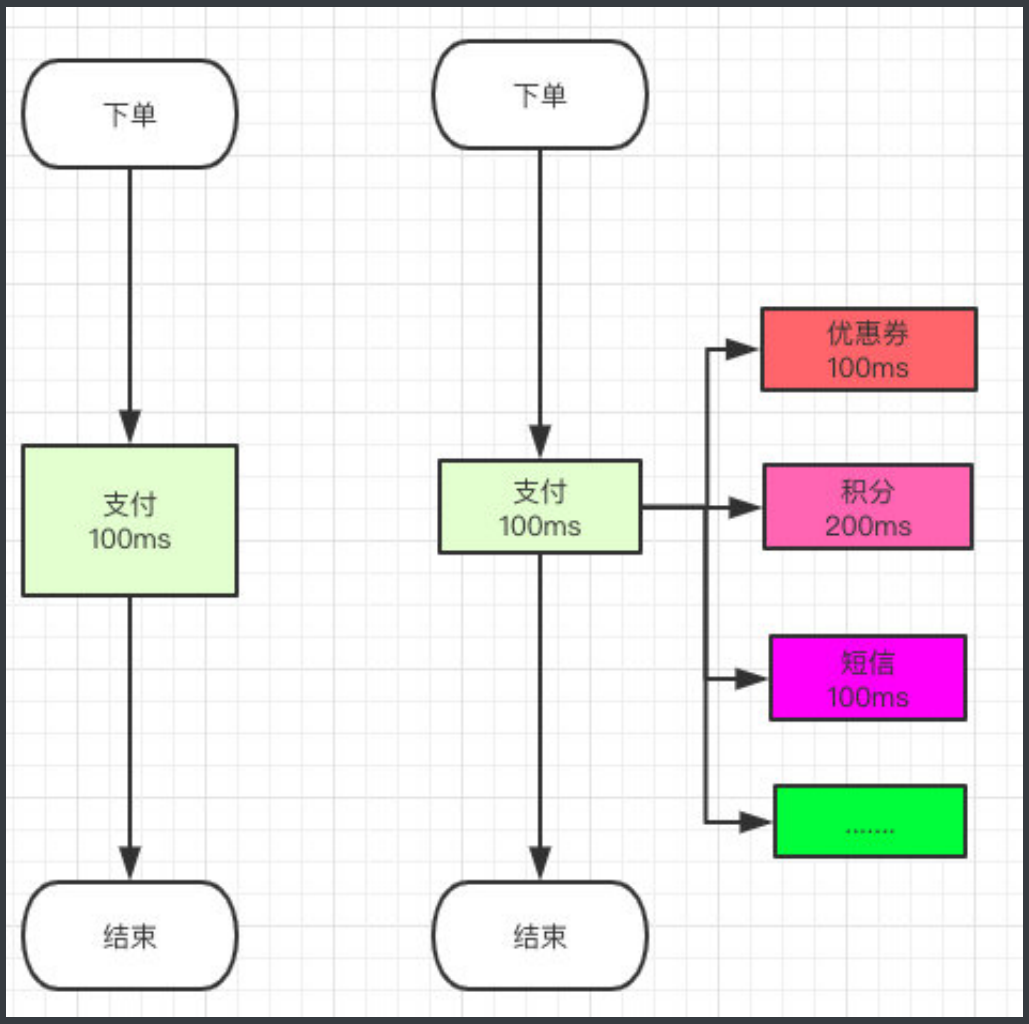
大家感受一下这个QPS。

嗯不错，链路长了就慢了，那你怎么解决的?

那链路长了就慢了，但是我们发现上面的流程其实可以**同时做**的呀，你支付成功后，我去校验优惠券的同时我可以去增减积分啊，还可以同时发个短信啊。

那正常的流程我们是没办法实现的呀，怎么办，**异步**。

你对比一下是不是发现，这样子最多只用100毫秒用户知道下单成功了，至于短信你迟几秒发给他他根本不在意是吧。



小伙子我打断你一下，你说了异步，那我用线程，线程池去做不是一样的么？

诶呀，面试官你**不要急嘛**，我后面还会说到的，骚等。

解耦：

既然面试官这么问了，我就说一下为啥我们不能用线程去做，因为用线程去做，你是不是要写代码？

你一个订单流程，你扣积分，扣优惠券，发短信，扣库存。。。等等这么多业务要调用这么多的接口，**每次加一个你要调用一个接口然后还要重新发布系统**，写一次两次还好，写多了你就说：老子不干了！

而且真的全部都写在一起的话，不单单是耦合这一个问题，你出问题排查也麻烦，流程里面随便一个地方出问题搞不好会影响到其他的点，小伙伴说我每个流程都**try catch**不就行了，相信我别这么做，这样的代码就像个**定时炸弹**💣，你不知道什么时候爆炸，平时不炸偏偏在你做活动的时候炸，你就领个**P0故障**收拾书包**提前回家过年**吧。

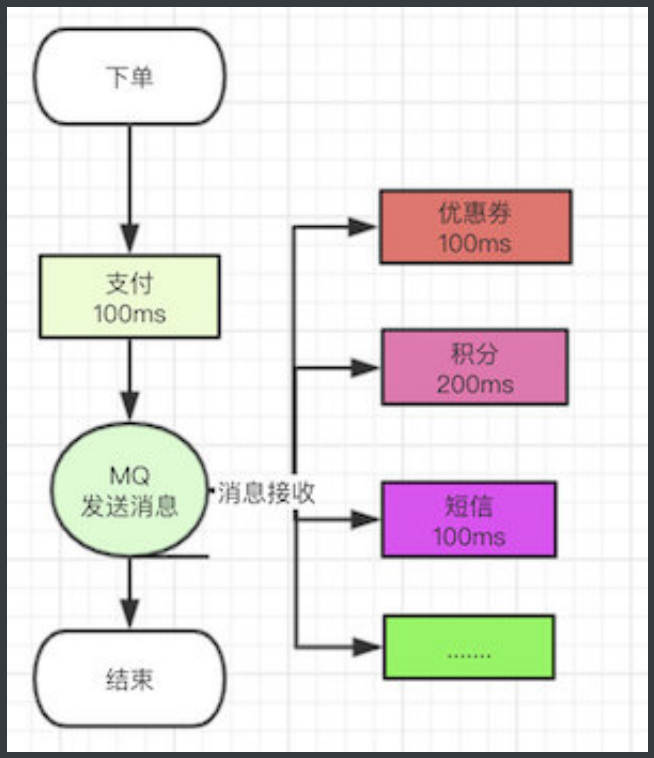
Tip：P0—PN 是互联网大厂经常用来判定事故等级的机制，P0是最高等级了。

但是你用了**消息队列**，耦合这个问题就迎刃而解了呀。

哦，帅丙怎么说？

且听我娓娓道来：

你下单了，你就把你**支付成功的消息告诉别的系统**，他们收到了去处理就好了，你只用走完自己的流程，把自己的消息发出去，那后面要接入什么系统简单，直接订阅你发送的支付成功消息，你支付成功了我**监听就好了**。



那你的流程走完了，你不用管别人是否成功么？比如你下单了积分没加，优惠券没扣怎么办？

问题是个好问题，但是没必要考虑，业务系统本身就是自己的开发人员维护的，你积分扣失败关我下单的什么事情？你管好自己下单系统的就好了。

Tip：话是这么说，但是这其实是用了消息队列的一个缺点，涉及到**分布式事务**的知识点，我下面会提到。

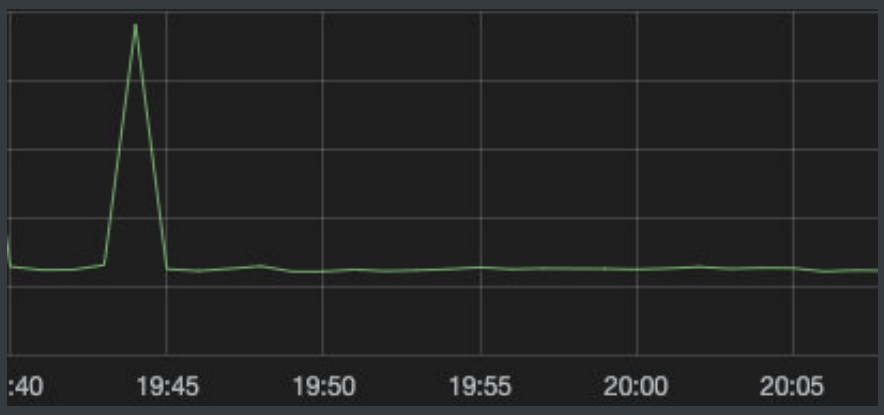
削峰：

就拿我上一期写的秒杀来说（暗示新同学看我上一期），你平时流量很低，但是你要做秒杀活动00：00的时候流量疯狂怼进来，你的服务器，**Redis**，**MySQL**各自的承受能力都不一样，你直接**全部流量照单全收**肯定有问题啊，直接就打挂了。

那怎么办？

简单，把请求放到队列里面，然后至于每秒消费多少请求，就看自己的**服务器处理能力**，你能处理5000QPS你就消费这么多，可能会比正常的慢一点，但是**不至于打挂服务器**，等流量高峰下去了，你的服务也就没压力了。

你看阿里双十一12：00的时候这么多流量瞬间涌进去，他有时候是不是会慢一点，但是人家没挂啊，或者降级给你个友好的提示页面，等高峰过去了又是一条好汉了。



听你说了辣么多，怎么都是好处，那我问你使用了消息队列有啥问题么？

诶，看过前面我写的文章的**人才**都知道，我经常说的就是，**技术是把双刃剑**！

没错面试官，我使用他是因为他带给我们很多好处，但是使用之后问题也是**接踵而至**。

同样的暖男我呀，也从三个点介绍他主要的缺点：

系统复杂性

本来蛮简单的一个系统，我代码随便写都没事，现在你凭空接入一个中间件在那，我是不是要考虑去维护他，而且使用的过程中是不是要考虑各种问题，比如消息**重复消费**、**消息丢失**、**消息的顺序消费**等等，反正用了之后就是贼烦。

我插一句嘴，上面的问题（重复消费、消息丢失、顺序消费）你能分别介绍一下，并且说一下分别是怎么解决的么？

不要！我都说了敖丙下一章写啥？

其实不是**暖男**我不想在这里写，这三个问题我想了下，统统都是**MQ**的**重点问题**，单独拿一个出来就是一篇文章了，**篇幅实在太长了**，我会在下一章挨个介绍一遍的。

数据一致性

这个其实是分布式服务本身就存在的一个问题，**不仅仅是消息队列的问题**，但是放在这里说是因为用了消息队列这个问题会暴露得比较严重一点。

就像我开头说的，你下单的服务自己保证自己的逻辑成功处理了，你成功发了消息，但是优惠券系统，积分系统等等这么多系统，**他们成功还是失败你就不管了？**

我说了保证自己的业务数据对的就好了，其实还是比较不负责任的一种说法，这样就**像个渣男，没有格局，这样呀你的路会越走越窄的**。

所有的服务都成功才能算这一次下单是成功的，那怎么才能保证数据一致性呢?

分布式事务：把下单，优惠券，积分。。。都放在一个事务里面一样，要成功一起成功，要失败一起失败。

Tip:**分布式事务**在互联网公司里面实在常见，我也不在这里大篇幅介绍了，后面都会专门说的。

可用性

你搞个系统本身没啥问题，你现在突然接入一个中间件在那放着，万一挂了怎么办？我下个单**MQ挂了**，优惠券不扣了，积分不减了，这不是杀一个程序员能搞定的吧，感觉得杀一片。

至于怎么保证高可用，还是那句话也不在这里展开讨论了，我后面一样会写，像写**Redis**那样写出来的。

放心敖丙我不是渣男来的，我肯定会对你们负责的。点赞!

看不出来啊，你有点东西呀，那我问一下你，你们是怎么做技术选型的?

目前在市面上比较主流的消息队列中间件主要有，**Kafka**、**ActiveMQ**、**RabbitMQ**、**RocketMQ** 等这几种。

不过敖丙我想说的是，**ActiveMQ**和**RabbitMQ**这两着因为吞吐量还有**GitHub**的社区活跃度的原因，在各大互联网公司都已经基本上绝迹了，业务体量一般的公司会是有在用的，但是越来越多的公司更青睐**RocketMQ**这样的消息中间件了。

Kafka和**RocketMQ**一直在各自擅长的领域发光发亮，不过写这篇文章的时候我问了蚂蚁金服，字节跳动和美团的朋友，好像大家用的都有点不一样，应该都是各自的中间件，可能做过修改，也可能是**自研**的，大多**没有开源**。

就像我们公司就是是基于**Kafka**和**RocketMQ**两者的优点自研的消息队列中间件，吞吐量、可靠性、时效性等都很可观。

我们回归正题，我这里用网上找的对比图让大家看看差距到底在哪里：

特性	ActiveMQ	RabbitMQ	RocketMQ	Kafka
单机吞吐量	万级，比 RocketMQ、Kafka 低一个数量级	同 ActiveMQ	10 万级，支撑高吞吐	10 万级，高吞吐，一般配合大数据类的系统来进行实时数据计算、日志采集等场景
topic 数量对吞吐量的影响			topic 可以达到几百/几千的级别，吞吐量会有较小幅度的下降，这是 RocketMQ 的一大优势，在同等机器下，可以支撑大量的 topic	topic 从几十到几百个时候，吞吐量会大幅度下降，在同等机器下，Kafka 尽量保证 topic 数量不要过多，如果要支撑大规模的 topic，需要增加更多的机器资源
时效性	ms 级	微秒级，这是 RabbitMQ 的一大特点，延迟最低	ms 级	延迟在 ms 级以内
可用性	高，基于主从架构实现高可用	同 ActiveMQ	非常高，分布式架构	非常高，分布式，一个数据多个副本，少数机器宕机，不会丢失数据，不会导致不可用
消息可靠性	有较低的概率丢失数据	基本不丢	经过参数优化配置，可以做到 0 丢失	同 RocketMQ
功能支持	MQ 领域的功能极其完备	基于 erlang 开发，并发能力很强，性能极好，延时很低	MQ 功能较为完善，还是分布式的，扩展性好	功能较为简单，主要支持简单的 MQ 功能，在大数据领域的实时计算以及日志采集被大规模使用
社区活跃度	低	中	高	高

大家其实一下子就能看到差距了，就拿**吞吐量**来说，早期比较活跃的**ActiveMQ** 和**RabbitMQ**基本上不是后两者的对手了，在现在这样大数据的年代**吞吐量是真的很重要**。

比如现在突然爆发了一个超级热点新闻，你的APP注册用户高达亿数，你要想办法第一时间把突发全部推送到每个人手上，你没有**大吞吐量的消息队列**中间件用啥去推?

再说这些用户大量涌进来看了你的新闻产生了一系列的附带流量，你怎么应对这些数据，**很多场景离开消息队列基本上难以为继**。

就**部署方式**而言前两者也是大不如后面两个**天然分布式架构的哥哥**，都是高可用的分布式架构，而且数据多个副本的数据也能做到0丢失。

我们再聊一下**RabbitMQ**这个中间件其实还行，但是这玩意开发语言居然是**erlang**，我敢说绝大部分工程师肯定不会为了一个中间件去刻意学习一门语言的，开发维护成本你想都想不到，出个问题查都查半天。

至于**RocketMQ**（阿里开源的），git活跃度还可以。基本上你push了自己的bug确认了有问题都有阿里大佬跟你试试解答并修复的，我个人推荐的也是这个，他的架构设计部分跟同样是阿里开源的一个**RPC**框架是真的很像（**Dubbo**）可能是因为师出同门的原因吧。

Tip：**Dubbo**等我写到**RPC**我会详细介绍的。

Kafka我放到最后说，你们也应该知道了，压轴的这是个大哥，大数据领域，公司的日志采集，实时计算等场景，都离不开他的身影，他基本上算得上是**世界范围级别的消息队列标杆**了。

以上这些都只是一些我自己的**个人意见**，真正的选型还是要去**深入研究**的，不然那你公司一天UV就1000你告诉我你要去用**Kafka**我只能说你吃饱撑的。

记住，没有最好的技术，只有最适合的技术，不要为了用而用。

面试结束

嗯，小伙子不错不错，分析得很到位，那你记得下期来说一下消息队列的高可用，重复消费、消息丢失、消息顺序、分布式事务等问题?

嗯嗯好的面试官，不过不确定能不能一口气说完，毕竟敖丙还没开始写，而且读者还有可能白嫖，动力不一定够。

嗯嗯这倒是个问题，不过啊在看的都是人才肯定会给你点赞👍的!

我也这么认为。

总结

消息队列的基础知识我就先介绍这么多，消息队列在面试里面基本上也是跟我前面写的**Redis**一样必问的。

面试的思路还是一样，**要知其然，也要知其所以然，就是要知道为啥用，用了有啥好处，有啥坑。**

面试官不喜欢**只知道用**的，你只会用那哪天线上出问题怎么办？你难道在旁边拜佛？

我是敖丙，一个在互联网苟且偷生的工具人。

创作不易，不想被白嫖，各位的**「三连」**就是丙丙创作的最大动力，我们下次见！