**B简历：**

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

**1) 基本信息**

|  |  |  |  |
| --- | --- | --- | --- |
| **基本情况** | | | |
| **姓名** | 焦启阳 | 性别 | 男 |
| **出生日期** | 1994-09-30 | 属相 | 狗 |
| **现住地** | 中原区 升龙又一城D区 |  |  |
| **大学情况** | | | |
| **学校** | 新乡学院 | 专业 | 计算机科学与技术 |
| **入学时间** | 2012-9 | 毕业时间 | 2016-7 |
| **专业课** | java语言程序设计、Java企业级开发、数据库技术与应用、大型数据库处理技术、网页设计与开发、计算机网络与通讯、软件工程 | | |
| **学历** | **本科** | 学位 | **学士学位** |
| **学校地址** | 新乡市红旗区金穗大道东段191号 | 校长 | **刘兴友** |
| **去学校的方式：** | **坐大巴到新乡汽车东站，乘60路公交车，路径7站，到留庄营新村下车就到了** | | |
| **上小学日期：** | **2000年9月   6岁六年** | | |
| **上初中日期：** | **2006年9月   12岁三年** | | |
| **上高中日期：** | **2009年9月   15岁三年** | | |
| **上大学日期：** | **2012年9月   18岁四年** | | |
| **毕业时间：** | **2016年7月   22岁** | | |
| **入学时间：** | **2012年9月** | | |
| **毕业时间：** | **2016年7月** | | |
| **在校学生：** | **20000多** | | |
| **有教师：** | **不太清楚** | | |

|  |  |
| --- | --- |
| 第一家公司 ： | **合其家物联网科技有限公司** |
| **地址：** | 五凤街道软件园b区12号 |
| **入职时间：** | **2016年4月** |
| **离职时间：** | **2018年6月** |
| **公司性质：** | **民营** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **林永生** |
| **薪资方面：** | **刚刚毕业转正以后工资是6K，过完年涨到7K，离职时工资是9K** |
| **路线：** | **住在公司不远的小区，凤仪家园，走路上班** |

|  |  |
| --- | --- |
| **第二家公司情况：** | **杭州熙盟科技有限公司**(熙盟) |
| **公司地址**： | 浙江省杭州市余杭街道文一西路1818-2号15幢（万达广场旁边） |
| **公司规模：** | **30人左右，我们项目组6人，一个项目经理，2名开发人员，1名前端，1个测试，1个运维** |
| **公司性质：** | **民营** |
| **公司行业：** | **技术开发、技术服务、技术咨询、成果转让：计算机软硬件、计算机系统集成、工业自动化技术** |
| **去公司路线：** | **当时住在马鞍山雅苑 出小区沿着文一西路向东约700米然后向北沿智溢路约300米** |
| **入职时间：** | **2018年8月** |
| **离职时间：** | **2021年3月底** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **习军哥** |
| **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：** | **15K** |
| **薪资的构成:** | **岗位工资(5K)+绩效工资(10K)** |

**面试常见问题：**

|  |
| --- |
| 四年涨薪：  **2016年工作的时候是5K，税后还是5K，转正6K左右**  **2017年初涨了1K左右，总7K左右**  **2017年12月涨了2k，总的9k左右**  **2018年8月跳槽涨了3K左右，总12K**  **2019年初涨1K，13K左右**  **2019年8月工作一年涨了1K 14k**  **2020年9月工作一年涨了1K 15k**  **2021年3月离职时工资为15K** |

|  |
| --- |
| **你为什么从上家公司离职？** |
| **之前在外包公司，最近公司接的项目比较少，没什么事情，太安逸了，想换一个新的平台去充实提升自己!** |
| **优点:** 团队协作能力强**抗压自学能力强（有一次项目时间很紧，经常工作到12点半之后，连续攻坚一个月终于把项目完美的完成了。），**自学能力强  **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: 这几年做程序，因为要不断的对代码进行验证确认，整体达到一定效果才可以，所以感觉自己现在有点强迫症** |
| **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。** |
| **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** |
| **能** |
| **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 15K 上家工资14K 税后13K左右 扣了1K左右的税** |
| **你的五年规划是啥？** |
| **近期以提升技术为主，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：**  14薪  **车补**  200左右  **项目奖金**  7k-10k，做完项目，收完尾款的时候。  **年终奖**  这个就是2个月的工资，  一般是过年的时候和当月工资一块发  **电话补助**  300  **饭/餐补**  30/天  **电脑补**  300 |

|  |
| --- |
|  |
| **你的优缺点是啥？** |

**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！**

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

您好！我是来面试咱们公司的java 开发的！我是17年毕业的，到今年呢已经是有3年多的工作经验了，在这3年里我大大小小做的有几个项目。在这四年里面我也掌握了不少技术，比如说你像一些主流的开发框架，像什么Spring、SpringMVC啊、MyBatis、Hibernate、MyBatis-Plus啊、SpringBoot啊、还有SpringCloud都接触过。这是主流的开发框架，然后你像一些常用的关系型数据库比如说MySQL、Oracle这些我也用的比较多，像MySQL的主从复制还有读写分离啊也做过。然后你像一些分布式缓存比如说Redis我也用的不少，像Redis集群我也搭建过。还有的话你像消息队列中间件比如说RabbitMQ之类的我也用过,微信支付或者是支付宝支付这块我之前也做过。然后像接口测试的话，你像Postman对接口进行功能测试，我也用过。还有就是像SQL优化啊、JVM优化啊、Tomcat优化之类的我也了解过。这就是我这几年来的一个工作经历。

主动说：

我最近做的一个项目是使用 Spring Boot+springCloud 来做的微服务项目的开发

首先先来说一下，这个springboot项目具体是怎么做的吧！

**框架篇**

**首先讲下对springboot的理解**开发springboot项目的时候，我发现springboot的核心就是启动类上的两行代码，启动springboot会首先加载启动类，启动类中可以看到两行代码,@SpringBootApplication和SpringApplication.run方法。

在 @SpringBootApplication 的内部包含了3个注解

@Configuration

@EnableAutoConfiguration

@ComponentScan

@Configuration是基于JavaConfig形式的Spring Ioc容器的配置类，可以把它看成xml配置文件中的beans标签。 @Configuration写到类上面，在类中的方法上如果写了 @Bean注解，那么它的返回值将作为一个bean注册到Spring的IoC容器，方法名默认作为bean的id。

@ComponentScan这个注解对应XML配置中的context:component-scan元素， 说白了它的作用就是自动扫描并加载符合条件的组件,比如 @Component和 @Service等或者bean定义，最终将这些bean定义加载到IoC容器中。

我们可以通过basePackages来指定 @ComponentScan 自动扫描的范围,如果不指定，则默认Spring框架实现会从声明 @ComponentScan所在类的package进行扫描。这也是SpringBoot的启动类最好是放在root package下的原因。

@EnableAutoConfiguration 这个注解是借助 @Import的帮助，将所有符合自动配置条件的bean定义加载到IoC容器中。

Springboot的简单实用：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。

**讲述对springcloud微服务的理解**

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发挥了请求转发和过滤的作用，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。

在具体开发一个微服务的时候，用的是SpringCloud，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，客户端通过ip地址访问服务端，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；如果服务端宕机了，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端注册到注册中心上去，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。客户端 也不需要 直接 调用服务端，而是通过注册中心获取地址列表，从地址列表中选一个服务端接口进行调用，后续的维护各方面也会比较方便。

心跳健康检测

在应用启动后，将会向Eureka Server发送心跳,默认周期为30秒，如果Eureka Server在90秒内没有接收到某个微服务的心跳，Eureka Server将会从服务注册表中把这个服务节点移除

但是当网络分区故障发生时，微服务与Eureka Server之间无法正常通信，以上行为可能变得非常危险了——因为微服务本身其实是健康的，此时本不应该注销这个微服务。

Eureka通过“自我保护模式”来解决这个问题

如果在15分钟内超过85%的客户端节点都没有正常的心跳，那么Eureka就认为客户端与注册中心出现了网络故障，Eureka Server自动进入自我保护机制，Eureka Server不再从注册列表中移除因为长时间没收到心跳而应该过期的服务。Eureka Server仍然能够接受新服务的注册和查询请求，但是不会被同步到其它节点上，保证当前节点依然可用。当网络稳定时，当前Eureka Server新的注册信息会被同步到其它节点中。

**讲项目的统一管理**

之前我们在做项目的时候，是把所有的接口都写到一个API接口项目中，那时候处理接口安全问题，我们是写了一个AOP切面或者拦截器，统一对控制层的各个方法进行安全验证，但现在用到SpringBoot+SpringCloud微服务，每个功能模块都是一个单独的微服务项目，如果还用之前AOP切面或者拦截器的方式，就会出现很多重复代码，维护起来也特别麻烦。

因为外部客户端的请求都是通过Zuul网关路由到具体的微服务，所以为了保证微服务的安全，我们就在Zuul中自定义了过滤器，对所有微服务的安全进行统一的处理，还有，因为涉及到前后端分离，前端项目访问后端微服务涉及到跨域问题，所以我们在Zuul中也自定义了关于跨域的过滤器，进行统一处理。

具体是这么做的，首先你得先定义一个继承于ZuulFilter的类，重写里面的filterType()方法filterOrder()方法，shouldFilter()方法，还有一个最核心的用来写具体业务逻辑的run()方法。其中filterType方法的作用是用来返回一个字符串，指明该过滤器的类型，经常用到的有pre类型，说白了就是在请求被发送到微服务之前调用；我们的微服务安全认证以及跨域这块都是用的pre类型的过滤器，这样对非法请求，就可以在发送到具体的微服务之前拒绝它；还有post类型，说白了就是微服务执行完后再执行该过滤器，filterOrder()方法返回一个int类型的值，用来指明该过滤器的执行顺序，数字越小表示优先级越高，就越先执行，shouldFilter()方法返回一个boolean值，用来指明该过滤器是否执行，true表示执行,false表示不执行。

run方法中就是之前说的，用来放具体的处理逻辑。

在run方法中，首先获取一个RequestContext对象，之后就可以通过调用它的getRequest()方法来获取request对象；这样就可以获取头信息，按照基于token的方式进行接口的安全验证。

过程中特别需要注意的就是当验证不通过的时候，需要通过fastjson将要响应的数据转换为json格式的字符串，之后设置响应的内容类型为application/json并且指定utf-8的编码方式，用来处理中文乱码问题；通过setResponseBody将Json格式的字符串设置为响应的内容，最后通过setSendZuulResponse为false，禁止路由转发。如果想要将zuul过滤器中的数据传递给后端微服务中使用，则需要通过addZuulRequestHeader方法来进行，而后端微服务中就可以通过request.getHeader来获取值。这里面需要特别注意的就是，如果传递的数据中含有中文则需要通过URLEncoder进行utf-8的编码，同样在获取数据后也需要通过URLDecoder进行解码。

最后想要使自定义的过滤器生效，得进行相关的配置。我通过创建一个配置类，并且在类上通过 @Configuration 和 方法上的 @Bean 结合起来，完成自定义filter的配置。

**讲微服务之间的相互调用及雪崩问题处理**

在整个微服务搭建的过程中有可能发生微服务雪崩的问题，说白了就是A->B->C,如果C这个微服务出现问题，就会影响到B,同样B出现问题，就会导致A也出问题，最后导致所有的微服务都不可用。就像咱们现实生活中，一个路口堵车，就会导致整条街慢慢拥堵，再后来影响的范围就会越来越大。可以用SpringCloud中的Hystrix来解决这个问题。

咱们都知道Hystrix可以进行熔断，降级和资源隔离。熔断其实就可以理解成咱们家里面的保险丝。保险丝是为了保证在高负载情况下电路的安全，如果某个电路出现了负载过大情况，保险丝就会熔断，从而保证其他电路不受影响。Hystrix的熔断功能比较强大，它还可以进行自动检测并修复。

具体的原理是这样的，默认情况下在10s内，每当20个请求中，有50%失败时，就会触发熔断，导致hystrix从闭合状态切换到打开状态，这样就不需要每次都去调用远程微服务而是直接走调用者本地的降级方法。5秒后hystrix会进入到半打开状态，这时如果有新的请求再过来hystrix就会尝试再次进行远程微服务调用，如果调用成功就会进入到闭合状态，如果不成功就会再次进入到打开状态，继续进行熔断。这个就是hystrix的自动检测并修复的能力。

降级说白了就是进行熔断后需要执行本地的方法。也是为了保证微服务的高可用。

[说白了就像咱们看的打仗时候的急行军，为了保证按时到达作战地点，我就把身上其他的干粮，背包全部丢掉，只带上最基本作战的武器，快速前进保证最终的结果。可以省略]

在Hystrix中, 有两种资源隔离方式，信号量隔离和线程池隔离，我们主要用的是线程池隔离. 通常在使用的时候我们会根据调用的远程服务划分出多个线程池. 比如调用产品服务的Command放入A线程池, 调用会员服务的Command放入B线程池. 这样就可以将运行环境隔离开。就算调用服务的代码存在bug或者由于其他原因导致自己所在线程池被耗尽, 也不会对系统的其他服务造成影响。

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,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的接口上通过 @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类中对应的降级方法，降级方法中，可以记录下日志并且返回null。这样使用Feign的客户端在调用 远程微服务 的方法后，判断返回值是否为null,如果是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。在 @FeignClient的注解上指明要调用的微服务的名字，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服务端，服务端项目中要保证 请求的方式，请求的地址以及方法名，返回值，参数都要和feign中定义的保持一致。

性能调优篇

因为做的是电商项目，所以我们对项目的多线程，高并发问题都很重视，针对这些也做了相应的处理。比如说在订单这个模块就会因为高并发出现一些问题；

**订单模块**

**当时做这个项目的时候，考虑了很多东西，这个模块相对比较复杂，像表设计，订单接口幂等性问题，商品的超卖问题，包括订单也都是用户登录后才能进行的操作等等。**

首先说下高并发情况下产生的超卖问题吧：超卖是这样产生的，同时下订单时先对商品库存进行查询、满足条件才能下单，但在高并发的情况下，可能好几个查到了同样的结果，同时下单，将数据库中的库存买到了负数。为了杜绝这种情况呢，我做了以下的处理：【超卖】

在sql语句中拼一个and条件，通过判断，用数据库的库存减去我们要买的数量大于等于0的话就能通过，从数据库根源上解决超卖问题

**【幂等性/重复下单】**

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。

所以我们借助redis通过我们前段传过来的keyId唯一标识，先取在放，判断redis是否为空，这样就可以让第一个请求能够过去 第二个请求过不去，但是容易出现线程安全问题，

第一个请求在取的时候还没走到存的地方，第二个请求也取出来了，所以就导致了线程安全问题。

我们通过redis中的分布式锁，getAndSet 来进行取存一体，因为redis是单线程的所以可以帮我们解决了这个线程安全问题

**购物车模块购物车这块也是我做的。同样为了提高性能，这个模块我当时做的就是基于redis的购物车，将购物车的所有数据都存储到redis中。这是个项目呢，当时从表设计到最后用postman进行接口的测试，都是我负责的。**

当时用了redis中的hash结构来存储所有用户购物车的数据，咱们都知道在使用hash的时候，涉及到key,field,value这三个方面的参数信息，key这块我用了 CartInfo 这个固定的字符串来表示，在field这块为了能够表示不同用户的购物车，我用来 member:会员id来表示，在value这块我是把会员对应的整个购物车的信息都转换为Json格式进行了存储。整个购物车对应的数据结构是这个样子的。

购物车中包含 商品总件数，总价格，以及 购物车的商品项列表，在商品项中包含 商品id,商品名，商品单价，商品件数，商品图片，商品小计 等，这里面特别要注意的就是关于价格这块在java中要用bigDecimal这个数据类型，这样可以防止精度丢失。

只有会员登录后才能操作购物车，比如添加商品，删除商品等。关于登录这块，我采用的是基于token的用户登录，类似于自己写了一个jwt【json web token】。

这样用起来更加的灵活方便，在自定义的拦截器中，当通过了层层关卡的验证后，将会员信息存入到request中，这样在其他的控制层方法中就可以通过request来获取当前登录的用户信息。为了方便在每个控制层方法中获取会员信息，我采用了 spring中自定义的参数解析器，这样就避免了在每个控制层方法中都使用requst.getAttribue来获取会员信息，而是在自定义的参数解析器中写一遍即可。

具体我是这样做的，自定义了一个实现HandlerMethodArgumentResolver接口的参数解析器 类，这里面最主要有两个方法，supportsParameter和resolveArgument，在supportsParameter中我通过methodParameter获取参数的类型和我自定义的会员类进行对比如果相等则返回true,证明找到了会员类型的参数，当返回true时就会走resolveArgument方法，在该方法中，通过request.getAttribute()获取拦截器中存入的会员信息，并将其强转为指定的会员类，进行返回。最后要想使自定义的参数解析器生效，则需要在spring-mvc的配置文件中配置mvc:annotation-driven，并在其中配置mvc:argument-resolvers，通过bean class指定自定义的参数解析器类，这样就可以在控制层的各个方法中，直接写上会员类参数，就可以通过会员类获取其中的会员id等信息。

这里面涉及到 添加商品到购物车的业务逻辑 是这样的。这个接口方法接受3个参数，其中商品id和商品个数是直接从前台传过来的，而会员类则是通过自定义的参数解析器来完成赋值的。首先我会根据商品id判断该商品是否存在，如果不存在则提示商品不存在的信息，如果存在则需要判断该商品的状态是否正常，比如是否是上架状态，如果状态是下架则提示该商品已经下架。

经过这些验证后，我会判断该会员在redis中是否有对应的购物车信息，是通过封装的工具类redisUtil中的hget来获取的，如果该会员对应的购物车信息为空，则首先要创建购物车，并且将当前商品添加到购物车中，最后还要计算购物车中的总件数和总价格，通过阿里巴巴的fastjson将其转换为json格式，并通过我封装的redisUtil的工具类hset方法，将其存入到redis中的hash结构里。

如果会员对应的购物车信息不为空，则分情况进行处理。首先看看该商品是否已经在购物车中了，如果是，则找到该商品并更改商品的数量为商品原有的数量加上前台传过来的商品的数量，还要重新计算该商品的小计，以及重新计算整个购物车中商品的件数以及总价格，最后更新redis中的数据。

如果该商品不存在购物车中，则将该商品加入会员的购物车，并重新计算整个购物车中商品的件数以及总价格，最后更新redis中的数据。

这就是整个往购物车中添加商品的逻辑，至于减少商品的个数，完全可以使用和添加商品一样的方法，只不过传过来的商品的个数为负数就行了。

而且还有一些小细节，就是如果不断的减少商品个数，当商品个数减少到零的时候，就将该商品一并从购物车中删除掉。当购物车中所有的商品都被删除的时候，也同时删除整个购物车。最后都得更新redis。

**线程**

线程的状态/生命周期:

新建状态：new了一个线程

就绪状态:?调用线程对象的start方法后

运行状态：当被操作系统选中，获取cpu的使用权，执行的是run方法

阻塞状态：因为某种原因，放弃了cpu的使用权

死亡状态：线程运行结束了或者运行过程中出现异常退出了

为啥会产生线程安全问题?

当?多个线程?操作?同一个资源?才会出现线程安全问题。

什么叫线程安全，什么叫线程不安全？

当?多个线程?操作?同一个资源，最终的结果和预期值

不一样，则为线程不安全，最终结果和预期值一样，

则为线程安全！！！

**线程池概念**

**对于高并发、多线程的问题，我们还通过线程池的方式提高程序的性能。**

首先线程池可以**避免频繁的创建和销毁线程所造成的性能损耗**，原理就和数据库连接池差不多，说白了就是项目启动的时候在线程池中就已经创建好了指定数量的线程，需要的时候直接去用，用完后再放回线程池供其他程序进行使用。再者用线程池中的多线程可以处理大批量的数据，比如我要将数据库中的图片加水印或者在硬盘上批量生成文件，这时候都可以用线程池。就像要洗100个碗，你可以让一个人去干，这一个人就像是一个线程，你也可以让10个人一块去干，这就是线程池中的多线程。相比而言多线程执行的时间更短，效率更高。在项目中我们是用**ThreadPoolExecutor**来创建线程池的，它里面有几个核心的参数信息，线程池的**核心大小**，**队列**以及**线程池的最大值**。线程池的工作原理是这样的，默认情况下，创建线程池之后，线程池中是没有线程的，需要提交任务之后才会创建线程。如果当前线程池中的线程数目小于核心线程数，则每来一个任务，就会创建一个线程去执行这个任务；如果当前线程池中的线程数目>=核心线程数，则每来一个任务，会尝试将其添加到队列中，若添加成功，则该任务会等待空闲线程将其取出去执行；当队列已满，添加失败，就会尝试创建新的线程去执行这个任务；这时候创建新的线程是根据线程池时的最大线程数为依据的，如果当前线程池中的线程数目达到最大线程数，则会采取任务**拒绝策略**进行处理；说的简单点就是先把核心线程数给占满了，不够用就开始往队列里面放，如果队列也占满了，就往创建的最大线程池里面放，如果也占满了就可以根据策略进行拒绝处理。

如果线程池中的线程数量大于 **corePoolSize**【核心线程】时，某线程的空闲时间超过**keepAliveTime**【存活时间】，线程将被终止，直至线程池中的线程数目不大于corePoolSize【核心线程】。

线程池中的队列，一般常用的有**ArrayBlockingQueue**【有界阻塞队列】：基于数组的队列，创建时必须指定大小；还有**LinkedBlockingQueue**【无界阻塞队列】：基于链表的先进先出队列，如果创建时没有指定此队列大小，则默认为Integer.MAX\_VALUE；

被动说：一般需要根据我们会根据任务的类型来配置**线程池大小**，如果是CPU密集型任务，就需要尽量压榨CPU，可以设为 CPU个数\*CPU核数+1，如果是IO密集型任务，参考值可以设置为2\*CPU个数\*CPU核数。

业务篇

在这个项目里我负责了会员登录的模块，购物车模块，订单模块

**会员登录模块**

我们之前做的登陆都是基于sesssion的登录，登录后把会员信息放在session中，我知道，session是存放在服务端的，当登陆的用户过多时，采用这种登陆方式就很浪费性能，所以这次我做的是基于jwt的登录，在网上有不少jwt（json web token）的登录，都有些缺陷，我就自己写了一套。

基于Token的登录呢,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，如果验证成功就使用JWT的工具类来进行数据加密 得出一个token值

把token值返回给客户端，客户端把他存到sessionstor里面，然后使用路由守卫获取sessionstorre里面的值判断值是否为空，为空就直接跳转到登录页面，不为空直接下一步，服务端的拦截器先获取传过来的token值判断token值是否为空，不为空在进行解密判断解密出来是否有数据，有数据就证明用户登录过了，就放开请求，又因为我们认证需要请求中带有token值，所有每个请求都带有token值特别繁琐，所以我们使用请求拦截器，所有的请求，请求之前都给他加上token值，就解决了每个请求都加token值的繁琐。

nginx负载均衡（高并发处理方式）我们项目在最后上线部署的时候，采用的是nginx+3个tomcat进行负载均衡，在我们不进行负载均衡之前，所有的请求都由一台tomcat进行处理，这样会    使我们的tomcat所承受的压力增大，而我们进行负载均衡之后，同样数量的请求经过nginx将其分发到多台tomcat进行处理，从而降低每台tomcat所承受的压力,而且当其中一台机器宕机时，其他机器还可以继续提供服务，保证服务不间断。

    项目在部署完成后，遇到这么个问题，用户登录输入验证码的时候，明明验证码输入的正确，但总是提醒说验证码不正确从而不能正常登录，经过分析后发现有可能第一次请求被发送到t1上，那么放在session中的验证码就被放到了t1上，当用户输入验证码点击登录时，新发送的请求有可能被发送到t2上面，而t2的session中不存在该验证码，这样在进行对比时就肯定会不一致从而提示验证码输入错误。

咱们都知道nginx有不同的负载均衡策略，默认是轮询策略，说白了就是将请求均匀的分发到不同的tomcat上，比如我们有两台tomcat，分别是t1和t2,当4个请求到来的时候，第一个请求被nginx发送到t1上，第二个请求被nginx发送到t2上，第三个请求又被发送到t1上，第四个请求又被发送到t2上，这就是nginx默认的轮询策略，当其中一台tomcat宕机时，会被自动剔除掉。

再者在企业中不同tomcat所在机器的配置和性能都不一样，这个时候如果用默认的轮询策略就显的比较浪费资源，我们可以通过weight这个关键字配置加权轮询，比如将性能高的那台tomcat的权重设置为4，将普通配置的那台tomcat设置为1，这样就能达到让性能高的那台tomcat处理更多的请求。

除此之外nginx还有一个IP\_Hash的负载均衡策略，这个策略说白了就是根据访问的客户端的ip地址，对其进行hash化，然后如果该客户端第一次请求被发送到t1上了，那么它就会和t1进行绑定，以后的每次请求也会被发送到t1上，只有t1宕机了，请求才会被发送到其他tomcat上。

所以我们就用ip\_hash这种负载均衡策略来代替默认的轮询策略，这样就解决了验证码错误问题。但这里面还有一个问题，如果被客户端绑定的这台tomcat宕机了，那么请求就会被发送到其他tomcat上，这个时候在用户进行操作时，请求会被拦截器拦截，拦截器中的代码会验证session中是否有当前登录的用户信息，如果没有则跳转到登录页面，因为这台tomcat上的session中不存在用户登录的信息所以就会跳转到登录页面，这就会导致用户的体验度非常不好，感觉也会很怪异。

为了最终解决这个问题，我通过上网查资料，就决定通过采用基于redis的分布式session来解决session共享的问题。

要想完成分布式session的编码工作，就必须需要搞清楚单机版的session是如何工作的，这样我们就可以模拟它工作的流程和机制，并且把相关的信息存储到redis这个缓存服务器中。

单机版session的工作原理是这样的，当访问页面时如果遇到getSession()这个方法，那么就会在服务端的内存中开辟一块空间用来存储以后要放到session中的数据，并且会生成一个唯一的  sessionId  作为和这块空间关联的依据，写入到客户端浏览器的会话Cookie中。说的通俗一点可以这样理解，就相当于我们去超市购物的时候一个个的存储柜，其中sessionId就是这个存储柜上的钥匙，而存储在session中的数据就是存储到这一个个存储柜中的物品。一把钥匙只能开一个柜子们，这样就保证了不同人拿着他们唯一的那把钥匙去开各自的柜子们，取到属于自己的物品。

当知道了这些以后，我们就清楚，这里面最主要涉及到会话cookie的操作以及将数据存储到redis中并且还得让会话cookie和redis中的key产生关联，这样每次发送请求时，会将本地存储cookie的域名和当前访问的域名进行对比，如果匹配则会将cookie自动提交到服务器端，这样就可以根据cookie中存储那个自定义的sessionId在redis中找到对应的数据，

首先我封装了一个cookie的工具类，然后在里面写了常见的 写cookie,读cookie,删cookie的常见操作。这里面要注意的是，cookie被分为会话cookie和持久化cookie，他们的区别就是会话cookie的生命周期和浏览器保持一致，一旦浏览器关闭了，就会消失；而持久化cookie是存储在硬盘上，他的生命周期取决于在编程的时候设置的maxAge的值；在具体编码的时候会话cookie不需要设置maxAge属性，而持久化cookie要根据项目的要求设置maxAge；再者在设置cookie时候还需要设置 域名以及Path,通常将域名设置为当前域名，将path设置为 / ；代表网站的根目录。写cookie通过的是response的addCookie方法，而读cookie用的是request的getCookies方法，返回一个cookie数组，然后根据cookie的name进行对比找到需要的cookie信息，删除cookie比较简单，说白了就是将指定name的cookie的maxAge设置为0,然后重新通过response的addCookie方法写入到客户端浏览器中，这样就完成了对指定cookie的删除工作。

在项目中我是通过jedis来作为中间的桥梁操作redis服务器的，并且考虑到性能问题配置了jedis连接池，这样就可以在大并发访问时候提高性能；并且把对redis的常规操作也都封装成了工具类，方便后续的使用。我将项目中单机版session的地方都改为了通过redis的进行信息的存储。首先在生成验证码的地方，我首先通过CookieUtil读取指定name的cookie，如果没有读取到则通过uuid生成一个唯一的标识作为cookie的值，用指定的LoginId作为cookie的name将其通过CookieUtil的writeCookie方法写入到客户端浏览器中，这样就保证了最开始没有LoginId的时候会生成新的,如果LoginId存在则无论怎么刷新会使用现存的LoginId而不会刷新一次就创建一个新的。并且会将Cookie的值作为redis中的key,将生成的随机数验证码作为值，调用redisUtil工具类存储到redis中，并设置过期时间为5分钟，因为验证码没必要一直存在占用内存空间。

在登录方法中首先通过CookieUtil的readCookie方法根据LoginId作为key，读取指定的cookie，获取cookie中的值，将该值作为redis中key,读取对应的验证码信息，和用户输入的验证码进行对比，如果一致则验证码正确进行后面的验证，否则验证码输入错误。在各项验证都成功后将获取的用户信息通过Gson转换为json格式的字符串作为值，用“user:”+LoginId作为key，将用户信息存入到redis中，并设置过期时间为30分钟，同时删除redis中的验证码，释放空间。

最后就是在拦截器中通过CookieUtil的readCookie方法读取到会话cookie中存储的LoginId的值，用“user:”+LoginId作为redis中的key读取用户信息,如果为null则证明用户没有成功登录跳转到登录页面，否则证明用户成功登录，调用redisUtil中的expire方法给redis中的用户信息续命，重新设置为30分钟，这也是参考本地session的工作机制。这样就完成将整个项目中的本地session切换为基于自定义cookie+redis的分布式session。

就解决了在进行负载均衡配置后，如果一台tomcat宕机，用户也无需重新登录的问题。

=============（补充点/被动说）=============日志管理模块

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。

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，这个切面类说白了就是一个普通的java类，在类上添加两个注解，分别是@Component @Aspect，这两个注解的作用分别是：

@Component：将本类交给ioc管理

@Aspect：声明这个是切面类。

在该类中新建方法，方法上添加注解，声明aop的通知，aop的通知共有五个，@Before前置通知，@After后置通知，@AfterThrowing异常通知， @Around环绕通知，@AfterReturning最终通知，设置参数，表明横切那个层，在这里我使用@Before前置通知，横切controller层。

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，把数据保存到我们日志里面

**（sql优化方面）**

1、对于大数据量的分页不要用常规的 limit 开始位置，每页条数；这样越靠后查询时间越长，效率越低，可以使用id>=的方法来替代，比如select 字段名 From 表名 where id >= (select id from 表名 limit 开始位置，1) Limit 每页条数

2、SELECT语句中避免使用'\*’，只查询需要返回的字段，这样可以减少解析sql语句的时间，以及减少 带宽，cpu,内存，io等 各方面的消耗。

3、 外键必须加索引，这样可以提升多表联查的性能

4、通过冗余字段避免多表连查，从而提高性能。比如在商品表中可以不仅有分类的id，还可以加上分类的名字，这样就不用为了显示分类名字而去进行多表联查了。又比如在会员表中不仅有地区的id，还可以有地区的名字这样也可以避免为了显示地区名而进行多表联查。

5、在 where，order by 以及GROPU BY 涉及的列上建立索引，为了避免全表扫描。

6、In和exists的区别：主要是就是驱动顺序的改变，这是性能变化的关键，如果是exists，那么以外层表为驱动表，先被访问，如果是IN，那么先执行子查询，以内层表为驱动表。所以IN适合于外表大而内表小的情况；EXISTS适合于外表小而内表大的情况。

7、通常要用小表去驱动大表，这样可以提高性能，在left join 中 左边的表为驱动表，所以应该让左边的表尽可能的为小表，在right join 中 右边的表为驱动表，所以应该让右边的表尽可能的小，在 inner join 中 mysql会自动选择较小的表为驱动表。

8、避免在 where 子句中对有索引的字段进行运算,这会导致索引失效，从而进行全表扫描。比如select 字段名 from 表名 where 库存量+1>=10,就需要改为 select 字段名 from 表名 where 库存量>=9

9、避免在where字句中对字段进行null值判断，这样会导致即便字段上有索引也会进行全表扫描，好的处理方式就是给字段设置默认值。

什么是索引

通过索引可以提高查询的速度，它就相当于书的目录；索引不是越多越好，因为索引会占用额外的空间，并且在进行增删改的时候还需要额外维护索引。创建索引的原则：给频繁查询的字段上创建索引根据当前字段的业务含义，来区分是创建唯一索引还是创建普通索引考虑到索引不是越多越好，也可以根据业务情况创建复合索引，

想要使其复合索引起作用，在查询的时候需要按照当时创建

复合索引的字段的顺序来,也就是最左匹配原则。

执行计划:通过explain查看sql语句的执行计划key:代表mysql实际使用的索引。type:这个字段比较重要，它表示MySQL在表中找到所需行的方式：

Const：主键在where条件中

Range：扫描部分索引，常见于between、<、>等的查询

Index：代表扫描全部 索引树

ALL：代表扫描全 表/全表扫描/不走索引。

创建索引-- 普通索引create index index\_status on t\_log(status);

-- 创建唯一索引create unique index index\_username on t\_user(userName);

https://www.cnblogs.com/summer0space/p/7247778.html-- 创建复合索引create index index\_status\_logincount on t\_user(status, loginCount);

索引的类型

【索引特性上来说】

唯一索引 普通索引

【字段的个数来说】

单个索引/单一索引 复合索引/组合索引

【字段的个数，如果对多个字段同时加索引则称之为复合索引】

聚集索引和非聚集索引：

聚集索引：主键就属于聚集索引。在一张表上聚集索引只能有一个。

非聚集索引: 除了主键以外的唯一索引，普通索引。在一张表上非聚集索引可以有多个。

唯一索引和普通索引的区别：

唯一索引所在字段中的值必须是唯一的。主键字段会自动创建唯一索引。

创建索引，不一定就能使用上索引；想要索引生效必须满足两个条件，第一，该字段上有索引；第二，所写的sql语句必须支持索引。

什么时候索引不起作用/什么时候索引会失效?

1.如果在有索引的字段上进行运算，那么索引失效，会进行全表扫描。

2.在使用like的时候如果%在前面则索引失效

全文检索技术

java:[开源的搜索引擎技术]

Lucene：

Solr【是对lucene的封装】：企业级的搜索引擎技术

ES【是对lucene的封装】：互联网级的搜索引擎技术

3.避免在查询条件中使用is null,可以采用默认值的方法来解决

大批量数据的插入/更新?

1.一条条插入

2.分段 批量 提交【推荐】【减少和数据库的交互次数】

3.批量插入和数据库交互一次

删除?delete ... in

**JAVA 反射**

java反射机制 说白了 就是在运行过程中，对于类来说，咱们可以获取它的属性和方法，对于对象来说，可以动态调用它的任意方法和属性。 动态获取信息 以及 动态调用对象的方法 就是反射机制！！！

我在项目中做导出excel功能的时候就是用的反射机制，这样就能达到导出excel的通用性，只用这么一个工具类，就可以导出不同的业务数据，比如商品，会员，品牌，分类，而且用起来也特别方便。

想要用反射首先得获取类，获取类的方式我知道的有这几种，可以通过对象的getClass()方法获取类，可以通过Class.forName("类的全称")来获取类，可以通过 类名.class 获取类。

之后可以通过 getDeclaredFields()获取类的所有属性，getDeclaredField(属性名)获取类中指定 名字的 属性,getDeclaredMethods()获取类的所有方法，获取属性后还可以通过getType()获取属性的类型，通过getName()获取属性名。

动态调用方法这块如果是get方法可以使用 属性.get(对象) 来调用get方法，如果是set方法可以通过 属性.set(对象，值)来调用，但因为get,set方法一般都是私有的，所以得先通过 属性.setAccessible(true) 来开启访问权限。如果是普通的方法可以通过 方法.invoke(对象，参数) 来调用。

=========（基础概念）==============

**SSM框架搭建的步骤**

1. 导入Spring、SpringMVC、MyBatis的依赖

2. 导入Spring、SpringMVC、MyBatis的配置文件

3. 在Spring的配置文件中配置如下内容：

(1)配置数据源

(2)配置MyBatis的会话工厂SqlSessionFactoryBean，并注入数据源

(3)配置MyBatis自动扫描DAO层Mapper接口和配置MyBatis映射文件的路径

(4)通过component-scan注解开启对service层的扫描

(5)配置事务的传播特性、隔离级别、回滚策略，让那些以add、delete、update开头的方法开启事务的提交和回滚，其它方法走的是只读事务。

(6)配置事务管理切面类，横切service层。

4. 在SpringMVC配置文件中配置如下内容：

(1)使用component-scan标签开启对controller层的扫描

(2)通过annotation-driven标签开启注解驱动

(3)配置视图解析器

(4)如果需要进行文件上传操作，还需要配置文件上传解析器，并指定文件上传的大小 限制

5. 在MyBatis的配置文件中配置如下内容：

(1)配置在控制台打印SQL语句

(2)配置开启二级缓存

(3)配置下划线自动转小驼峰

6. 在web.xml中配置如下内容

(1)配置Spring的监听，并加载Spring的配置文件

(2)配置SpringMVC的核心控制器DispatcherServlet并加载SpringMVC的配置文件

(3)为了防止中文乱码，一般还会配置一个编码过滤器并设置字符编码集为UTF-8

其实到此为止，咱们的SSM框架就算搭建完毕了，接下来就可以进行项目的开发了。

新建控制层并在类上加上@Controller注解，然后在控制层类的方法上添加@RequestMapping注解，接下来在控制层类中通过@Autowired或@Resource注解注入Service层，在Service层通过@Autowired或@Resource注解注入DAO层，这里需要注意的是，DAO层只有接口没有实现类，是通过MyBatis映射文件中的namespace指明实现的DAO层Mapper接口，并通过MyBatis映射文件中的SQL的ID指明调用DAO层Mapper接口方法时要执行的SQL语句。

**谈谈你对Spring的理解？**

Spring 的核心就是 IOC 和AOP。

IOC（Inverse Of Control）控制反转，就是将原本在程序中手动创建对象的控制权，交由Spring框架来管理。 Spring使用IOC容器来实现IOC， IOC容器实际上就是个Map（key，value）,key就是Bean的ID，value就是对应的Bean对象。通过将对象之间的相互依赖关系交给IOC容器来管理，然后IOC容器来完成对象的注入。这样可以很大程度上简化应用的开发，把应用从复杂的依赖关系中解放出来。IOC容器就像是一个工厂一样，当我们需要创建一个对象的时候，只需要配置好配置文件/注解即可，完全不用考虑对象是如何被创建出来的。 然后在需要的地方引用就行了，这大大增加了项目的可维护性且降低了开发难度。

AOP基于代理设计模式，代理分为动态代理和静态代理，在项目中默认使用的是基于JDK的动态代理，它需要接口的支持，这就是在写 service 业务逻辑层的时候通常先写接口，再写实现类的原因，如果没有接口只有类，这个时候可以使用 CGLib 这个动态代理来实现。AOP说白了就是面向切面编程，就是把一些非核心业务逻辑提取出来形成一个切面，从而让咱们程序员在编写代码时候，只用关注核心业务逻辑的处理，至于这些非核心业务逻辑统一在切面里面处理。 像日志记录，性能统计，事务处理，安全控制，这些都可以通过AOP的方式进行统一的处理。切面说的简单点就是在指定类的指定方法的前后执行特定的横切业务逻辑。切面由切点和通知构成，通知又包括方位和横切业务逻辑，切点就是为了定位指定类的指定方法，方位包括前置通知，后置通知，环绕通知，抛出异常通知，返回后通知这些。横切业务逻辑说白了就是一些公共的非核心业务代码。AOP的切面在项目中用的还是比较多的，咱们都知道事务管理就是通过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，我在项目中负责过日志管理模块，通过...[可以将基于AOP的日志统一处理融进去。

**Spring的优点？**

1) 方便解耦，简化开发：通过Spring提供的IOC容器，我们可以将对象之间的依赖关系交由Spring进行控制，避免硬编码所造成的过度程序耦合。

2) AOP面向切面编程：通过Spring提供的AOP功能可以让我们对系统中一些横切业务逻辑如安全、事务、日志等进行集中式管理，从而让这些功能得到更好的复用。

3) 声明式事务管理：Spring的声明式事务管理让我们无须通过编程的方式管理事务，只需要在配置文件中或者通过注解进行相关的规则声明，就可以将事务规则应用到业务逻辑中，提高开发效率和质量。

4) 方便集成各种优秀框：Spring不仅不排斥各种优秀的开源框架，相反，Spring可以降低各种框架的使用难度，Spring提供了对各种优秀框架（如Struts,Hibernate）等的直接支持。

**Spring的缺点？**

我说一下我的感受吧，现在Spring越来越大，支持的东西越来越多，这就导致大多数开发人员对Spring的体系及结构都一知半解，用到某个东西的时候在网上乱七八糟的文章来回粘来粘去，有些时候甚至搞得人都分不清楚到底哪种使用方式是对的，因为越来越臃肿的体系，导致Spring的学习成本倍增。

**Spring的Bean加载的过程？**

1) 通过反射实例化Bean;

2) 对bean的依赖属性进行填充

3) 如果实现了BeanNameAware接口,调用setBeanName设置Bean对象的ID或者Name;

4) 如果实现BeanFactoryAware接口,调用setBeanFactory 设置BeanFactory;

5) 如果实现ApplicationContextAware,调用setApplicationContext设置ApplicationContext

6) 调用BeanPostProcessor的预先初始化方法;

7) 调用InitializingBean的afterPropertiesSet()方法;

8) 调用定制init-method方法；

9) 调用BeanPostProcessor的后初始化方法;

**SpringMVC的运行原理**

整个处理过程从一个HTTP请求开始：

1. 客户端发送请求到 DispatcherServlet。

2. DispatcherServlet根据请求信息调用HandlerMapping，解析请求对应的Handler。

3. 解析到对应的Handler（也就是我们平常说的Controller控制器）后，开始由HandlerAdapter适配器处理。

4. HandlerAdapter会根据Handler来调用真正的处理器开处理请求，并处理相应的业务逻辑。

5. 处理器处理完业务后，会返回一个ModelAndView对象，Model是返回的数据对象，View是个逻辑上的View。

6. ViewResolver会根据逻辑View查找实际的View。

7. DispaterServlet把返回的Model传给View进行视图渲染。

8. 最后DispatcherServlet将渲染好的视图响应给客户端

**SpringMVC的常用注解有哪些？**

@RequestMapping：用来设置方法的请求地址。它呢可以用在类上也可以用在方法上，用在类上的话，则访问所有的请求方法时都需要加上该地址作为父路径。

@GetMapping：也是用来设置方法的请求地址的，但是只支持get请求。一般在进行查询操作的时候会用到这个注解。

@PostMapping：也是用来设置方法的请求地址的，但是只支持post请求。一般在进行新增操作或者传递的参数比较多的情况下会用到这个注解。

@DeleteMapping：也是用来设置方法的请求地址的，但是只支持delete请求。一般在进行删除操作的时候会用到这个注解。

@PutMapping：也是用来设置方法的请求地址的，但是只支持delete请求。一般在进行修改操作的时候会用到这个注解。

@ResponseBody：用于将服务器端返回java对象转为JSON数据。

@RequestBody: 用于将客户端传递过来的JSON数据转换为java对象。

@RestController: 它是一个组合注解，组合了@Controller和@ResponseBody这两个注解。如果说某个Controller所有的方法都需要返回JSON格式的数据，直接在Controller类上加上这个注解，这样就不用我们在每个方法上再单独添加@ResponseBody注解了。

@PathVariable：用来获取路径参数

@RequestParam：用来设置请求参数和请求方法里的参数的映射，一般用于请求参数名和请求方法中参数名不一致的情况。

**@Resource和@Autowired的区别？**

@Autowired是Spring提供的注解，@Resource是JDK提供的注解。

@Autowired只能按类型注入，@Resource默认按照名称注入，也可以通过类型注入。

@Resource默认按照名称进行注入，如果注入失败的话会转为按类型进行注入，如果按类型也注入失败会抛出异常。

**MyBatis是什么?**

MyBatis是一个持久化框架，是一个半自动ORM 框架，它的本质是对JDBC的封装。MyBatis不仅支持自定义SQL语句、存储过程，而且使用MyBatis不用我们手动设置SQL语句中参数，也不需要我们手动把查询出来的结果集映射为java对象。在使用传统JDBC时，我们往往需要写很多JDBC代码，比如说加载驱动、创建连接、创建statement。然后还需要自己写SQL语句以及自己手动给SQL语句中参数赋值，最后自己对结果集进行封装处理，而MyBatis帮我们简化了以上功能，只需要一些配置文件（xml）或是注解的方式即可完成对数据库的查询以及结果的映射封装。相比于原生JDBC而言，使用MyBatis让我们只需要关注SQL语句本身，而不需要花费精力去处理加载驱动、创建连接、创建statement等这些和SQL语句无关的繁琐过程。

**MyBatis的优点有哪些？**

第一实现了SQL语句和代码的分离，解除了SQL语句与代码的耦合。第二使用MyBatis可以自动将查询结果集转换为java对象。第三使用MyBatis无需我们手动设置SQL语句中的参数。第四MyBatis提供了很多xml标签方便我们编写动态SQL语句（比如说where,if,foreach,sql,include等标签）。

**MyBatis的缺点有哪些？**

第一就是SQL语句的编写工作量较大，尤其是字段多、关联表多时更是如此，对开发人员编写SQL语句的功底有一定要求。第二SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库，特别是在SQL语句中使用到一些函数的时候移植性更差。第三由于映射文件里SQL语句的id必须唯一，导致DAO层Mapper接口中的方法不支持方法重载。第四不支持级联更新、级联删除。

**MyBatis和Hibernate的区别(至少3点)?**

MyBatis容易掌握，只要会写SQL语句就行了。而Hibernate门槛较高，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡取得平衡，以及怎样用好Hibernate方面需要你的经验和能力都很强才行。

MyBatis可以进行更为精细的SQL优化，而Hibernate要进行SQL优化会比较困难一些。

Hibernate无需写SQL就可以完成单表简单增删改查，而MyBatis单表简单增删改查还需要自己写SQL语句。

Hibernate数据库移植性很好，不同的数据库需要写不同SQL。

Hibernate有更好的二级缓存机制，可以使用第三方缓存。MyBatis本身提供的缓存机制不佳。

**MyBatis#{}占位符和${}拼接符的区别？**

#传入的参数在SQL中显示为字符串，#方式能够很大程度防止sql注入；$传入的参数在SqL中直接显示为传入的值，$方式无法防止Sql注入。

Select ${cloumn1},${cloumun2} from ${table\_name}

**Mybatis中映射文件和Mapper接口有什么对应关系吗？**

映射文件中的mapper标签的namespace属性值必须要和对应的Mapper接口路径保持一致

映射文件中SQL语句的ID要和Mapper接口中的方法名保持一致

映射文件中SQL语句的resultType要和Mapper接口中的方法的返回值类型保持一致

映射文件中SQL语句的parameterType要和Mapper接口中方法的参数类型保持一致