小明 VS P9面试官

。

00:05

面试官，您好

飞扬

00:07

你好。OK，那您先做一下那个自我介绍介绍你擅长的项目，以及这个你的技术特长。

。

00:21

那个我叫小明，之前是在XX科技有限公司做那个java开发，现在是有三年多的开发经验，之前就是主要从事和在线教育相关的相关的业务。技术这块的话，就是有做过那个单体系统的开发，也有做过这个分布式和微服务系统的开发。然后主要的技术栈就是java，java相关的。

飞扬

00:51

明白。Java相关的就是因为比较宽泛哈，就我看我们的这个。讲的这些技术，你能讲几个你擅长的技术吗？因为Java它涉及的技术栈太多了。

。

01:07

就是对于，就是如果对于java。

飞扬

01:11

我我我举个例子，咱咱咱说就是因为我看你是在在线教育和电商，就是因为他的场景就会比较高并发或者是高负载这种情况会比较多一些，就是你是比较擅长于这个。写多读少的情况，还是读多写少，或者是高并发，或者是要求这个高事务的这个场景多，你接触的哪项目是偏，哪些会多一些？

。

01:43

写多读少和读多写少在之前的工作中都有遇到过，就是那种强一致性的事物的话，场景比较少一点。

飞扬

01:57

明白。

飞扬

02:00

OK，那你选一个项目，咱深入聊一下。

飞扬

02:09

你觉得你做的有成就感，或者是你觉得就是这个项目有技术含量，你觉得比较高的一个项目。

。

02:18

那个我，我们就是因为简历上第二个项目是，是我当时是我自己独自就是独自设计和编码开发实现的，然后后面还负责到了一些就是运维的东西，这个的话我觉得就是也也会有一点一些技术亮点。

飞扬

02:40

那行，那我们就聊这个项目，就是音频音视频处理系统,对吧。

。

02:46

对。

飞扬

02:47

行，那你可以先介绍一下这个技术架构，然后里面的技术和技术难点，你用什么技术解决了什么技术难点。

。

02:57

可以的，就是我我可能这个项目的话，我需要先说一下这个背景。

飞扬

03:04

好。

。

03:04

就是。所以，因为。

。

03:07

当时那个部门儿里边儿有是有一个音视频的小组的，他们技术栈用python，但是写了之后，后面这个小组就是人都不在了之后，就这个系统经常出现问题，然后我们也不能很快的去解决。然后后面后面就是公司里面有一个很重要的业务，就是那个微信机器人儿就是。大概有七八千台，就是微信机器人这种。就是用来服务我们的用户，他们。

飞扬

03:40

七八千台微信机器人是虚拟节点还是实际的节点？

。

03:46

有，有一部分是虚拟的，有一部分是真实，就是真实的手机。

飞扬

03:51

那我们虚拟的是怎么虚拟出来的是放在需那个虚拟机里面还是放在这个容器里面做的

。

03:58

虚拟虚拟的那块儿，它是有专门的云服务厂商，像华为云、百度云这种，他们会提供这种云云手机的。

。

04:10

就像允许建的服务，对对。

飞扬

04:11

类似于模拟器这种对吧？

。

04:11

对对。

。

04:14

然后他他们就是会采集到大量的语音聊天儿的东西，但是这个微信语音它本身是不能就是他不是MP3格式的，他原原始的格式是没有任何的播放器可以播放的，但是。公司里的运营什么，他们需要听这

些语音，然后的话就需要把它转成一个MP3格式的，所以就是有这样有这样一个业务背景。就是所以我们就是用Java去做了一下，这个重写了一下，然后又加上了对这个音视频转码的支持。

飞扬

04:56

那我们有几个问题，我问一下，第一个就是我们的这个实时性要求有多高。

。

05:05

它的实时性就是对于因为他目前是承载两个业务方的业务，第一个就是这个微信机器人的，他的微信语音转码，这个因为它属于内部人员使用，然后它的延时就可以接受的，岩石可以稍微大一点。但是第二类就是一个英语学习应用，他的视频合成类的，它的延时要求是比较高的，但是比较高，不是说像我们普通接口那种可能100毫秒之内要返回，他是说会把。可能十几个或者20几个比较短的视频，有老师的，有学生的，按照顺序把它加上水印拼接，合成一个大视频，就针对这个这个需求的话，用户他本身就是录视频的时候已经花了很长时间，然后他把视频全部上传之后，他就也可以接受，就等待比如十几秒20几秒这种的时间的。对，就是接口性能的话，是有这样一个要求。

飞扬

06:07

明白。那那这个我们的这个数据量有多大。

。

06:14

这个数据量的话。每天就是语音这块儿的话，他可能会有100多万的数据量，然后这个。视频合成这一块儿，他可能会有十到20万的量，但是主要这一点，他的他的量，每天虽然每天的量不是很多，但是他那个因为是。就是。教育系统基本上都是这样，他的就是他发生的时间就比较集中，可能这100万就是中午那会儿，或者是下午那会儿，或者是周末某个时间点，会有很突然就进来很多的这种需要处理的任务。

飞扬

06:59

明白，那你在这里面遇到什么具体的问题吗？

。

07:04

具体的问题就一开始，为了就是验证，我就是我们用家去做这个东西，它是否可行，刚开始就就是使用了，因为我举里举，我拿一个例子来举，就是比如这把一条儿微信的原始语音。转成内可以播放的MP3格式的，它要经过好多步骤，第一个步骤我们要把它那个把这个原始文件从阿里云上下载下来然

后还要把它解封装成一个原原始的PCM文件最后接下来要把这个PCM文件重新就是封装成MP3的文件

。

07:46

服装城 Pmp3文件之后，我们还要把它上传到一个地方，也是阿里云的oss，但是，这个上传的地址。他可能是业务方提供的，所以还要去回调业务方的接口儿，然后把它上传就上传，上传好之后，需要回调一下处理的结果，去把一些临时文件给删除，这个中间还涉及到一些日志的上报，也就是。记录一下他这个整个任务他的处理的流程，到时候这个也是方便那个排查问题的时候做的一些东西。

飞扬

08:26

对，我这我的意思是说，就是咱咱回归一下哈，就是因为你你聊的流程比较多，我就是说你用这个框架做了以后，你用具体的技术解决了哪些实际的问题。

。

08:41

刚刚那个大纲那个大流程的话，一开始我们做的第一版的话，就是用单线程。不是用单线程，就是用一个完整的线程去把这些步骤全部都给，就是完全七八步全都交给他自己做。

飞扬

09:00

串行的做对吧?

。

08:58

然后这种的话，他只是方便，我们就是一块对实现起来比较简单，然后快速验证一下，但是做完了之后这种话题就发现他整个，吞吐量很低，然后它的cpu，还有，这主要是cpu还有网络发现都占用很低，就达不到一个占用很高的那个那个水平上。

。

09:23

然后就是为了提升他这个吞吐量，还有他这个。性能这块儿的话，第二把我们就使用了，就是封装了一个通用的，就是一个一个一个组件算那个基于是就是用那个disrupt那个。就是线程间的一个消息传递的一个库，封装了一个发布订阅的一个组件儿。这个组件的作用就主要是我们首先，我我介绍一下这个组件儿，这个组件儿。他首先他会有一个，就是会有一层负责分发的一个ring buffer数据结构。然后这个的话在他下面，在她下面会分成多个多个channel，然后每个channel它，它自己绑定一个单独的ring buffer。然后我们就把它整个运行的，它整个运行的流程的话，先先进到一个分发的remember里面，然后再进到再根据他这个不同的事件的类型，进入她，进入到他那个不同的channel。绑定的remember里面，然后这个负责处理具体处理消息的这个ring buffer，而他会我们也可以给他挂一些，就是自定义的业务线程池。去去并行的去处理。然后像刚刚内内原来是把内七八个步骤的话他。你交给一个线程整体制作，现在的话，他可能就是我们，我们都把它分成一个很小的就是。

。

11:03

很小的步骤，就每一个步骤。每每一个步骤，它可能一个任务来了，把它分成七八步，在最开始的时候。他只需要就封装一个，就可能下载一个。

。

11:17

下载一个M那个音频文件的一个事件。然后，然后他就会进入到刚刚那个发布订阅类似就是观察者模式的那个封装那个组件中。当当当这个每个事件，它是有一个消费者与之对应的，但消费者先听到这个事件之后，他就把负责把他东西下载下来。然后再去发布封装成另外一个，就是把这个Amr的音频文件解封装成这个PCM的文件就像而且这种的话，它是可能有同时有很多个，有很多个这种任务进来。

。

11:58

然后。这样的话就他就会比我们三个县城儿就做完一个任务，因为单个比如你有比如有有16个线程它。做同样的任务的话，每个县城都独自做完，他可能同时只能跑这个16个这个任务来处理，但是这样的话。他可能这样，但是就是这样的话，我们他这样的话，他可能也是可能成倍的就是两倍，三倍的同时，在处理的任务任务量。因为就，就是。比如他第一第一步的话，第一步它它它可以源源不断的，就是始终有任务在跑着，不会有那个线程有空闲什么的。

飞扬

12:52

明白。

飞扬

12:55

OK，那那我想问一下，就是我我们当时我们为什么选型这个软件。

。

13:04

当时为什么选。因为因为当时就是想做一下，就是异步化的这个异步处理的这个东西，就开始调研，调研一些。就是。那不是那就是调研一些这种进程，就是线程间通信的一些库，然后就会这就是对比了，对比了其他的就是JDK他自己的一些阻塞，阻塞队列这种。然后就会发现这个disrupt他比较高效，然后首先它它是它是没有锁的，就是没有锁的，另外一个就是它很多的，就像每个rain buffer里面的。很多的元素就是每个事件它是就是提前。是提前创建好的，之后都是复用这个对象也能避免，就是垃圾回收。这些和频繁的就是创建对象。另外就是它的就是很多，有很多的其他的开源软件里边儿也用到他也用到，也有用到，他就好像log4j机里边儿也有这个disrupt去一步一步的去打日志。

飞扬

14:23

但是他这个是对整个包括他的这个事务，这些要求是不高的，对，也不需要就是说它的这个流程的这个安全性不需要保证，但是我们的这个场景，如果说要去对这个流程的安全性做保证，你怎么去保证它的这个安全性。

。

14:41

这个流行的安全性、安全性具体是您您具体是指哪方面的？

。

14:41

这个流行的安全性、安全性具体是您您具体是指哪方面的？

飞扬

14:47

比如说我们之前比如说可能要通过九步来完成这个整个的这个流程对。你怎么去保证？因为你已经。你已经因为无所状态，然后你都发出去任务，对不对。

。

15:03

对，首先内内这个我举个例子，就我们最开始有一个feature线程去，任务之后，他会有一个就是信号量去去限制一下，他最大最大整最大的能够就是同时处理的。这个这个任务，因为他单台机器不可能就是无无限的让他同时处理那么东西，第二第二点的话。就是我们。每个每个每一个步骤处理完成之后，它会。他会有记录日志去上报给，因为他会上报给那个，这个这个系统它是有有三个，有三个有三个服务组成的，就是一个就是负责。入口的一个服务，然后刚刚咱们说的逻辑是在那个音视频处理服务，在这个服务内部的接口，另外还有一个就是公网的一个notegs的代理服务。像就是您说的那个。首先每一步处理完成之后，它会它会记录一些，记录一些日志，然后会上报的，上报这个上报的这个入口，服务入口，服务里面，再接入服务内部，它可能会有，他会首先它会有一个，如果是处理失败的话，某个步骤儿，他可能这会儿正好，比如网络问题，网络不稳定，它下载东西确实没下载，没下载到来，然后失败了，他会把这个失败的信息，就是以日志的城市上报，给这个入口服务，但是入库服务如果是这种失败的问题，因为网络的问题下载不下来，失败，那他可能会。在在在那个服务里面，重新让他入一下队列，然后这个音视频处理的服务，还可以再把他拉回来处理一次，如果是网络问题的话，这样应该能解决，另外一个就是。比如他超时了或者什么的，然后也会把这个结果去超时了，可能就没有结果，没有结果这种话。可能就是然后在那个入口服务，他会有一个有一个有一个定时任务去查一下库里面就是。

。

17:17

已经超时了的，就是过了，过了，因为有他的那个收到这个任务的时间，就是过了那么久之后，还没有还没有结果返回的，也会把它给重入一下队列。因为这个处理的话，他不要求幂等性的，就是这种的话，它不会影响原来的结果，她，就算之前那个他。可能稍微晚，但是戴助理成功了，他他也也是不影响的。然后。

。

17:46

有一有一个点，就比如像那个学生学生和老师的视频，他可能每个步骤就就拿下载的这一步来说，它有可能要一下下载十几个。就是十几个视频那种，但是这十几个视频，他要等到最后一个下载完之后，这个步骤他才能，他才算能结束，他才能到能才才能流转到下一个步骤，这一块儿的话，就是我们，我们。

。

18:12

在事件里面会有一个cutdownlatch，根据他的那个需要下载的视频的数量，有一个cutdownlatch去控制这样一个东西，就每次下载完一一个之后，你就判断一下他是不是零，如果是是零的话，他才会去触发第二部这种。

飞扬

18:33

那他岂不是，如果是判断它是不是零，他岂不是他就是一个。那个他那个卡点。

。

18:41

对，是是是，是那种是一个那种意思的。

飞扬

18:45

对，然后如果说你的这个场景是一个多生产者和多消费者的情况下，那么这个框架他怎么能保证多线程的数据安全？因为他有锁吗？

。

18:59

对，您是您是问这一点多线程的这个他这个框架他因为没锁，但是他。他底层就是也是用那个安抚去保证它那个就是无所画的。通过就是它，它是。通过unsave去操作，去直接操作字段，然后获取它就是这个某个字段，然后在这个。

。

19:28

对象他的偏移位置，通过安飞屋提供的一些compare and set，这个方法去保证他。保证他的多线程在多线程下是安全的，就是。

飞扬

19:42

他是怎么去保证，就是您说的这个unsave，它就通过什么原理或者什么机制来保证。

。

19:54

这块儿他就是。我这块儿有点儿有点儿就是记不太清楚了，因为这个unsave再往unsave再往下面，它应该是就是和那个。

。

20:08

内存内存，内存一次性相关的一些东西。

飞扬

20:13

内存一致性相关。那那你看我们举个例子哈，就是因为我看你写的这个。里面你你也用到了队列，对，kafka。

。

20:25

对这个入口服务。

飞扬

20:25

对？那，不。

。

20:28

您说。

飞扬

20:28

我们kafka的话，其实队列里面它是有锁机字来实现的，对。

飞扬

20:36

就是他的这个锁机制和我们这个。现场安全的这个机制有什么区别吗？

。

20:46

就是kafka的那个里边儿的队列的那个锁机制可能有点儿。之前不是太了解。

飞扬

20:55

但是我看你里面用到了这项技术。

。

21:00

对，这个是作为在入口服务上，那个业务方去，可以选择通过调用接口或者是投递卡夫卡消息来去投递这个任务。

飞扬

21:13

明白。

飞扬

21:14

我再问一下，就是刚才我听您说这个这个软件我们用到了这个观察者的模式对。

。

21:21

对。

飞扬

21:22

那你能介绍一下吗？他在用这个观察的这个模式是应用到什么场景，然后它是怎么去实现。

。

21:31

运用到场景的话，他还是借助那个刚刚说的那个那个那个库described库去实现的，因为它本身它它那个那个那个队列，就是内存队列，它是一个多多生产者，多消费者，这这就本身就是一种发布订阅的一种模式。然后像我们这个事件的话，我们写一个事件，写一个，写一个监听器写一个listener，也可以写多个listener。

。

22:02

然后当我们发布事件的时候。他通过他。它里面从他raing buffer里面不断的get。就是不断的获取，获取到这个事件之后，去去回调这个。他这个事件，这种事件类型对应的所有的listener，然后这种的话，我认为它就是一种，就是观察者模式，虽然他可能不是，就是像说的那么不是像那个书里面说的观察者模式，那么规范，但是这这是一种观察者的效果，就是。

飞扬

22:37

明白，但我刚刚听您说，就是它是通过听器对，那为什么为什么他设计的时候不用监听器的模式，而用观察者的模式来做这个事儿。

。

22:50

因为监听就是监听器，就是他可能监听的那种，他就需要有有那种用callback的机制啥的，这种的话就对比现在这种方式，我认为就是现在这种方式，它是比较简单的一种方式。

。

23:05

实现起来。

飞扬

23:07

明白。

飞扬

23:08

OK。我看了你还用了mango DB是，在这个里面。

。

23:16

对。因为这个库用的是mango DB。

飞扬

23:21

为什么要用mango DB?

。

23:24

因为就是。在公司里面的话，我们的经常就是使用的库，要么是mysql，要么是mango DB。然后这块儿的话就考虑他那个。他可能因为，因为说要一个任务投递进来的时候，在入口服务这里，它是需要写库的，我们会同时把它写到一个redis的队列里面，但是还是需要写库的，就考虑到他就是。

。

23:53

支持的QPS或者TPS这块的话，我们选用的是mangodb，其实这块儿用mysql也可以。

飞扬

24:03

那你但是，但是你这样的形式成本就高，就是你弄的这个分布式文件存储的这个数据库，然后又用了mysql对，那你你相当于你维护成本就很高。

。

24:17

对，目前就是他直接，因为他作为一些，就是存一些任务的，每条任务，每条投递大都是任务信息，还有一些日志是直接用了mangodb，它没有没有没有没有mysql了就是只选了一个

飞扬

24:37

只用了它来存这些是。

。

24:39

对。

飞扬

24:39

但是，但是问题是，就是你用了mango DB解决，就是跟redis不是mysql的话有什么本质区别吗？你为什么一定要选这个mangodb呢？

。

24:52

他没有，他他本质区别就是考虑两点，第一就是它数据量比写它写入量可能会比较大。然后想着mangodb，首先他对他对，就是写入的并发这块儿肯定会比mysql高一点，另外一点就是它涉及到一些日志。然后本身用mysql来用，如果这些用mysql来存的话，本身就不太适合的，人家mango的话，虽然他是一个就是文档型的数据库，但是如果用它存一些日志什么的，就是这个量还是可以接受的。

。

25:30

所以选的是什么。

飞扬

25:30

对那那那有一个问题就是。那么那个mysql的话，和这个mangodb他mysql的话，关系型数据库的话，它关联查询，它能很有效的很快的能查到，但是mango DB它就有一定难度了，那你怎么去解决这个问题？

。

25:50

这个这这个的话是在这种场景里边儿，它关联查询，因为它就非常简单，就只有两张表儿，不是就只有三张表儿，然后他没有这种也考虑到它的业务比较简单，他没有这种说要关联查询的这种场景，他基本上都是可以根据ID去操作这个文档。

飞扬

26:13

没有通过时间吗？或者是通过其他的这个关键字。

。

26:20

基本上是通过通过ID去定位的，像像像记录日记这些啥的，然后像刚刚说的，就是它有一个定时任务，去查一下最近。最近的任务有哪些是超时的，这种它会根据跟跟它会根据比如时间字段去查询。

飞扬

26:41

时间字段查询，那么通过时间差字段查询的话，他就有可能跨文件或者跨这个表的。

飞扬

26:49

查询了。

飞扬

26:53

那怎么解决这个问题？那这个如果说大家都是按ID来存储对，有点像列式存储吗。那你横向去查询的话，你就会遇到一些问题。

。

27:06

因为这块儿它主要是就是单表的操作，然后像漫步DB它底层的文，他底层就是数据。

。

27:16

存储数据的那个文件。文件文件的格式，他组织的那种可能不太熟，就是蒙DB存储数据的那个文件的那块儿。

飞扬

27:30

行，没没事儿，然后我们再聊一下那个，我看你用写里面用的redis你用来做什么？

。

27:39

redis的话，这块儿就是它的流程，就是会有这个业务方把这个任务投递过来，用ready就是。有一个队就有一个list类型的数据结构。然后他不断地入队、入队、入队，然后就各个音视频处理服务的实力，他去来拉东西的时候，来fetch的时候，我们就是从头从前面就用用Lrange去把它给取取任务，然后最后再把用LTM这个命令的把它这个任务给删除。就是因为这样的话就避免去再去去，每每次来发起调用都都要去库里面去查。

飞扬

28:27

那就是说缓存一些部分数据对吧？

。

28:30

对。

飞扬

28:32

那我们的数据量大吗，缓存的。

。

28:36

因为这个东西它是一直首先每天的任务量也不算特别的大，而且的话就是，他是不断的在在处理着的，就是有东西在写，然后也也都也有东西在处理着的。所以这个量他应该不会，他不会达到那种大K的那种那种级别。

飞扬

28:57

明白，那么我们redis是单台机器还是有主从还是一个集群？

。

29:03

redis这块的话，它就是一个单台的机器就是。是一个那个，哨兵也不算单台，就是一个哨兵类型的。

飞扬

29:15

哨兵类型是。

。

29:16

他不住不自己选对。

飞扬

29:19

那我们那个。Red is是能介绍一下它里面的哈希槽你是怎么用的吗？

。

29:28

它里面的哈希，哈希槽这块儿应该适合他的，集群相关的就是。你说的是那个集群，为了就是保证保证那个数据均匀分布，通过那个比如16384个slot去。去给不同的集群节点分配，分配某个段的曹魏，然后去。去，然后用K去做哈希，看他落到哪个槽位去实现就是。这个数据在集群中比较均匀地分布那种。

飞扬

30:04

对，但是他怎么通过哈希槽实现数据共享？

。

30:10

如何通过哈希草实现数据共享？

飞扬

30:14

因为他集群，对吧。

飞扬

30:19

集群的话，他肯定是要组成一个这个数据共享的一个。

。

30:19

这块儿。

飞扬

30:27

地方，然后让这个数据进行散列的这个排列，然后不管是查询还是这个，然后。或者增删改查，他都能够去通过一个机制来来维护这个对。

。

30:45

那这块儿的机制我可能之前不是很了解。

飞扬

30:49

OK，那那你对这个一次性哈希了解吗。

。

30:54

一致性哈希。这块的话，我们公司之前有用过那个，我大概知道他他是。就是些。那个因为这个一般都是如果有这种需求，我们直接用了库，就比如那个摸摸那个摸摸，哈西那种它实现的库去去做这种一致性哈希。

飞扬

31:22

什么库？没，没听清。

。

31:24

就是有一个就是mur hash那个库。

飞扬

31:30

你直接掉那个库是吧。

。

31:32

对。

飞扬

31:33

你掉那个库的话，它是里面的机制，你你大概了解吗？就是它通过什么样的一个。

。

31:38

机制的话，大概他就是那个。因因为就是散裂的话，它就需要比较多的节点，它就是类似哈希环的那种那种东西。就是我我们我们去计算出来，一个一个哈一个哈希值，它可能。他在在落到怀上的某个位置，但是如果这怀上的节点就是。比较少比较少的什么的时候，他他可能就会不均匀什么的。然后这种的话就是可以引入一些就是虚拟的节点，这种就保证它的数据均匀。

飞扬

32:17

但是虚拟的节点会会打得很散。就是如果说举个例子，比如说有一个节点。当机了节点，他怎么去？去去那个。保证这个数据的完整性。

。

32:37

去你点点，如果有个节点，真实的节点宕机之后。

飞扬

32:41

对。因为你真实的节点就会打散成很多虚拟的节点，对。

。

32:52

这块儿的话，我可能还要去看一下资资料。

飞扬

32:56

明白。没事儿，我我们再聊一下您说的这个秒杀项目哈，这个没事儿，反正那个因为你们不是已经做完了吗，我就了解了解一下，就是你们是怎么设计的，然后里面大概聊一下，我们聊一下技术。您可以介绍一下，就是这个项目。

。

33:17

这个项目它是有两大块内容，组成一块儿就是秒杀这一块儿的内容，另一块儿就是页面静态化的一些东西。

飞扬

33:27

明白。

。

33:28

然后它整体的，您说。

飞扬

33:31

您是怎么考虑的？就是场景是什么样的，比如说我是要做一个什么活动，然后有多少礼品要去秒杀，秒杀时间多少，用户有多少。

。

33:45

对他就是他就是以活动的那个形式去展开的，就是活动，然后活动里面会。挂一些就是活动的商品，或者是课程，什么什么就是类似的东西。然后，亮亮的亮块儿的话，因为这块儿就是。具体的数据可能还不太清还不太清楚。就是QPS那些。

飞扬

34:18

那就是为了做一个这个产品是。

。

34:21

对。

飞扬

34:26

那你介绍一下，就是你设计这个。这个系统的时候，设计这个系统的时候，用这些技术是出于什么考虑来设计这个？这个系统的秒杀这个系统。

。

34:39

首先就是除以它那个，就是秒杀的时候，他会时间很短，然后瞬时流量很大就得去。用一些组件去扛掉它的那个瞬时流量。

。

34:52

然后。就就拿库存，就拿库存这块儿来说，就是秒杀的东西，一般他都是有，就是固定数量的库存，然后这里面库存这块的话，它是使用了redis的一个，使用了多台redis，就把这个库存进行分片存储，要到时候执行抢购的时候他可以。就是多个redis，多台redis机器，它可以同时去处理这个抢购的请求。

飞扬

35:25

redis是处理抢购的请求，没太明白，redis不是读取读取的吗？他又不是写入的。

。

35:28

是不是？对。

。

35:33

不是那个我可能表达有些问题，就是他如果有就是他把那个库存分到多个redis的实例，就可以我们我们的服务去。

。

35:45

执行lua脚本去扣减这个库存的时候。他可以分别扣减，不同redis啊可以同时扣减，就是多个多个服务实例，它可以同时扣减，不同的就是。

。

35:58

不同节点redis上的实例，然后这样。就比单台的比单台单台主从，这样能够扛住的亮，并发量会大很多。

飞扬

36:09

明白，但有一个问题，就是你redis的话，你即便去扣减他们的这些实例，你怎么去维护他的锁？分布式锁？

。

36:21

这块儿其实在扣减的时候就没有进，没有对他进行加锁，就就利用他就是执行lua脚本那种原则性，就保证他这个。

。

36:33

扣减成功就是在lua脚本里面写了一些扣减库存的逻辑，就是比如把它就是目前的目前的库存数量给获取到，然后去去减了之后去看一下他是不是。这不是比邻是不是，是不是不小于零，然后。就是他把这个原子这个操作的原子性封装到这个lua脚本里面了。

飞扬

37:03

但是脚本，脚本的话，你是有多个脚本还是一个脚本？

。

37:07

扣减的话，它只有这一个脚本，就是扣减。

。

37:10

他，他这个。

飞扬

37:11

那那你这个脚本的作用是不是相当于一个队列的这种角色？

。

37:17

也不是说一个队列，因为这就是在这个活动开始之前，他会把所有的库存都给均匀的分，其实就是数字，就某个ID的商品，它的对应的秒杀库存是多少，全部同步到redis上了，但秒杀开始之后，他只有只有一。操他的操作就只有一个操作，就是他从他去扣减库存，就是把redis的扣减，扣减库存，如果扣减成功了，就表示他这个抢购成功了，如果扣减失败。就像刚刚说的那个lua脚本的逻辑里边儿，如果扣减之后发现它的它比零小了，这种他其实是已经没有库存了。

飞扬

37:59

但是会不会出现这个最终一致性的一些问题。就是如就是你的这个缓存里面的库存数据和库数据库是有不一致的。

。

38:12

是是这样的是是这个在秒杀配置秒杀活动的时候他，会把就是商品的真实库存就一下一批就，比如这个活动有有提供100个秒杀商品，它会把这个100个库存，就是提前在商品系统里面的库存给锁定住。然后等到秒杀开始之前，再把这一百一一百个库存同步到red is里面，当秒杀结束之后，他会处理这些库存，就是如果是真的是秒杀完了，他会把那个。就是商品系统里边儿。锁定的内100个库存。就是给给真的给剪掉，如果是如果是一般一如果是没有没有没有秒杀完的话，可能再会给他加上这种。就是她她跟她这在秒杀这个环节，他没有和其他的交互，就是为了简化，他只让他这个。

。

39:12

扣减redis的里面的这个库存。就是没有没有别的。和数据库相关的了。

飞扬

39:21

你就是说都在这个缓存里面存储是。

飞扬

39:25

和这个更新。

。

39:25

对。

飞扬

39:27

那有没有可能出现超卖的情况?

。

39:31

超卖的情况也也是通过就是这个lua脚本的逻辑去去让他保证了。

飞扬

39:38

那我们这个脚本是不是相当于就是一把锁。

。

39:39

就是。

飞扬

39:42

就是它是它是一个它相当于是一个数据，这个微信的存储对吧。

。

39:42

对。

。

39:50

对，因为那个red is的话，他去执行一个lua脚本，就不管它里面有有get，有多少个get和set这样的操作，他这个对对这个脚本来说他对red is来说，他执行这个脚本，他就是原子的，它不会说像我们用用Java代码，我我第一行先get一个值，第二行set一个值，然后。然后就或者是更新一个值，然后它它是有那个这两行代码，它它是两个不同的操作，它是不不是原子的那种，通过就是通过这个脚本去保证，保证他的一个原子性，他既不会被超，不会超卖这种。

飞扬

40:33

对，但是我有一个问题，就是为什么我们当时不直接用red is分布式锁来做这个事儿，要到最后面，比如说这个到这个，你的这个脚本端才去做？因为你这样的话，岂不是给red is的压力会更大一些。你直接在前面给拦直接用分布式锁做了不就。可以吗？你没必要那么多那么大的访问量去访问这个脚本。

。

41:01

这块儿就是他两种方案都可以行，当时可能就是觉得这种脚本的方式比较简单，然后就选用了这种方式。

飞扬

41:12

明白，我再问一下，就是我们当时设计的时候，在前端做了哪些处理，来来这个，一个是负载，还有一个就是对我们的这个。量比较大的时候会出现，比如超卖情况，或者是数据不一致的情况，前端做哪些处理？

。

41:35

前端这块儿的话，首先是他他有大量访问的时候，对页面儿做了一些静态页面静态化的措施，然后。就他在抢购的时候，他必须要先就是包括一些登录的验证，还或者一些图形验证码这样的，这样一些就是蓝调一些流量的。难道一？

飞扬

41:56

我们的秒杀是先登录进来是吗？秒杀不是说秒杀到了再登录是吗。

。

42:03

对。然后就通过这样一些措施，就蓝调一些，先拦到一些流量，然后的话他他会再再秒，在秒秒杀开始之前也也有一个，就是那种类似预约的功能。然后只有你预约之后才有资格去抢购的这样一种措施。

飞扬

42:28

预约是提前多久预约？

。

42:30

就是这个活动。这个活动可以在活动添加，就是创建活动的时候可以指定这个可以配置一下这个时间。

飞扬

42:45

没太明白，就是预约，比如我提前一天预约还是怎么。

。

42:50

就是根据这个这个活动，他可能是设置比如。今天你可以看到就是明天有一场秒杀活动，然后它最早就是用户这个活动用户，用户什么时候可以看到，有一个对于活动来说，就是有有这个活动。这个用有他开始的时间，也有它用户可以看到的事件，就他在看，用户可以看到之后，然后他就可以点击预约。

飞扬

43:17

明白，就是你配置什么商品，然后什么时候能看到它的秒杀活动对。那那没有全局的吗？比如说我就是100台这个手机或者是多少。

。

43:31

这个全这个全局的，但是还没有，他都是根据那个活动，就是跟着就是根据一个活动来的。

飞扬

43:40

明白，那如果如果说我要设计一个全局的话，你会考虑哪些方面。

。

43:40

就是因为有的。

。

43:47

您是说。

飞扬

43:50

比如说我就是小米手机有100步，然后比如说1月1号要开抢，对，可能会来很多人，然后你怎么去设计这些。这个这个保证这个性能一个是稳定，一个是系统不被崩了，然后，同时还能保证这一前100个人是能拿到这个手机的。

。

44:15

这块儿的话。我想一下组织一下，组织一下这个这个语言。

。

44:24

第一步，您说的。

飞扬

44:24

对，其实他其实他跟你的秒杀系统一样，只不过是少了个预约功能。

。

44:33

其实就是现在目前这个的话，设计师是是是支持这种情况的。

。

44:42

就是它支持，就是刚刚您说的这种场景的话，场景的是是是支持的。

。

44:57

对这块的话。

。

45:03

他这么做。怎么把就是怎么保证，它只有就是刚刚的那个库存的话，他可能还是就是用枷锁去扣减库存，或者是用脚本去去扣减库存，保证它，它它就是不会被超卖，然后针对流量访问大的时候，因为可能有很多人来抢，很多人来抢。

飞扬

45:24

还有人攻击对，还有人模拟用户，对。

。

45:28

对，然后会在，首先会在接入层那边会做一个限流，因为它我们已经知道了，就是这他只有100个商品，这个商品数量秒杀的数量是可以拿到的，然后可以其实可以就是不要放那么多请求进来，也许100个商品的话，我们可以放1000个请求，或者2000个请求，或者500个请求都可以，就最终放进来，就是到达我们后端服务的这个请求的数量。然后就是。

飞扬

45:56

但但是比如说你不放这么多进来的话，他有可能一秒就有很多可能几十万，上百万一秒，就有很多人来，你怎么去判断这些人，因为你没办法区分，对，因为它是并发进来。就是他一秒的时间，他就会有很多。你就没法控制这些人？就是说一下拦截出去。

。

46:28

这种的话。每秒可能可能，对于就是每秒百万，这个QPS的话会会会有些。

。

46:41

对，可能nginx也因为目前是用这个lvs和nginx作的接入层，如果每秒百万的话，这种是不是可以考虑一下，就是购买一些云服务这种。

飞扬

46:48

明白。

飞扬

46:56

对，你们现在是用的是本地服务器是吗？

。

47:02

对，就是就是一些公司里的机房就是租的，比如像世纪互联什么这种机房，然后自己的。

飞扬

47:11

没有在云上面做。

。

47:13

有一其实有有一部分业务在在那个。在云上，在在在云上有一部分是在自己的部署在自己的机房里，但还是以那种就是自己机房为主的那种。

飞扬

47:29

明白了。

飞扬

47:32

以自己机房为主，那如果你上这个秒杀系统的话，会有单独的域名或者是单独的这个服务器吗？

。

47:40

他会有单独的域名儿的。

飞扬

47:44

服务器没有单独的服务器，还是共用是？

。

47:47

服务器，它可能它，它也它也是就是比如JVM虚拟机，这种机器。

飞扬

47:55

虚拟机。

飞扬

47:58

虚拟机，明白。行。那个我在问了解一下，就是你这里用了rocket

的MQ是用来做什么？

。

48:09

这个rocket mq的话。就是。就比如，秒杀成功之后，他去进行业务下单的时候，它不是它不是就是直接。就是可能有的时候商品秒杀的这个下单抢购成功，虽然就是抢购成功的可能也很多，但是。他没有直接说这一抢购成功了，然后立马就去调用订单接口去创建订单，这里的话是用rocketmq给他削了一下风，就是他只是抢购成功之后，他并没有真实的去创建订单，而是发了一个消息，然后有这个消费者去消费这个消息，去再去具体的去创建这个秒杀的订单。

飞扬

48:59

那你为什么不直接就用release里面的队列就可以，这个很简单，就是我看到这个场景。为什么要引用rocket mq自己来做这个事儿？就是用到rocket mq里面的这个什么特性吗？

。

49:14

也没有什么，也没有什么特特性，对这块儿其实可能一开始反应，如果用队列的话，就直接把rocketmq给删了。就没有考虑用red is的队列那种。

飞扬

49:29

那。

飞扬

49:31

就是为什么会当时选择rocket mq是你们别的项目在用是吗？

。

49:37

对。

飞扬

49:38

你就是在复用一下是。那我们的数据库试用的，买搜狗数据库对不对？

飞扬

49:49

那我们的这个买搜狗数据库，比如说并发量高了。他的表是怎么保证一致性的？

飞扬

49:58

表数据。

。

49:58

并发到高的话，它会有，它就是就是它的隔离级别，有一个MVC的一个，一个机制。

飞扬

50:15

他怎么保证？

。

50:15

就类似，就是。

。

50:17

具体怎么保证的话。

飞扬

50:23

因为MVC的话，他其实像版本，就是乐观锁的概念吗，版本的这种方式来来进行锁定的对。

。

50:38

具体的话。

飞扬

50:41

这块可能研究不深，对。

。

50:45

对。

飞扬

50:46

明白。

飞扬

50:47

再问最后一个问题，就是刚刚在前面这个音频系统里面，我听到你说了一下这个GC这一块的，这块你熟吗?

。

50:57

GC这一块儿有了解过一些。

飞扬

51:01

你可以讲一下你的这个理解，就是就是这个。比如我们的这个GVM里面JC的一个大概的这个原理。

。

51:12

就是因为我们就是代码里面会代码是跑在虚拟机里GVM里面的，然后会创建，肯定会涉及到频繁的创建就创建对象什么的，但是这个内存它有限，这块的话，他可能就是把JVM的堆内存，就是纷纷围成多个区域，比如就是年轻的年老代驾，然后刚开始是在年轻戴的话。创建的对象放到年轻代，然后这个时候当他可能当他满了之后，会触发一一次，一次将GC这种，然后他他就是有不同的那个垃圾回收器，它可能会有，就是不同的找到这个判定这个。这个对象是否是是否是垃圾的那种东西，然后。然后GC的这块儿，我们因为这块儿配置的话，基本上都是直接用了就是。

。

52:11

记忆垃圾回收器，然后。然后就是简单的配置一下那种。

飞扬

52:18

明白，那如果说你遇到比如说这个性能有问题。

飞扬

52:23

那你调优会从哪几个方向去调。

。

52:27

调优的系统有问题的话，他可能。第一反应的话，第一个第最先出现的现象可能就是接口的延时比较大了。这这个时候的话，我们就要去看一下他那个就JVM的监控，或者是首先就是看一下内部的监控，看他找一下具体的原因。然后的话也需要关注一下他是不是确认一下他是不是触发，触发full gc，然后导致就是。导致就是就是类似那种stopped the word这种，这种情况发生了。

飞扬

53:11

明白。

。

53:13

就如果是有这种情况发生，可能就是需要检查一下代码，看哪里，就是是不是有大量创建对象的，大量创建对象的。

。

53:24

地方。

飞扬

53:26

比如说这个老年代。

。

53:26

或者是。

飞扬

53:29

很大，但新生代比较小的时候怎么办？

。

53:32

对我就就是这种比例，可能还需要观察一下，就是不需要调整，这下这种比例可能就像您刚刚说的，就是如果老年代它占用比较大，但其实。他其实没有必要占占，占那么大，就可以把那个年轻的你稍微放大一点，然后很多的对象，让他在这个年轻代能够被及时的垃圾回收掉，它就不会说进入老年代，最后再触发fullgc的那种过程，就是这个比例可以根据那个代码的场景。就是去。去去调整一下比例。

。

54:12

然后再去观察观察一下情况。

飞扬

54:12

点。

飞扬

54:16

好。那我这边暂时没什么问题，你有什么问题吗。

。

54:22

我这我这块儿的话。然后就是。就是您觉得就是有哪方面欠缺的吗？因为刚刚可能回答问题，就是还不是特别理想，没有很流畅。

飞扬

54:38

我觉得你的这个。技术的全面性都还可以，就是都接触的比较多。但是就是有一些深度的话，可能需要再往里钻一钻，因为有很多东西的技术的这个深度点其实是共通的，比如刚刚我问你的这个red is里面的哈希槽，包括一致性哈希，这里面都是其实都是分布式的概念，不只是在red is里面。包括这个分布式队列里里面的这个原理，因为它的队列的原理跟你刚刚说。

飞扬

55:17

那个。病发的那个框架里面的这个CAS的这个机制，保证这个线程安全的机制，它有异曲同工之妙，当然它有各有各的长处哈，就是它其实有很多这个。这个原理是要深入进去，你能发现它其实设计的这个思路是非常类似的，只不过是在不同的这个框架或者不同的这个场景里面用到了不同的特点。

。

55:46

好好好的，那以后的话就是。

飞扬

55:47

对对对。

。

55:50

就是尽量尽量就是把把那个技术某一个技术，他比较底层核心的东西先给搞明白。

飞扬

56:00

对对对，因为你，你只要对一两种这个框架研究的很深入以后，其实很多原理都能通。

。

56:12

谢谢您老师。

飞扬

56:13

没事儿没事儿。那那那今天先这样。