因为java面试题是一个较为大的体系，所以我们会拆分为几季来讲解，预计是会推出3季，本次是第一季，第一季课程包含的内容如下：

# 1、分布式系统

## 1.1 为什么要进行系统拆分？

（1）为什么要进行系统拆分？如何进行系统拆分？拆分后不用dubbo可以吗？dubbo和thrift有什么区别呢？

## 1.2 分布式服务框架

（1）说一下的dubbo的工作原理？注册中心挂了可以继续通信吗？

（2）dubbo支持哪些序列化协议？说一下hessian的数据结构？PB知道吗？为什么PB的效率是最高的？

（3）dubbo负载均衡策略和高可用策略都有哪些？动态代理策略呢？

（4）dubbo的spi思想是什么？

（5）如何基于dubbo进行服务治理、服务降级、失败重试以及超时重试？

（6）分布式服务接口的幂等性如何设计（比如不能重复扣款）？

（7）分布式服务接口请求的顺序性如何保证？

（8）如何自己设计一个类似dubbo的rpc框架？

## 1.3 分布式锁

（1）使用redis如何设计分布式锁？使用zk来设计分布式锁可以吗？这两种分布式锁的实现方式哪种效率比较高？

## 1.4 分布式事务

（1）分布式事务了解吗？你们如何解决分布式事务问题的？TCC如果出现网络连不通怎么办？XA的一致性如何保证？

## 1.5 分布式会话

（1）集群部署时的分布式session如何实现？

# 2、高并发架构

## 2.1 如何设计一个高并发系统？

## 2.2 消息队列

（1）为什么使用消息队列啊？消息队列有什么优点和缺点啊？kafka、activemq、rabbitmq、rocketmq都有什么优点和缺点啊？

（2）如何保证消息队列的高可用啊？

（3）如何保证消息不被重复消费啊（如何进行消息队列的幂等性问题）？

（4）如何保证消息的可靠性传输（如何处理消息丢失的问题）？

（5）如何保证消息的顺序性？

（6）如何解决消息队列的延时以及过期失效问题？消息队列满了以后该怎么处理？有几百万消息持续积压几小时，说说怎么解决？

（7）如果让你写一个消息队列，该如何进行架构设计啊？说一下你的思路

## 2.3 搜索引擎

（1）es的分布式架构原理能说一下么（es是如何实现分布式的啊）？

（2）es写入数据的工作原理是什么啊？es查询数据的工作原理是什么啊？底层的lucene介绍一下呗？倒排索引了解吗？

（3）es在数据量很大的情况下（数十亿级别）如何提高查询效率啊？

（4）es生产集群的部署架构是什么？每个索引的数据量大概有多少？每个索引大概有多少个分片？

## 2.4 缓存

（1）在项目中缓存是如何使用的？缓存如果使用不当会造成什么后果？

（2）redis和memcached有什么区别？redis的线程模型是什么？为什么单线程的redis比多线程的memcached效率要高得多？

（3）redis都有哪些数据类型？分别在哪些场景下使用比较合适？

（5）redis的过期策略都有哪些？手写一下LRU代码实现？

（6）如何保证Redis高并发、高可用、持久化？redis的主从复制原理能介绍一下么？redis的哨兵原理能介绍一下么？

（7）redis的持久化有哪几种方式？不同的持久化机制都有什么优缺点？持久化机制具体底层是如何实现的？

（8）redis集群模式的工作原理能说一下么？在集群模式下，redis的key是如何寻址的？分布式寻址都有哪些算法？了解一致性hash算法吗？如何动态增加和删除一个节点？

（9）了解什么是redis的雪崩和穿透？redis崩溃之后会怎么样？系统该如何应对这种情况？如何处理redis的穿透？

（10）如何保证缓存与数据库的双写一致性？

（11）redis的并发竞争问题是什么？如何解决这个问题？了解Redis事务的CAS方案吗？

（12）生产环境中的redis是怎么部署的？

## 2.5 分库分表

（2）为什么要分库分表（设计高并发系统的时候，数据库层面该如何设计）？用过哪些分库分表中间件？不同的分库分表中间件都有什么优点和缺点？你们具体是如何对数据库如何进行垂直拆分或水平拆分的？

（3）现在有一个未分库分表的系统，未来要分库分表，如何设计才可以让系统从未分库分表动态切换到分库分表上？

（4）如何设计可以动态扩容缩容的分库分表方案？

（5）分库分表之后，id主键如何处理？

## 2.6 读写分离

（1）如何实现mysql的读写分离？MySQL主从复制原理的是啥？如何解决mysql主从同步的延时问题？

# 3、高可用架构

## 3.1 如何设计一个高可用系统？

## 3.2 限流

1. 如何限流？在工作中是怎么做的？说一下具体的实现？

## 3.3 熔断

（1）如何进行熔断？熔断框架都有哪些？具体实现原理知道吗？

## 3.4 降级

（1）如何进行降级？

Day3

这个课程针对的人群

（1）好多同学出去面试，尤其是面互联网行业的Java职位，经常会问一些消息队列、缓存、分库分表、分布式系统之类的问题，但是可能不少同学之前没有搞过这些，或者略知一二，但是被面试官几个连环炮就给打蒙了

（2）或者是不少同学很长时间不面试了，最近要找工作，需要短期突击快速准备一下常见的面试题，至少先给自己热个身，不至于面试的时候一问三不知

（3）或者是不面试，但是对这套技术体系不怎么会的同学，可以用这个课程算是扫盲，快速夯实一下互联网公司技术栈的一些基础

（4）如果你看一眼，觉得自己基本都会，那么可以绕行，这个课程应该不适合你，可以加我助理QQ，未来我会出一些高级的免费课程

这个课程的定位：

（1）两周让你快速突击完整的互联网java工程师面试中要掌握的技术体系

（2）针对的是互联网行业的普通java岗位常问的一些面试题，不是针对一些高级、资深、专家等高阶岗位的面试，因为你要是面那些高阶岗位，靠两三周突击是不可能的

（3）所以这里很多问题以及问题的解答，基本可能都是属于科普性的，有一些技术实力比较好的同学，不要看到后面来骂我，说讲的啥，这么简单，都是些入门和基础，确实是，我也觉得这些问题都很简单，但是确实大量的java同学面试被问到可能都答不出来，所以这个课程是有针对人群的。觉得这个课程讲解内容过于简单的同学，请绕道，这个课程就别看了，但是欢迎关注我们的公众号，后面我们可能还会推出更加高级的一些免费课程。

（4）讲解方式走短平快路线，大白话+手工画图，语言精简不拖拉，不说废话，每个问题都直击问题要点即可。课程出发点是帮助那些想要在2周之内完成面试准备，出去找工作的那帮同学

（5）对每个面试题都给你先剖析一下面试官问这个面试题背后的愿意是什么？他为什么要问你这个问题？

（6）能解答每个问题背后的基本原理，至少被问到这些问题别一无所知，自己能回答出来一些原理以及技术方案，代表至少你有这个知识积累

（7）能在面试现场动手画图讲解

（8）能对常见的要求写代码的环节现场手写代码

（9）试想，面试考察范围那么大，如果你每个技术都想要深入学习达到精通，上手能用，能做项目，那你任何一个技术都要学习很长时间。但是面试的时候可不是那么问的，不会问你api和具体操作步骤，主要考察思想、原理以及经验。所以这里我们为了面试准备，就是对每个技术讲解面试的时候常问的一些问题。

能达到这个标准，那么可以短期内快速提升你对常见面试题的应答能力，让很多同学也许在一些互联网java面试时，也许原本很多题目都几乎说不出什么回答，到现在学完这个课程之后可以基本回答常见的技术体系内的面试题

这个课程，主要设计思想就是针对有面试需求的同学，短期内，短平快的突击一下，大幅度增强你在面试现场的一个表现，增加你通过面试的机会，让你跟很多普通的工程师在面试时的表现迅速拉开差距

所以针对这个定位，我进行几点说明：

1、面试题剖析如何回答时，我会剔除掉很多不必要的细节，就突出很重点。比如有的技术，网上一些博客也许会零零散散写几十点，但是面试的时候你不是背书，不是背博客来回答的，就是讲出来几个重点就可以了。所以有些东西也许我没讲，但是大家要明白不是我不讲，是因为我们要挑重点了解，在面试的时候挑重点来说。

2、对大部分技术点，不会动手去落地实践，比如说RabbitMQ的高可用性，我们就是剖析MQ高可用部署的一个原理，现场大量手工绘图讲清楚如何回答这个东西，确保你被问到的时候，能说清楚整个高可用部署架构以及现场动手画图。但是我不会带着你去安装虚拟机，然后安装rabbitmq，然后再部署高可用架构，然后再进行高可用容灾演练。因为我们的定位是短期内快随突击面试题，转为面试准备，不是动手项目实战课程。

3、对很多技术，我们的定位就是面试被问到的时候，你可以回答出其基本原理，我们不可能对有些技术点刨根问底，深挖到源码级别，那是需要大量时间去讲解的，不是所谓的两三周短平快可以突击的完的。所以确实有可能对于一些要求很高的公司，是可能存在一个问题，人家问你一个问题，你答出来了，接着深挖，你答不出来了，那么此时是没办法的，你要想对各种技术都有深入底层的功底，那不是两三周的事儿，而是两三年的苦功夫才能得到的。

4、有些特别简单的问题，明显只要是个java工程师，学过java的都该知道的问题，我就不讲了。比如说equals和hashcode是啥关系？string类为啥是final的？这类问题，实在是太过基础。

所以特别用几分钟时间跟大家说明白这个问题，大家了解一下这个课程定位，后面学习的时候相信就不会产生什么歧义、误会以及不满了。

最后一点说明，这个课程会作为“石杉码农学院”一个长期对外开放的免费课程，大家在学习这个课程之后，去外面面试，如果遇到这个课程里没有涵盖掉的问题，然后你觉得是比较难的比较典型的问题，可以通过微信公众号给我们留言，我们会定期免费持续更新这个课程的，持续解答更多的面试题

同时我们这里还会定期发布一些同学给我们反馈的互联网公司的java面试经历，我们会整理成文章后发布出来，供大家参考

Day4

面试官：你好

候选人：你好

大家寒暄一下。。。

（面试官在你的简历上面看到了，呦，有个亮点，就是你在项目里用过MQ，比如说你用过ActiveMQ）

面试官：你在系统里用过消息队列吗？（面试官在随和的语气中展开了面试）

候选人：用过的（此时感觉没啥）

面试官：那你说一下你们在项目里是怎么用消息队列的？

候选人：巴拉巴拉，我们啥啥系统发送个啥啥消息到队列，别的系统来消费啥啥的（很多同学在这里会进入一个误区，就是你仅仅就是知道以及回答你们是怎么用这个消息队列的，用这个消息队列来干了个什么事情？）

比如我们有个订单系统，订单系统会每次下一个新的订单的时候，就会发送时一条消息到ActiveMQ里面去，后台有个库存系统负责获取了消息然后更新库存。

面试官：那你们为什么使用消息队列啊？

（你的订单系统不发送消息到MQ，直接订单系统调用库存系统一个接口，咔嚓一下，直接就调用成功能了，库存就更新了）

候选人：额。。。（楞了一下，为什么？我没怎么仔细想过啊，老大让用就用了），硬着头皮胡言乱语了几句

（面试官此时听你楞了一下，然后听你胡言乱语了几句，开始心里觉得有点儿那什么了，怀疑你之前就压根儿没思考过这问题）

面试官：那你说说用消息队列都有什么优点和缺点？

（面试官此时心里想的是，你的MQ在项目里为啥要用？你没考虑过，那我稍微简单点儿，我问问你消息队列你之前有没有考虑过如果用的话，优点和缺点分别是啥？）

候选人：这个。。。（确实平时没怎么考虑过这个问题啊。。。胡言乱语了）

（面试官此时心里已经更觉得你这哥儿们不行，平时都没什么思考）

面试官：kafka、activemq、rabbitmq、rocketmq都有什么区别？

（面试官问你这个问题，就是说，绕过比较虚的话题，直接看看你对各种MQ中间件是否了解，是否做过功课，是否做过调研）

候选人：我们就用过activemq，所以别的没用过。。。区别，也不太清楚

（面试官此时却是觉得你这哥儿们平时就是瞎用，根本就没什么思考，觉得不行）

面试官：那你们是如何保证消息队列的高可用啊？

候选人：这个。。。我平时就是简单走api调用一下，不太清楚消息队列怎么部署的。。。

面试官：如何保证消息不被重复消费啊？如何保证消费的时候是幂等的啊？

候选人：啥？（mq不就是写入和消费就可以了，哪来这么多问题）

面试官：如何保证消息的可靠性传输啊？要是消息丢失了怎么办啊？

候选人：我们没怎么丢过消息啊。。。

面试官：那如何保证消息的顺序性？

候选人：顺序性？什么意思？我为什么要保证消息的顺序性？

面试官：如何解决消息队列的延时以及过期失效问题？消息队列满了以后该怎么处理？有几百万消息持续积压几小时，说说怎么解决？

候选人：不是，我这平时没遇到过这些问题啊，就是简单用用，知道mq的一些功能

面试官：如果让你写一个消息队列，该如何进行架构设计啊？说一下你的思路

候选人：。。。。。我还是走吧。。。。

我个人的意见，包括我常年在BAT公司里面干了很多年，各种同学都面过，一种比较类似的一种面试风格。就是一般我们不是发散的，我们是从点，铺开，比如说我们会可能跟你聊聊高并发话题，就这个话题里面跟你聊聊缓存、MQ等等东西吧。

对于每个小话题，比如说MQ，我们会从浅入深，这个上面的这个面试套路其实是蛮典型的，如果你确实说自己会MQ，你出去面试，去一些大公司面试，有些面试官可能就是这种风格，就是一步一步深挖

其实上面是一个非常典型的关于消息队列的技术考察过程，好的面试官一定是从你做过的某一个点切入，然后层层展开深入考察，一个接一个问，直到把这个技术点刨根问底，问到最底层。

如果没有刻意的对这种面试方式锻炼一下，出去面试碰到难一点的面试，大多会手忙脚乱，基本面试以失败为告终。

但是如果你把这些常见问题都掌握了，哪怕是面试官没问到你这么深入，他问你一个消息队列问题，你就自己给他说出自己的一整套见解，那么恭喜你，就是plus加分项了

Day5

1、面试题

为什么使用消息队列啊？消息队列有什么优点和缺点啊？kafka、activemq、rabbitmq、rocketmq都有什么区别以及适合哪些场景？

2、面试官心理分析

其实面试官主要是想看看：

（1）第一，你知道不知道你们系统里为什么要用消息队列这个东西？

我之前面试就见过大量的候选人，说自己项目里用了redis、mq，但是其实他并不知道自己为什么要用这个东西。其实说白了，就是为了用而用，或者是别人设计的架构，他从头到尾没思考过。

没有对自己的架构问过为什么的人，一定是平时没有思考的人，面试官对这类候选人印象通常很不好。因为进了团队担心你就木头木脑的干呆活儿，不会自己思考。

（2）第二，你既然用了消息队列这个东西，你知道不知道用了有什么好处？

系统中引入消息队列之后会不会有什么坏处？你要是没考虑过这个，那你盲目弄个MQ进系统里，后面出了问题你是不是就自己溜了给公司留坑？你要是没考虑过引入一个技术可能存在的弊端和风险，面试官把这类候选人招进来了，基本可能就是挖坑型选手。

就怕你干1年挖一堆坑，自己跳槽了，给公司留下后患无穷

（3）第三，既然你用了MQ，可能是某一种MQ，那么你当时做没做过调研啊？

你别傻乎乎的自己拍脑袋看个人喜好就瞎用了一个MQ，比如kafka。甚至都从没调研过业界到底流行的MQ有哪几种？每一个MQ的优点和缺点是什么？每一个MQ没有绝对的好坏，但是就是看用在哪个场景可以扬长避短，利用其优势，规避其劣势。

如果是一个不考虑技术选型的候选人招进了团队，面试官交给他一个任务，去设计个什么系统，他在里面用一些技术，可能都没考虑过选型，最后选的技术可能并不一定合适，一样是留坑

3、额外的友情提示

同学啊，如果你看到这里，连activemq、rabbitmq、rocketmq、kafka是什么都不知道？连个hello world demo都没写过？那你。。。

通过网上查阅技术资料和博客，用于快速入门，是比较合适的，但是如果要比如系统梳理你的面试技术体系，或者是系统的深入的研究和学习一些东西，看博客实际上是不太合适的

那也没事，我们这个课程的定位是不会去讲这些的，建议你马上暂停一下课程，然后上百度搜一下，这4个东西是什么？每个东西找一个教你hello world的博客，自己跟着做一遍。我保证你1个小时之内就可以快速入门这几个东西。

等你先知道这几个东西是什么，同时写过hello world之后，你再来继续看我们的课程

4、面试题剖析

（1）为什么使用消息队列啊？

其实就是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

面试官问你这个问题，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

先说一下消息队列的常见使用场景吧，其实场景有很多，但是比较核心的有3个：解耦、异步、削峰

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。。。

面试技巧：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，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。

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。。。

（2）消息队列有什么优点和缺点啊？

优点上面已经说了，就是在特殊场景下有其对应的好处，解耦、异步、削峰

缺点呢？显而易见的

系统可用性降低：系统引入的外部依赖越多，越容易挂掉，本来你就是A系统调用BCD三个系统的接口就好了，人ABCD四个系统好好的，没啥问题，你偏加个MQ进来，万一MQ挂了咋整？MQ挂了，整套系统崩溃了，你不就完了么。

系统复杂性提高：硬生生加个MQ进来，你怎么保证消息没有重复消费？怎么处理消息丢失的情况？怎么保证消息传递的顺序性？头大头大，问题一大堆，痛苦不已

一致性问题：A系统处理完了直接返回成功了，人都以为你这个请求就成功了；但是问题是，要是BCD三个系统那里，BD两个系统写库成功了，结果C系统写库失败了，咋整？你这数据就不一致了。

所以消息队列实际是一种非常复杂的架构，你引入它有很多好处，但是也得针对它带来的坏处做各种额外的技术方案和架构来规避掉，最好之后，你会发现，妈呀，系统复杂度提升了一个数量级，也许是复杂了10倍。但是关键时刻，用，还是得用的。。。

（3）kafka、activemq、rabbitmq、rocketmq都有什么优点和缺点啊？

常见的MQ其实就这几种，别的还有很多其他MQ，但是比较冷门的，那么就别多说了

作为一个码农，你起码得知道各种mq的优点和缺点吧，咱们来画个表格看看

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特性 | ActiveMQ | RabbitMQ | RocketMQ | Kafka |
| 单机吞吐量 | 万级，吞吐量比RocketMQ和Kafka要低了一个数量级 | 万级，吞吐量比RocketMQ和Kafka要低了一个数量级 | 10万级，RocketMQ也是可以支撑高吞吐的一种MQ | 10万级别，这是kafka最大的优点，就是吞吐量高。  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，几千个的级别，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从几十个到几百个的时候，吞吐量会大幅度下降  所以在同等机器下，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，延迟是最低的 | ms级 | 延迟在ms级以内 |
| 可用性 | 高，基于主从架构实现高可用性 | 高，基于主从架构实现高可用性 | 非常高，分布式架构 | 非常高，kafka是分布式的，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 |  | 经过参数优化配置，可以做到0丢失 | 经过参数优化配置，消息可以做到0丢失 |
| 功能支持 | MQ领域的功能极其完备 | 基于erlang开发，所以并发能力很强，性能极其好，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，是事实上的标准 |
| 优劣势总结 | 非常成熟，功能强大，在业内大量的公司以及项目中都有应用  偶尔会有较低概率丢失消息  而且现在社区以及国内应用都越来越少，官方社区现在对ActiveMQ 5.x维护越来越少，几个月才发布一个版本  而且确实主要是基于解耦和异步来用的，较少在大规模吞吐的场景中使用 | erlang语言开发，性能极其好，延时很低；  吞吐量到万级，MQ功能比较完备  而且开源提供的管理界面非常棒，用起来很好用  社区相对比较活跃，几乎每个月都发布几个版本分  在国内一些互联网公司近几年用rabbitmq也比较多一些  但是问题也是显而易见的，RabbitMQ确实吞吐量会低一些，这是因为他做的实现机制比较重。  而且erlang开发，国内有几个公司有实力做erlang源码级别的研究和定制？如果说你没这个实力的话，确实偶尔会有一些问题，你很难去看懂源码，你公司对这个东西的掌控很弱，基本职能依赖于开源社区的快速维护和修复bug。  而且rabbitmq集群动态扩展会很麻烦，不过这个我觉得还好。其实主要是erlang语言本身带来的问题。很难读源码，很难定制和掌控。 | 接口简单易用，而且毕竟在阿里大规模应用过，有阿里品牌保障  日处理消息上百亿之多，可以做到大规模吞吐，性能也非常好，分布式扩展也很方便，社区维护还可以，可靠性和可用性都是ok的，还可以支撑大规模的topic数量，支持复杂MQ业务场景  而且一个很大的优势在于，阿里出品都是java系的，我们可以自己阅读源码，定制自己公司的MQ，可以掌控  社区活跃度相对较为一般，不过也还可以，文档相对来说简单一些，然后接口这块不是按照标准JMS规范走的有些系统要迁移需要修改大量代码  还有就是阿里出台的技术，你得做好这个技术万一被抛弃，社区黄掉的风险，那如果你们公司有技术实力我觉得用RocketMQ挺好的 | kafka的特点其实很明显，就是仅仅提供较少的核心功能，但是提供超高的吞吐量，ms级的延迟，极高的可用性以及可靠性，而且分布式可以任意扩展  同时kafka最好是支撑较少的topic数量即可，保证其超高吞吐量  而且kafka唯一的一点劣势是有可能消息重复消费，那么对数据准确性会造成极其轻微的影响，在大数据领域中以及日志采集中，这点轻微影响可以忽略  这个特性天然适合大数据实时计算以及日志收集 |

综上所述，各种对比之后，我个人倾向于是：

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；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

Day5