# B简历：

# -------------------------------------------

### 基本信息

|  |  |  |  |
| --- | --- | --- | --- |
| 基本情况 | | | |
| **姓名** | 刘文尚 | 性别 | 男 |
| **出生日期** | 1995-10-14 | 属相 | 猪 |
| **现住地** | 丰台区 东营里 2号院 |  |  |
| 大学情况 | | | |
| **学校** | 西安欧亚学院 | 专业 | 软件工程 |
| **入学时间** | 2013-09 | 毕业时间 | 2017-07 |
| **专业课** | [程序设计语言](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E8%AF%AD%E8%A8%80/2317999" \t "https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B/_blank)、[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93/103728" \t "https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B/_blank)、[软件开发工具](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E5%BC%80%E5%8F%91%E5%B7%A5%E5%85%B7/4605523" \t "https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B/_blank)、[系统平台](https://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E5%B9%B3%E5%8F%B0/3766067" \t "https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B/_blank) | | |
| **学历** | **本科** | 学位 | **学士学位** |
| **学校地址** | 陕西省[西安市雁塔区东仪路8号](https://ditu.so.com/?k=%E8%A5%BF%E5%AE%89%E5%B8%82%E9%9B%81%E5%A1%94%E5%8C%BA%E4%B8%9C%E4%BB%AA%E8%B7%AF8%E5%8F%B7&src=baike_address" \t "https://baike.so.com/doc/_blank) | 校长 | 胡建波 |
| **去学校的方式：** | 1. 从**西安北站**出发 2. 步行约780米至北客站 3. 乘坐[地铁2号线](http://poi.elong.com/traffic/%E8%A5%BF%E5%AE%89/metro/%E5%9C%B0%E9%93%812%E5%8F%B7%E7%BA%BF/)至三爻站下车(共17站) 4. 从A口出,步行约370米到三森国际家居站 5. 乘坐[525路](http://poi.elong.com/traffic/%E8%A5%BF%E5%AE%89/bus/525%E8%B7%AF/)至欧亚学院站下车(共4站) 6. 下车后,步行约140米   抵达欧亚学院 | | |
| **上小学日期：** | **2001年9月 6岁六年** | | |
| **上初中日期：** | **2007年9月 12岁三年** | | |
| **上高中日期：** | **2010年9月 15岁三年** | | |
| **上大学日期：** | **2013年9月 18岁四年** | | |
| **毕业时间：** | **2017年 7月 22岁** | | |
| **在校学生：** | **5000左右** | | |
| **有教师：** | **不太清楚** | | |
|  |  | | |

|  |  |
| --- | --- |
| 第一家公司 ： | 西安和仁科技 |
| **地址：** | 西安市 雁塔区 宝德·云谷国际（只说大厦名称） |
| **入职时间：** | **2017年9月初** |
| **离职时间：** | **2019年1月** |
| **公司性质：** | **民营** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **杜晓虎** |
| **薪资方面：** | **刚刚毕业转正以后工资是6K左右，过完年大概三月份涨到7K左右** |
| **路线：** | 住在公司附近 步行几分钟就到了 |
|  |  |

|  |  |
| --- | --- |
| **第二家公司情况：** | **西安鲸鱼互动信息技术有限公司(**只说公司简称) |
| **公司地址**： | **陕西省西安市高新区科技路创业广场** |
| **公司规模：** | **30人左右，我们项目组10人，一个项目经理，4名开发人员，2名前端，1个测试，2个运维** |
| **公司性质：** | **民营** |
| **公司行业：** | [技术咨询](https://b2b.11467.com/search/-6280672f54a88be2.htm" \o "技术咨询公司) , 技术服务 , 计算机系统服务 , 基础软件服务 , 应用软件服务 |
| **去公司路线：** | **住在公司附近 步行十分钟左右就到了** |
| **入职时间：** | **2019年3月** |
| **离职时间：** | **2021年3月底** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **于少鹏** |
| **税前：** | **15K** |
| **税后：** | **14K** |
| **缴税：** | **1K左右(具体原因不清楚当时公司就是这样扣的税)** |
| **期望薪资：** | **17K** |
| **薪资的构成:** | **岗位工资(5K)+绩效工资(10K)** |
|  |  |

**面试常见问题：**

|  |
| --- |
| **你为什么从上家公司离职？** |
| **上家公司待的时间长了，公司比较照顾老员工，让后把我分配去维护之前的项目，让后工作太轻松，环境太安逸，**  **然后就换个公司，想换个新的环境挑战一下自己！** |
| **你交社保了吗？为啥没交？** |
| **没交**  **在上家公司的时候，人事说咱们公司的五险一金如果要上的话都是从自己工资里面扣的，当时感觉没啥必要也就没上。** |
| |  | | --- | | **三个词形容自己**： 交流表达能力强，团队协作能力强，自学能力强 (**可以综合自己自学的知识表达**) | | **你的优缺点是啥？** |   **优点:**  **抗压自学能力强（有一次项目就要上线了，连着加班了一周，每天都到夜里很晚，但我最后也是成功完成任务。），**  **喜欢钻研技术，**  **自学能力强** (  **1.从官网上下载需要的文件。**  **2.导入相关的文件到项目中。**  **3.在需要用到的文件中引入具体的js，css。具体在做例子的时候先做静态例子，再和后台结合做出来动态数据的例子。特别在做动态数据例子的时候，我们可以通过结合官网上提供的demo,通过f12观察这个几个问题:**  **<1>js插件怎么向后台发送请求；**  **<2>传递了哪些参数以及参数的含义;**  **<3>后台怎么接受参数;**  **<4>后台接受到参数后怎么处理,才是js组件所需的响应格式;**  **<5>如何将json数据响应给前台;**  **<6>前台如何接受到该json数据；**  **<7>接受到数据之后如何处理，才能将数据展现出来。**  **按照这个思路和流程，我们在学习第三方的js组件时，就能够快速的掌握并融入到项目中。**  **)**  **缺点: 研究技术时候，想把这个技术搞的很清楚，结果才开始的时候影响了项目开发的进度，我也意识到这个问题，所以后来对于我比较感兴趣的技术，我会利用自己的业余时间去钻研它，在工作时以项目任务为主。** |
| **你知道五险一金都是啥吗？** |
| **养老保险、医疗保险、失业保险、工伤保险、生育保险，和住房公积金。** |
| **你能接受加班吗** |
| 遇到这个问题，有些人毫不犹豫说“能”，好像这么回答会证明自己能吃苦、不怕累。如果问“能以公司为家吗”，估计也会有人说“能”。我表示服气！  你应该先问一下公司加班的时长和频率，如果工作需要的合理加班，当然可以接受。如果公司以加班为荣，为了加班而加班，你为啥要接受？  当然，我承认，现在互联网公司普遍加班。如果这家公司你非常想去，这家也有加班文化，看在钱、或者镀金的份上，从就从了吧。 |
| **你能接受出差吗** |
| **能** |
| 你期望薪资多少？你上家工资多少？税后拿到手的有多少？扣了多少钱的税？ |
| 期望薪资 16K 上家工资15K 税后14K左右 扣了1K左右的税 |
| 你的五年规划是啥？ |
| **近期的话我就是想先让自己的技术在得到以进一步的提升，然后对公司的业务也要尽快掌握，然后向项目经理方面发展。** |
| 是统招吗 是是本科吗 是能在学信网查到吗 我当时上的是民办大学，能在民教网查到！ |
| 你们公司还有什么福利？多少薪： 13薪  **车补**  300左右  **项目奖金**  7k-12k，做完项目，收完尾款的时候。  **年终奖**  这个就是1个月的工资，  一般是过完年之后发  **电话补助**  200  **饭/餐补**  30/天  **电脑补**  400 |

**公司内其他部门：**

**技术部（开发部/研发部），行政部，销售部，财务部**

**---------------------------------------------------------------------**

**其他问题：**

**笔试问题：我工作这么多长的时间了，理论上的知识忘得差不多了，你和你们经理联系一下直接面试吧，不行的话那么我就去直接去下一家！**

**-----------------------------------------------------------------------------------------------------------------**

坐稳再开口！！ 主动性！！ 开门见山！！大白话！！三秒钟原则！！学会反问！！

**Hr人事，leader领导，tean团队，pm项目/产品经理，cto首席技术执行官，coo首席运营官，CEO董事长、**

## 主动说：

**你好！**我叫刘文尚，今天来应聘咱们公司 java 开发！

从事java开发行业至今有四年了，

我最近使用 **Spring Boot 来做微服务项目**

之前做项目的时候 负责一些 **常规模块、性能方面的、安全方面**的 等等

**基于token的登陆:**

针对于安全这块我当时做了一个**基于token的登陆**，

在我之前做的项目中我负责了前台的会员注册，会员登录以及后台的会员管理，当时项目前台这块采用的是前后端分离的方式进行开发的，我负责的是后台接口的编写。

在之前我用的是基于session的登录方式，是在用户登录成功后将用户信息存入到session中，这样不利于程序的横向扩展， 并且随着登录用户的增加，会不断的占用服务端的内存资源；而现在这种基于token的登录方式，是在登录成功 后将用户信息存入到客户端的中，不会额外占用服务端的内存资源，并且通过签名和验签可以保证数据不被篡改。

在之后的会员登录这块我就采用了基于Token的登录方式。说白了就是用户在前台输入用户名密码后通过ajax发送请求到后台接口，后台接口在进行层层验证后，如果验证成功则直接响应相关的信息给前台，前台在ajax的成功回调函数里面接受到数据，将其暂时存入cookie中，之后在后续发送请求时，再从cookie中取出数据将其写到自定义的header中，我当时自定义的header叫(x-auth/auth-data/auth-token/x-token),以后数据就通过header传递给后台接口的拦截器进行各种验证，通过验证后才能访问登录后的资源。

具体在做的时候需要解决几个问题。

首先是ajax跨域请求的问题，因为采用了前后端分离的开发方式，所以前台页面和后台接口在两个不同的项目中，因为端口号和ip地址不同所以就涉及到了跨域问题，咱们都知道ajax是不允许跨域的，解决这个跨域问题可以通过jsonp,也可以通过cors,因为jsonp发送的请求只能是get请求，所以我们项目中当时采用了cors这种解决方案，这样就可以发送各种请求方式的请求，像get,post,delete,put这些。

具体在项目中做的时候，要么可以通过spring给咱们提供的一个叫 @CrossOrigin的注解来实现，要么可以自己写一个CorsFilter来实现，当时的项目中因为涉及到所有的请求都会先经过springMVC的拦截器进行验证，所以我就将跨域处理的相关代码写到了拦截器里面，这样也就不需要在每个Controller中添加 @CrossOrigin这个注解了。

具体在做的时候是通过调用response的setHeader方法添加几个头信息来允许ajax的跨域请求，具体记得不是太清楚了， 好像叫allow\_methods,allow\_headers还有allow\_origin;其中allow\_methods指明能够接受的跨域的请求类型，比如 put,get,post,delete,options这些，allow\_headers指的是允许接受的自定义的头信息，allow\_origin表示允许 跨域的远程ip地址和端口号。

再者ajax跨域的时候要传递头信息，我们为了让每个ajax发送请求时候都能自动加上这个自定义头信息进行传递，我就写一个公共的js文件，并且在这个js文件中通过了$.ajaxSetup来定义全局的ajax设置，并且通过beforeSend 函数来在所有ajax发送请求前，通过xhr.setRequestHeader传递自定义的头。这样在每个前台页面中引入这个公共的js文件就可以了。

再者就是后台登录接口的处理逻辑，先进行各种非空判断以及用户名和密码的验证，如果都成功则将会员id, 会员名，以及一个uuid组成json格式的字符串，并通过base64进行编码；之后将base64编码后的用户信息结合服务端定义的秘钥用md5生成签名，目的是为了保证数据不被篡改，因为秘钥在服务端；之后再将会员名和uuid组合起来作为key,设置指定的过期时间，将其存入到redis中，这样做是为了后期在拦截器中进行过期判断。最后将前面用户信息的base64编码的内容和签名信息的base64编码内容通过.作为分隔符连接起来，响应给 前台，前台就可以在登录的回调函数中将这个信息暂时存入cookie中，方便后期取出cookie中的数据放到ajax的请求头中。

最后最重要的就是后端接口中拦截器的处理逻辑。

1.跨域处理

2.options特殊请求的处理

3.对于不需要拦截的请求直接放行

4.判断头信息是否完整

5.验签

6.判断是否过期

7.将会员信息存入request作用域，方便在其他控制层获取会员信息

跨域处理之前已经说过了。

关于options特殊请求是这个样子的。之所以会发送options请求，是因为我们前台客户端在发送ajax请求的时候传递了自定义的头信息，这就会导致发送真实的请求前先发送一个options请求，但这个options的请求不会传递 任何实质的数据，所以在后台拦截器处理的时候我们可以通过request.getMethod来判断下请求方式是否是options,如果是就不放行，也就是return false,处理完后前台就会自动发送真实的请求过来。

在项目中有些请求是不需要登录就能访问的，有些请求是登录后才能访问的，为了对处理这类请求，我写了一个自定义注解，然后把注解添加到登录后才能访问的方法上，这样在拦截器中，我就可以通过反射获取方法上是否有这个自定义注解，如果没有证明不需要进行拦截验证，就可以直接return true;放行。

接下来我会判断前端传过来的请求中是否包含自定的头信息，如果没有则抛出异常，告诉前端头信息不完整，因为在项目中我还通过springMVC的 @ControllerAdvice 注解定义了一个全局的异常统一处理器，所以当抛出异常后会被这个全局异常处理器拦截到，将其转换为json格式的数据包含状态码和错误消息传递给前端，这样前端就可以根据这个进行判断，进行相关的处理和信息提示。

如果有头信息就到了最关键的验签过程，目的 就是防止内容被篡改，具体的逻辑是这样处理的，首先将传过来的头信息根据.进行切割，切割后就取到了经过base64编码的用户信息和经过base64编码的签名，之后用经过base64编码的用户信息和服务端的秘钥经过md5得到签名，再对base64编码的签名进行解码，然后两个签名进行对比如果一致，则证明数据没被篡改，否则抛出异常提示数据被篡改。

验签后就要判断存储在redis中的key是否过期，这个比较简单，就是将base64编码的用户信息进行解码得到 json格式的字符串，并通过阿里巴巴提供的fastjson这个第三方jar包，将json格式的字符串转换为对应的 会员对象，这样就能从中获取到会员的相关信息包含会员名，会员id，以及uuid这个标识，紧接着根据 会员名和uuid组合key，调用redis工具类判断该key是否还存在，如果不存在则抛出异常提示数据过期了。

如果存在就会给对应的key进行续命，其实也特别简单就是调用redis工具类中的expire方法就行了。这个续命说白了有点类似于session的过期时间处理，比如session的过期时间是30分钟，如果到最后一分钟了 又有客户端请求发送过来则会重置为30分钟。

经过这些层层判断后就可以将会员信息存入到request作用域中，这样做是为了方便在其他控制层的方法中获取会员的相关数据，最后通过return true进行 放行 就行了。

**日志记录：**

在项目中我做过后台管理系统的日志管理模块，日志管理模块的作用说白了就是记录用户的操作，这样就知道谁在什么时候干了什么事情。

之前做其他项目的时候，日志处理模块通常都是在控制层结合log4j进行日志的 控制台打印以及日志文件的存储，除此之外还会把日志信息插入到mysql数据库中存储起来，方便查看。

但这样做会有个问题，就是需要在每个Controller类的每个方法中都写上相关的日志记录代码，这样就会出现大量的代码重复，以后维护起来也特别麻烦。

我通过和我们团队的人讨论，最终决定通过自定义注解+aop来完成这个改进。

在写自定义日志注解时候，通过 @Target 设置为Method指明该注解只能用在方法上面，通过将 @Retention设置为RunTime指明将注解保留至运行时。这样就可以通过反射去获取注解信息。

在注解中声明了一个String类型的value来让开发人员手工设置日志的信息，之所以采用value，是因为value这个字段有特殊的含义，它可以在使用自定义注解给日志信息赋值的时候省略不写，用起来更加方便。之后就可以在Controller中的方法上加入自定义注解并且对value进行日志信息的赋值，如 @Log("增加商品")。

最后在AOP的日志切面类中通过获取方法签名得到Method，通过Method的isAnnotationPresent判断该方法上面是否加入了自定义日志注解，如果 是 则再通过Method的getAnnotation来获取自定义的日志注解，最后再通过.value()方法获取自定义注解中日志信息的值，这样在记录日志的时候就可以显示更加人性化的信息。

所以说后面在做日志管理模块的时候，我就考虑到可以使用AOP做统一日志的处理，这样就可以让我们程序员在工作的时候把精力花在核心业务代码的处理上。

具体在做的时候我是这样写的。首先写一个日志切面类，在类上添加两个注解，分别是@Component @Aspect

，@Component：将本类交给ioc管理；

@Aspect：声明这个是切面类。

在这个类中我会自定义一个横切逻辑，但这个方法中需要特别注意几点，因为当时在项目中我使用的是环绕通知，所以这个方法返回一个Object类型的值，在方法上添加参数JoinPoint，使用它，来获取方法的签名，再根据方法签名来查找到所有的方法。

这时候我们通过getDeclaredAnnotations 获取方法上面的注解，判断方法上是否有注解如果有注解的话就进行下面的操作，把数据保存到我们mysql数据库里面

**.订单**

订单这个模块我在之前的项目中也做过，这个模块相对比较复杂，考虑的事情也比较多。像表设计、订单接口幂等性的问题，商品超卖的问题，包括订单也都是用户登录后才进行的操作等。

项目中很多接口是用户登录后才能进行操作的，所以我们采用的是基于token的用户登录，并且自定义了一个注解，这样在需要登录的控制层的方法上直接加上这个自定义注解配合。自定义的拦截器，就可以达到先登录才能访问接口的目的，使用起来非常方便！！！

先说下订单表的设计吧，订单这个模块中包括 订单表和订单明细表，像订单表我当时设计的主要有这些字段：

订单id、 用户id、 订单支付方式、 订单的总金额、 订单中商品的总件数、 订单的创建时间、 订单的支付时间、 订单交易关闭的时间、 订单发货时间、 订单交易成功的时间、 订单完成评价的时间、

订单的状态 【未支付10  已支付20   已发货30   交易成功40     交易关闭50  已完成评价60】、

订单状态描述、 收件人手机号、 收件人地址、 邮编、 收件人名字、 邮费、 是否需要发票。

在这里面 订单id   作为主键，它的生成方法是年月日时分加上雪花算法生成的标识，这样既能从一个订单id中看到这个订单是啥时候生成的也能通过雪花算法保证整个订单编号的唯一性。

用户id就是当前登录的会员id，是为了指明这个订单是哪个会员的订单。

订单的支付方式，是考虑到我们的系统要支持好几种支付途径，比如1代表微信支付，2代表支付宝支付。

订单的总金额，总件数这个没啥要特别说的，这个就是当时购物车中添加的商品总价格及其总件数，但如果添加到购物车的商品在提交订单进行购买时发现库存不足，我当时的做法就是找到库存不足的商品，弹出提示框告诉用户，并在订单的总金额和总件数中要减去这些库存不足的商品，当然这些库存不足的商品也就不再插入订单明细表了。

订单的创建时间这个字段比较重要，因为在用户下单后可能不及时支付订单，那我们的系统就是在1个小时后会自动取消订单。所以就可以用当前时间减去订单创建时间，如果大于1小时就将订单状态设置为交易关闭，并在订单状态描述中记录“订单支付超时取消订单”的描述。

除此之外还有订单的支付时间，交易关闭的时间，交易成功的时间，发货时间，完成评价的时间。

订单的状态我当时设置的有

         未支付  已支付   已发货   交易成功   交易关闭 已完成评价

取消订单的时候会进入交易关闭状态，可以是人工取消订单也可以是系统在1个小时后自动取消订单。

当用户收到货物后点击确认收货，会进入到交易成功状态，如果用户没有点击确认收货，那么是在发货时间15天后自动将订单变为交易成功状态，只有交易成功后才能对商品发表评论。

还有一些收件人的信息 包括手机号，地址，邮编，名字等等，这样做是通过这些个冗余字段，避免为了获取这些信息还要进行多表联查，因为多表联查会降低性能而通过冗余字段就可以避免多表联查，无形中也就提升了性能。

订单明细表:

订单id、 商品id、用户id、 商品名、 商品价格、 商品个数、 商品小计、 商品图片。

我当时采用 订单id和商品id 作为联合主键，说白了也就是用多个字段来保证一条数据唯一性。

除此之外还融入了用户id这个字段，目的也是为了通过该冗余字段就能知道某个用户购买的商品明细，而不需要再连接订单表，避免多表联查。

表设计完之后就是正常的写接口，订单接口要考虑重复下单的问题，超卖的问题，以及把redis中存储的购物车数据取出来，插入订单明细表，订单表以及支付日志表。

关于重复下单这块，说白了就是要考虑接口的幂等性。幂等性我是这样理解的，就是对接口调用一次和调用多次，产生的结果都是一样的。像查询方法，就天生具有幂等性的特点。

但下订单属于添加操作，对于添加操作本身不具备幂等性的特点，所以需要我们特别处理下，如果不处理就会出现因为网络抖动或者服务器重发请求，造成虽然用户只操作了一次，但是可能最终发送了多次添加订单的请求，这样就会让用户感觉到很奇怪，明明我就下了一个订单，怎么会出现两个一模一样的订单呢。

对于接口的幂等性我是基于redis+token的方式解决的。具体的思路是这样的。

首先我会写一个创建token的接口，这个接口的代码也特别简单，就是生成一个uuid作为token，然后响应给客户端，并且将该token存储到redis中，而且设置它的过期时间为10分钟，这样在客户端中就可以调用这个接口生成一个唯一的token标识,并在后续的请求中将该token作为header中的信息传入到接口的服务端进行相关验证。

接着我写了一个自定义的注解，这个注解没有什么实质性的内容，说白了就是一个标识，只要是加上了这个注解的方法，都会在我自定义的拦截器中进行幂等性的处理。

幂等性拦截器中的判断是最核心的。首先我会判断请求中是否包含token头信息，如果没有就响应给客户端一个提示，头信息丢失。如果存在则获取该token的内容，调用redis中的exist方法来判断在redis中是否存在该token，如果不存在则提示token不存在。最后就是调用redis的delete方法，将该token作为key进行删除操作，如果返回的值等于0,则证明已经有其他请求发送过来了，该请求就不在是第一个发送过来的请求，就提示请求重复，否则证明是第一个请求，放行即可。

之后我会在下订单的方法上加入自定义的幂等性注解，就解决了重复下单的问题。

接下来解决的是订单超卖的问题。

下订单的业务逻辑是这样处理的，首先从redis中取出用户的购物车数据，如果为空，则响应给客户端一个用户购物车为空的提示信息，如果不为空则取出用户对应的购物车的json数据，将其转换为java对象。

首先通过年月日时分的时间戳加上雪花算法生成一个订单的唯一标识，然后从购物车对应的java对象中取出购物车中的商品列表，这个商品列表其实对应的就是订单明细，循环遍历商品列表，减库存，如果减库存成功，则将该商品插入订单明细表，如果不成功就不需要插入订单明细表了，而且还需要从购物车中剔除该商品，更新购物车的总价格和总件数，并将该库存不足的商品加入一个列表，用于响应给前端进行提示。

将更新后的购物车信息中的总价格和商品总件数插入到订单表中，并设置创建时间为当前时间，当前订单状态为 未支付。

最后插入支付日志表，这个表记录的是支付时候的相关信息，比如outTradeNo,payMoney等，

将支付日志表中的订单id设置为刚才插入的订单id,将支付状态设置为 未支付，并将该支付日志插入到redis缓存中，方便后续在支付的时候直接从缓存中取出来进行使用。紧接着就清空该用户对应的redis中的购物车数据。因为当时购物车数据是以hash的方式存储的，所以通过hdel进行删除就行了。

在这个过程中要注意，

如果整个购物车中的商品都库存不足，则直接提示前端，下单失败，库存不足；也就不需要再插入订单表和支付日志表了。

我当时采用的是基于数据库乐观锁的方式进行订单超卖的处理。核心思路就是在update语句中加入库存量是否大于或等于购买量的where条件。比如原来更新数据库中的商品库存我这样写，先查询下该商品对应的库存量，如果库存量大于等于购买量则执行update 商品表 set 库存=库存-购买量 where 商品id=购物中的商品id;否则提示库存不足。

表面看上去没有任何问题，但在遇到多线程高并发的时候，就会出现多个线程同时查询商品对应的库存量，都符合条件，所以多个线程都会依次进行update语句，最后就造成商品可能为负数，出现商品超卖的情况。

我是通过将update语句改造为 update 商品表 set 库存=库存-购买量 where 商品id=购物中的商品id and 库存量>=购买量，然后根据update的返回值，如果返回值为0，则证明没有执行该sql语句，证明库存已经不足了。这样即便在多线程的时候也不会出现超卖问题。

**幂等性：**

在订单的⽣成的模块中我们考虑了⾼并发下访问⽣成订单接⼝的幂等性。为了保证保证接⼝的幂等性，⾸先我们写了个⾃定义注解及⼀个保证幂等性的AOP拦截器。还有⼀个⽣成⼀个标志位的⽅法。当⽤⼾点击结算按钮从结算⻚⾯跳转到新增订单⻚⾯时会触发该⽅法，该⽅法内通过UUID⽣成⼀个32位的flag作为key存储到Redis中，然后将flag传送到前端⻚⾯，随着结算⻚⾯跳转将该flag转发到提交⽣成订单的⻚⾯。当⽤⼾点击提交订单时，会的带着商品id的集合与这个flag标志进⼊后台⽬标⽅法。在进⼊⽬标⽅法之前会被保证幂等性的AOP拦截，AOP⽅法内判断是否有该flag，如果flag则查询redis是否有flag作为key的数据，且进⾏删除，如果删除该条数据的返回值为true则说明第⼀次进⼊该⽅法，则放过该条请求使其直接进⼊⽬标⽅法。反之则返回值为false则进⾏拦截。从⽽通过删

除redis数据数据是否成功来保证访问接⼝的幂等性。此种⽅式还能保证了⾼并发下的接⼝访问的幂等性。当有多个并发线程进⼊AOP中，由于Redis是单线程的，删除数据时第⼀个线程删除成功，返回成功标志位，其他并发的线程则均删除失败，从⽽根据此情况保证⾼并发下访问接⼝幂等性。

**未支付订单的处理**

我们订单生成后回去扣除库存这个时候就会遇到一个问题，就是用户未支付，也不去处理那我们的订单如何进行库存的回滚。我当时考虑用到的是spring的定时任务这种方式比较简单，就是定时扫描数据库获取未支付并且已经超时的订单，然后将状态修改成已取消。这种方法确定就是消耗系统资源，会有很多无用的操作执行。后来也是在之前的开发中用过RabbitMQ消息队列，它里边有延时队列的支持，就在我们项目中使用这种方式。首先我们在docker中安装RabbitMQ,并且安装延时插件，安装插件是为了在使用时更方便，这些网上的安装步骤都是非常齐全的。我们在项目中的订单模块中，首先创建交换机使用new CustomExchange()，主要参数有交换机名称，type声明是延时队列，并且设置交换机持久化，还要指明交换机和队列的绑定方式时我这用的是direct直连方式，接着就是使用new Queue()声明对面主要参数是队列名称，持久化等。然后将我们的交换机和队列使用BindingBuilder.bind绑定起来这样我们就可以在下订单使用这个队列了。我们订单生成后我们需要调用增加数据到RabbitMq的方法，传过去的是参数是订单编号，使用Message对象作为信息交换的载体，调用rabbitTemplate.convertAndSend发送消息，这里的方法参数要重写一下MessagePostProcessor，然后使用message.getMessageProperties().setHeader("x-delay",expTime);设置延时的时间我们这里一般设置的是2小时，这样生产者就能讲数据发送过去了。接着编写消费者使用springboot集成RabbitMQ，集成方式和生产者一样，编写消息队列的监听类，在方法上加上@RabbitListener(queues = "ewtp\_order\_Delay\_queue")指明监听的队列名称，然后队列中存放的数据到2小时后就会被监听消费，获取消息body中orderId,查询数据库的订单状态，如果订单已经完成或者取消就不再执行后续业务，如果订单还是未支付的状态就需要将订单中涉及的商品库存进行回滚，完成未支付订单的后续处理。

**SpringBoot、springCloud：**

我们在项目中是用Feigin进行微服务之间的调用的，又因为Feign集成了Hystrix(黑丝追克斯),所以我们当时通过在配置文件中开启hystrix,并且在具体使用Feign的类上通过 @FeignClient 中的 Fallback指定降级时候需要执行的类即可。这样在熔断后就会自动调用Fallback类中的降级方法，降级方法中我们也没有做特殊的处理，就是记录下日志并且返回null。这样使用Feign的客户端根据调用方法时候的返回值，如果为null,则证明走了降级方法，进行特殊处理就行。

Feign还拥有负载均衡的特性，说白了它是靠Ribbon进行的负载均衡，只不过对Ribbon做了封装，让我们用起来更方便。具体在做的时候，我们会单独提取出来一个API项目，该项目中包含了实体类以及公共的接口，这个公共接口中其实就是原来微服务中controller层中的方法。并且还包含该接口对应的FeignClient的客户端，一定要注意FeignClient的客户端也必须是一个接口，这个时候可以让FeignClient的接口继承于公共接口，在FeignClient的客户端上加入 @FeignClient的注解，并指明要调用的服务名，这样就可以通过该服务名从注册中心Eureka中获取对应的地址列表，方便进行负载均衡的调用。

其次我们会创建对应的提供者项目，提供这项目和上面所说的流程一致，

只不过在controller类中去实现API项目中的公共接口，重写里面的方法。

在作为客户端的消费者项目中，首先在pom文件中引入Eureka和Feign的依赖，

接着在application.properties中指明注册中心的地址方便从注册中心获取提供者的地址列表，

然后在启动类上要添加 @EnableFeignClients，还需要在 @EnableFeignClients中加入要扫描的FeignClient的包路径。之后在需要用到调用远程提供者的地方,注入API项目中已经事先写好的FeignClient，直接进行调用即可。通过Feign我们能让微服务之间的调用变的像本地服务之间的调用一样方便。在这整个过程中为了方便取出服务端返回的结果，我们还将ServerResponse/TResponse这个类进行了泛型化。

在使用Feign作为微服务调用的时候一定要特别注意，公共接口中的方法如果用到 @RequestParam,@PathVariable 就必须指明value值，否则启动时候会报错，再者就是在controller中实现公共接口方法时候也要在方法的参数中再次用到公共接口中方法里面一样的注解，如果公共接口中的方法里有 @PathVariable而controller类中的方法中没写这个注解，则接受不到客户端传递的参数信息，这个也是需要特别注意的。

在具体开发一个微服务的时候，用的是SpringBoot，首先在项目的pom文件中同样需要引入Eureka的依赖，并且在application.properties文件中通过spring.application.name指明自身的服务名以及注册中心的地址，这样才能将其注册到注册中心中被其他消费者访问。

在项目中还使用到了 @SpringBootApplication，之所以可以省略那么多的配置文件，

靠的就是这个注解。这个注解的本质是包含了 @Configuration， @EnableAutoConfiguration， @ComponentScan这三个注解，通过这些注解就起到了扫描类以及自动加载配置信息的作用。

还在biz/service业务层上通过 @Transactional(rollbackFor = Exception.class)指明了事务的控制，使其在抛出Exception异常时能够回滚。默认情况下事务只对运行时异常进行回滚所以才会通过rollbackFor = Exception.class将其改为对所有的异常都进行回滚。还需要在Mapper接口上通过注解的方式完成持久层的常规操作；如通过 @select 进行查询， @Results 将查询后的结果和实体类中的属性进行映射， @insert 进行插入， @update 进行更新， @delete 进行删除， @selectKey 可以在插入后获取新插入的记录的id值并将其赋值给实体类中的id属性。想要在service层注入 Mapper 层的接口，则还需要在启动类上加入 @MapperScan这个注解对其Mapper层的接口进行扫描。

在项目中我们用Eureka建立注册中心项目，在pom文件中引入Eureka的依赖，之后在

启动类中加入 @EnableEurekaServer的注解证明该项目的作用是作为注册中心使用，

并且在对应的application.properties中指明Eureka注册中心服务器的地址，方便提供者和消费者对其进行访问，为了增强注册中心的高可用性，我们做了Eureka集群，防止因为单台Eureka导致的单点故障问题。

如果没有注册中心，服务的调用者通过ip地址直接访问远程的提供者，如果服务越来越多，调用也会越来越复杂，一旦ip地址发生变动，则所有用到的地方都需要发生改变，维护起来会特别麻烦；如果访问的提供者宕机了，所有访问的客户端也都会受到影响。通过注册中心可以让服务的提供者注册到注册中心上，并且可以进行负载均衡；这样就可以达到高可用以及提高并发的 目的。服务的调用者也不需要 直接 调用服务的提供者，而是通过注册中心获取地址列表，从提供的地址列表选一个提供者接口进行调用，后续的维护各方面也会比较方便。

总体来说在项目中我们使用SpringBoot开发微服务并结合SpringCloud的注册中心Eureka和

支持负载均衡的客户端Feign来完成整个项目的开发,使用Zuul充当网关；其中Feign是为了在微服务之间进行调用的时候使用，而Zuul发挥了反向代理和负载均衡的作用，前端发送的所有请求经过Zuul，进行安全验证，通过后再转发到具体的微服务上。

**JVM虚拟机调优**

我们的linux服务器,安装centos6.5, 64位的操作系统和64位的软件，配置了4个cpu,64G内存。

我们在jvm优化的时候是这样做的

首先JVM将内存划分为：

年轻代

年老代

永久代

其中年轻代和年老代属于堆内存，永久代不属于堆内存，有虚拟机直接分配。

年轻代：年轻代用来存放JVM刚分配的Java对象

年老代：年轻代中经过垃圾回收没有回收掉的对象将被Copy到年老代

永久代：永久代存放Class类、Method方法元信息，其大小跟项目的规模、类、方法的量有关，一般设置为128M就足够，设置原则是预留30%的空间。

jvm的垃圾回收算法有:

串行算法（单线程）

并行算法

并发算法

**吞吐量优先的并行收集器**

**响应时间优先的并发收集器**

我们的项目设置的是响应时间优先的并发收集器，

将堆大小通过 -Xms3000m -Xmx3000m设置为3000m,将年轻代通过 -Xmn2g 设置为2g,设置年老代为并发收集，当时设置的是运行6次GC以后对内存空间进行压缩、整理。

打开对年老代的压缩，可以消除碎片 。

**被动说：**

**spring中的事务以及事务的传播特性隔离级别**

**Spring AOP事务的描述：**

**我们项目是在spring的配置文件里，通过设定一个表达式，对service里add ,delete,update说白了就是和增删改相关的方法进行事务拦截。我们需要配置事务的传播特性（propagation="REQUIRED"）,把增,删,改以外的查询操作配置成只读事务read-only="true"）.只读事务可以提高性能。之后引入tx:advice,在tx:advice引用 transactionManager（事务管理）,最后通过引入txAdvice。**

**关于事务这块有这么几个关键点，**

**1.判断事务是否配置成功的关键点是在出现异常时，事务是否会回滚！！！ 2.事务通常控制在service业务逻辑层，这样在service层中调用dao层操作数据库，把整个业务逻辑涉及到的数据库操作当成一个整体，只要有一步抛异常出错，前面的操作也都会进行相应的回滚，只有都执行成功才会进行commit提交！！！ 3.通常对增删改进行事务的控制，对查询方法进行只读事务的控制，因为查询方法不会对数据库的数据造成影响不涉及到回滚操作而增删改这些写操作的方法会对数据造成修改所以进行事务控制后就可以在出现异常时候进行回滚！！！4.默认情况下事务只对运行时异常【RuntimeException】 进行回滚！！！通过在配置中加入rollback-for=Exception 来对所有异常进行回滚！！！**

**除此之外咱们还知道，事务还有四大特性**

**1. 原子性（Atomicity）**

**原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。**

**2. 一致性（Consistency）**

**事务必须使数据库从一个一致性状态变换到另外一个一致性状态。(数据不被破坏)**

**3. 隔离性（Isolation）**

**事务的隔离性是指一个事务的执行不能被其他事务干扰.**

**4. 持久性（Durability）**

**持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的.**

**事务的传播特性以及隔离级别**

**spring中事务的传播特性好像有5个左右，我做项目的时候使用最多的就是PROPAGATION\_REQUIRED，它所代表的意思支持当前事务，如果当前没有事务，就新建一个事务。spring中事务的隔离级别有5个，默认使用的是ISOLATION\_DEFAULT,它代表使用数据库默认的事务隔离级别，也是我们项目中最常使用的。除此之外还有 读未提交： 它充许另外一个事务可以看到这个事务未提交的数据， 这种隔离级别会产生脏读，不可重复读和幻像读。 读提交： 保证一个事务修改的数据提交后才能被另外一个事务读取， 也是大多数数据库的默认值。可以避免脏读，但会产生不可重复读和幻像读。 不可重复读： 在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。 串行化： 顺序执行事务。除了防止脏读，不可重复读外，还避免了幻读。 并发性也最低，但最安全。 不可重复读的重点是修改?:? 同样的条件?,?你读取过的数据?,?再次读取出来发现值不一样了?。 幻读的重点在于新增或者删除?： 同样的条件?,?第?1?次和第?2?次读出来的记录数不一样。**

**1.java中实现线程的方式**

**在java中实现线程有两种方式：继承Thread类，实现Runable接口，一个java?main程序默认会开启两个线程一个是主线程，一个垃圾回收线程。**

**2.线程不安全与安全:**

**多个线程访问同一个资源，导致结果和期望值不同，我们就说它是?非线程安全的（线程不安全），反之我们就说它是?线程安全的。**

**3.线程的状态**

**1、新建状态(New)：新创建了一个线程对象。**

**2、就绪状态(Runnable)：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。**

**3、运行状态(Running)：就绪状态的线程获取了CPU，执行程序代码,对应线程中的run方法。**

**4、阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：**

**(一)、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。**

**(二)、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。**

**(三)、其他阻塞：运行的线程执行sleep()或join()方法，JVM会把该线程置为阻塞状态**

**5、死亡状态(Dead)：线程执行完了或者因异常退出了run()方法**

**4.解决线程安全的问题的方案:**

**a.通过加锁（synchronized）的方式解决线程安全问题**

**1.synchronized?方法?**

**2.synchronized?块(同步代码块)**

**b.避免使用全局变量**

**c.使用ThreadLocal**

**线程池**

**java线程池是为了让多个线程同时工作，充分的利用系统的资源，特别是多个cpu的资源，从而提高应用程序的性能。**

**同时也可以避免持续地创建新线程，限制最大线程数量。**

**可以通过Executors/ThreadPoolExecutor来创建固定数量的线程池。**

**阻塞队列**

**我知道阻塞队列有：**

**ArrayBlockingQueue：**

**是一个基于数组结构的有界阻塞队列。在使用时必须指定队列的大小。**

**LinkedBlockingQueue：**

**一个基于链表结构的阻塞队列，吞吐量通常要高于ArrayBlockingQueue。**

**Executors.newFixedThreadPool()使用了这个队列，它大小默认值为     Integer.MAX\_VALUE，**

**所以我们在使用LinkedBlockingQueue时也会手动传值，为其提供我们所需的大小，**

**避免队列过大造成机器负载过大或者内存爆满。**

**如果所有的线程都在执行任务，而且工作队列也被请求填满了，**

**此时对于新到达请求，我们会直接抛出异常，丢弃任务。**

**同时会将丢弃的做好记录，方便在线程池闲下来的时候继续进行处理。**

**1 锁：**

**对于项目开发中，经常用到的一些锁，简单说一下我对锁的一个理解。**

**从数据库层面来说有悲观锁，乐观锁。从数据库层面来说有悲观锁，乐观锁。**

**悲观锁,**

**每次去查询数据的时候都认为别人会修改，**

**所以每次在查询数据的时候都会上锁，**

**这样别人想拿这个数据就会阻塞直到它拿到锁。**

**传统的关系型数据库里边就用到了这种锁机制，**

**比如通过select ....for update进行数据锁定。**

**乐观锁,**

**每次去查询数据的时候都认为别人不会修改，**

**所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有**

**去更新这个数据，可以使用版本号，时间戳等机制。**

**乐观锁适用于多读的应用类型，这样可以提高吞吐量。**

**从java代码层面来说有synchronized同步锁**

**和synchronized同步块，它们都是为了**

**解决多线程访问时，线程安全的问题，**

**像java里面的hashtable和vector都是**

**用的synchronized来保证线程安全。**

**除此之外，还有读写锁**

**读写锁表示有两个锁，一个是读操作相关的锁，**

**也称为共享锁；另一个是写操作相关的锁，也叫排它锁。**

**多个读锁之间不互斥，读锁与写锁互斥，**

**写锁与写锁互斥。**

**也就是说多个Thread可以同时进行读取操作，**

**但是同一时刻只允许一个Thread进行写入操作。**

**通过lock()方法加锁，通过unlock()方法解锁。**

**除此之外还有分布式锁，普通的锁是为了解决多线程访问**

**产生的安全问题，分布式锁是为了解决多进程访问产生的**

**安全问题。我们在项目中使用Redisson来完成分布式锁。**

**2在linux服务器上部署项目的时候，通常会使用的一些命令**

**Linux中常用的命令:**

**1. 在部署springboot开发的微服务时，**

**通过Nohup  java  -jar jar包的名字 > 日志文件名 &   让程序以后台方式运行**

**2. 通过netstat -anp | grep 端口号 可以查看端口对应的进程号，通过netstat -anp | grep 进程号 可以查看进程对应的端口号，之后可以结合ps  -ef | grep 进程号， 来查看进程信息。**

**3.通过wget下载相关的文件**

**4.通过yum  install 在线安装软件，比如我在linux上**

**安装mysql时/ 比如在linux上安装gcc**

**5. 通过 rpm -ivh 进行rpm软件的安装**

**6. 通过make 和 make intall 安装nginx**

**5.通过ps -ef | grep 进程名 来查看进程，**

**比如我通过 ps -ef | grep java来查看**

**tomcat是否启动,通过 ps -ef | grep redis**

**来查看redis是否启动。**

**6.通过vi /etc/sysconfig/iptables来编辑文件**

**开启防火墙的端口号，最后通过service iptables restart**

**来重启防火墙使其配置生效**

**7.通过 kill -9 进程号来强制杀死进程**

**8.通过 cat /proc/cpuinfo查看cpu**

**9.通过 cat /proc/meminfo查看内存**

**10.通过 df  -h 查看硬盘信息**

**11. 通过 top命令,可以查看正在运行的进程和系统负载信息**

**12.find / -name 名字 : 查找指定名字的文件,可以使用\*通配符**

**13.vi 文件名:编辑文件**

**命令模式【默认情况】:通过Esc从编辑模式回到命令行模式**

**dd:删除行**

**yy:复制**

**p:粘贴**

**数字 dd:删除指定行的数据**

**gg:首行**

**shift+g:尾行**

**/查找的内容：查找**

**n:下一个**

**shift+n:上一个**

**编辑模式【按i进入】：**

**底部命令模式: 通过 ：使其从 命令模式 切换到 底部命令模式**

**:wq!:强制保存并退出。**

**:q! :不保存并退出。**

**Set,List,Collection,Collections的区别？**

**List和Set都是接口，他们都继承于接口Collection,List是一个有序的可重复的集合，而Set的无序的不可重复的集合。 Collection是集合的顶层接口，Collections是一个封装了众多关于集合操作的静态方法的工具类,因为构造方法是私有的，所以不能实例化。**

**List接口实现类有ArrayList,LinkedList,Vector。ArrayList和Vector是基于数组实现的,所以查询的时候速度快，而在进行增加和删除的时候速度较慢LinkedList是基于链式存储结构，所以在进行查询的时候速度较慢但在进行增加和删除的时候速度较快。又因为Vector是线程安全的，所以他和ArrayList相比而言，查询效率要低。**

**map概述**

**Map是一个以键值对存储的接口。Map下有两个具体的实现，分别是HashMap和HashTable.**

**2.HashMap是线程非安全的，HashTable是线程安全的，所以HashMap的效率高于HashTable.**

**3.HashMap允许键或值为空，而HashTable不允许键或值为空.**

**HashMap的底层代码/原理**

**HashMap的底层原理**

**HashMap底层就是一个数组结构，数组中的每一项又是一个链表。**

**当新建一个HashMap的时候，就会初始化一个数组。**

**?**

**Entry就是数组中的元素，每个?Entry?其实就是一个key-value对，**

**它持有一个指向下一个元素的引用，这就构成了链表。**

**?**

**HashMap?在底层将?key-value?当成一个整体进行处理，这个整体就是一个?Entry?对象。HashMap?底层采用一个?Entry[]?数组来保存所有的?Entry?对象，**

**当需要存储一个?Entry?对象时，会根据hash算法来决定其在数组中的存储位置，**

**再根据equals方法决定其在该数组位置上的链表中的存储位置；当需要取出一个Entry时，**

**也会根据hash算法找到其在数组中的存储位置，**

**再根据equals方法从该位置上的链表中取出该Entry。**

**?**

**?**

**默认是构建一个初始容量为?16，负载因子为?0.75?的?HashMap。**

**?**

**?**

**也就是说，默认情况下，数组大小为16，那么当HashMap中元素个数超过16\*0.75=12的时候，就把数组的大小扩展为?2\*16=32，即扩大一倍，然后重新计算每个元素在数组中的位置，而这是一个非常消耗性能的操作，所以如果我们已经预知HashMap中元素的个数，**

**那么预设元素的个数能够有效的提高HashMap的性能。**

**?**

**?**

**?**

**ConcurrentHashMap原理**

**在java并发包里面，ConcurrentHashMap是线程安全的hashMap，通过**

**引入分段锁的概念将一个大的Map拆分成多个小的HashTable.在存值**

**和取值得时候通过key.hashCode()来计算key及其对应的值应该放到**

**哪个HashTable中。默认情况下ConcurrentHashMap会创建16个分段数组集合。**

**这样在进行操作的时候如果有3个线程A?B?C，这时候A?B两个线程根据key.hashCode()**

**可能被分配到同一个hashtable中，这样A在进行操作的时候B就会阻塞。但C这个**

**线程可能被分配到了另外一个hashtable中，这样C就可以直接执行而不会阻塞。**

**所以效率就得到了极大的提升。默认情况下可以提高16倍。**

**String,StringBuffer,StringBuilder的区别**

**String???????字符串常量???不可变??使用字符串拼接时是不同 的2个空间**

**StringBuffer??字符串变量???可变???线程安全??字符串拼接 直接在字符串后追加**

**StringBuilder?字符串变量???可变???非线程安全??字符串拼 接直接在字符串后追加**

**1.StringBuilder执行效率高于StringBuffer高于String.**

**2.String是一个常量，是不可变的，所以对于每一次+=赋 值都会创建一个新的对象，StringBuffer和StringBuilder 都是可变的，当进行字符串拼接时采用append方法，在 原来的基础上进行追加，所以性能比String要高，**

**又因为 StringBuffer??是?线程安全的而StringBuilder是线程非安 全的，所以StringBuilder的效率高于StringBuffer. 3.对于大数据量的字符串的拼接，采用 StringBuffer,StringBuilder.**

**SpringMVC的运行原理**

**1.Tomcat在启动时加载解析web.xml,找到spring?mvc的 前端总控制器DispatcherServlet,并且通过 DispatcherServlet来加载相关的配置文件信息。**

**2.DispatcherServlet接收到客户端请求，找到对应 HandlerMapping，根据映射规则，找到对应的处理器 （Handler）。**

**3.调用相应处理器中的处理方法，处理该请求后，会返回 一个ModelAndView。**

**4.DispatcherServlet根据得到的ModelAndView中的视 图对象，找到一个合适的ViewResolver（视图解析器）， 根据视图解析器的配置，DispatcherServlet将要显示的数 据传给对应的视图，最后显示给用户。**

**jdk1.8新特性:**

**在项目中/在工作中 我用lambda表达式还有stream比较多，你像结合foreach使用lambda表达式可以对list和map进行循环遍历，使用起来特别简单，用一行代码就完成了以前几行代码才能做的事情。**

**比如要对map进行循环遍历 就可以调用map.forEach((x,y)->system.out.println(x,y)),其中x,y就是map中的键和值。还有使用到线程的话就可以直接用()->{},在{}中写上要执行的**

**逻辑，其实就相当于以前run方法的内容。**

**stream的作用就是对集合类进行操作，它不会改变原有的集合类，支持filter过滤，这个**

**可以理解为sql语句中的where条件，支持map进行集合的转换，比如可以结合map将list转换为list**

**Spring中的AOP和IOC**

**IOC我的理解就是说原来咱们想要创建一个类的对象，得自己通过new的方式进行，**

**而现在可以通过在spring的配置文件中写上一段的配置，**

**从而让创建对象实例的过程交给spring容器来完成，而我们在程序中可以直接获取**

**创建好的对象实例进行使用。这就是IOC的控制反转。**

**AOP的切面在项目中用的还是比较多的，咱们都知道spring中的事务管理就是通过AOP切面的方式来实现的。切面的好处就是把分散在代码中的重复代码提取出来进行统一的维护和控制。我们在项目中就使用aop完成日志的统一处理。原来是把日志记录的相关代码分散到各个控制层的相关方法中，但这就会导致程序员在开发时候不能将精力集中到业务逻辑的处理上，还得考虑记录日志，工作效率就大打折扣，我在项目中通过....[将aop中日志的统一处理融进去]**

**jdk1.8新特性:**

**在项目中/在工作中 我用lambda表达式还有stream比较多，你像结合foreach使用lambda表达式可以对list和map进行循环遍历，使用起来特别简单，用一行代码就完成了以前几行代码才能做的事情。**

**比如要对map进行循环遍历 就可以调用map.forEach((x,y)->system.out.println(x,y)),其中x,y就是map中的键和值。还有使用到线程的话就可以直接用()->{},在{}中写上要执行的**

**逻辑，其实就相当于以前run方法的内容。**

**stream的作用就是对集合类进行操作，它不会改变原有的集合类，支持filter过滤，这个**

**可以理解为sql语句中的where条件，支持map进行集合的转换，比如可以结合map将list转换为list,支持skip,limit进行类似的分页操作，支持distinct进行去重操作，支持count进行聚合，foreach进行遍历输出，collect将流中的元素累积成一个list或者set集合。**