# B简历：

## （1）基本信息

|  |  |  |  |
| --- | --- | --- | --- |
| 基本情况 | | | |
| **姓名** | 刘文浩 | **性别** | 男 |
| **出生日期** | 1995-11-03 | 属相 | 猪 |
| **现住地** |  |  |  |
| **大学情况** | | | |
| **学校** | 青岛滨海学院 | **专业** | 信息管理与信息统计 |
| **入学时间** | 2013-9-1 | 毕业时间 | 2017-7-1 |
| **专业课** | 面向对象程序设计、数据结构、数据库原理与应用、大数据技术原理与应用 | | |
| **学历** | **本科** | 学位 | **学士学位** |
| **学校地址** | 山东省青岛市黄岛区嘉陵江西路425号  (公共汽车站台青职学院站) | 校长 | 忘了（韩方希） |
| **去学校的方式：** | 坐火车到青岛,坐长途汽车到黄岛汽车站->直接打的1,20元 | | |
| **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：** | **20000左右** | | |
| **有教师：** | **不太清楚，大概1000多个** | | |
| **分数线：** | **普本471** | | |

|  |  |
| --- | --- |
| 第一家公司 ： | 济南地平线信息科技有限公司（济南地平线） |
| **地址：** | 注册地址：山东省济南市历下区文化东路78号万豪中心7号楼913室，2015年注册  工作地址：历下香格里拉,东楼还是北楼记不清了;(详细:济南历下区汇展香格里拉(北门)东北塔30楼3024) |
| **入职时间：** | **2017年8月** |
| **离职时间：** | **2018年11月** |
| **公司性质：** | **民营** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **徐海洋** |
| **薪资方面：** | **毕业实习4.5K左右,转正6000，干了一年7000** |
| **路线：** | 忘了怎么走了，当时也是在香格里拉那租的房子，就在公司下边 |

|  |  |
| --- | --- |
| **第二家公司情况：** | 山东云驰信息技术有限公司(山东云驰)为济南源创软件有限公司同团队运营的别名公司，2002年成立 |
| 公司地址： | 注册地址：山东省济南市历下区解放东路56号金泉大厦B座1704室  工作地址：东环国际广场或者东环国际; (详细:济南历城区东环国际广场B座1404室) |
| **公司规模：** | **7，80人左右，我们项目组9人，一个项目经理，4名开发人员，2名前端，1个测试，1个运维** |
| **公司性质：** | **民营** |
| **公司行业：** | 主要业务方向包括政府综合应用、云服务、大数据、物联网等 |
| **去公司路线：** | 住百花小区走到路口，直行，公共汽车：46路/70路/93路;走路4,50分钟。 |
| **入职时间：** | **2018年12月** |
| **离职时间：** | **2021年3月初** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **江澈** |
| **初入职工作：** | **10k** |
| **离职前税前：** | **15K** |
| **离职前税后：** | **13K多点** |
| **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：**

|  |  |
| --- | --- |
| **你为什么从上家公司离职？** | |
| **想换个工作环境，不断的提高充实自己，父母也在这边** | |
| **你交社保了吗？为啥没交？** | |
| **没交过**  **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一部分,要是不上可以把钱返回来，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，抗压自学能力强 (**可以综合自己自学的知识表达**) | | **你的优缺点是啥？** |   **优点:**  **虚心受教，抗压能力强，乐于分享，团队协作能力强，**  **缺点:平时话比较少，不过现在已经改进的差不多了，** |
| **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** |
| **看具体情况，一般加班都可以接收** |
| **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？ |
| **继续加强自己的技术功底，以及综合能力，然后向架构方向扩展** |
| 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**  220左右  **项目奖金**  8k-10k，做完项目，收完尾款的时候发。  **年终奖**  就是1个月的工资，  最后一个月和工资一块儿发/年后发  **电话补助**  260  **饭/餐补**  **15、**30/天  **电脑补**  310 |

**公司内其他部门：**

**技术部，人事部，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！**

## 业务功能模块

### 电商项目：

**下订单（幂等性，减库存），支付接口（支付宝，微信），JwtToken登录，aop日志记录，**

**多线程poi导出（线程池），定时器**

**Springcloud微服务架构，springboot基础服务，Springmvc，分布式框架，跨域，restful，**

**dubbo，rabbitMQ，Jdk1.8（lambda，函数式接口），反射，注解，容器（hashmap），事务，**

**锁（同步锁，死锁），**

**Mysql，mnongDB，redis，**

**docker，git，**

在上家公司最近做的是一个电商项目，当时使用的是分布式框架，我负责的主要是在后端这边的东西，在初期的设计方面：后端我们计划使用基于springboot的Spring Cloud微服务架构模式，然后前端那边使用的是Vue来做的，这就产生了一个问题，(**跨域**)由于浏览器的同源策略限制。同源(是指在同一个域)就是两个页面具有相同的协议(protocol）)，主机(host)和端口号(port)，如果不同源，那么同源策略它就会阻止一个域的js脚本和另外一个域的内容进行交互，这是浏览器最核心也是最基本的安全功能，所以在正常情况下，Vue的请求是传不到后台的，在以往的单体架构中，是见不到这种情况的，它的解决方法也很简单，方法也有好几种，比如jsonp这种现在用的比较少了，原因之一就是它只支持get请求，不支持post请求，最简单的方法就是在控制层加个@CrossOrigin注解来为整个控制层都进行跨域，但是这种方法集成程度太高了，出了问题不好找，还有就是通过CorsFilter跨域过滤器来解决，Spring MVC在 4.2以后内置了一个CorsFilter专门来处理CORS请求问题，也可以通过配置这个Filter，对AllowedOrigin进行设置来进行统一控制跨域请求，我平时这种方法使用的比较多，还有一些比如Vue前台代理，在Nginx进行统一配置，另外spring mvc还提供了一种WebMvcConfigurer的方法，总之跨域的解决方法还是挺多的，再回到项目设计上，解决了跨域的问题，我们的前台请求就可以访问到后台的接口了，而不管是在前后台的请求还是SpringCloud各个微服务之间的调用，我们都默认遵循一个**restful**风格接口，restful风格接口其实就是规定我们不同的请求调用不同的接口，不同的接口执行不同操作，比如前台发送get请求，我们后台接口中就使用@GetMapping接收，这个接口执行的一般就都是查询的操作，post请求就用@PostMapping接收，执行的一般是添加，常用的还有Put和Delete等等，这种风格的接口可以让我们一眼就看出来这个接口大概是用来干什么的，如果使用了restful风格的接口，前台请求的方法和后台映射路径的注解不对应的话，会报错。

#### 多线程poi导出（线程池），

* 我们用多线程导出 使用了 线程池 注解 java反射
* 首先我们自己设置每个文档要创建多少个sheet页 每个sheet页中有多少数据
* 第一步先查询出我们数据库中有多少数据
* 第二部 通过总条数除掉设置的每个sheet页多少条数据 计算出要创建多少sheet页 先% 有余数就加一；
* 第三部 通过总条数除掉计算出来的总sheet页 计算出要创建多少excel文档 先% 有余数就加一；
* 第四步 因为我们的需求是多线程进行导出 所以 要等所有的excel文档创建完成以后在进行下一步添加到zip压缩包中 所以就用到了 coutdownlatch 来进行计算 将创建excel文档的数量作为总数量 然后 对excel的总数量进行循环遍历 因为频繁的创建和销毁线程会浪费资源 并且每次创建线程都会浪费性能 所以我用了线程池 这样每次用的时候去线程池里面取就行了 不用频繁的创建 既节约资源 还提高了性能 。
* 第五步 多线程中 创建excel文档 我用的xsshwork 创建完XSSHFWORK对象 然后计算出当前文档所需要的数据 limit所需要的起始值和每页条数 然后循环遍历要创建的sheet页条数 创建sheet页 第一行需要的是他们每一列字段的名称 我们用的是注解的方式 在需要导出的字段上家上一个自定义注解 然后获取查询出来数据的某一条数据 根据java反射机制来获取他上面的标识 循环遍历 设置到第一行 当excel页创建完成后 我们就可以通过zip流将excel所保存的文件夹 写入压缩包中

#### Springcloud微服务架构，

##### 1.Eureka

Eureka是服务注册中心，分别是eurekaserver服务端和eurekaclient客户端

eurekaserver服务端：里面有个注册中心，注册中心保存的是各个服务所在的机器和端口号

eurekaclient客户端：将服务注册到注册中心中并且发现服务

工作原理：在启动应用后，eureka客户端会向服务器注册自己的服务，同时服务端的信息会缓存到本地,客户端会向服务器周期性的心跳交互更新服务信息。

##### 2.ribbon组建介绍

ribbon在springcloud中主要做负载均衡的作用.它是基于http协议和tcp协议的客户端负载均衡,使得面向REST请求时变成客户端的负载均衡调用.

##### 3.feign组建介绍

feign 组建其实是整合和ribbon的.feign主要是进行远程调用,它通过@FeginClient的注解,进行动态代理(要是某个接口上面标注有这个注解,那么feign会对这个接口进行动态代理)

feign的动态代理,会根据你请求的@RequestMapping 和参数,来动态构造你要请求的服务地址,然后根据地址发送请求.

如果服务部署了多台机器,那么feign会通过负载均衡ribbon的轮询机制确实服务机器(ribbon默认采用轮询机制).

##### 4.Hystrix组建

在微服务架构中,一个系统会有多个服务,服务之间的相互调用如果不做任何保护就会导致服务器雪崩问题.那么Hystrix就是用来解决这些问题的.Hystrix是隔离,熔断以及降级的一个框架,

在某个服务挂掉之后,Hystrix会熔断这个服务,直接返回.(降级:在这个服务熔断之后,记录这期间服务请求的信息,以便后期维护).

##### 5.zuul网关

zuul:微服务网关,主要是负责网络路由的.

在微服务的架构中,可能后台会部署几百个服务,所有的服务名称前端不可能全部记住.那么现在就需要网关,前端通过浏览器发送请求,进入到网关zuul,网关会根据请求的一些特征,将请求转发到后端的服务.

网关还可以做统一的服务降级,限流,认证权限安全等等.

##### 1.springcloud 与dubbo的区别

服务调用方式不同,dubbo是采用rpc远程调用,srpingcloud是采用rest api远程调用.

注册中心:dubbo是采用zookeeper,springcloud是采用eureka

服务网关不同:dubbo本身是没有实现的,只是整合了第三方的技术.而springcloud是有一个组建zuul作为路由网关,springcloud还支持断路器,与git完美集成分布式配置文件支持版本控制.

##### 2.微服务之间是如何独立通信的?

feign远程调用.优点:没有中间代理件,简单常见.缺点:可用性低,在请求过程中服务端和客户端都必须是可用的

消息中间件.优点:解耦,提高可用性,可以支持多种相应模式(如:发布/订阅,请求/异步响应等等).缺点:增加系统的复杂度,万一消息中间件挂了,整个服务就挂了.

#### Springboot基础服务，

快速构建项目和简化开发，最大的的特点就是自动配置，项目里面我们不需要再去进行大量的xml配置，只需要引入相应的启动类，有需要的就在配置文件中进行相应的配置，搭建速度非常快，项目启动也非常快，并且开箱即用，部署也非常简单，因为springboot内嵌了tomcat，我们只需要吧把项目打成jar包，使用java-jar包名.jar一键式启动项目

##### @springbootapplication

1. @SpringBootConfiguration又组合了@Configuration注解，也就是把该类变成一个配置类所以实现配置文件的功能。

2.@EnableAutoConfiguration：打开自动配置的功能，也可以使用@SpringBootApplication注 解的exclude属性也可以关闭某个自动配置的选项，如关闭数据源自动配置功能： @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。

3.@ComponentScan注解就是进行Spring组件扫描的。该注解默认会扫描该类所在的包下所有的配置类

#### Springmvc，

是一个控制层框架; 主要用来接参传参;页面跳转;

**SpringMVC 的运行原理**

1. 客户端发出请求第一个接受请求的组件是DispatcherServlet.(前端控制器模式)  
   2、DispatcherServlet开始查找一个或多个HandlerMapping。一个HandlerMapping的工作主要是将URL映射到一个控制器对象。  
   3、一旦DispatcherServlet找到了一个控制器对象，它将请求分派给这个控制器，让它根据设计的业务逻辑处理这个请求。

4。完成业务逻辑后，控制器返回一个ModelAndView给DispatcherServlet。ModelAndView携带的是一个视图对象，或者是一个视图对象的逻辑名  
5。如果MedelAndView对象携带的是一个视图对象的逻辑名，DispactherServlet需要一个ViewResolver来查找用于渲染回应的视图对象。最后，DispatcherServlet将请求分派给ModelAndView对象制定的视图对象。视图对象负责渲染返回给客户的回应。

**springMVC搭建**

导入jar包

在web中配置核心控制器，也就是核心请求分发器

把spring-mvc.xml文件放到resources目录下

配置开启注解和扫描注解(一般扫描到controller包下)以及视图解析器

新建controller目录，目录下创建\*controller类 在类上加@Controller注解声明是一个控制类 @RequestMapping注解可加可不加，在controller类的方法上一定要加上@RequestMapping， 因为前台请求是根据requestmapping中的值来找到对应的方法的

Springmvc接收值得几种方式？

对象接收

属性接收

HttpServletRequest接收

@RequestParam接收

**SpringMVC跳转页面的方式**

1，通过modelandview 构造函数 可以指定返回的页面 也可以通过setViewName方法跳转

2，返回类型是String的时候直接return页面 可以把要返回的数据放入到model或者modelMap中

3，通过request.getRequestDispartcherServlet返回

4，如果返回json的话 可以加@responsebody注解 把数据返回到当前页面

**Springmvc是如何跳转页面的**

1. 默认是forward

2. 重定向 返回的时候在return或者setViewName中 加上redirect:请求

**forward和redirect的区别**

(1)forward(转发)：

1.是服务器内部的重定向，客户端浏览器的网址是不会发生变化的。

2.关于request: 由于在整个定向的过程中用的是同一个request，因此forward会将request的信息带到被重定向的jsp或者servlet中使用。

(2)redirect（重定向）：

1.是客户端的重定向，浏览器中显示的url网址会发生变化。

2.因为这种方式比forward多了一次网络请求，因此效率会低于forward。

#### 分布式框架，

无论是那种SOA的架构设计，都离不开几个模块的功能，即Provider，Consumer，Registry，Gateway，负载均衡，服务分流，监控等，

（1）Provider：服务提供者，无论是业务服务，还是一个系统中公用的SAAS，都属于Provider

（2）Consumer：即发起调用的客户端

（3）Registry：服务注册中心，是分布式服务系统中的一个重要组成模块，管理Provider的Manager，在实际的运行环境中，服务注册中心Registry被动通知或Consumer主动询问，在Provider有节点宕机或新增节点时，客户端也可实时感知到，从而避免了某个Provider被无限调用或是无限闲置

（4）Gateway：网关也是分布式服务框架中不可或缺的部分，每种系统与框架都有自己的一套协议，当异构系统互相调用时，网关的作用即显现出来，Gateway接受各种外部HTTP请求，完成相应的权限校验，报文适配，路由转发到对应的Provider，再将Provider返回的结果传递给异构系统的Consumer，完成异构系统的互相调用

（5）负载均衡，服务分流：Consumer从Registry获得具体的Provider列表后，如何选取合适的Provider，取决与一定的负载均衡算法，常见的算法有轮询法，随机法，源地址哈希，加权轮询，加权随机等

（6）监控：接收来自Consumer和Provider异步上报的性能监控数据，对有风险的节点发出告警

#### Shiro

Shiro是apache旗下一个开源框架，它将软件系统的安全认证相关的功能抽取出来，实现用户身份认证，权限授权、加密、会话管理等功能，组成了一个通用的安全认证框架。

Shiro可以完成，认证，授权，加密，会话管理，Web集成 ，缓存等方面

#### 分布式事务，

分布式事务顾名思义就是要在分布式系统中实现事务，它其实是由多个本地事务组合而成。

##### 什么是Seata？

Seata是阿里巴巴推出的一款用来解决分布式事务问题的框架，他经过天猫双十一的考验，很有可能成为解决分布式事务问题的主流框架

分布式事务产生的背景？

在传统的单体项目中，多个不同的业务逻辑使用的都是同一个数据源，使用的都是同一个事务管理器，所以不会存在事务问题。

在分布式或者微服务架构中，每个服务都有自己的数据源，使用不同事务管理器，如果A服务去调用B服务，B服务执行失败了，A服务的事务和B服务的事务都会回滚，这时候是不存在事务问题的，但是如果A服务B服务执行成功之后出现异常，A服务的事务会回滚，但是B服务的事务不会回滚，此时就存在分布式事务问题。

（在单体的项目中，有多个不同的数据源，每个数据源中都有自己独立的事务管理器，互不影响，那么这时候也会存在多数据源事务管理：解决方案jta+ atominc）

##### Seata有三个组成部分：

事务协调器TC：协调

事务管理器TM：发起方

资源管理器RM：参与方

只要写网络通讯框架就要用到netty

##### Seata实现原理

1.TM（发起方）连接到我们的TC事务协调者，创建一个全局的事务的xid，保存到ThreadLoacl中；

2.TM（发起方）和RM(参与方)都被Seata的数据数据源实现代理，在原生的sql之前和之后保存原来和修改后日志到undo\_log中，方便后期实现回滚。

3.TM（发起方）使feign客户端调用接口时候，在ThreadLoacl中获取xid，设置到请求头中；

4.RM(参与方)从请求中获取到该xid，设置到ThreadLoacl中，同时也会向seataserver注册该分支事务。

5.TM(发起方)将当前本地事务的结果，告诉给协调者TC，协调者TC在通知所有的分支是否回滚。

6. TM(发起方)如果调用接口成功之后抛出异常的情况下，告诉给协调者TC，协调者TC在通知所有的分支根据根据全局的xid和分支事务的id 查询分支数据源的undo\_log日志表逆向生成sql语句实现回滚，同时删除对应的undo\_log日志

7. TM(发起方)如果调用接口成功之后没有抛出任何的异常，告诉给协调者TC，协调者TC在通知所有的分支根据根据全局的xid和分支事务的id 查询分支数据源的 删除对应的undo\_log日志表

##### Seata原理分析

1.发起方TM会向我们的TC协调者申请一个全局的事务id，保存到threadlocal中；

2.TM和RM都会被Seata代理数据源，写入原来和修改后内容保存到undo\_log表中；

3.TM从请求头中传递该全局的事务id给RM，RM从请求中头中获取到该全局事务id，并且注册该分支。

4.如果TM调用接口成功之后，如果报错的情况下则通知给协调者，协调者在告诉所有的分支都开始回滚，直接根据本地事务id+xid查询undo\_log表 ，逆向生成sql语句回滚，同时删除该undo\_log日志。

5.如果TM调用接口成功之后，如果没有报错的情况下则通知给协调者，协调者在告诉所有的分支都开始提交事务，直接根据本地事务id+xid删除对应的undo\_log表记录即可。

#### dubbo，

Dubbo是阿里巴巴开源的基于 Java 的高性能 RPC 分布式服务框架，

#### rabbitMQ，

主要作用：解耦、异步、削峰

消息队列服务通常有三个概念： 提供者、消息队列、消费者者

RabbitMQ 在这个基本概念之上在提供者和队列之间,加入了交换器 (Exchange)。这样提供者和消息队列就没有直接联系转而变成提供者把消息发给交换器，交换器根据调度策略再把消息转发给消息队列

## 常见问题

###### Spring Bean生命周期？

答：首先Spring对Bean进行实例化（默认是单例的），然后Spring IOC容器注入Bean的属性，然后Spring调那个初始化方法使其准备就绪，最后就是如果不需要这个Bean的话就进行销毁。再就是如果想自定这个Bean的话就需要实现一些接口，比如说Bean实现了DispostbleBean接口，Spring将调用它的destory方法，这个方法的作用与在配置文件中对Bean使用destory-method属性的作用一样，都是在Bean实例销毁前执行的方法。

1、需要进行实例化以使其进入可用状态。

2、属性注入：Spring IoC容器注入bean的属性。

3、发布初始化：Spring容器调用其postProcessAfterinitalization（）方法。

4、如果不再需要bean，则容器将删除它。

###### 2、vue.js生命周期？

答：vue生命周期总的来说有三个阶段：

①创建期间（

beforeCreate：实例刚在内存中被创建出来，此时，还没有初始化好 data 和 methods 属性;

created：实例已经在内存中创建完毕，此时 data 和 methods 已经创建完毕，此时还没有开始编译模板;

beforeMount：此时已经完成了模板的编译，但是还没有挂载到页面中;

mountde：此时已经将编译好的模板，此时才会渲染页面，挂载到了页面指定的容器中显示;）

②运行期间的生命周期函数（

beforeUpadate：状态更新之前执行此函数， 此时 data 中的状态值是最新的，但是界面上显示的 数据还是旧的，因为此时还没有开始重新渲染DOM节点;

updated：实例更新完毕之后调用此函数，此时 data 中的状态值 和 界面上显示的数据，都已经完成了更新，界面已经被重新渲染好了！）

③销毁期间的生命周期函数（

beforeDestroy：实例被销毁之前调用，在该函数中，实例仍然完全可用。

Destroyed：Vue 实例销毁后调用，调用该函数后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。）

###### 3、队列是什么？

队列，又称为伫列（queue），是[先进先出](https://zh.wikipedia.org/wiki/%E5%85%88%E9%80%B2%E5%85%88%E5%87%BA)（FIFO, First-In-First-Out）的[线性表](https://zh.wikipedia.org/wiki/%E7%BA%BF%E6%80%A7%E8%A1%A8)。在具体应用中通常用[链表](https://zh.wikipedia.org/wiki/%E9%93%BE%E8%A1%A8)或者[数组](https://zh.wikipedia.org/wiki/%E6%95%B0%E7%BB%84)来实现。队列只允许在后端（称为rear）进行插入操作，在前端（称为front）进行删除操作。

###### 解决nginx负载均衡的session共享问题

当nginx服务器进行负载均衡后，用户登录系统那么如果负载均衡机制将用户登录的请求分发给s1服务器的话，那么当用户操作个人中心的时候负载均衡机制将请求分给了s2服务器，那么保存在s1中的用户session信息将无法同步到s2服务器，为了解决这一问题可以使用以下四种方案：

①不适用session，使用cookie（把用户访问页面产生的session（服务器中的域）信息放到cookie（浏览器中的域）中，当被分到用户一开始没有访问过的服务器时就会先判断本次访问的服务器有没有session信息，如果没有就去客户端cookie里面看有没有session信息，如果没有说明真的不存在，如果有的话那就把cookie里面的用户session信息同步到用户本次分发使用的服务器）

缺点：这种方法实现起来简单，方便，也不会加大数据库的负担，但是如果客户端把cookie禁掉了的话，那么session就无从同步了，这样会给网站带来损失；cookie的安全性不高，虽然它已经加了密，但是还是可以伪造的，所以这种方式也是不推荐的。

②将session存在数据库中（session保存在数据库中，是把session表和其他的数据表存放在一起，那么当用户只要登录后随便操作了些什么就要去数据库验证一下session的状态，这样无疑加重了mysql数据库的压力；如果数据库也做了集群的话，那么也就是说每个数据库集群的节点都得保存这个session表，而且要保证每个集群的节点中数据库的session表的数据保持一致，实时同步

缺点：session保持在数据库，加重了数据库的IO，增大数据库的压力和负担，从而影响数据库的读写性能，而且mysql集群的话也不利于session的实时同步）

③ session存在缓存memcache或者redis中

④ ip\_hash技术

###### 5、数据库优化

①SQL语句优化（

1. 应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。  
    2）应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：  
   select id from t where num is null  
   可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：  
   select id from t where num=0  
    3）很多时候用 exists 代替 in 是一个好的选择  
    4）用Where子句替换HAVING 子句 因为HAVING 只会在检索出所有记录之后才对结果集进行过滤）

②索引优化

③数据库结构优化

④服务器硬件优化

###### jquery框架常用的点击事件

①点击事件 click

②失焦事件 blur

③提交事件 submit

###### JVM类加载

①类的生命周期（加载，链接（校验、准备、解析），初始化）

②类加载机制（虚拟机（jvm）把描述类的数据从 Class 文件加载到内存，并对数据进行校验、准备、解析和初始化，最终形成可以被虚拟机直接使用的 Java 类型，这就是虚拟机的类加载机制。）

③JVM的类加载机制主要有如下3种。

全盘负责：所谓全盘负责，就是当一个类加载器负责加载某个Class时，该Class所依赖和引用其他Class也将由该类加载器负责载入，除非显示使用另外一个类加载器来载入。

双亲委派：所谓的双亲委派，则是先让父类加载器试图加载该Class，只有在父类加载器无法加载该类时才尝试从自己的类路径中加载该类。通俗的讲，就是某个特定的类加载器在接到加载类的请求时，首先将加载任务委托给父加载器，依次递归，如果父加载器可以完成类加载任务，就成功返回；只有父加载器无法完成此加载任务时，才自己去加载。

缓存机制。缓存机制将会保证所有加载过的Class都会被缓存，当程序中需要使用某个Class时，类加载器先从缓存区中搜寻该Class，只有当缓存区中不存在该Class对象时，系统才会读取该类对应的二进制数据，并将其转换成Class对象，存入缓冲区中。这就是为很么修改了Class后，必须重新启动JVM，程序所做的修改才会生效的原因。

###### spring设计模式

①工厂设计模式（简单工厂和工厂方法）

②单例设计模式

③代理设计模式

④模板方法设计模式

⑤观察者设计模式

⑥适配器设计模式

⑦装饰者设计模式

⑧策略设计模式

单例模式：这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。

优点：1、在内存里只有一个实例，减少了内存的开销，尤其是频繁的创建和销毁实例（比如管理学院首页页面缓存）。2、避免对资源的多重占用（比如写文件操作）。

缺点：没有接口，不能继承，与单一职责原则冲突，一个类应该只关心内部逻辑，而不关心外面怎么样来实例化。









###### spring mvc执行流程

步骤：

用户发送请求，前端控制器拦截请求，解析请求的url路径，请求处理器映射器，处理器映射器根据用户的请求返回一个执行链，前端控制器请求适配器执行方法，适配器调用处理器执行，最终返回一个逻辑视图，前端控制器请求视图解析器解析视图，返回一个用户真正需要跳转的视图，前端控制器渲染改视图，做出响应

###### mybatis的${} 、#{}

①#相当于对数据加上双引号，$相当于直接显示数据。

②#{} ： 根据参数的类型进行处理，比如传入String类型，则会为参数加上双引号。#{} 传参在进行SQL预编译时，会把参数部分用一个占位符 ? 代替，这样可以防止 SQL注入。

③${} ： 将参数取出不做任何处理，直接放入语句中，就是简单的字符串替换，并且该参数会参加SQL的预编译，需要手动过滤参数防止 SQL注入。

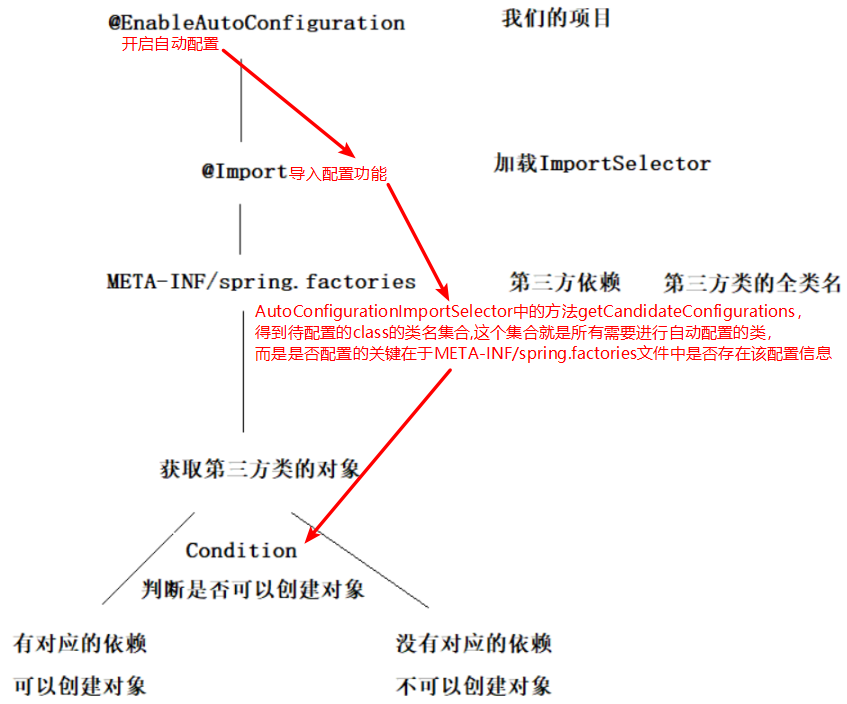
###### 乐观锁和悲观锁的区别

①读取频繁使用乐观锁，写入频繁使用悲观锁

②乐观锁lock/unlock（总是认为不会产生并发问题，每次去取数据的时候总认为不会有其他线程对数据进行修改，因此不会上锁，但是在更新时会判断其他线程在这之前有没有对数据进行修改，一般会使用版本号机制或CAS（）操作实现。）

③悲观锁synchronized（顾名思义，就是很悲观，总是假设最坏的情况，每次取数据时都认为其他线程会修改，所以都会加锁（读锁、写锁、行锁等），当其他线程想要访问数据时，都需要阻塞挂起。可以依靠数据库实现，如行锁、读锁和写锁等，都是在操作之前加锁，在Java中，synchronized的思想也是悲观锁。）

###### spring boot自动配置实现原理



###### 线程安全的集合有哪些

线程安全Vector ，HashTable ，StringBuffer

线程不安全ArrayList、LinkedList、HashMap、HashSet、TreeMap、TreeSet、StringBulider

Vector与ArrayList比较：

 1. 性能上

　　ArrayList底层数据结构是数组，适合随机查找和遍历，不适合插入和删除，线程不安全，效率高。。LinkedList底层数据结构是链表， 适合数据的动态插入和删除，随机访问和遍历速度比较慢，线程不安全，效率高。。

 2. 同步性

    Vectors是可同步的，是线程安全的。ArrayList是不可同步的，不是线程安全的。所以，一般单线程推荐用ArrayList，多线程中则用Vector

 3. 数据增长

    往一个ArrayList或者Vector里插入一个元素时，如果内部数组空间不够，ArrayList或Vector会扩展它的大小。Vector在默认情况下增长一倍的大小，而ArrayList增加50%的大小。

###### New字符串和等于一个字符串的区别

①String str1 = "ABC"; 可能创建一个对象或者不创建对象。

如果"ABC" 这个字符串在（堆内存中）String池中不存在，会在String池中创建一个String str1= "ABC"的对象。然后把str1指向这个内存地址（栈内存中）。之后用这种方式创建多少个值为"ABC"的字符串对象。始终只有一个内存地址被分配，之后都是String的copy。这种被称为‘字符串驻留’，所有的字符串都会在编译之后自动驻留。

②String str2 = new String("ABC"); 至少会创建一个对象，也可能2个。

因为用到了new的关键字，肯定会在堆内存中创建一个str2的对象。它的value值是"ABC"，同时如果这个字符串在string池中不存在，会在string常量池中创建这个string对象"ABC"。

String str1 = new String("ABC");

String str2 = new String("ABC");

System.out.println(str1.equals(str2)); // true equals 比较值

System.out.println(str1 == str2); // false == 比较的是内存地址。

###### 如何让两个线程交替执行

由于线程争用cpu的随机性，就需要A线，B线程执行，在B执行完一次进入等待之前唤醒A，如此往复，那么这里就要用上notify和wait了。

解决方法：①采用Lock锁对象，公平锁解决两个线程交替执行

②采用同步代码块synchronized，然后采用wait()等待和notify()唤醒方法解决两个线程交替执行

###### equals可以比较基本类型吗

答：不可以，

①String 的equals方法：

如下：重写object的equals方法，首先进行内存地址的比较，然后进行数据类型的比较，然后进行字符串长度的比较，最后再进行每一个字符的比较

###### String，StringBuilder，StringBuffer三者区别

1. 常见的字符串拼接该用谁？

①String是一个不可改变的字符序列

②StringBuilder是一个可以改变的字符序列

推荐使用StringBuilder，因为拼接效率比String高

1. StringBuilder和StringBuffer的区别是什么

相同点：两者的功能是完全一致的

不通点：①StringBuffer是jdk1.0出现的，线程安全（同步）的

②StringBuilder是jdk1.5出现的，线程不安全（不同步）的

安全的效率低，不安全的效率高，推荐使用StringBuilder

1. 在工作中如果在业务层去拼接sql语句，选用谁？

①如果sql语句拼接内容少的话使用String

②如果sql语句拼接内容多的话使用StringBuilder

###### 说下事务

1、什么识事务

事务是应用程序中一系列严密的操作，所有操作必须成功完成，否则在每个操作中所作的所有更改都会被撤消。也就是事务具有原子性，一个事务中的一系列的操作要么全部成功，要么一个都不做。

2、事务的四大特性

①原子性：事务是数据库的逻辑工作单位，而且是必须是原子工作单位，对于其数据修改，要么全部执行，要么全部不执行。

②一致性：在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。

③ 隔离性（isolation）：一个事务的执行不能被其他事务所影响。

④ 持久性（durability）：一个事务一旦提交，事物的操作便永久性的保存在DB中。即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

3、隔离级别

脏读:  指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据， 那么另外一 个事务读到的这个数据是脏数据，依据脏数据所做的操作可能是不正确的。

不可重复读:  指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。 那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的。这样就发生了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。

幻觉读:指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及 到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，就会发生操作第一个事务的用户发现表中还有 没有修改的数据行,也就是说幻像读是指同一查询在同一事务中多次进行，由于其他提交事务所做的插入操作，每次返回不同的结果集，此时发生幻像读,就好象发生了幻觉一样。

###### ArrayList和LinkedList的迭代器那个速度快

①如果是数组结构的集合，如ArrayList存放元素大小是固定的，并且在事先已经指定并开僻了指定数量的容量的数组空间，实际上只是对数组进行了操作。则使用for循环会更快（随机获取）

②如果是链表结构的集合，如LinkedList 是采用链表实现的，在事先无法指定容量，每添加一个数据都得去开僻新的空间。则使用迭代器遍历，速度会更快(顺序获取)。

###### 19、Spring MVC框架中使用的注解

@Controller：用于定义控制器类

@RestController：是@Controller和@ResponseBody的合集

@RequestMapping：提供路由信息，负责URL到Controller中的具体函数的映射

@GetMapping：是@RequestMapping(method = RequestMethod.GET)的缩写。

@PostMapping：是@RequestMapping(method = RequestMethod.POST)的缩写。

@ResponseBody：表示方法的返回结果直接写入HTTP response body中

@RequestBody：将前端传入的json参数转化为java对象

@PathVariable：获取路径参数

@RequestParam：用在方法的参数前面

###### 20、了解过redis吗？

NoSQL 一种非关系性数据库,通常我们用于缓存数据 和关系型数据库互补,关系型数据库通常用来数据存储,NOSQL用来备份存储关系型数据库的数据,其他的还有MangoDB,Tokyo

①缓存穿透（查询的数据在数据库没有，redis中也没有，然后如果有人恶意访问的话会导致数据库访问压力增大从而导致宕机。解决办法：布隆过滤器或者将它存入缓存中值为null，设置有效时间短一点）

②缓存雪崩（由于原有缓存失效，信缓存未到期间导致原本访问缓存的却去访问数据库，给数据库造成压力导致宕机。解决办法：将这些缓存有效时间设为不同）

③缓存击穿（指一个key非常热点，大并发集中对这个key访问，当这个key失效时，仍然有大并发访问过来这个时候由于缓存中没有就会访问数据库导致数据库压力过大从而当值宕机）

④目前为止Redis支持的键值数据类型如下：

1) 字符串类型 string

2) 哈希类型 hash(map类型)

3) 列表类型 list(linklist双向链表格式)

4) 集合类型 set

5) 有序集合类型 sortedset，每个元素都要关联一个double类型的分数(score)，redis正是通过分数来为集合中的成员进行从小到大的排序。

###### 21、说下Mybatis

（1）Mybatis是一个半ORM（对象关系映射）框架，它内部封装了JDBC，开发时只需要关注SQL语句本身，不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。程序员直接编写原生态sql，可以严格控制sql执行性能，灵活度高。

（2）MyBatis 可以使用 XML 或注解来配置和映射原生信息，将 POJO映射成数据库中的记录，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。

（3）通过xml 文件或注解的方式将要执行的各种 statement 配置起来，并通过java对象和 statement中sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回。（从执行sql到返回result的过程）。

###### 22、JDK1.8新特性

Lambda表达式

函数式接口

\*方法引用和构造器调用

Stream API

接口中的默认方法和静态方法

新时间日期API

###### shiro的执行流程以及核心组件?

Subject：是安全管理中直接操作的对象

SecurityManager:安全管理器,所有与安全有关的操作都会与SecurityManager 交互；且其管理着所有 Subject；它是 Shiro的核心，它负责与 Shiro 的其他组件进行交互，它相当于 SpringMVC 中DispatcherServlet 的角色

Realm:返回安全数据，我们通常自定义realm，继承AuthorizingRealm类重写其中两个方法doGetAuthorizationInfo和doGetAuthenticationInfo，前者进行权限管理，后者进行角色认定，两个方法将用户信息权限等数据封装起来交由SecurityManager来进行安全管理使用，相当于dao。

②其他权限框架：

SSM + Shiro

Spring Boot/Spring Cloud + Spring Security

③Spring Security核心组件及认证流程

核心组件：AuthenticationMannager：其中可以包含多个AuthenticationProvider

AuthenticationProvider：主要用来进行认证操作的类 调用其中的authenticate()方法去进行认证操作

UserDetilService ：通过扩展这个接口来显示获取我们的用户信息，用户登陆时传递的用户名和密码也是通过这里这查找出来的用户名和密码进行校验，但是真正的校验不在这里，而是由AuthenticationManager以及AuthenticationProvider负责的，需要强调的是，如果用户不存在，不应返回NULL，而要抛出异常UsernameNotFoundException

操作流程：web将用户名和密码组装成一个UsernamePasswordToken对象请求=>AuthenticationMannager（委托对应的AuthenticationProvider）=>UserDetilService（查询是否有，没有就报异常返回）

###### Spring IOC是用来干什么的？

控制翻转思想 Ioc意味着将你设计好的对象交给容器控制，而不是传统的在你的对象内部直接控制。（将创建对象得权力交给spring 降低耦合度）

IOC工作流程：

一、准备配置文件：就像前边Hello World配置文件一样，在配置文件中声明Bean定义也就是为Bean配置元数据。

二、由IoC容器进行解析元数据： IoC容器的Bean Reader读取并解析配置文件，根据定义生成BeanDefinition配置元数据对象，IoC容器根据BeanDefinition进行实例化、配置及组装Bean。

三、实例化IoC容器：由客户端实例化容器，获取需要的Bean。

###### 了解的加密算法有哪些

Md5 BcryptPassword RSA 传智健康使用的是MD5

###### HashMap

①遍历方式有四种

1. 先根map.keySet获取到键的集合，然后遍历键，最后根据键取到值
2. 先用map.entrySet获取到键值对对象，然后根据迭代器遍历取到键和值
3. 先用map.entrySet获取到键值对对象，然后根据增强for进行遍历取到键和值
4. 先用map.values获取值集合（Conllection），然后用增强for或者迭代器遍历取到值（只能取到值）

②Map的实现类

1. TreeMap：使用二叉树进行存储key-value
2. LinkdeHashMap：底层使用链表来维护key-value的次序
3. HashTable：jdk1.0出现的，线程安全（同步的），效率低，不能使用null作为key和Value，
4. Properties：也是以key-value作为键值对存储信息
5. HashMap线程不安全（不同步），允许存储null键和null值，效率高

③Map的特点

1. 双列集合，将键映射到值的一个对象
2. 数据结构只针对键有效，跟值没关系
3. 一个映射不能包含重复的键，每个键最多只能映射到一个值

④HashMap集合的数据结构

1. 哈希表结构：数组+链表
2. 当我们条用添加的方法想集合中添加元素的时候，首先调用该自定义对象的hashCode方法计算出一个哈希值，然后拿着哈希值去集合中查找看是否有相同的，如果没有的话直接存储，如果有的话就调用equals方法逐个比较元素的内容

⑤jdk1.8版本后在哈希表中引入了二叉树

1. 当某一个索引下的节点数量达到了一定程度的时候会将原本链表结构改为二叉树结构提升了查询的效率
2. 二叉树结构做到了排序（左小右大）

先去节点左边的，当左边的全部取完再取自己最后去节点右边的

###### jvm GC 垃圾回收机制

①什么样的对象会被当做垃圾回收（当一个对象的引用（地址）没有变量记录的时候）

如：当我创建了一个对象，然后我将这个对象赋值为null，该对象就会变成垃 圾对象，并在垃圾回收器空闲的时候对其清扫。

②如何校验对象是否被回收

可以重写Object类中的finalize方法，这个方法在垃圾回收器执行的时候会被垃圾回收器自动调用

③怎么通知垃圾回收器回收对象

调用System类的静态方法gc（）通知垃圾回收器去清理垃圾

④JVM GC 垃圾回收机制

概述：引用计数算法给对象加一个计数器，呗引用时加1，取消引用时减1，为0的时候回收。

⑤尽量不要创建很大的对象

原因：GC回收算法从来不对大对象（>=85000字节）堆进行内存压缩整理，在堆中大的内存块会浪费太多CPU时间

⑥不要频繁的new生命周期很短的对象

这样频繁垃圾回收频繁压缩有可能会导致很多内存碎片

###### 数据库三大范式

第一范式：数据表字段保持原子性，不可再分

第二范式（2NF）要求实体的属性完全依赖于主关键字

第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息（一般使用 id 关联）。从而减少数据库的冗余。

###### 负载均衡的实现

①四层负载均衡，也就是主要通过报文中的目标地址和端口，再加上负载均衡设备设置的服务器选择方式，决定最终选择的内部服务器。

②七层负载均衡，也称为“内容交换”，也就是主要通过报文中的真正有意义的应用层内容，再加上负载均衡设备设置的服务器选择方式，决定最终选择的内部服务器。

③静态负载均衡算法：轮询，比率，优先权

④动态负载均衡算法：最少连接方式，最快没事，观察模式，预测模式，动态性能分配，动态服务器补充，服务质量，服务类型，规则模式，ping侦测，TCP Open侦测，HTTP URL侦测

###### 并发合并行

并发：指在一个cpu中同时处理多项进程

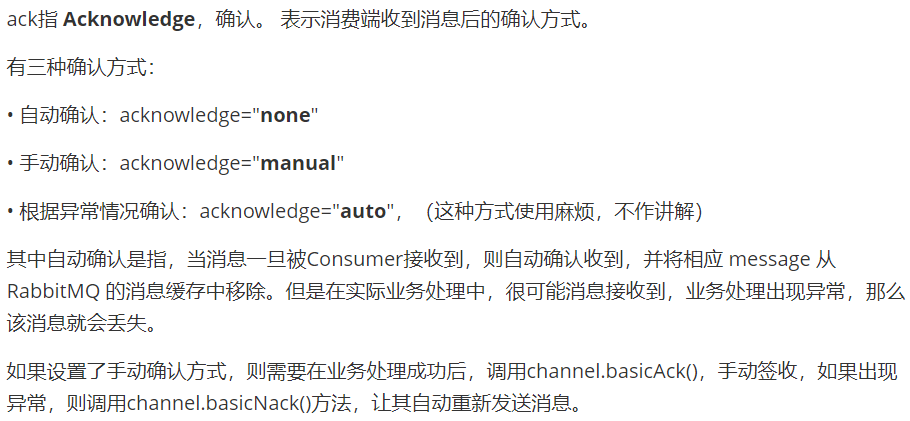
并行：指在多个cpu上同时执行多个进程

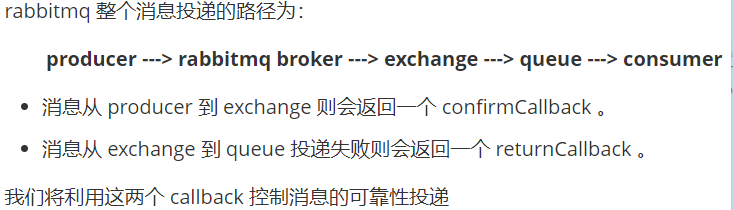
###### java为什么可以一处编写多处运行

Java编译产生的不是针对特定平台的机器码，而是一种与平台无关的字节码文件（即\*.class文件）。相同的字节码在不同平台上直接运行原本是不可能的，但通过中间的转换器实现了“一次编译，到处运行”的效果，JVM就是这个转换器。不同平台上的JVM是不同的，但他们提供给Java字节码程序的接口是完全相同的。因此，这些字节码不面向任何平台，只面向JVM，也就是说，JVM充当了中介或者叫做翻译的角色。

###### MQ的ack

为了保证数据不被丢失，RabbitMQ支持消息确认机制，即ack。发送者为了保证消息肯定消费成功，只有使用方明确表示消费成功，RocketMQ才会认为消息消费成功。中途断电，抛出异常等都不会认为成功——即都会重新投递。保证数据能被正确处理而不仅仅是被Consumer收到，我们就不能采用no-ack或者auto-ack，我们需要手动ack(manual-ack)。在数据处理完成后手动发送ack，这个时候Server才将Message删除。





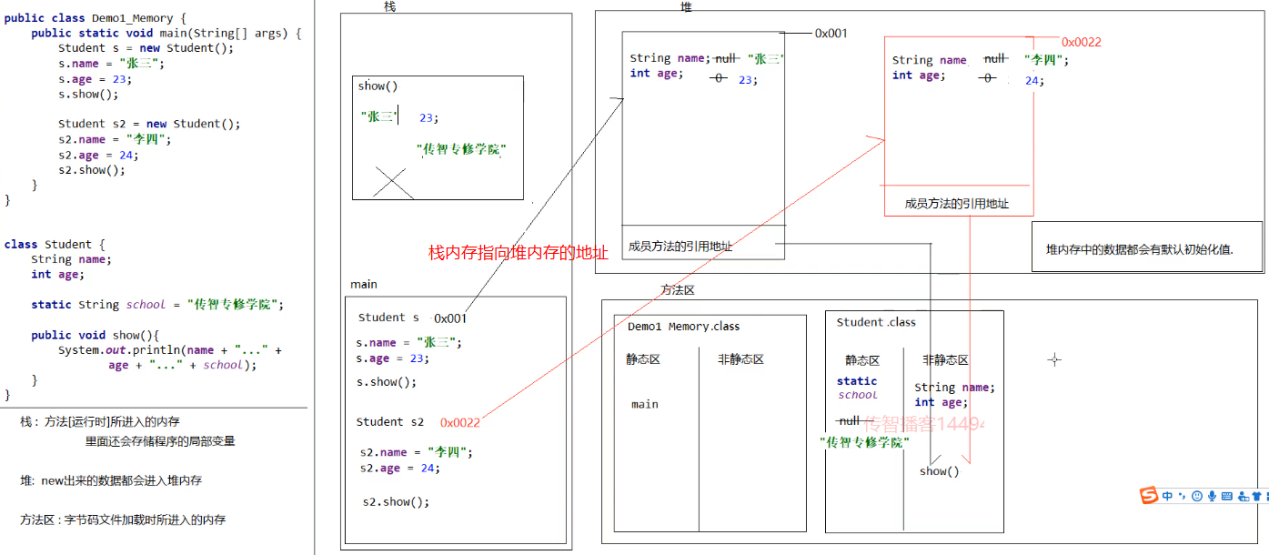
###### JVM（虚拟机）的内存划分

①java虚拟机有哪几块内存空间

栈内存：方法运行时所进入的内存，里面还会存储程序的局部变量

堆内存：new出来的数据都会进入堆内存

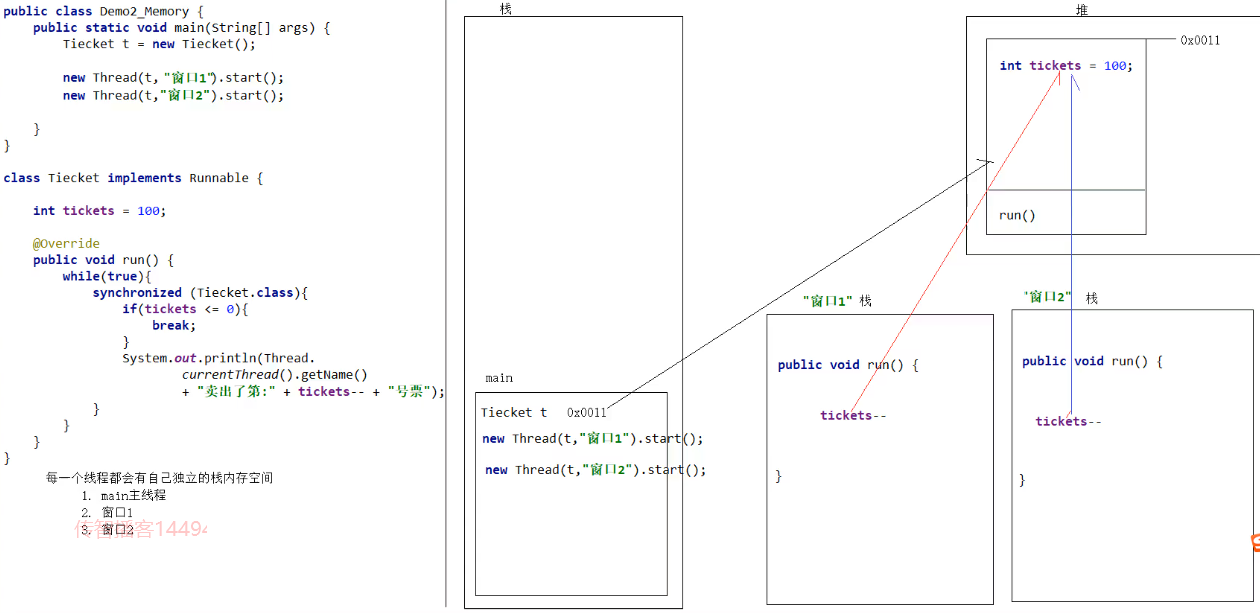
方法区：字节码文件加载时所进入的内存



②带有线程的内存图

每一个线程都会有自己独立的栈内空间

堆内存中的数据，是被多个线程所共享的

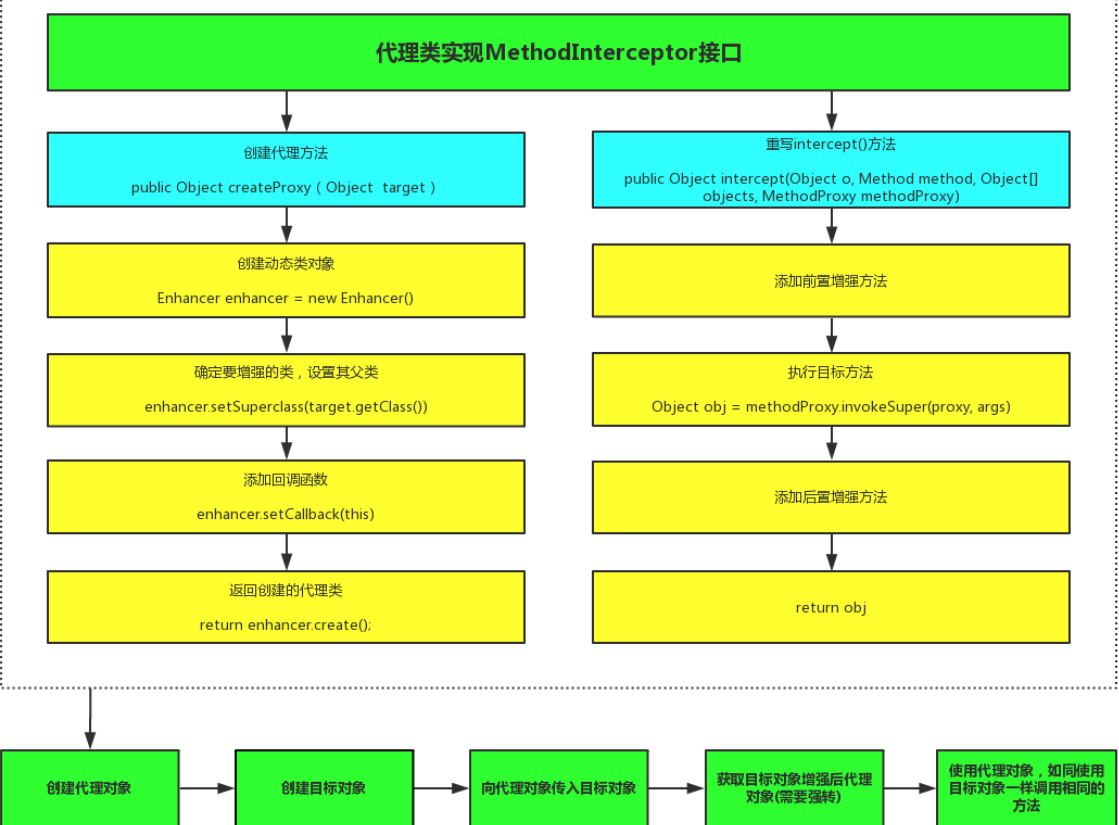


（多线程消费同一个商品，可以将该商品定义为共享资源，存放于堆内存中。）

###### aop底层动态代理

Spring AOP使用了两种代理机制，一种是基于JDK的动态代理（动态代理就是为了解决静态代理不灵活的缺陷而产生的。静态代理是固定的，一旦确定了代码，如果委托类新增一个方法，而这个方法又需要增强，那么就必须在代理类里重写一个带增强的方法。而动态代理可以灵活替换代理方法，动态就是体现在这里。），

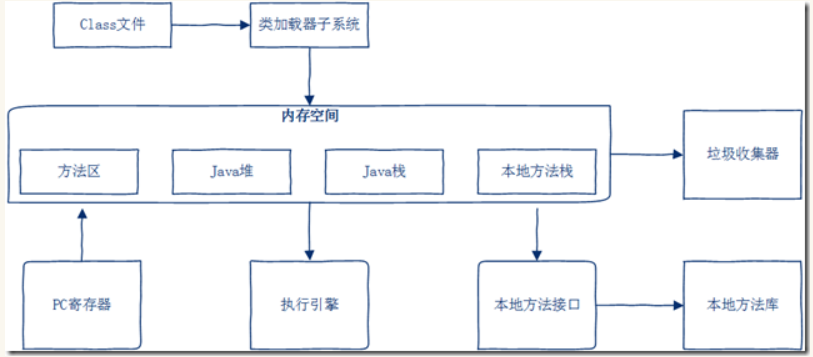
另一种是基于CGLib的动态代理（不需要类继承任何接口，字节码技术），之所以需要两种代理机制，很大程度上是因为JDK本身只提供基于接口的代理，不支持类的代理



###### sql分组函数



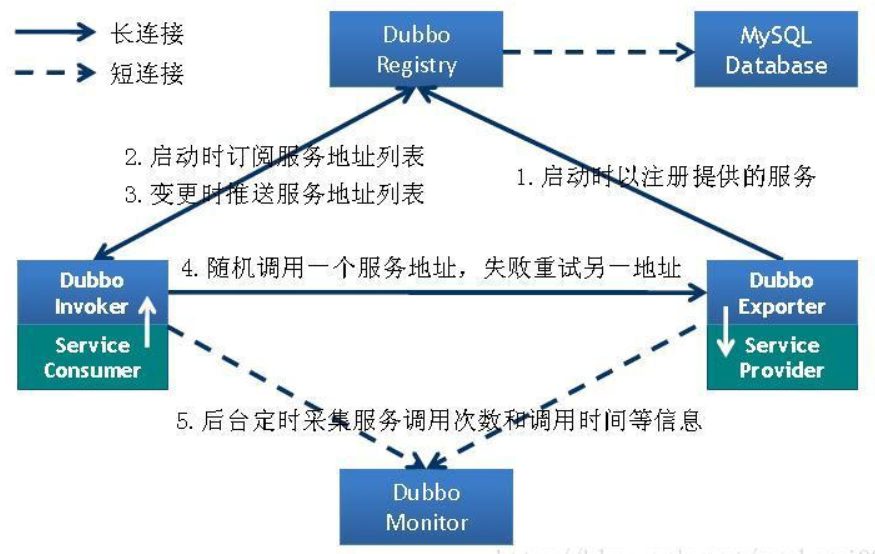
###### JVM浅谈



class文件被jvm装载以后，经过jvm的内存空间调配，最终是由执行引擎完成class文件的执行。当然这个过程还有其他角色模块的协助，这些模块协同配合才能让一个java程序成功的运行。

###### Dobbo

Apache Dubbo是一款高性能的Java RPC框架。其前身是阿里巴巴公司开源的一个高性能、轻量级的开源Java RPC框架，可以和Spring框架无缝集成。





调用关系说明:

服务容器负责启动，加载，运行服务提供者。

服务提供者在启动时，向注册中心注册自己提供的服务。

服务消费者在启动时，向注册中心订阅自己所需的服务。

注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。

服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

###### 异步请求怎么控制的

promise机制本身就是用于异步编程的，如需顺序请求可以在前一个promise的then方法中执行第二个promise

###### Controller和RestController

①相同点：都是用来表示Spring某个类的是否可以接受HTTP请求

②不通点：

1、是否可以返回页面（如果只是使用@RestController注解，则其方法无法返回指定页面，此时配置的视图解析器 InternalResourceViewResolver不起作用，返回的内容就是 return 里的内容（json）。 如果需要返回到指定页面，则需要用 @Controller配合视图解析器InternalResourceViewResolver才行。）

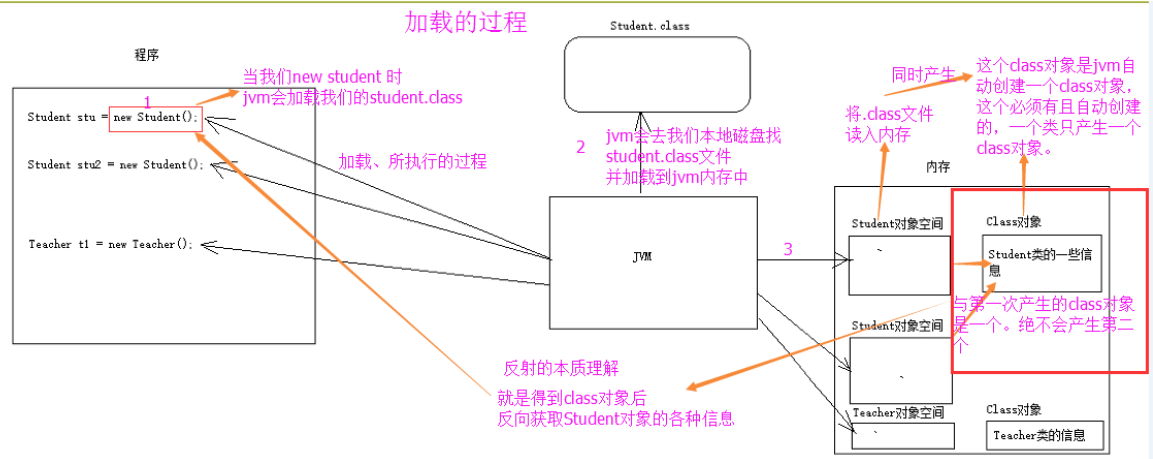
2、返回内容（如果需要返回JSON，XML或自定义mediaType内容到页面，@RestController自己就可以搞定,这个注解对于返回数据比较方便，因为它会自动将对象实体转换为JSON格式。而@Controller需要在对应的方法加上@ResponseBody注解。）

###### 39、反射机制

①什么是反射

反射是java的特征之一，是一种间接操作目标对象的机制，核心是JVM在运行的时候才动态加载类，并且对于任意一个类，都能够知道这个类的所有属性和方法，调用方法/访问属性，不需要提前在编译期知道运行的对象是谁，他允许运行中的Java程序获取类的信息，并且可以操作类或对象内部属性。程序中对象的类型一般都是在编译期就确定下来的，而当我们的程序在运行时，可能需要动态的加载一些类，这些类因为之前用不到，所以没有加载到jvm，这时，使用Java反射机制可以在运行期动态的创建对象并调用其属性，它是在运行时根据需要才加载。

（此图是类正常加载和反射本质）



②反射优缺点：

优点：使用反射，我们就可以在运行时获得类的各种内容，进行反编译，对于Java这种先编译再运行的语言，能够让我们很方便的创建灵活的代码，这些代码可以在运行时装配，无需在组件之间进行源代码的链接，更加容易实现面向对象。

缺点：（1）反射会消耗一定的系统资源，因此，如果不需要动态地创建一个对象，那么就不需要用反射；（2）反射调用方法时可以忽略权限检查，因此可能会破坏封装性而导致安全问题。

③反射的用途

1、反编译：.class-->.java

2、通过反射机制访问java对象的属性，方法，构造方法等

###### 40、多线程

①什么是线程

线程是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位。

②线程和进程有什么关系

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。别把它和栈内存搞混，每个线程都拥有单独的栈内存用来存储本地数据。

③多线程的实现方式

（1）自定义一个类继承THread类，重写run（）方法，然后创建自定义类对象，start（）启动线程（然后由JVM调用此线程的run（）方法）

（2）自定义一个类实现Runnable接口，然后重写run（）方法，然后创建自定义类对象，创建Thread类的对象，把自定义对象作为构造方法的参数，然后启动线程

###### 41、抽象类

①抽象类中是可以有静态方法的，在被子类继承后可以通过子类的实例化对象调用；也可以直接通过抽象类和子类的类名来调用，就像其他的静态方法一样。同样的静态变量也可以使用相同的方法访问。

②抽象类中不能有静态的抽象方法（原因：抽象类是不能实例化的，即不能被抄[分配内存](https://www.baidu.com/s?wd=%E5%88%86%E9%85%8D%E5%86%85%E5%AD%98&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank);而static修饰的方袭法在类实例化之前就已经别分配了内存，这样一来矛盾就出现了：抽象类不能被分知配内存，而static方法必须被[分配内存](https://www.baidu.com/s?wd=%E5%88%86%E9%85%8D%E5%86%85%E5%AD%98&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)。所以抽象类中不能有静态的抽象方法。）

###### 雪崩效应的解决方法

①服务雪崩：多个微服务之间调用的时候，假设微服务A调用微服务B和微服务C，微服务B和微服务C有调用其他的微服务，这就是所谓的”扇出”，如扇出的链路上某个微服务的调用响应式过长或者不可用，对微服务A的调用就会占用越来越多的系统资源，进而引起系统雪崩，所谓的”雪崩效应”

②请求熔断：当Hystrix Command请求后端服务失败数量超过一定比例(默认50%), 断路器会切换到开路状态(Open). 这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时间后(默认5秒), 自动切换到半开路状态(HALF-OPEN).

　　　 这时会判断下一次请求的返回情况, 如果请求成功, 断路器切回闭路状态(CLOSED), 否则重新切换到开路状态(OPEN). Hystrix的断路器就像我们家庭电路中的保险丝, 一旦后端服务不可用, 断路器会直接切断请求链, 避免发送大量无效请求影响系统吞吐量, 并且断路器有自我检测并恢复的能力.

③服务降级：Fallback相当于是降级操作. 对于查询操作, 我们可以实现一个fallback方法, 当请求后端服务出现异常的时候, 可以使用fallback方法返回的值. fallback方法的返回值一般是设置的默认值或者来自缓存.通知后面的请求告知这服务暂时不可用了