# B简历：

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

## 基本信息

|  |  |  |  |
| --- | --- | --- | --- |
| **基本情况** | | | |
| **姓名** | 崔少权 | 性别 | 男 |
| **出生日期** | 1996-08-16 | 属相 | 鼠 |
| **现住地** | 天通苑 天通北苑三区 |  |  |
| **大学情况** | | | |
| **学校** | **武昌工学院** | 专业 | **计算机科学与技术** |
| **入学时间** | 2014-9 | 毕业时间 | 2018-7 |
| **专业课** | **[java语言](https://www.baidu.com/s?wd=Java%E8%AF%AD%E8%A8%80&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)程序设计、Java[企业级开发](https://www.baidu.com/s?wd=%E4%BC%81%E4%B8%9A%E7%BA%A7%E5%BC%80%E5%8F%91&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)、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** | 武汉市洪山区白沙洲大道110号 | 校长 | **李勇** |

|  |  |
| --- | --- |
| **去学校的方式：** | **从菏泽火车站坐火车到武昌站（k1073 十个小时 86￥ 22：25-08：26）**  **公交：中山路武昌火车站638/906/202/至白沙洲大道青凌乡 步行364m 工学院路** |
| **上小学日期：** | **2002年9月 6岁** |
| **上初中日期：** | **2008年9月 12岁** |
| **上高中日期：** | **2011年9月 15岁** |
| **上大学日期：** | **2014年9月 18岁** |
| **毕业时间：** | **2018年 7月 22岁** |
| **入学时间：** | **2014年9月** |
| **毕业时间：** | **2018年7月** |
| **在校学生：** | **5000左右** |
| **有教师：** | **不太清楚** |

|  |  |
| --- | --- |
| 第一家公司 ： | 河南星河同频高科技发展有限公司 |
| **地址：** | 郑州金水区金岸大厦(政七街)515 |
| **入职时间：** | **2017年8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：** | **刘伟华** |
| **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：** | 金龙小区，就在公司附近也就两公里，自己骑电动车 |

|  |  |
| --- | --- |
| **第二家公司情况：** | 杭州本恒科技有限公司(只说公司简称) |
| **公司地址**： | 杭州萧山区农业大厦(市心北路)2幢28L |
| **公司规模：** | **80人左右，我们项目组8人，一个项目经理，4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：** | **互联网和虚拟现实的软件产品的开发。** |
| **去公司路线：** | **城中花园（市心中路）地铁2号线 人民广场→振宁路 步行500米** |
| **入职时间：** | **2019年2月** |
| **离职时间：** | **2021年3月初** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **陈月崇** |
| **税前：** | **15K** |
| **税后：** | **13K** |
| **缴税：** | **2K左右(具体原因不清楚当时公司就是这样扣的税)** |
| **期望薪资：** | **16K** |
| **薪资的构成:** | **岗位工资(5K)+绩效工资(10K)** |

|  |
| --- |
| **四年涨薪：**  **2017年工作的时候是5.5K，税后大概5.4K左右，**  **转正6K左右**  **2018上半年涨了1K左右 ，总7K左右。**  **2019年2月跳槽涨了3K左右 总10K**  **2019年12月涨1K总 11K左右**  **2020年3月工作一年涨了2K左右**  **2020年11月工作一年涨了2K左右**  **2021年3月初离职时工资为15K** |

**面试常见问题：**

|  |
| --- |
| **你为什么从上家公司离职？** |
| **上家公司呆的时间也挺久的，想换个工作环境，不断的提高充实自己**  **你的优缺点是啥？** |
| **优点:** **善于倾听，团队协作能力强，责任心强。**  **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 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 |

**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、**

**介绍流程：**

**先主动介绍：**

**您好，今天我是来应聘咱们公司的Java开发工程师，然后**

**我最近的一个项目是spring boot+spring cloud来做的一个微服务项目开发**

**那么首先我先介绍一下spring boot吧**

Spring Boot 是 Spring 开源组织下的子项目，是 Spring 组件一站式解决方案，主要是简化了我们使用 Spring 的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手。开箱即用，约定大于配置，内置web容器

我们在使用的时候构建spring boot项目有两种方式：

* + 1. 第一种 是通过springboot官方提供的方式进行一键式创建，但是前提是需要有外网
    2. 第二种是 是先创建一个maven项目，然后在我们的pom文件中声明springboot的启动器，声明jdk版本，引入启动器

SpringBoot运行原理:

我们在启动springboot项目的时候可以在启动类中看到最核心的两行代码， @SpringBootApplication注解和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。

--用于定义配置类，指出该类是 Bean 配置的信息源，相当于传统的xml配置文件，一般加在主类上。如果有些第三方库需要用到xml文件，建议仍然通过@Configuration类作为项目的配置主类——可以使用@ImportResource注解加载xml配置文件。--

@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。

--组件扫描。让spring Boot扫描到Configuration类并把它加入到程序上下文

@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。

允许 Spring Boot 自动配置注解，开启这个注解之后，Spring Boot 就能根据当前类路径下的包或者类来配置 Spring Bean。

**讲述对springcloud微服务的理解**

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。

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。

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，我们是写了一个发切面或者拦截器，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。

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

# 功能模块

## 基于token的登陆

用户点击登录 把用户名和密码传过去 服务端接收账号密码进行判断账号是否存在，

存在的话再判断密码是否正确，如果正就使用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。

## 权限管理的实现

权限管理对我们目前来说主要通过三块来进行实现。

1设型:RBAC 模型

RBAC 模型:也就是基于角色的权限控制，有别于传统模型中的直接把权限赋予账号. RBAC 模型增加了“角色"的概念，把权限赋予角色，再把角色赋予用户，提高了账号的管理效率，降低了出错的概率。

2

数据库设计:权限五表<用户表，角色表，权限表，用户角色关系表，用户权限关系表>用户表、角色表、权限表都根据需求进行相应设计，同时由于用户和角色角色和权限都是多对多的关系，所以引入了角色用户关系表和角色权限关系表来配置他们之间的多对多关系。

3、 后台系统设计:三个模块<用户模块、角色模块、权限模块>

1:用户新增或者修改的时候可以对角色进行勾选进行关联，角色是通过数据库查出来，然后展示在新增或者修改的弹窗中，以复选框或者下拉框进行展示，方使用户勾选。

2)角色模块:用户在新增或者修改角色的时候可以对权限勾选进行关联，权限是通过数据库查出来，然后以 zTree 插件展示在新增或者修改的弹窗中，另外对角色进行副除时还需要查看此角色是否被用户占用，如占用则不能删除。

3)权限模块:使用zTree插件进行展示，同时在删除时要把选中节点的所有子节点一并删除，另外还需要查看此权限是否被角色占用，如占用则不能删除，同时在新增或者修改时必须选中一个节点才可以新增或者修改，否则将进行弹窗提示。

## 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。

## 多线程导出

线程的实现方式的话就是实现一个Runnable接口，还有继承Thread类重写它的run方法，在里面写具体实现，在主线程中创建线程调用一个start方法就创建好了，线程的五个状态的话就是，你创建Thread的时候就是新建状态，然后调用start方法的时候就就绪状态，当cpu把时间片分配给你的时候，这个线程就是运行状态，其他线程的话就在就绪状态中排队，当你执行完就到了死亡状态，如果你没有执行完，cpu把时间片分给了别人，这时候你就会回到就绪状态，等待着cpu选到你，还有个阻塞状态的话，就是当你加锁的情况下，a线程拿到了锁，但是cou分配的时间到了，你没执行完，就会把资源锁住，这时候b线程被cpu选到了，它去执行一看被锁了，这时候b线程就会进入到阻塞状态，直到等到a线程执行完，把锁给释放了b线程才会从阻塞状态出来

我当时的话是用它做了一个poi的导出，当时一开始是正常的通过反射加自定义注解通过这个HSSFWorkbook做一个导出，但是发现效率太慢，数据过多，之后我就把每创建一个Excel就启动一个线程，然后每次查询只查一个sheet页的数据，这样确实快了很多，但是还是不完美，之后我又加了线程池，把线程数设置最大线程数，这样线程也不需要频繁的创建和销毁了，就快完成的时候了，一下载发现导出来的是个空的，之后一想，多线程是异步的，线程还没完成，主线程是不等待的，直接执行了，然后我就百度搜索解决方案，最后找到了CountDownLatch这个方法，把线程的总数给它 ，在主线程最后下载的地方调用一个await方法，这个方法就是让你等待，当CountDownLatch数量到0的时候才会执行， 然后再线程内调用countDown方法，这个就是每次执行了这个方法就会线程减一当，减到0就证明所有线程走完了，然后主线程就会执行，加了这个之后就把这个问题解决了

定时任务

（发邮件 发短信 会员续费）

我当时是做一个定时发短信的任务，就是用户生日的时候发一封短信，

我们当时是分布式框架spring+springboot+springCloud定时器我们用的是springboot的@Scheduled 定时器我们只需要创建一个类加上三个注解@Component 交给ioc进行管@Configuration声明这是一个文件@EnableScheduling //启用调度器然后规定一个定时时间定时扫描就行了当时我们是把所有用户的信息查询出来，然后进行便利看对比今天的时间是哪个用户的生日，然后就进行起一个线程去发邮件，然后再继续便利，单个服务的时候就好好的，就是集群的时候，它到了这个时间段就一下发了好多封短信，每个集群的服务都调用了这个方法，这时候我们就借助了redis，通过哪个getAndSet方法取和存都是一步操作，然后就判断没值的时候就往下运行，有值的时候就直接返回，正当发送的时候，执行完getAndset报错了，找到错之后去运行发现，现在判断根本就过不了了，redis中数据一直存在了，这就是另一个小问题，就相当于你a进程拿到了这个锁，但是他死了，锁也就跟着没了，下面的进程永远都进不去了，这样的话永远也发不了短信了，然后我就百度了一下知道了redis中能给key设置一个失效时间，这样就算他报错了，一定时间，锁也会自动消失

## aop日志记录

首先，日志记录我们是通过aop+自定义注解来做的

为什么用aop呢，就是因为aop是不改变原代码的情况下增加新的功能，如果你不用的话你需要再每个方法都进行记录一下日志，这样是非常繁琐的，所以我用了aop思想，

然后我们创建一个类，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

反射的常用方法

①、得到 Class 的三种方式

//1、通过类对象调用 getClass() 方法来获取

　　Person p1 = new Person();

　　Class c1 = p1.getClass();

//2、直接通过 类名.class 的方式得到,该方法最为安全可靠，程序性能更高

　　Class c2 = Person.class;

//3、通过 Class 对象的 forName() 静态方法来获取，用的最多，

// 但可能抛出 ClassNotFoundException 异常

　　Class c3 = Class.forName("com.ys.reflex.Person");

其中有很多方法，比如说

getModifiers();获取类上的修饰符

　　getName()：获得类的完整名字。

getPackage(): 获得该类所在的包路径

getDeclaredAnnotations(); 获取该类上所有注解

　　getFields()：获得类的public类型的属性。

　　getDeclaredFields()：获得类的所有属性。包括private 声明的和继承类

　　getMethods()：获得类的public类型的方法。

　　getDeclaredMethods()：获得类的所有方法。包括private 声明的和继承类

　　getConstructors()：获得类的public类型的构造方法。

## 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。

## Redis篇：

### 1.redis是什么

redis是非关系型数据库 也是一个key value的数据库

redis效率非常快 读取11000次每秒 写入是8100次每秒

为什么效率这么快 因为redis是基于内存的 所以效率快

redis有两种持久化方案 rdb和aof

reb相当于快照形式 效率快 但是丢失数据会丢失一整片

aof是日志形式的 效率慢 但是最多丢失最后一条数据

redis的所有操作都是原子性的 redsi是单线程 事务单个执行

### 2、redis的数据类型

**String字符串**

我们用string类型做过解决接口幂等性 还可以用来计数 incr key自增 decr key自减

还存放了我们的地区数据 部门数据 **这些数据的特性是什么？**（数据访问量大切数据不经常改变）

**Hash（哈希）购物车**

我们的购物车数据是用hash存在redis中的

**List（列表）优惠券**

就rpop(弹出)redis数据库中存储的数据 就是去除数组中的最后一条数据

**zset(sorted set：有序集合) Set（集合）**

不好意思 没用过！！！

### 缓存【redis】穿透：

缓存穿透是指查询一个一定不存在的数据，比如商品最多只有1000个，我查询1001,1002这些一定不存在的id,这个时候因为查询的信息在缓存中不存在，大量的并发请求就会直接到数据库上，而数据库中也不存在这样的数据，所以缓存就形同虚设，这个id一定不存在的请求以后都会到数据库上，增加数据库的压力。

为了解决这个问题，可以缓存空值,也就是说即便在数据库中 查询的数据不存在，也会在缓存中缓存个空值，这样下一次就可以走缓存而不是数据库了，但太多的空值也会占用内存空间，导致内存的浪费，可以通过设置过期时间比如3-5分钟，这样当到期后，空值对应的key就会消失；释放空间；但导致的另外一个问题就是如果在这3-5分钟内后台增加数据，这个时候就有id为1001的数据了，而缓存中还是空，所以就会导致数据库和缓存中数据的不一致，为了解决这个问题可以在后台增加数据的同时主动更新缓存中的数据。

### 缓存击穿：

缓存击穿是指当某一key的缓存过期时大并发量的请求同时访问此key，

瞬间击穿缓存服务器直接访问数据库，让数据库处于负载的情况。

如何解决缓存击穿一般有2种解决办法，分别是异步定时更新和互斥锁

异步定时更新

在缓存处理上，同理，比如某一个热点数据的过期时间是1小时，那么每59分钟，通过定时任务去更新这个热点key，并重新设置其过期时间。

互斥锁

就是当Redis中根据key获得的value值为空时，先锁上，然后从数据库

加载，加载完毕，释放锁。若其他线程也在请求该key时，发现获取锁失败，则先阻塞。

### 缓存【redis】雪崩：

如果缓存在一段时间内同时失效，发生大量的缓存穿透，所有的查询都落在数据库上，造成了缓存雪崩。

解决这个问题可以

1.为不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀

2.做双缓存策略。

A1为原始缓存，A2为拷贝缓存，A1失效时，可以访问A2，

A1缓存失效时间设置为短期，A2设置为长期。

### redis持久化：

有AOF和RDB两种方式,我们一般使用默认的RDB方式。因为它的效率会更高，

RDB持久化是在指定的时间间隔内将内存中的 数据 写入到硬盘的rdb文件中，实际操作过程是fork[分]一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用 二进制 压缩存储。

具体是在redis.conf中配置的，找到save关键字， 指定每隔多长时间有多少个key发生变化，就将其持久化。

AOF：是将每个 写操作对应的指令 保存到后缀名为aof的 文本 文件中。

rdb的性能要高于aof：

因为rdb中直接存储的是数据，所以在redis服务器启动的时候，直接加载rdb文件中的数据到内存中就行了。

而aof中存储的是写指令，所以在redis服务器启动的时候，需要先加载aof

文件中的指令，之后还需要再次执行这些指令。

aof的 实时性、可靠性 要高于rdb：

因为每个写操作指令都会及时写入aof文件中，所以aof这种持久策略不容易丢失数据，而rdb是在指定的时间间隔后才会写入数据到文件中，所以有可能会丢失一部分数据。

## 优化：

创建索引的时候就要遵守创建索引的原则：

1. 给频繁查询的字段上创建索引

2. 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

3. 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。

打个比方，我们都知道我们地区的三级联动，这个时候就可以使用我们的复合索引，这样就可以加快我们的查询速度，而且我们也不可能直接跳过省分去查市

从索引的特性上来说索引可以分为普通索引和唯一索引：

普通索引字段的值可以重复,而唯一索引字段的值则不能重复，必须保证唯一，而且我们的主键会自动生成唯一索引。

创建索引的具体语法是这样的：

唯一索引：create unique index 索引名 on 表名(字段名);

普通索引只是少了我们这个unique这个唯一的关键字，而创建复合索引则只是比创建普通索引多了字段名，也是说字段名可以设置多个

普通索引：create index 索引名 on 表名(字段名);

复合索引：：create unique index 索引名 on 表名(字段名1，字段名2);

但是我们这个索引也不是说我们只要创建就一定能使用上索引，想要使索引生效，还要满足两个条件，

第一，该字段上有索引；第二，所写的sql语句必须支持索引。

那么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。

这个当时的解决方案就是不在我们的索引字段上进行运算

2.在使用like的时候如果%在前面则索引失效，当时是这样解决的

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。

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。

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。

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。以上为我所了解的sql优化，

## Spring：

### 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 中的 bean 的作用域有哪些?

singleton : 单例，每个bean都只会创建一个实例，Spring 中的 bean 默认都是单例的。

prototype : 原型，每次从IOC容器中获取都会都会产生一个新的bean实例。

request : 每次HTTP请求都会产生一个新的bean实例，这个实例仅在当前请求内有效。

session : 每个HTTP会话都只会创建一个 bean实例，这个实例只在当前会话内有效。

global-session： 所有的HTTP会话都只会创建一个bean实例。

### Spring中的Bean为什么默认是单例的？有什么缺点吗？

一句话就是为了提高性能，具体点：

1. 由于不会每次都新创建新对象，所以就减少了新生成实例的消耗。因为spring会通过反射或者cglib来生成bean实例这都是耗性能的操作，其次给对象分配内存也会涉及复杂算法。
2. 减少JVM垃圾回收，由于不会给每个请求都新生成bean实例，所以自然回收的对象少了。

第三，可以快速获取到bean，因为单例的获取bean操作除了第一次生成之外其余的都是从缓存里获取的所以很快。

缺点就是在并发环境下可能会出现线程安全问题。

### 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

## tomcat调优

1.启用线程池，默认的tomcat没有启用线程池，

在tomcat中每一个用户请求都是一个线程，所以可以使用线程池提高性能。

我们在项目中将maxThreads[最大线程数]设置为1000，将排队数acceptCount和maxThreads设置相等。

当tomcat的线程数达到maxThreads后，新的请求就会排队等待，超过排队数的请求会被拒绝。

2.使用64位的tomcat和jdk，禁用AJP协议。

3.开启APR通讯模式，支持高并发。因为默认tomcat采用的是性能最低的BIO【阻塞IO】模式。

4.tomcat中设置JVM参数 通过-server开启server模式 通过-Xms4000m和-Xmx4000m设置初始堆大小和最大堆大小，

通常将两个值设置为一样，避免堆空间不断增大和缩小所带来的性能损耗。

## 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。

# 被动说：

## 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 key-value 对，

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。

Jdk1.8中引入了红黑树，目的是避免单条链表过长而影响查询效率。

## 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。

## Hashtable与HashMap的区别

1.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

4.HashTable之所以是线程安全的是因为在它的方法上加上了synchronized锁。

5.LinkedHashMap是有序的

## Set,List,Collection(可来可神),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(v可特)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。

## String,StringBuffer,StringBuilder的区别

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.对于大数据量的字符串的拼接，采用StringBuffer,StringBuilder.

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<bean id="" class=""/>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]