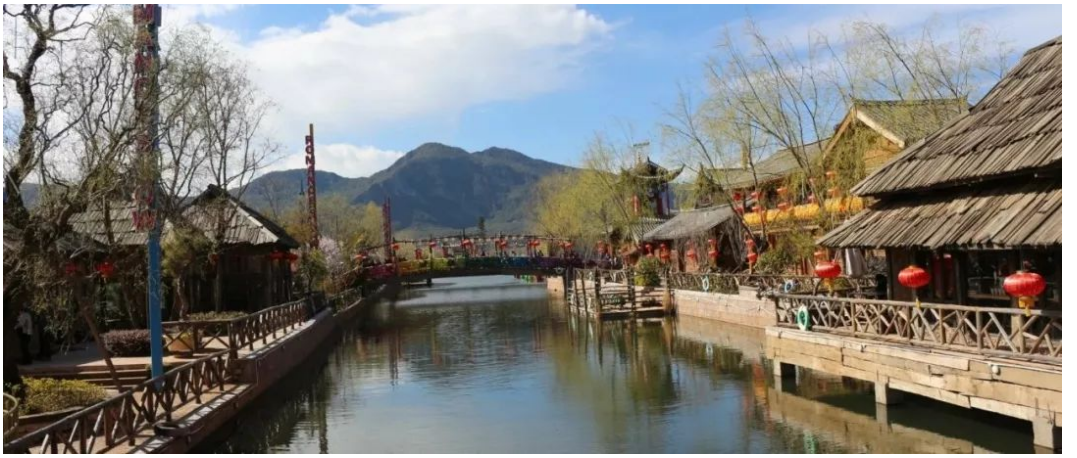
https://mp.weixin.qq.com/s/xr479YxQ14KeoxL0KdptXA

记录一次成都阿里一面的经历

乔二爷 纯洁的微笑 2019-04-24



上周在拉勾上收到一个蚂蚁金服的大哥要我的简历，当时很惊讶，居然有蚂蚁金服的找到我，然后想都没想就给了。

受宠若惊呀，我知道自己的水平跟阿里的差距有多远，以前一直没用勇气去投，连试试都不敢。这次居然主动找过来了，当时就再想，难道阿里这么缺人么？还是只是为了完成某些KPI，当然了，我这种想法比较幼稚。

没想到的是第二天居然收到了阿里巴巴的面试邀请邮件，里面说到会在10个工作日内进行第一次面试。

不管怎么样吧，既然面试来了，就试试吧。能面一次这种级别的技术公司，看看自己真实的差距，也是三生有幸了。从离职的这段时间也补了补JVM，基础的数据结构算法什么的，还有一些高频的Java 基础问题。

周一晚上 8：25 接到了蚂蚁大哥来的电话，果然，跟传说中的一样，还在上班。当时很意外，没想到是电话面试，因为邮件中没提到，后面回想，貌似一面基本上都是电话面试，有些朋友二面也是电面。

进入正题，下面是整个内容。顺序有些不一致，我按照模块来整理的。

自我介绍

自我介绍就先介绍 多大了、毕业多久了、做了什么些东西、最近做的什么内容，擅长的部分呀等等。

这里说一下，自我介绍的内容如实说就好，不要太过于夸大，自我介绍的内容建议大家提前准备好，不要说的时候想到哪儿说到哪儿。

线程部分

1、多线程的实现方式有哪些?

这个题目在一面的时候基本上都会碰到吧，继承 Thread 类、实现Runnable 接口，最后调用的是 start() 方法来启动线程。

这里还有个知识点是 start() 跟 run() 方法的区别和联系。

直接调用 start() 方法，此时线程处于一个就绪（可运行）的状态，但是并没有真正的运行。而是得到CPU 的时间片后，开始执行 run() 方法，run() 方法里面的是我们的线程体。

我们直接 运行 run() 方法，它其实就是一个普通的方法调用，在主线程中执行，**是不会开启多线程的**。

2、描述一些线程死锁的情况?

这个问题在平常项目基本上没怎么接触到，但是我有过部分了解。回答的是：两个线程在持有自己的锁的时候，还要去持有对方持有的锁时，由于别人的锁已经被对方持有，造成彼此等待对方释放锁的情况。回答得比较片面，还有一些类型的死锁问题没有答出来，后面直接交底了，面试官说没关系的。

建议大家在准备这个问题的时候能说出来产生死锁的条件、现象、解决办法等。然后配上一些实例说明，在面试过程中，面试官就提到说根据我们平常遇到死锁问题的场景实例来说。

大家可以搜一下下面这两个死锁场景问题：

1、三个人 三根筷子：每个人需要拿到身边的两根筷子才能开始吃饭

2、银行转账问题：线程 A 从 X 账户向 Y 账户转账，线程 B 从账户 Y 向账户 X 转账，那么就会发生死锁。

3、项目中有没有用过线程池？怎么用的？

回答了我们项目里面有些接口需要组装多个服务的数据进行封装，然后返回。这里面我们会使用多线程去并行拉取数据，减少接口响应时间。

面试官说：“ok，那么你有没有看过线程池里面的源码呢？有哪几种线程池？”

源码这里我迟疑了一下，我说不太熟，然后我说了几种类型的线程池 **newSingleThreadExecutor**、**newFixedThreadPool**、**newCachedThreadPool** 但是还漏了一种 **newScheduledThreadPool** 没想起来。

4、线程池的原理是什么样子？底层方法的参数分别是什么意思？

回答这个问题的时候，当时我卡住了。我知道这几个底层都是对 调用的 **ThreadPoolExecutor**，但是我死活没有想起来名字，这时候面试官提醒了一下，然后说没关系的。

接着就问：“那你知道他的参数都有哪些吗？都分别代表什么意思吗？”

我回答的是 有个 线程的个数 和 线程存活的时间，其他的没说上来。然后面试官说：“没关系的”。

补充一下：线程池底层都是通过 ThreadPoolExecutor 来实现的。

```
public ThreadPoolExecutor( int corePoolSize,
                           int maximumPoolSize,
                           long keepAliveTime,
                           TimeUnit unit,
                           BlockingQueue<Runnable> workQueue,
                           ThreadFactory threadFactory,
                           RejectedExecutionHandler handler)
```

几个参数的意思分别为：

- corePoolSize：线程池里最小线程数
- maximumPoolSize：线程池里最大线程数量，超过最大线程时候会使用 RejectedExecutionHandler
- keepAliveTime：线程最大的存活时间，超过这个时间就会被回收
- unit：线程最大的存活时间的单位
- workQueue：缓存需要执行的异步任务的队列
- threadFactory：新建线程工厂
- handler：拒绝策略，表示当workQueue已满，且池中的线程数达到maximumPoolSize时，线程池拒绝添

> 加新任务时采取的策略。DiscardPolicy：抛弃当前任务，DiscardOldestPolicy：扔掉最旧的，CallerRunsPolicy：由向线程池提交任务的线程来执行该任务，AbortPolicy：抛出RejectedExecutionException 异常。

问到这里，我回答的确实太有限，面试官就没有再细问了，还是说：“没关系的”。

如果你这里答出来了，那么我认为你还需要掌握的是，**这几种线程池在哪些情况下使用什么类型的，以及要注意什么问题，很大可能面试官会继续深挖。**

这里就不给出答案了，我相信你自己去搜一下，体会会更深刻些。

MyBatis 部分

5、mybatis 的 $ 与 # 的区别?

回答：他们两都可以来传递参数，不过 # 可以方式 sql 注入，而 $ 就是字符串拼接的方式处理，可能会有sql 注入的问题。

上面还有一个关键的点没有答出来，那就是 #{} 在预处理时，会把参数部分用一个占位符 ? 代替，变成了如下的 sql 语句：

```
select * from user where name = ?;
```

而 ${} 则只是简单的字符串拼接，在动态解析阶段就直接拼接成了 最终的sql 语句：

```
select * from user where name = 'zhouq';
```

6、$ 跟 # 的使用场景 ?

这个问题我没有怎么理解得到，然后回答的就是 $ 在拼接表名的时候用，其他时候传递参数值的时候用 #。

7、mybatis 的 dao 接口跟 xml 文件里面的sql 是如何建立关系的?

这里问到的时候比较蒙圈，然后回答的是：mybatis 会先解析这些xml 文件，xml 文件里面有命名空间 （namespace），这里可以跟dao 建立关系，然后 xml 中的每段 sql 会有一个id 跟 dao 中的接口进行关联。。。

然后面试官说: "如果 我有两个这个xml 文件 都跟这个dao 建立关系了，那不是就是冲突了？",然后，我认怂了。

我上面的回答太笼统，肯定是有问题的，建议你好好去了解一下mybatis 的原理。

mybatis 到这里就没了。

数据库

先问的是，你平常使用得做多的是什么数据库，当然了，mysql 。

8、mysql 锁机制 ？

面试官问的是，你了解mysql 的锁机制么？我就只答出来一个行锁。然后其他的没想起，就认了，其他的忘记了。

建议你去了解了解还有表锁、页面锁 等等。

9、排它锁 & 共享锁你了解吗 ？

这个地方我想了一会，说平时了解得不多。实时上，平常我们的小业务系统基本上没有用到这些，可能有用到的地方，也没有去在意吧。

接着，面试官说了下面这个场景题，然后让出解决方案。

10、场景问题：在A线程处理一条数据，比如扣款，或者是更新状态时候，其他的线程比如 B 需要对它进行阻塞，不能够再对这条数据进行操作，包括查询也不行，得等A 线程处理完成以后，B才能进行处理。A 跟 B 是同样的业务代码产生的，非不同的业务。要使用数据库的锁来实现，怎么实现 ？

问这个问题的时候，面试官很耐心的解释了这个场景，然后问我有没有想起点什么来？其实就是想考察上面的关于数据库锁的问题。

11、mysql 索引是怎么实现的?

回答的是 B+ 树，接着面试官继续问：“能否大致描述一下 B+ 树的大致结构 ？”。这块内容没怎么了解，直接认怂了。

缓存相关

这块内容是我项目上写得有使用了多级缓存的方案，然后面试官就这一块问了下面的这些关于使用缓存可能会遇到的问题。

12、缓存击穿、缓存穿透 、缓存雪崩 ？

13、热点数据失效怎么解决?

这两个问题，以前好好了解过，但是没整理成自己的东西，面试的时候也说得云里雾里。

14、先删缓存还是先更新数据库，为什么?

这里我说的：**是先删缓存，然后再更新数据库**，**但这是错误的**，这里有非常大的问题。

想想这样一个场景：

如果一个线程 A 先把缓存删除了,然后去更新数据库，那么在它删了缓存还没有更新到数据库的这个中间时间，线程B进来了，发现缓存没有，就去读库，这时候还是读取还是旧的数据，然后又更新到缓存去了。此时A 才把新数据写到数据库。

此时就产生了一个典型的问题就是“**双写不一致**”。

关于这块问题的讨论：《缓存更新的套路》-陈皓老师

kafka

15、kafka 的架构，包含了哪些角色?

这个问题我开始不知道怎么回答，就说了个 Broker，然后面试官提醒了一下：“不是我们平常还有生产者，消费呀什么的吗？”额，我说还有生产者、消费者、主题呀等等。

这过程中面试官还提到说平常我们在搭建的时候要配置写什么东西呀等等，按照官网的介绍说也行。

这里还有其他的比如Partition、消费者组、还有一个主要的 就是 zk 了。

这里建议大家好好的把 kafka 里面的这些概念、属于、架构图好好自己画一下。不然真是关键时候真说不出来，但是他一提你又明白。这样子肯定是不行的，面试是你说，不是面试官说。

16、kafka 的最小工作单元?

这个问题问得也是蒙圈，其实就是说我们在写代码的时候，要用kafka的时候。我们需要使用那些最基础的组件，比如生产者、消费者、主题、偏移量 等等。

这个问题如果你们遇到，最好向面试官问清楚。

17、kafka 消息重复消费的问题？幂等怎么做的？

刚开始面试官说，你知道kafka 消息重复的问题吗？有没遇到。

我回答的是，会存在消息重复消费的问题。我们在消费数据这端做了幂等处理来解决。

然后面试官继续才问的是：幂等怎么来做的， 我说通过设置数据版本号，还有数据库唯一索引等等。

面试官：“ok”。

这个问题，如果你能告诉面试官产生重复消费的情况，比如说投递的时候重复了，消费的时候由于 offset 没处理好等等问题导致的话，我想可能会更好。

18、kafka ack 机制？集群中的ack 是怎么实现的？

这里我只回答上了 ack 机制是啥，但是实现原理没有回答上来。

Redis

19、Redis 中有哪些数据结构

平常使用得最多的是 String，还有 List 、Hash、Set、ZSet 。

没有再问其他的内容。

但是像Redis 为什么这么快这种问题，我认为你应该要去了解，其他小伙伴经常遇到。也就是多路复用是个什么玩意儿？

源码

20、这里面试官问 你平常有没有看过一些源码？框架的也行？JDK 的也行。

然后我说看了 HashMap 的源码,然后就巴拉巴拉的说了一哈大体的 put、get 流程 ， 它的结构是什么样子的。

这过程中还问到了 怎么判断两个对象是否相等？也就是 == 和 equals 的知识点。

其他的就没再继续问了。到这里整个电面过程结束了，说10个工作日内会给我答复此次面试情况。整个过程大概持续了40分钟的样子。

我知道，凉凉。

最后总结一下

上面的模块虽然顺序有变化，但是每个大块里面的问题都是按照顺序来的，基本上都是由浅入深、循序渐进的来问。

像数据库锁、线程池、缓存问题 **这些内容几乎都是那种连环炮的形式，直到摸到你的底。**

通过这次面试，亲身体会到了差距。不过，更有方向了。

告诫大家一点东西：

- 平常多积累输出：输出或者教会别人是最好的学习方式，光看不练，几天就忘。
- 先深后广：深入学习，而不是只停留在使用API 的层面，一块一块系统的深入了解，再去搞其他的。
- 建立知识体系：把自己学习的内容形成博客也好，什么导图也罢，记得把这些零散的内容，整理成自己的知识。
- 别抱有侥幸心理：别裸面，如果自己有整理的还是多看一下，多准备准备。大厂的面试会挖到你最深的部分，不要觉得只背一些面试题就是 ok的，题是背不完的。临时抱佛脚基本上过不了关。如果你是靠背面试题进去的，那么你厉害，佩服。
- 隔一段时间就去面试吧：不要学我，待一家公司三年多，中途都没有出去面过，出去面面才知道，哪些是需要去补充的。

有些问题可能答案这些不是太全面，需要你自己去动手。

希望这篇文章对你有帮助，哪怕只有一个点，都是值得的。如果其中有一些点你不了解，那么你是时候要去补充了。

别在自己的舒适区待太久，不然出不来。出来混，迟早是要还的！

作者介绍：乔二爷，在成都乔二爷这个名字是之前身边的同事给取的，也不知道为啥。也习惯了他们这样叫我。

一直待在相对传统一点的企业，有四年半的 Java 开发经验，会点大数据的内容，也跟客户打过一年的交道，还带过 10个月 10人＋的技术团队，有一定的协调组织能力，能够理解 boss 的工作内容，也能很好的配合别人做事。

作者赞赏码

Java 极客技术公众号，是由一群热爱 Java 开发的技术人组建成立，专注分享原创、高质量的 Java 文章。如果您觉得我们的文章还不错，请帮忙赞赏、在看、转发支持，鼓励我们分享出更好的文章。

阅读全文