# B简历：

## 基本信息：

|  |  |  |  |
| --- | --- | --- | --- |
| 基本情况 | | | |
| **姓名** | **赵梦帆** | 性别 | 男 |
| **出生日期** | 1996-03-19 | 属相 | 鼠 |
| **现住地** | 郑州市二七区养老公寓旁边 |  | 中原区 升龙又一城b 3栋4楼9层 |
| 大学情况 | | | |
| **学校** | 河南科技学院新科学院 | 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** | **C语言、数据结构课程设计、数据库原理课程设 计、操作系统实训、Java程序设计** | | |
| **学历** | **本科** | 学位 | **校长** |
| **学校地址** | 河南省新乡市红旗区新中大道中段 | 学士 | 刘鸣韬 |
| **去学校的方式：** | **在火车站乘232路公交车到枫林大道下车** | | |
| **上小学日期：** | **2002年9月 6岁六年** | | |
| **上初中日期：** | **2008年9月 12岁三年** | | |
| **上高中日期：** | **2011年9月 15岁三年** | | |
| **上大学日期：** | **2014年9月 18岁四年** | | |
| **毕业时间：** | **2018年 7月 22岁** | | |
| **在校学生：** | **5000左右** | | |
| **有教师：** | **不太清楚** | | |

|  |  |
| --- | --- |
| 第一家公司 ： | 河南开云信息技术有限公司 |
| **地址：** | 郑州高新区汉威国际传感器科技园附近 |
| **入职时间：** | **2017年10月初** |
| **离职时间：** | **2019年4月中旬** |
| **公司性质：** | **民营** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **尚海峰** |
| **薪资方面：** | **刚刚毕业转正以后工资是5K左右，过完年大概三月份涨到6K左右，走的时候8000** |
| **路线：** | 一号线到河南工业大学站，做长春路站的B202公交车到莲花街雪松路站下 |

|  |  |
| --- | --- |
| **公司地址**： | 铂元智能科技（北京）有限公司  北京大兴区大族广场T5号楼2302 |
| **公司规模：** | **80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：** | **互联网和虚拟现实的软件产品的开发。** |
| **去公司路线：** | 昌平区风光雅苑 ，亦庄线，荣京东街地铁站 |
| **入职时间：** | **2019年4月** |
| **离职时间：** | **2021年4月初** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **于笑** |
| **税前：** | **16K** |
| **税后：** | **15K** |
| **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：** | **16K** |
| **薪资的构成:** | **岗位工资(5K)+绩效工资(11K)** |

|  |
| --- |
|  |

**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** |
| **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，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:**  **抗压能力强（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。），**  **乐于分享，**  **团队协作能力强**  **缺点: 我这个人说话比较直，你像在和团队成员探讨问题的时候，容易得罪人，**  **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** |
| **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？ |
| **前2,3年 继续加强自己的技术功底，**  **然后朝着项目经理(技术经理,产品经理)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**  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:**

-->日志管理-->负载均衡加redis分布式session-->基于Token的用户登录【jwt】--> 基于redis的购物车-->订单

-->项目的重构【从ssmp框架升级为springboot+springcloud组成的微服务】

-->sql优化-->通过线程池提高程序的性能

**核心概念：**

**咱们也知道这几年这个微服务springboot和springcloud用的非常多，而我在上一个项目中使用的也是这个springboot+springcloud进行的开发，那我就先说一下这个我对这个微服务框架的理解吧**

# Springboot

我先给你说一下springboot吧，Spring Boot是由Pivotal团队提供的全新框架，不再需要定义样板化的配置，内嵌了使用工具（比如内嵌了Tomcat、Jetty等），方便开发人员快速搭建和开发的一个框架。

1 快速搭建spring 框架 2 约定大于配置 3 开箱即用 4 嵌入了web容器

* 搭建项目快，几秒钟就可以搭建完成；
* 让测试变的简单，内置了JUnit、Spring Boot Test等多种测试框架，方便测试；
* Spring Boot让配置变的简单，Spring Boot的核心理念：约定大约配置，约定了某种命名规范，可以不用配置，就可以完成功能开发，比如模型和表名一致就可以不用配置，直接进行CRUD（增删改查）的操作，只有表名和模型不一致的时候，配置名称即可；
* 内嵌容器，省去了配置Tomcat的繁琐；
* 方便监控，使用Spring Boot Actuator组件提供了应用的系统监控，可以查看应用配置的详细信息；
* ***缺点 :***
* (1) 书籍文档少且不够深入;
* (2) 如果你不认同Spring框架,这也许是它的缺点,但建议你一定要使用Spring框架.

***需要的环境：***

Spring Boot 2.0.4

JDK 1.8

IDE：IntelliJ IDEA 2018.2

注意：Spring Boot 2.x 要求Java8及以上版本，不再支持Java6、Java7.

# Springcloud

Springcloud 组件中的组件有 -eureka 注册中心，Feign， Ribbon，Hystrix熔断，Hystrix监控中心，zull网关

### spring-cloud-eureka：

spring-cloud-eureka是spring-cloud-nettfix微服务套件中的一部分， 它基于nettfix-

eureka做了二次封装，主要负责微服务架构中的服务治理功能。

服务治理一般都会有两个功能：服务注册、服务发现。通常会有一个注册中心，每个服务单

元向注册中心登记自己信息，比如提供的服务，ip, 端口以及一些附近加信息等。注册中心

会将新的服务实例发送给其它依赖此服务的实例。

核心注解@EnableEurekaClient

### springcloud-Feign:

Feign是一个声明式WebService客户端。使用Feign能让编写Web Service客户端更加简单, 它的使用方法是定义一个接口，然后在上面添加注解，同时也支持JAX-RS标准的注解。Feign也支持可拔插式的编码器和解码器。Spring Cloud对Feign进行了封装，使其支持了Spring MVC标准注解和HttpMessageConverters。Feign可以与Eureka和Ribbon组合使用以支持负载均衡。

Feign也是运行在消费者端的，使用 Ribbon 进行负载均衡，所以 Feign 直接内置了 Ribbon。而与Ribbon不同的是，通过feign只需要定义服务绑定接口且以声明式的方法，优雅而简单的实现了服务调用。

核心注解：@EnableFeignClients//声明fegin

### Spring Cloud Ribbon:

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端负载均衡的工具。

简单的说，Ribbon是Netflix发布的开源项目，主要功能是提供客户端的软件负载均衡算法，将Netflix的中间层服务连接在一起。Ribbon客户端组件提供一系列完善的配置项如连接超时，重试等。简单的说，就是在配置文件中列出Load Balancer（简称LB）后面所有的机器，Ribbon会自动的帮助你基于某种规则（如简单轮询，随机连接等）去连接这些机器。我们也很容易使用Ribbon实现自定义的负载均衡算法。核心：

启动类中注入bean

@Bean

@LoadBalanced //声明 ribbon （并对resttemmplate进行负载均衡 调用的时候直接写服务名就行 不用写ip和端口）

RestTemplate restTemplate() {

return new RestTemplate();// RestTemplate对httpclient发送请求 进行封装了

}

### Spring Cloud Hystrix：

Hystrix中文是豪猪的意思，在Spring Cloud中防雪崩的利器就是Spring Cloud Hystrix，Spring Cloud Hystrix是基于Netflix Hystrix实现的。在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故

Hystrix能干啥

l 服务熔断

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).

l 服务降级

Fallback相当于是降级操作, 我们可以实现一个fallback方法, 当请求后端服务出现异常或者服务熔断后的时候, 直接调用本地fallback方法，告知后面的请求服务不可用了，不要再来了。

l 服务容错

l 线程和信号隔离

在Hystrix中, 主要通过线程池来实现资源隔离. 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.比如说，一个服务调用两外两个服务，你如果调用两个服务都用一个线程池，那么如果一个服务卡在哪里，资源没被释放。后面的请求又来了，导致后面的请求都卡在哪里等待，导致你依赖的A服务把你卡在哪里，耗尽了资源，也导致了你另外一个B服务也不可用了。这时如果依赖隔离，某一个服务调用A B两个服务，如果这时我有100个线程可用，我给A服务分配50个，给B服务分配50个，这样就算A服务挂了，我的B服务依然可以用。

l 请求缓存

比如一个请求过来请求我userId=1的数据，你后面的请求也过来请求同样的数据，这时我不会继续走原来的那条请求链路了，而是把第一次请求缓存过了，把第一次的请求结果返回给后面的请求。

l 请求合并

我依赖于某一个服务，我要调用N次，我发了N条请求发然后拿到一堆结果，这时候我们可以把多个请求合并成一个请求，这样我们只需调用一次远程服务，提升了效率。

l 近乎实时的服务监控

### Hystrix 客户端核心：

@EnableHystrix// 熔断 启动类中

对你要熔断的方法

// 此注解标识是要熔断 fallbackMethod 熔断后进行执行的方法

@HystrixCommand(fallbackMethod = "errorMethod",commandProperties = {

@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds",value = "3000")

}) //对此方法进行熔断

Hystrix 服务端：

启动类

@EnableHystrixDashboard

启动类中

/\* 提供监控平台的路径\*/

@Bean

public ServletRegistrationBean getServlet() {

HystrixMetricsStreamServlet streamServlet = new HystrixMetricsStreamServlet();

ServletRegistrationBean registrationBean = new ServletRegistrationBean(streamServlet);

registrationBean.setLoadOnStartup(1);

registrationBean.addUrlMappings("/hystrix.stream");

registrationBean.setName("HystrixMetricsStreamServlet");

return registrationBean;

### Spring Cloud zull：

Zuul是Netflix开源的微服务网关，他可以和Eureka,Ribbon,Hystrix等组件配合使用。Zuul组件的核心是一系列的过滤器，这些过滤器可以完成以下功能：

身份认证和安全:

识别每一个资源的验证要求，并拒绝那些不符的请求审查与监控：

***动态路由：***

动态将请求路由到不同后端集群

***压力测试：***

逐渐增加指向集群的流量，以了解性能

***负载分配：***

为每一种负载类型分配对应容量，并弃用超出限定值的请求

***静态响应处理：***

边缘位置进行响应，避免转发到内部集群

# 购物车

前台定一个死值 kid=-1 然后判断此商品是否为sku商品 如果是将kid覆盖 如果不是 把-1传过去

后台接收前台传过来三个参数 pid kid count

首先判断用户是否登录 通过token判断 没登陆打回登录页面

然后判断kid是否等于-1 如果等于-1 说明此商品为非sku商品 那么我们查出这个非sku商品的库存 并设置filed值

如果是sku属性 那么我们查出这个sku商品的库存套餐和价格 然后根据pid查询单条数据，把sku商 品的库存套餐和价格都放进去

然后判断redis中是否有此商品 如果有然后判断此商品库存是否够 如果够则库存+1 没有的话就新增一条redis数据

## 订单

后台接收前台传过来的 order对象 goodsInfo字符串 和keyId

首先判断用户是否登录 通过token判断 没登陆打回登录页面

kid是前台传过来的唯一标识uuid 解决幂等性 防止用户一直发送请求 然后生成订单 把这个订单的OID返回过来

然后把goodsInfo字符串转成json数组循环遍历出来获取它的价格标题和ID

然后判断id是否包含冒号 如果包含则取冒号后面的ID 次ID为sku商品ID 根据这个skuID查询单条数据

然后判断库存是否够 够的话就口库存

然后声明一个OrderInfo集合 再new一个Order对象 把价格标题和ID放到Order对象中 再把Order对象放到List集合中

然后批量新增OrderInfo信息 再清空删除购物和信息

string stringbuffer stringbuilder区别

string是字符串常量 值不可变 内存地址不会发生改变

stringbuffer 值可变 线程安全 效率低

stringbuilder 值可变 线程不安全 效率高

## 幂等性

通俗的说就是一个接口, 多次发起同一个请求, 必须保证操作只能执行一次

订单接口, 不能多次创建订单

支付接口, 重复支付同一笔订单只能扣一次钱

解决方法

页面加载成功后生成一个随机数 发起请求的时候将随机数传到后台

在后台用redis的getandset方法加锁 ---> 释放

超卖

sql语句加条件来实现库存不能为负数

# 消息队列 RabbitMQ

消息队列（Message Queue）是一种应用间的通信方式，消息发送后可以立即返回，由消息系统来确保消息的可靠传递。消息发布者只管把消息发布到 MQ 中而不用管谁来取，消息使用者只管从 MQ 中取消息而不管是谁发布的。这样发布者和使用者都不用知道对方的存在。

Direct Exchange

直连型交换机，根据消息携带的路由键将消息转发给对应的消息队列。

大致流程，有一个队列绑定到一个直连交换机上，同时赋予一个路由键 routing key 。

然后当一个消息携带着路由值为X，这个消息通过生产者发送给交换机时，交换机就会根据这个路由值X将消息转发给绑定值也是X的队列。

Fanout Exchange

扇型交换机，这个交换机没有路由键概念，就算你绑了路由键也是无视的。 这个交换机在接收到消息后，会直接转发到与他绑定的所有队列。

Topic Exchange

主题交换机，这个交换机其实跟直连交换机流程差不多，但是它的特点就是在它的路由键和绑定键之间是有规则的。

简单地介绍下规则：

当一个队列的绑定键为 "#"（井号）或者\* (星号) 的时候，这个队列将会无视消息的路由键，接收所有的消息。

Dead Letter Exchange

死信交换机，与这个交换机绑定的队列，出现消息过期，或者消息总大小超过队列最大限制等，就会把消息删除，如果指定了交换机，过期就会把消息转发到指定的正常交换机和正常队列中，就会达到一个延迟队列的效果

# 事务

### 1、什么是事务？

事务是一组逻辑操作单元，即一组sql语句。要么都执行，要么都不执行。

### 事务的四大特性

四大特性分别为：

原子性，一致性，隔离性，持久性

原子性：事务要么都执行，要么都不执行；

一致性：事务使数据库从一种一致性状态到另外一种一致性状态；

隔离性：事务之间是相互隔离的。一个事务内部使用的数据以及操作对其他 并发执行的事务是相互隔离的，不能相互干扰；

持久性：事务一旦提交对数据的改变就是永久的，接下来的任何故障都不会 对其有影响；

### 3事务的传播特性

1. PROPAGATION\_REQUIRED（required）: 如果存在一个事务，则支持当前事务。如果没有事务则开启

2. PROPAGATION\_SUPPORTS(supports): 如果存在一个事务，支持当前事务。如果没有事务，则非事务的执行

3. PROPAGATION\_MANDATORY(mandatory): 如果已经存在一个事务，支持当前事务。如果没有一个活动的事务，则抛出异常。

4. PROPAGATION\_REQUIRES\_NEW(requires\_new): 总是开启一个新的事务。如果一个事务已经存在，则将这个存在的事务挂起。

5. PROPAGATION\_NOT\_SUPPORTED(supported): 总是非事务地执行，并挂起任何存在的事务。

6. PROPAGATION\_NEVER(never): 总是非事务地执行，如果存在一个活动事务，则抛出异常

7. PROPAGATION\_NESTED(nested)：如果一个活动的事务存在，则运行在一个嵌套的事务中. 如果没有活动事务,则按TransactionDefinition.PROPAGATION\_REQUIRED 属性执行

### 4事务的隔离级别

5大隔离级别：

.default(默认隔离级别),**read\_uncommitted(**读未提交**),read\_commited(**读已提交**),**

**repeatable\_read(**重复读取**),serlalizable(**串行化**)**

**1、DEFAULT（default）**

默认隔离级别，每种数据库支持的事务隔离级别不一样，如果Spring配置事务时将isolation设置为这个值的话，那么将使用底层数据库的默认事务隔离级别。顺便说一句，如果使用的MySQL，可以使用"**select @@tx\_isolation**"来查看默认的事务隔离级别

**2、READ\_UNCOMMITTED(read\_uncommitted)**

读未提交，即能够读取到没有被提交的数据，所以很明显这个级别的隔离机制无法解决脏读、不可重复读、幻读中的任何一种，因此很少使用

**3、READ\_COMMITED(read\_commited)**

读已提交，即能够读到那些已经提交的数据，自然能够防止脏读，但是无法限制不可重复读和幻读

**4、REPEATABLE\_READ(repeatable\_read)**

重复读取，即在数据读出来之后加锁，类似"select \* from XXX for update"，明确数据读取出来就是为了更新用的，所以要加一把锁，防止别人修改它。REPEATABLE\_READ的意思也类似，读取了一条数据，这个事务不结束，别的事务就不可以改这条记录，这样就解决了脏读、不可重复读的问题，但是幻读的问题还是无法解决

**5、SERLALIZABLE(serlalizable)**

串行化，最高的事务隔离级别，不管多少事务，挨个运行完一个事务的所有子事务之后才可以执行另外一个事务里面的所有子事务，这样就解决了脏读、不可重复读和幻读的问题了

### 5并发事务导致的问题

在许多事务处理同一个数据时，由于没有采取有效的隔离机制，并发处理数据时，会带来一些问题。

①、第一类丢失更新：撤销一个事务时，把其他事务已提交的更新数据覆盖。

②、脏读：一个事务读取到另一个事务未提交的更新数据。

③、幻读也叫虚读：一个事务执行两次查询，第二次结果集包含第一次中没有或某些行已经被删除的数据，造成两次结果不一致，只是另一个事务在这两次查询中间插入或删除了数据造成的。

④、不可重复读：一个事务两次读取同一行的数据，结果得到不同状态的结果，中间正好另一个事务更新了该数据，两次结果相异，不可被信任。

⑤、第二类丢失更新：是不可重复读的特殊情况。如果两个事物都读取同一行，然后两个都进行写操作，并提交，第一个事物所做的改变就会丢失

# 基于token的登录认证

### 1、Jwt

做token登录验证的话我们得先知道jwt,那么什么是jwt呢？

JWT就是一个开放标准(RFC 7519)，它定义了一种紧凑的、自包含的方式，用于作为JSON对象在各方之间安全地传输信息。该信息可以被验证和信任，因为它是数字签名的。说白了就和加密一回事

#### JWT的构成

第一部分我们称它为头部（header),

第二部分我们称其为载荷（payload, )，

第三部分是签证（signature).

### 2、基于token的鉴权机制

用户使用用户名密码来请求服务器

服务器进行验证用户的信息

服务器通过验证发送给用户一个token

客户端存储token，并在每次请求时附送上这个token值

服务端验证token值，并返回数据

## 基于token登录和认证的具体步骤

#### 登录

在坐登录注册的业务时，我决定用redis作为注册时发送验证码的缓存，在业务中首先是验证手机号格式

从redis缓存获取用户输入的手机号是否已经发送过验证码，验证手机号是不是已经注册过，防止手机号重复，然后用邮箱的方式给用户发送验证码，在redis的缓存中设置验证码的存储时间为一分钟，

当用户输入验证码传到后台后，从redis获取验证码是否一致，如果一致则为注册成功，当用户点击注册按钮时，发送到后台的用户名则判断是否唯一或者不可以是空值，密码使用MD5加密方式可以保证一定的安全度，在用户注册成功之后会发送注册成功的邮件到用户的邮箱使用了多线程可以，提高了用户体验度，

用户登录时后台根据用户名获取用户数据，判断用户的密码是否正确，用JWT令牌生成token代表用户的身份信息存储到sessionStore中，因为JWT是基于json的一种公开规范，非常方便解在用户退出登录时删除存储的token信息的一套登录注册业务；

#### Token认证

1. 用户发送携带用户名及密码的请求到后台服务；
2. 后台服务进行验证成功后，调用JWT的sign方法生产token令牌；
3. 将token值返回给前台浏览器，并存入sessionStorage；
4. 后续客户端发起请求时都需要携带该token参数；
5. 后台拦截器进行token的uusign解密验证，若没有token值或解密失败则进行拦截；

# 定时任务

我们项目中有些需求需要特定发送，比如用户生日祝福发送，发短信等，我们这时候就要用到定时器。

拿用户生日祝福来说吧，我们首先导入mail包用来发送发送邮件，在配置文件中简单配置就可以使用，要每天都要发送生日祝福邮件的话，我们可以创造一个定时器实现

### 创造定时任务

1. 然后新建我们的定时器类TimerTask
2. 在需要定时执行的方法上添加我们的注解 来设置我们的定时时间
3. @Scheduled(cron = "0 30 22 ? \* \*")
4. 设置上之后 这个方法的方法体就变成了我们定时任务的执行体 等定时时间达到之后 就会去执行方法体里面的内容
5. 注意类上需要添加我们的component注解 component注解是吧我们这个类 交给aop进行管理

### 发送邮件

发送邮件创建一个线程池来发送邮箱 线程池核心线程数根据自己的用户数量和服务器配置来均衡一个中间点，我们这用到了三个核心线程数。

循环我们的用户数据，创建我们的线程穿过去几个参数1.我们邮件的标题2.邮件的内容3.邮件接收者，之后再我们的线程执行体中（run方法中）去调用一下我们邮件的工具类，工具类中配置号我们发送方的邮箱号，设置我们的发送方的邮箱类型（**smtp.qq.com**）配置发件人的邮箱密码 （密码是我们在账号设置中设置的smtp的密匙）

设置发送人，接收人，标题，内容，这就实现了我们的邮箱发送

### 分布式事务

我们需要在定时任务的执行体里面添加一个分布式锁，目的是为了防止多台服务器集群同时触发我们的定时任务造成的邮件重复问题，这个分布式锁 我们用到了redis中的getset方法，

redis的getset方法redis中string类型下的 她会存进去新的值并且返回我们原来的值，原来没有值的话返回null；

通过getset方法把我们的当前时间作为key存到我们的redis中

当时没有给key设置过期时间，然后出现了问题，我们当时发现一个请求成功后，在发送请求无法进入方法，经过检查发现key一直存在拦截了后面的请求，然后我就把那个key设置了过期时间就顺利的解决了这一问题

### 多线程

为什么使用多线程？要用这个疑问的话首先要知道什么是多线程:

我举例来说：我们打开腾讯管家，腾讯管家本身就是一个程序，也就是说它就是一个进程，它里面有很多的功能能查杀病毒、清理垃圾、电脑加速等众多功能。

按照单线程来说，无论你想要清理垃圾、还是要病毒查杀，那么你必须先做完其中的一件事，才能做下一件事，这里面是有一个执行顺序的。

如果是多线程的话，我们其实在清理垃圾的时候，还可以进行查杀病毒、电脑加速等等其他的操作，这个是严格意义上的同一时刻发生的，没有执行上的先后顺序。

我们生日发送用多线程原因是：就是说我们任务执行的时间有些长，为了不让用户长时间等待，我们将这个时间繁琐点的任务放到多线程中去执行 主线程直接返回给用户

#### 创建线程方式

1）继承Thread类创建线程类

继承thread重写run方法

直接new 线程对象.start执行

2）通过Runnable接口创建线程类

实现runnable重写run方法

new thread

new 线程类 吧线程类放到thread中

然后通过start启动

3）通过线程池创建

线程池就可以理解为我们的线程容器

线程池的目的是为了减少我们线程的运行时间

不需要每次都去创建线程 执行后再销毁线程了 而是每次过来都去线程池中取一个线程，执行完之后吧线程归还给线程池中

#### 创建个线程池

ExecutorService executorService = Executors.newFixedThreadPool(3);//设置主线程3

### 5、Java线程具有五中基本状态：

五个状态：新建状态，就绪状态，运行状态，阻塞状态，死亡状态

1）新建状态（New）：当线程对象对创建后，即进入了新建状态，如：Thread t = new MyThread()；

2）就绪状态（Runnable）：当调用线程对象的start()方法（t.start();），线程即进入就绪状态。处于就绪状态的线程，只是说明此线程已经做好了准备，随时等待CPU调度执行，并不是说执行了t.start()此线程立即就会执行；

3）运行状态（Running）：当CPU开始调度处于就绪状态的线程时，此时线程才得以真正执行，即进入到运行状态。注：就

绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行，首先必须处于就绪状态中；

4）阻塞状态（Blocked）：处于运行状态中的线程由于某种原因，暂时放弃对CPU的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才 有机会再次被CPU调用以进入到运行状态。

根据阻塞产生的原因不同，阻塞状态又可以分为三种：

332

a.等待阻塞：运行状态中的线程执行wait()方法，使本线程进入到等待阻塞状态；

b.同步阻塞 – 线程在获取synchronized同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态；

c.其他阻塞 – 通过调用线程的sleep()或join()或发出了I/O请求时，线程会进入到阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

5）死亡状态（Dead）：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

### 6、线程安全问题

在我们运行过程中可能会遇到线程安全问题

什么是线程安全问题？

（1）运行结果错误

我们多线程成一起对变量进行操作 进行i++ 一千个线程同时执行 得到的结果不一定是1000 会有得到重复数据的 是因为同时读到变量的值 （200）同时加一 最终的结果都是201

我们可以加一个synchronized锁 目的是为了让线程走到有锁的地方单个顺序执行

synchronized是加在容易产生线程安全问题的地方

但是枷锁也会碰到逻辑安全问题

死锁

死锁是一种状态，当两个（或多个）线程（或进程）相互持有对方所需要的资源，却又都不主动释放自己手中所持有的资源，导致大家都获取不到自己想要的资源，所有相关的线程（或进程）都无法继续往下执行，在未改变这种状态之前都不能向前推进，我们就把这种状态称为死锁状态，认为它们发生了死锁。

还有优先级不够 一直拿不到cpu资源 从而导致进程不运行

# 权限管理

### 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。

# Mysql优化

## Mysql优化哪些方面？

**parameterType 接参**

**resultType 传参**

## 表设计上 存储引擎和字段类型

* 1. 存储引擎常用的有两种MylSam和InnoDB
     1. Mylsam不支持事务，不支持外键，采用表锁，侧重于性能；
     2. InnoDB支持事务，支持外键，采用行锁，侧重于事务；根据需求选择合适的存储引擎；
  2. 字段类型 给字段设置合适的数据类型，在合适的情况下使用整数型来代替字符串

## 功能上 创建索引

1. 索引简单说在列上添加标识，帮助Mysql高效获取数据。索引就是一种数据结构。
2. 索引的优点：

提高数据检索的效率，降低数据库的IO成本。

通过索引列对数据进行排序，降低数据排序的成本，降低了CPU的消耗

1. 索引的缺点：

虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT、UPDATE和DELETE。因为在更新表时，MySQL不仅要保存数据，还要保存一下索引文件每次更新添加了索引列的字段，都会调整因为更新所带来的键值变化后的索引信息。

## 3、索引分类：

* **单值索引：**

即一个索引只包含单个列，一个表可以有多个单列索引。

* **唯一索引**

索引列的值必须唯一，但允许有空值

* **主键索引**

设定为主键后数据库会自动建立索引

* **复合/组合索引**

即一个索引包含多个列

## 4、创建索引命令：

|  |  |
| --- | --- |
| 创建 | create [unique ] index indexname on table\_name(column)); |
| 删除 | drop index indexname on mytable; |
| 查看 | show index from table\_name\g; |
| 使用alter命令 | alter table tbl\_name add primary key (column\_list) : 该语句添加一个主键，这意味着索引值必须是唯一的，且不能为 null。 |
| alter table tbl\_name add primary key (column\_list) |
| alter table tbl\_name add index index\_name (column\_list): 添加普通索引，索引值可出现多次。 |
| alter table tbl\_name add fulltext index\_name (column\_list):该语句指定了索引为 fulltext ，用于全文索引。 |

1. 什么情况下应该创建索引
   1. **主键自动建立唯一索引**
   2. **频繁作为查询条件的字段应该创建索引**
   3. **查询中与其它表关联的字段，外键关系建立索引**
   4. **单键/组合索引的选择问题， 组合索引性价比更高**
   5. **查询中排序的字段，排序字段若通过索引去访问将大大提高排序速度**
   6. **查询中统计或者分组字段**
2. **什么情况不应该建立索引**
   1. **表记录太少**
   2. 经**常增删改的表或者字段**
   3. **Where 条件里用不到的字段不创建索引**
   4. **过滤性不好的不适合建立索引(字段重复的值比较多，超过50%)**

## 5、SQL语句上 合理的sql

1. 对查询进行优化，尽量避免全表扫描，首先在where和order by涉及的字段上添加索引；
2. 避免在索引列上进行计算，导致索引失效；
3. 避免在where条件上对字段的空值进行判断，否则引擎放弃索引而进行全表扫描；

# Aop日志

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。

之前在做其他项目的时候，日志模块通常都是在控制层结合log4j进行日志的控制台打印及日志文件的存储，除此之外还会把日志信息插入到mysql数据当中存储起来，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，以后维护起来也比较麻烦。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处理上。

具体在做的时候我是这样想的：

### 日志记录步骤

要做aop日志记录，首先我们需要的是自定义注解，自定义注解可以说它是一个标记点，可以添加在类、方法、属性上，注解不会对所修饰的代码产生直接的影响。便于在其他地方找到这个标记点，得到这个标记点所在位置的各项数据。创建自定义注解，先创建@Interface类，创建aop日志记录的自定义注解要使用的相关的注解有@Target，@Rentention，这两个注解的作用分别是：

@Target：是专门用来限定某个自定义注解能够被应用在哪些Java元素上面的（类、方法、属性）。

@Rentention：声明这个自定义注解的生命周期，我在使用的时候一般使用RUNTIME。

创建好自定义注解后，将其添加到controller层中需要进行日志记录方法上，之后就可以写日志记录的核心代码了。

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，后面会通过配置文件的配置让他具备切面类的功能，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，

定义一个环绕通知的方法，在方法上使用@Around(e ruan de)来定义切点表达式

在被切那个切点层的类上加上自定注解，就可以来获取方法的签名，再根据方法签名来查找到所有的方法。

根据java反射可以获取访问的方法名，参数等

而我们用户登录时，我把他们的信息存放在全局session HttpServletRequest

中，现在就可以取出来加入到日志信息中。

用户请求时间为方法执行前的毫秒数减去方法执行后的毫秒数，new一个logBean

对象，将用户请求的信息存在进去，调用service层的新增方法实现了日志 记录的新增

#### 反射

一般来说自定义注解会和反射一起使用，以此发挥出最大的作用。所谓反射，便是：对类来说，能知道它的所有属性和方法；对于对象来说，能得到所有的属性值和调用的方法，对象，即万物皆对象，注解也不例外。在此，使用getAnnotation，便可以得到添加自定义注解方法的各个数据，如操作数据，请求路径等。

# 多线程导出

我们项目中的用户数据是使用的多线程导出压缩包

### 场景分析

（为什么使用多线程)

应为我们的用户数据量庞大，为了防止一次性导出导致内存溢出，和单个excel数据量巨大无法读取的错误，所以我考虑使用多线程导出

将数据进行分批，通过不同的线程，下载到不同的excel下的多个sheet页中

最后统一打包到zip文件中进行下载

### 导出步骤

通过用户传递的count,size定制用户需要的每个excel需要多少sheet页（count）每个sheet需要多少条数据（size）

首先我们查询数据库用户的总数确定我们需要生成多少个sheet页，这些sheet页需要生成几个excel，为了保证我们的cpu资源的高利用率，提升文件的下载速度，我使用了多线程的方式进行下载，但我们最后要统一的返回一个zip压缩包，我们的主线程需要等待分支线程全部完成才能进行打包操作，这里我使用了countDownLatch，它就是声明一个变量，主线程中使用await()方法等待分支线程，每当我们一个支线程完成，通过countDown()方法进行-1，当countDownLatch为0时，在进行主线程操作

我也创建了线程池，来回收线程资源，节约线程频繁创建和销毁的资源损失

在分支线程中，我需要sheet页个数（size），几个sheet页（count），第几个excel（num），countDownLatch这些参数，这里我的线程工具类没有交给ioc去管理无法注入dao层，所以我们页通过参数传递dao层接口进去

首先创建workbook对象

根据count和num我们可以计算出页数下标，循环count次，创建count个sheet页

然后查询不同sheet页的数据

遍历数据填补到对应的row行中，全部循环结束则线程对应的一个excel生成完毕

new一个FileOutputStream流将workbook写入我们项目中的file临时文件中，关流

通过countDownLatch.countDown();告诉主线程完成

全部分支线程完毕，主线程将所有临时文件生成一个zip下载，并删除所有的临时文件

# redis非关系型数据库

### 1 什么是redis

redis 是基于key-value存储系统的非关系型数据库（缓存） redis mogodb ) 内存快 数据类型丰富 可持久化

1. 读写快

因为redis是基于内存读写的所以速度快

性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s

（2）数据类型丰富5种

丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作

1. 可持久化

(rdb 按照一定时间的key改变 进行持久化 丢失数据会丢失一片

aof 日志 写操作(增删改操作)的指令会被记录 丢失只会丢失很少的数据)

1. 原子性

原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。 单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。

（2-1）数据类型丰富5种

string（字符串）

数据结构：key value，命令 ： set 和get ，设置值： set key value，取值：get key

在哪用：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）

数据结构：key filed value，设置值：hmset key filed value，取值：hget key filed

在哪用：我们的购物车数据是用hash存在redis中的

list（列表）

设置值 ： lpush key value，value，取值 ： lrange key 下标，下标

在哪用：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

set（集合）

设置值：sadd key value ,value ，取值smembers key

zset(sorted set：有序集合)。

设置值 zadd key score value

zadd key 分值

取值 zrange by score key 小分 大分

zrange key 小分 大分

不好意思 没用过！！！

### Redis使用场景

经常访问但不会轻易改变的数据 都可以使用来减轻数据库的压力

(购物车( 电商项目使用aof方案 ) 地区 热点数据 缓存)

### **redis三大问题**

穿透 击穿 雪崩

#### 穿透

缓存穿透 根本不存在的key 是穿透

把传过来的key 设置个失效时间

##### 如何解决缓存穿透？

一般有2种解决办法，分别是缓存空值和过滤器缓存空值在缓存中，之所以会发生穿透，就是因为缓存没有对那些不存在的值得Key缓存下来，从而导致每次查询都要请求到数据库。 那么我们就可以为这些key对应的值设置为null并放到缓存中，这样再出现查询这个key 的请求的时候，直接返回null即可。但是还需要注意的就是需要有一个失效时间，因为如果不设置失效的话，如果哪天有值了就会导致问题。

#### 击穿

缓存击穿（热点数据集中失效）

缓存的key时间到期 直接去查数据库了

##### 如何解决

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

#### 雪崩

redid大量的key同时失效使得大量请求进入到数据库造成雪崩

##### 如何解决

设置不同的过期时间

为了避免大量的缓存在同一时间过期，可以把不同的key过期时间设置

成不同的， 并且通过定时刷新的方式更新过期时间。

使用缓存集群

在缓存雪崩问题防治上面，一个比较典型的技术就是采用集群方式部

署，使用集群可以避免服务单点故障。

### 集群

使用缓存集群 3种 主从复制 哨兵 集群

#### 主从复制

主从复制是指将一台Redis服务器的数据，复制到其它的Redis服务器。前者称为主节点(master)，后者称为从节点(slave)；数据的复制是单向的，只能由主节点到从节点。

默认情况下，每台Redis服务器都是主节点，且一个主节点可以有多个从节点（或没有从节点），但一个从节点只能有一个主节点。

修改配置文件

slaveof 主服务的IP 主服务端口

**优点：**

主从 1主多从 读写分离 主服务提供写服务 从服务提供读功能

因为能写仍然只有主服务一个，可以将读的操作全部移交到从服务上，变相提高了写能力。

### 哨兵

Redis 的 Sentinel 系统用于管理多个 Redis 服务器（instance）， 该系统执行以下三个任务：

**监控（Monitoring）**： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。

**提醒（Notification）**： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。

**自动故障迁移**（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作， 它会进行选举，将其中一个从服务器升级为新的主服务器， 并让失效主服务器的其他从服务器改为复制新的主服务器； 当客户端试图连接失效的主服务器时， 集群也会向客户端返回新主服务器的地址， 使得集群可以使用新主服务器代替失效服务器。

### 集群

数据分区(或称数据分片)是Redis集群最核心的功能

一个 Redis 集群包含 16384 个哈希槽（hash slot)， 数据库中的每个键都属于这 16384 个哈希槽的其中一个，集群使用公式CRC16(key) % 16384 来计算键key属于哪个槽，其中 CRC16(key)语句用于计算键key的CRC16校验和 。集群中的每个节点负责处理一部分哈希槽。

一般是3主 3从 主服务死亡 从服务顶上 如果都死亡 其他的主服务将 死亡的服务占据的哈希槽 在进行分配

# Mongodb

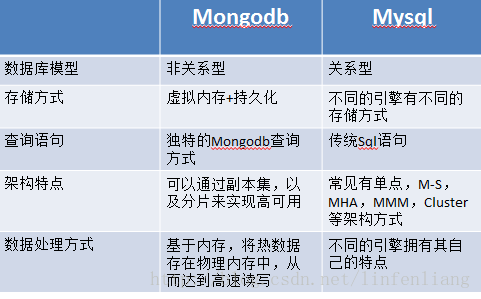
### 什么是mongodb

* 1. MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统。
  2. MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。它支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。

### MongoDB的特点

* 1. 文档数据类型
     1. SQL类型的数据库是正规化的，可以通过主键或者外键的约束保证数据的完整性与唯一性，所以SQL类型的数据库常用于对数据完整性较高的系统。MongoDB在这一方面是不如SQL类型的数据库，且MongoDB没有固定的Schema，正因为MongoDB少了一些这样的约束条件，可以让数据的存储数据结构更灵活，存储速度更加快。
  2. 即时查询能力
     1. MongoDB保留了关系型数据库即时查询的能力，保留了索引（底层是基于B tree）的能力。这一点汲取了关系型数据库的优点，相比于同类型的NoSQL redis 并没有上述的能力。
  3. 复制能力
     1. MongoDB自身提供了副本集能将数据分布在多台机器上实现冗余，目的是可以提供自动故障转移、扩展读能力。
  4. 速度与持久性
     1. MongoDB提供了Journaling日志的概念，实际上像mysql的bin-log日志，当需要插入的时候会先往日志里面写入记录，再完成实际的数据操作，这样如果出现停电，进程突然中断的情况，可以保障数据不会错误，可以通过修复功能读取Journaling日志进行修复。
  5. 数据扩展
     1. MongoDB使用分片技术对数据进行扩展，MongoDB能自动分片、自动转移分片里面的数据块，让每一个服务器里面存储的数据都是一样大小。

### ****MongoDB与MySQL的区别****



### Mongodb常用的命令

* 1. 切换数据库 sue dbs
  2. 插入语法
     1. db.users.insert({username:"smith"})
     2. db.users.save({username:"smith"})
     3. 区别若新增的数据中存在主键 ，insert() 会提示错误，而save() 则更改原来的内容为新内容。
  3. 查找语句
     1. db.users.find()
     2. db.users.count()
  4. 集合的相关操作

db.createCollection() 创建集合

db.集合名称.drop() 删除集合

show collections( tables) 查看所有集合

db.集合名.insert(文档) 集合添加数据 save insertone insermany

* 1. 文档的操作

db.集合名.insert(文档) 集合添加数据 save insertone insermany

db.集合名.remove

db.集合名.update

db.集合名.find()

# Docker容器引擎

## 1、docker是什么？

Docker是一款可以解决运行环境和配置问题的软件容器引擎；

Docker的组成部分（容器，镜像（image），仓库）

## 2、docker的相关命令

docker version 查看docker版本

docker info 查看docker的详细信息（cpu 内存 容器 镜像）

镜像的操作命令：

docker images 列出 本机的所有镜像 镜像名 版本号 id 创建时间 大小

docker search 镜像名 搜索镜像 收藏数

docker pull 镜像名:tag 下载一个centos 的一个镜像 docker pull centos 下载镜像不指定版本

docker rmi 镜像名（id） 注意正在被容器使用的镜像是无法删除

容器的操作命令 （有镜像才能创建容器）

docker run -it --name 容器名称（唯一的） 镜像（创建容器 并进入终端 一旦退出 容器结束）

docker run -itd --name 容器名称（唯一的） 镜像（创建容器 进入后台运行不会进行终端 （exec 进入终端） 终端退出 容器继续运行）

进入后台运行容器的终端 docker exec -it 容器名称（容器id） /bin/bash

退出容器 Ctrl+P+Q

查看运行中的容器 、docker ps

查看所有的容器docker ps -a

删除容器docker rm 容器id（名称）（运行中的容器不能删 如果想把运行中的容器也删掉 加参数 -f）

停止容器 docker stop 容器id（名称）

启动容器docker start 容器id（名称）

拷贝 宿主机向容器中拷贝文件

docker cp 宿主机的文件 容器id:目录

拷贝 容器中的文件 拷贝到宿主机上

docker cp 容器id:目录 宿主机的目录

## 3、数据卷怎么创建 (创建一个容器 并和宿主机共享目录)

docker run -itd --name 容器名 -v 宿主目录:/容器目录 镜像 /bin/bash

数据卷容器 （多个容器共享一个目录 （修改一个目录 多个容器跟着改变））

先创建一个共享的容器

docker run -v 宿主名录:容器目录 --name 容器名称 镜像 /bin/bash

创建容器指定共享的容器 实现共享目录

docker run -itd --volumes-from 共享容器名 --name容器名称 镜像 /bin/bash

## 4、制作镜像

commit

build（Dockerfile）

编写Dockerfile文件

语法 ：

FROM 指定基础镜像

MAINTAINER 指定镜像维护者信息

RUN 用于执行指定脚本命令

CMD 指定启动容器时执行的命令

EXPOSE 指定容器暴露的端口

ENV 指定环境变量

ADD 将文件从宿主机复制到容器指定位置，同时对压缩文件有自动解压功能

COPY 将文件从宿主机复制到容器指定位置

ENTRPOINT 设置容器启动时需要运行的命令

WORKDIR 为后续的如RUN、CMD、ENTRYPOINT、COPY、ADD指定工作目录

# mvc思想

MVC并不是一个啥新技术，它就是一种设计模式或者说是一种编程思想，使用MVC的目的是将M和V的实现代码分离。

M是指模型层也就是业务逻辑层，V是指视图层也就是用户界面，C则是控制层也就是控制器。它是在上世纪八十年代发明的一种软件设计模式，至今已被广泛使用

## 1视图

刚刚提到视图，视图呢详细说的话就是是用户看到并与之交互的界面（它可以包括一些可以显示数据信息的页面，或者展示形式。例如jsp，html，asp，php）

MVC一个大的好处是它能为你的应用程序处理很多不同的视图。在视图中其实没有真正的处理发生，不管这些数据是联机存储的还是一个雇员列表，作为视图来讲，它只是作为一种输出数据并允许用户操纵的方式。

## 2模型

还有模型，模型表示企业数据和业务规则（可以说就是后端接口，用于业务处理）。在MVC的三个部件中，模型拥有最多的处理任务。被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型能为多个视图提供数据。由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

3控制器

最后还有个控制器，控制器接受用户的输入并调用模型和视图去完成用户的需求（接受客户发送的请求，根据请求调用所对应的接口，然后模型业务处理后返回的数据，由控制器决定调用那个View展示）

## 总结

通过我们上面说的就可以了解到mvc的整个处理过程，MVC的处理过程就是，首先控制器接收用户的请求，并决定应该调用哪个模型来进行处理，然后模型用业务逻辑来处理用户的请求并返回数据，最后控制器用相应的视图格式化模型返回的数据，并通过表示层呈现给用户。

## Mvc缺点

MVC的缺点是由于它没有明确的定义，所以完全理解MVC并不是很容易。使用MVC需要精心的计划，由于它的内部原理比较复杂，所以需要花费一些时间去思考。

根据我个人经验，由于我们将一个应用程序分成了三个部件，所以使用MVC同时也意味着你将要管理比以前更多的文件，这一点是显而易见的。这样好像我们的工作量增加了，但是请记住这比起它所能带给我们的好处是不值一提。

# Model

model层就是写数据库的增删改查操作是跟数据库打交道

我用过 的model层框架有:

Jdbc,mybatis ,hibernate, jpa

## Jdbc

jDBC（Java DataBase Connectivity,java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序，但是这个东西的话现在基本没人用，（他问为什么:）jdbc的灵活就比较差，搭建也比较麻烦，没有mybatis搭建方便，也没有mybatis灵活（如果问为啥,就把介绍mybatis）

(如果不问就说下面的)

因为好久都没有用过了现在只能记得它访问数据库的过程：

***访问数据库的过程***

主要分为六大步骤：

1.加载sql驱动 （关键词：class.forName）

2.使用DriverManager获取数据库连接 （关键词：DriverManager.getConnection）

3.使用Connecttion来创建一个Statement对象 Statement对象用来执行SQL语句 （关键词：con.createStatement）

4.执行SQL语句 （关键词：execute）

5.操作结果集 （关键词：while）

6.回收数据库资源 （关键词：close）

## Mybatis

mybatis是一个持久层框架，使用sql语句操作的是数据库，底层都是JDBC。

MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久

层框架。MyBatis 避免了几乎所有的 JDBC 代码和手工设置参数以

及抽取结果集。MyBatis 使用简单的 XML 或注解来配置和映射基本

实体，将接口和 Java 的普通的 Java对像 映射成数据库中的记录。

mybatis很灵活:

mybatis让程序员将主要精力放在sql上，通过mytabis提供的映射方式，自动生成满足需要的sql语句，不满足的话也可以自己去写sqL

## hibernate

***hibernate 简介：***  
hibernate是一个开源框架，它是对象关联关系映射的框架，它对JDBC做了轻量级的封装，而我们java程序员可以使用面向对象的思想来操纵数据库。  
***hibernate核心接口***  
session：负责被持久化对象CRUD操作  
sessionFactory:负责初始化hibernate，创建session对象  
configuration:负责配置并启动hibernate，创建SessionFactory  
Transaction:负责事物相关的操作  
Query和Criteria接口：负责执行各种数据库查询

***hibernate中session中的常用方法***

save():保存

saveOrUpdate()保存或者修改

delete()删除

update()修改

get()根据唯一标示获取对象 立即加载

load()根据唯一标示获得对象 ，延迟加载

***hibernate运行原理***

1.通过configuration加载了hibernate.cfg.xml文件。

2.加载后创建sessionFactory.(sessionFactory是线程安全的)

3.然后通过sessionFactory获得session。（创建会话，相当于jdbc中的connection 是线程不安全的）

4.通过session操作数据库，最后通过transaction来进行事物的控制.

## Jpa

JPA即Java持久化API，JPA的出现主要是为了简化持久层（Dao层）开发以及整合ORM技术，结束Hibernate、TopLink(套铺赁克)、JDO等ORM框架各自为营的局面。

***Spring Data JPA的核心接口***

l Repository(瑞胖丝特瑞): 所有接口的父接口，而且是一个空接口，目的是为了统一所有Repository的类型，让组件扫描的时候能进行识别。

l CrudRepository：是Repository的子接口，提供CRUD(增删改查)的功能。

l Paging(培经)AndSorting(消停)Repository：是CrudRepository的子接口，添加分页和排序的功能。

l JpaRepository：是PagingAndSortingRepository的子接口，增加了一些实用的功能，例如批量操作。

l JpaSpecification(丝拍谁非K鑫)Executor(A克丝Q特儿)：用来做负责查询的接口。

l Specification(丝拍谁非K鑫)：是Spring Data JPA提供的一个查询规范，要做复杂的查询，只需围绕这个规范来设置查询条件即可。

***JPA中常用的注解***

@Entity【嗯忒忒】 标识当前类为实体类

@Table 配置与实体类相对应的数据库表信息(如表名)

@Id 标识当前字段为主键

@Generated(杰呢瑞忒德)Value 配置主键生成策略

@Column(靠乐木) 配置属性对应的数据库字段信息(如字段名)

@ManyToOne【美图玩】 配置一对多关系

@ManyToMany【没图没】 配置多对多关系

@Transient这个注解表示忽略这个属性，那么建表的时候就不会在数据库表中生成一个对应的字段了！

# controller

## servlet

Servlet是sun公司提供的一门用于开发动态web资源的技术。  
Sun公司在其API中提供了一个servlet接口，用户若想用发一个动态web资源(即开发一个Java程序向浏览器输出数据)，需要完成以下2个步骤：  
　　1、编写一个Java类，实现servlet接口。  
　　2、把开发好的Java类部署到web服务器中。  
　　按照一种约定俗成的称呼习惯，通常我们也把实现了servlet接口的java程序，称之为Servlet

Servlet的运行过程

Servlet程序是由WEB服务器调用，web服务器收到客户端的Servlet访问请求后：  
　　①Web服务器首先检查是否已经装载并创建了该Servlet的实例对象。如果是，则直接执行第④步，否则，执行第②步。  
　　②装载并创建该Servlet的一个实例对象。   
　　③调用Servlet实例对象的init()方法。  
　　④创建一个用于封装HTTP请求消息的HttpServletRequest对象和一个代表HTTP响应消息的HttpServletResponse对象，然后调用Servlet的service()方法并将请求和响应对象作为参数传递进去。  
　　⑤WEB应用程序被停止或重新启动之前，Servlet引擎将卸载Servlet，并在卸载之前调用Servlet的destroy()方法。

 Servlet的优点

1、  是mvc的基础，其他的框架比如struts1，struts2，webwork都是从servlet基础上发展过来的。所以掌握servlet是掌握mvc的关键。

2、  Servlet把最底层的api暴漏给程序员，使程序员更能清楚的了解mvc的各个特点。

3、  程序员可以对servlet进行封装。Struts2就是从servlet中封装以后得到的结果。

4、  市场上任何一个mvc的框架都是servlet发展过来的，所以要想学好struts2这个框架，了解servlet的运行机制很关键。

2   Servlet的缺点

1、  每写一个servlet在web.xml中都要做相应的配置。如果有多很servlet，会

导致web.xml内容过于繁多。

2、  这样的结构不利于分组开发。

3、  在servlet中，doGet方法和doPost方法有HttpServletRequest和HttpServletResponse参数。这两个参数与容器相关，如果想在servlet中作单元测试，则必须初始化这两个参数。

4、  如果一个servlet中有很多个方法，则必须采用传递参数的形式，分解到每一个方法中。

## springmvc

在web模型中，MVC是一种很流行的框架，通过把Model，View，Controller分离，把较为复杂的web应用分成逻辑清晰的几部分，是为了简化开发，减少出错。还是为了组内开发人员之间的配合。总之就是一种分层工作的办法。

springMVC，是spring的一个子框架，当然拥有spring的特性，如依赖注入，正因为这一点，通过简单的配置，就能够让SpringMVC框架和Spring容器一起使用。

***springMVC的运行原理***

1. 客户端请求提交到DispatcherServlet(第四排客车Servlet)

   2. 由DispatcherServlet控制器查询一个或多个HandlerMapping(憨的了卖屏)，找到处理请求的handler(憨的了)。

   3. DispatcherServlet将请求转发给到handler。

   4. 通过HandlerAdapter(憨的了额搭铺特)执行handler调用业务逻辑处理后，返回ModelAndView。

   5. DispatcherServlet查询一个或多个ViewResolver()视图解析器，将model(查询得到的数据)渲染到views(前端页面)。

   6.将views结果显示到客户端。

***SpringMVC框架的优点***：

1）SpringMVC框架提供了一整套完善的组件。

不同于以前框架只提供接口，然后让使用者自己根据接口写实现类来实现功能，SpringMVC提供了一整套完善的组件，可以直接使用。这就相当于以前的框架只给你提供设计图纸，你需要根据设计图纸自己把工具造出来，而现在SpringMVC直接给你一套现成的工具箱，里面有你用得到的各种工具。

而且使用SpringMVC框架提供这些工具也巨简单，通过一些简单的注解就搞定了，不需要编写繁琐冗长的代码

2）SpringMVC是以强大的Spring容器为基础的框架。

我们都使用过Spring容器，那你一定知道Spring容器的对象托管功能有多么的好用。它不需要你自己去创建和释放对象，Spring容器帮你全搞定! 同样地，能够放入Spring容器里面的这个SpringMVC工具箱，里面各种工具类的管理也是不需要操心的。而这省心省力的优点，是其他框架都没有的

3）框架的配置简单又不失灵活性。

SpringMVC框架的常用配置项就已经涵盖了项目中80%的配置需求，简单的项目甚至可以做到零配置，拿过来就用。

4）代码的可重用性很高。

翻译成大白话就是，一个成熟的，上线稳定运行过的SpringMVC项目，经过简单修改，甚至不用修改，就可以作为另一个新项目开发的基础。这样能在开发新项目时省很多力气。

5）可扩展性好。

如果SpringMVC框架提供的工具不能100%满足你的需求，你需要定制开发，SpringMVC框架同样也提供了相应的接口，可以进行工具的升级。这样你在遇到强力精英怪的时候，照样能用SpringMVC框架打败它。

springboot

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化Spring应用初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。Spring Boot其实就是一个整合很多可插拔的组件（框架），内嵌了使用工具（比如内嵌了Tomcat、Jetty等），方便开发人员快速搭建和开发的一个框架。

Spring Boot使创建独立的、基于生产级Spring的应用程序变得很容易，您可以“直接运行”这些应用程序。 我们对Spring平台和第三方库有自己的见解，这样您就可以轻松入门了。大多数Spring引导应用程序只需要很少的Spring配置

创建Spring boot的两种方式：

1、spring boot官网：<https://start.spring.io/>

2、使用Idea创建spring boot项目

springboot特点：

1 快速搭建spring 框架 2 约定大于配置 3 开箱即用 4 嵌入了web容器

Spring 最初最核心的两大核心功能 Spring Ioc 和 Spring Aop 成就了 Spring，

Spring 在这两大核心的功能上不断的发展，才有了 Spring 事务、Spring Mvc 等

一系列伟大的产品，最终成就了 Spring 帝国，到了后期 Spring 几乎可以解决企

业开发中的所有问题。

Spring Boot 是在强大的 Spring 帝国生态基础上面发展而来，发明 Spring Boot 不

是为了取代 Spring ,是为了让人们更容易的使用 Spring

# shiro 安全框架

* Apache Shiro是一个强大而灵活的开源安全框架，它能够干净利落地处理身份认证、授权、企业会话管理和加密。相比spring security框架更简单灵活，spring security对spring依赖较强。shiro可以实现web系统、c/s、分布式等系统权限管理。

## **功能**

1. 登录认证
2. 权限认证
3. 加密

## 优点

1控制在线人数和同时在线问题

2所有的后端管理框架都可以使用

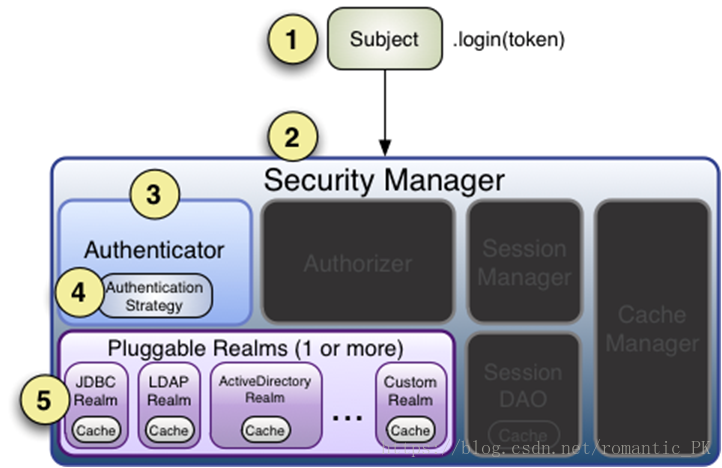
## **shiro的初步理解**

1. 默认基于cookie加session的模式完成登录认证.
2. 也可以重写一些方法和接口改成jwt的token验证.
3. 基于缓存的安全认证框架，他可以使用第三方缓存如redis.

## shiro的组成部分

1. Subject：主体，即当前操作用户
2. 2.(瑟Q弱tei)SecurityManager：安全管理器，它是Shiro框架的核心，典型的Facade模式，Shiro通过SecurityManager来管理内部组件实例，并通过它来提供安全管理的各种服务
3. Realm：领域，充当了Shiro与应用安全数据间的“桥梁”或者“连接器”。也就是说，当对用户执行认证（登录）和授权（访问控制）验证时，Shiro会从应用配置的Realm中查找用户及其权限信息
4. Authenticator(奥森特tei 特r)：认证器，AuthenticationStrategy(死抓特zhei)如果存在多个realm，则按着具体的策略进行登录控制，例如：如果有一个realm成功即可登录、必须所有realm都成功才能登录等
5. Authorizer（奥斯ruai zer）：授权器，决定subject能拥有什么样角色或者权限。
6. SessionManager：session管理器，创建和管理用户session。通过设置这个管理器，shiro可以在任何环境下使用session。
7. (卡诗)CacheManager：缓存管理器，可以减少不必要的后台访问。提高应用效率，增加用户体验。
8. Cryptography（抠铺淘客fei）：Shiro的api大幅度简化java api中繁琐的密码加密。

## **shrio的认证流程**



subject(主体)请求认证，调用subject.login(token)

SecurityManager (安全管理器)执行认证

SecurityManager通过ModularRealmAuthenticator进行认证。

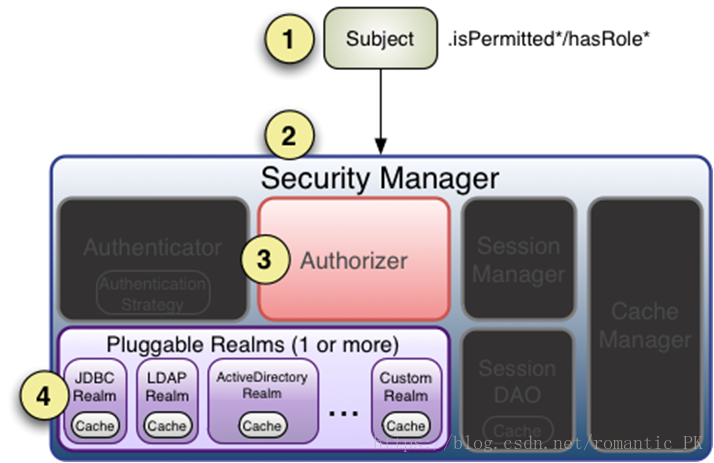
ModularRealmAuthenticator将token传给realm，realm根据token中用户信息从数据库查询用户信息（包括身份和凭证）

realm如果查询不到用户给ModularRealmAuthenticator返回null，ModularRealmAuthenticator抛出异常（用户不存在）

realm如果查询到用户给ModularRealmAuthenticator返回AuthenticationInfo(认证信息)

ModularRealmAuthenticator拿着AuthenticationInfo(认证信息)去进行凭证（密码）比对。如果一致则认证通过，如果不致抛出异常（凭证错误）。

## **Shiro授权流程：**



对subject进行授权，调用方法isPermitted("")或者hasRole("")

SecurityManager执行授权，通过ModularRealmAuthorizer执行授权

ModularRealmAuthorizer执行realm（自定义的CustomRealm）从数据库查询权限数据调用realm的授权方法：doGetAuthorizationInfo

realm从数据库查询权限数据，返回ModularRealmAuthorizer

ModularRealmAuthorizer调用PermissionResolver进行权限串比对

如果比对后，isPermitted中"permission串"在realm查询到权限数据中，说明用户访问permission串有权限，否则没有权限，抛出异常。

# Linux

### Linux的常用命令

* 1. Ls查看当前文件夹下的内容
  2. pwd查看当前所在文件夹
  3. cd 切换目录
  4. mkdir[目录名] 创建文文件夹
  5. rm[文件名] 删除指定文件
  6. clear清空屏幕
  7. # 打包文件 tar -cvf 打包文件.tar 被打包的文件／路径...
  8. # 解包文件 tar -xvf 打包文件.tar
  9. # 解压缩文件 tar -zxvf 打包文件.tar.gz
  10. cp复制文件或者目录
  11. 查看某个服务是否已经启动 ps -ef |grep 服务名
  12. 启动redis ./redis-server ../redis.conf
  13. 杀死某个进程 kill -9 进程号
  14. 查看防火墙状态
      1. 查看防火墙状态 systemctl status firewalld
      2. 开启防火墙 systemctl start firewalld
      3. 关闭防火墙 systemctl stop firewalld
      4. 若遇到无法开启
      5. 先用：systemctl unmask firewalld.service
      6. 然后：systemctl start firewalld.service
  15. 对外开发端口
      1. 查看想开的端口是否已开：
      2. firewall-cmd --query-port=6379/tcp
      3. 添加指定需要开放的端口：
      4. firewall-cmd --add-port=123/tcp --permanent
      5. 重载入添加的端口：
      6. firewall-cmd --reload
      7. 查询指定端口是否开启成功：
      8. firewall-cmd --query-port=123/tcp

# Nginx

### ****Nginx是什么****

* 1. 在服务器集群中，Nginx起到一个代理服务器的角色（即反向代理），为了避免单独一个服务器压力过大，将来自用户的请求转发给不同的服务器。

### Nginx负载策略

* 1. 轮询 这也是nginx的默认负载策略
  2. Weight 这是权重策略 根据权重进行分配 类似7,3开
  3. ip\_hash IP策略 就是所用户访问了一台服务之后就一直访问这个服务 可以保证session回话
  4. least\_conn 这是最少链接的策略 也就是说那台服务器连接数少就分配到那一台

# Maven

刚刚有提到我们用maven来搭建我们的springboot 现在说一下maven吧

### 什么是maven

maven是用来构建项目和管理依赖的工具。构建项目就是创建项目，而且可以在项目编码完成之后,对项目进行编译、测试、打包、部署操作。管理依赖呢，就是对项目中所需要的jar包进行管理。

### **1.**   **为什么要使用Maven(Maven的好处)**

1.    使用maven创建出来的项目具有统一的结构

2.    规范jar包的下载

3.    节约项目的空间，让项目不是那么臃肿

4.    Maven可以把一个jar包所依赖的jar也一并下载下来

5.    通过maven可以对项目进行编译、测试、打包、部署操作

### **2.**   **Maven仓库分为哪几种?**

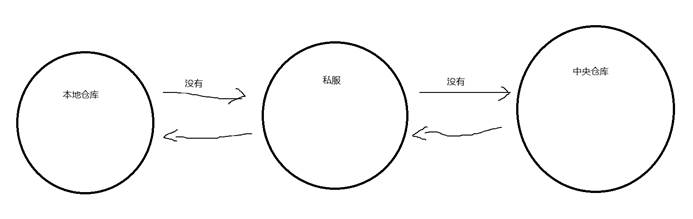
**本地仓库**:自己电脑上的一个文件夹，本地仓库默认设在了${user.home}/.m2/repository，本地仓库设在C盘有一点不好的地方，就是重装操作系统之后C盘被格式化了，之前仓库里的jar包就没了，所以咱们通常会修改本地仓库的位置，在maven安装目录下的conf/setting.xml中可以更改本地仓库的位置。

**中央仓库:**maven官方维护的一个仓库，里面包含了市面上绝大部分开源项目的jar包。但是中央仓库服务器所在遥远的美国，由于我们都懂的原因，访问速度比较慢，还有可能会中断。想要连接中央仓库必须得保证电脑联网了。

**第三方仓库(也就是私服)**:是架设在局域网的仓库，通常情况下公司会搭建自己的Maven私服，这样可以保证开发人员拿到的jar包都是统一的。而且我们班级之前也老师搭建的也有maven私服，大家都连接这个maven私服就可以下载jar包了。

### **3.**   **Maven获取jar包的顺序**

当本地仓库没有你需要的jar包时而且你也没有配置私服仓库地址时，maven会去中央仓库下载jar包。如果你的电脑上配置有私服仓库地址，则maven会去私服里下载jar包，如果私服也没有jar包，则私服会去中央仓库下载jar包。



### **4.**   **Setting(塞挺).xml和pom(炮木).xml**

settings.xml是maven的全局配置文件。Settings.xml中包含类似**本地仓库位置**、**修改远程仓库服务器**、**认证信息**等配置。而**pom.xml**主要描述了**项目的maven坐标**，**依赖关系**以及**其他所有的项目相关因素**，是项目级别的配置文件。

### **5.**   **Maven常用命令(掌握5个)**

mvn clean(科林)    清空产生的项目( target里)

mvn compile(凯木排欧) 编译源代码

mvn package(排K继)     打包

mvn test           运行测试

mvn test-compile   编译测试代码

mvn -Dtest package  只打包不测试

mvn  source(扫斯).jar   源码打包

# 前端框架

vue是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。

### （1）属性

{{info}} 取值

el:"#",绑定元素

data:{} 绑定数据

methods:{}绑定元素中包含的函数

Watch 监听属性

Computed 计算函数

### （2）VUE的常用指令

v-model指令 创建双向数据绑定

v-if指令，v-else指令 控制元素 有或没有

v-show指令，v-else指令 控制数据显示或不显示

v-for指令 遍历数组元素

v-bind指令 绑定标签的属性

v-on指令 用于给监听事件

### （3）生命周期 钩子函数

beforeCreate（创建前），created（创建后），

beforeMount（载入前），mounted（载入后），

beforeUpdate（更新前），updated（更新后），

beforeDestroy（销毁前），destroyed（销毁后）

### （4）什么是路由：

在我们浏览网页的过程中，进入一个网页，大多时候就会发起一次http请求，而前端中的路由，就是实现在页面中跳转而不会再次发送请求，有时我们再网页中看到 地址中带有#符号，这就代表了当前使用了路由

### （5）组件：

组件是可复用的 Vue 实例, 如果网页中的某一个部分需要在多个场景中使用，那么我们可以将其抽出为一个组件进行复用。组件大大提高了代码的复用率用来做分布式开发的

需要自定义标签 $emit子调父 $props父传子

### （6）什么是跨域

当一个请求url的协议、域名、端口三者之间任意一个与当前页面url不同即为跨域

### （7）为什么会出现跨域问题

出于浏览器的同源策略限制。同源策略（Sameoriginpolicy）是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，则浏览器的正常功能可能都会受到影响。可以说Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。同源策略会阻止一个域的javascript脚本和另外一个域的内容进行交互。所谓同源（即指在同一个域）就是两个页面具有相同的协议（protocol），主机（host）和端口号（port）

### （8）跨域解决方法：

JSONP 是服务器与客户端跨源通信的常用方法。最大特点就是简单适用，兼容性好（兼容低版本IE），缺点是只支持get请求，不支持post请求

Urus：

@CrossOrigin

以便启用CORS（默认情况下，@CrossOrigin允许在@RequestMapping注解中指定的所有源和HTTP方法

### （9）什么是同源（域）：

相同的协议

相同的ip

相同的端口

# Seata

Seata的分布式事务解决方案是业务层面的解决方案，只依赖于单台数据库的事务能力。Seata框架中一个分布式事务包含3中角色：

* **Transaction Coordinator (TC)： 事务协调器，维护全局事务的运行状态，负责协调并驱动全局事务的提交或回滚。**
* **Transaction Manager (TM)： 控制全局事务的边界，负责开启一个全局事务，并最终发起全局提交或全局回滚的决议。**
* **Resource Manager (RM)： 控制分支事务，负责分支注册、状态汇报，并接收事务协调器的指令，驱动分支（本地）事务的提交和回滚。**

其中，TM是一个分布式事务的发起者和终结者，TC负责维护分布式事务的运行状态，而RM则负责本地事务的运行。

下面是一个分布式事务在Seata中的执行流程：

1. **TM 向 TC 申请开启一个全局事务，全局事务创建成功并生成一个全局唯一的 XID。**
2. **XID 在微服务调用链路的上下文中传播。**
3. **RM 向 TC 注册分支事务，接着执行这个分支事务并提交（重点：RM在第一阶段就已经执行了本地事务的提交/回滚），最后将执行结果汇报给TC。**
4. **TM 根据 TC 中所有的分支事务的执行情况，发起全局提交或回滚决议。**
5. **TC 调度 XID 下管辖的全部分支事务完成提交或回滚请求。**

## 为什么Seata在第一阶段就直接提交了分支事务？

Seata能够在第一阶段直接提交事务，是因为Seata框架为每一个RM维护了一张UNDO\_LOG表（这张表需要客户端自行创建），其中保存了每一次本地事务的回滚数据。因此，二阶段的回滚并不依赖于本地数据库事务的回滚，而是RM直接读取这张UNDO\_LOG表，并将数据库中的数据更新为UNDO\_LOG中存储的历史数据。

如果第二阶段是提交命令，那么RM事实上并不会对数据进行提交（因为一阶段已经提交了），而实发起一个异步请求删除UNDO\_LOG中关于本事务的记录。

## **Seata执行流程**

首先TM 向 TC 申请开启一个全局事务，全局事务创建成功并生成一个全局唯一的 XID。

XID 在微服务调用链路的上下文中传播。

RM 开始执行这个分支事务，RM首先解析这条SQL语句，生成对应的UNDO\_LOG记录。

RM在同一个本地事务中执行业务SQL和UNDO\_LOG数据的插入。在提交这个本地事务前，RM会向TC申请关于这条记录的全局锁。如果申请不到，则说明有其他事务也在对这条记录进行操作，因此它会在一段时间内重试，重试失败则回滚本地事务，并向TC汇报本地事务执行失败。

RM在事务提交前，申请到了相关记录的全局锁，因此直接提交本地事务，并向TC汇报本地事务执行成功。此时全局锁并没有释放，全局锁的释放取决于二阶段是提交命令还是回滚命令。

TC根据所有的分支事务执行结果，向RM下发提交或回滚命令。

RM如果收到TC的提交命令，首先立即释放相关记录的全局锁，然后把提交请求放入一个异步任务的队列中，马上返回提交成功的结果给 TC。异步队列中的提交请求真正执行时，只是删除相应 UNDO LOG 记录而已。

RM如果收到TC的回滚命令，则会开启一个本地事务，通过 XID 和 Branch ID 查找到相应的 UNDO LOG 记录。将 UNDO LOG 中的后镜与当前数据进行比较，如果有不同，说明数据被当前全局事务之外的动作做了修改。这种情况，需要根据配置策略来做处理。否则，根据 UNDO LOG 中的前镜像和业务 SQL 的相关信息生成并执行回滚的语句并执行，然后提交本地事务达到回滚的目的，最后释放相关记录的全局锁。

# Java基础

1. **什么是Java?**

Java是一门面向对象的编程语言；

Java的三大特征：

1）封装（私有化属性、生成set get方法、生成有参午餐构造），

2）继承（子类继承父类拥有父类的所有属性和方法），

3）多态（重写：子类重写父类的方法，重载：同一类中方法名相同参数列表不同）

1. **Java的数据类型：**

基本数据类型：类型 位数 取值范围 包装类

整数型：byte 1个字节 -128——127（字节型） Byte

short 2个字节 -2的15次方——2的15次方-1（短整型） Short

int 4个字节 -2的31次方——2的31次方-1（整数型） Integer

long 8个字节 -2的63次方——2的63次方-1（长整型） Long

浮点型：float 4个字节 精度：小数点后7位（单精度） Float

Double 8个字节 精度：小数点后15位（双精度） Double

字符型：char 2个字节 只存储一个字符 Character

布尔型：boolean 1字节 只有 true 或false Boolean

引用数据类型：

类 Class、 接口 Interface、 数组Array

1. **包装类的常用方法**

Integer parseInt(String) String转Integer

valueOf(String或者int) String或者int 转 Integer

toString(Integer) Integer 转 String

Double parseDouble(String) String转 Double

valueOf(String) String 转 Double

toString(Double) double 转String

1. **基础语法**

if语句：

if（条件){方法体}else{}，if（条件）{}else if（条件）{}。。。。

For循环

For(初始值，循环条件，步长){}

While循环

while(条件){方法体；步长}//循环开始会先执行一次方法体

1. **String的常用方法：**

charAt（下标） 返回下标对应的字符；

indexOf（字符串) 返回字符串对应的下标；（不存在返回-1）

Length（） 返回字符串的长度；

Replace（字符串1，字符串2）将字符串1替换为字符串2；

Contains（）判断字符串是否包含参数中的字符串（返回true或false）；

toUpperCase()将字符串的字母转为大写；

toLowerCase()将字符串的字母转为小写；

Tirm()去掉字符串前后的空格；

Split(字符) 按字符将字符串进行分割得到字符串数组；

Substring(start,end) 从开始下标截取到结束下标的前一个字符（包前不包后）；

Concat(字符串) 将字符串进行拼接得到新的字符串；

1. **Date的相关方法**

getTime()获取毫秒值

转换类SimpleDateFormat，parse(String)将字符串转换成Date，format(Date)将date转换成String

1. **File的常用方法**

Exist() 判断文件夹是否存在 mkdir()创建文件夹 delete()删除文件

1. **Io流**

输入流 input （用于将本地文件上传到服务器中）

1. inputStream（字节输入流）
2. reader （字符输入流）

输出流 out（用于从服务器下载文件）

1）outputStream（字节输入流）

2）writer （字符输入流）

8、**继承的特点 extends**

①写在子类中

②格式：public class 子类类名 extends\* 父类类名{}

③子类继承父类，拥有父类所有的属性和方法；

但不能继承父类私有的特性和构造方法；

④创建子类对象时，会默认先调用父类的构造方法；

⑤一个子类只能有一个父类，但一个父类可以有多个子类（单继承）；

⑥将类中相同的属性和方法，向上提取并封装成父类，增加代码复用性，减少代码量

**9、重载的特点**

在同一个类中，方法名相同，参数的个数、类型、顺序至少有一个不同，就构成方法的重载，重载与修饰符和返回值类型无关。

**10、重写的特点**

①发生在继承关系中；

②方法名、返回值类型、参数列表必须与父类完全一致；

③子类不能重写父类私有方法和static方法；

④子类重写父类 访问修饰符权限不能小于父类，

不能报出比父类更大的异常。

**11、重写和重载的区别**

①重载要在同一个类中

重写要在存在继承关系的两个或多个类中，必须时非私有非静态的方法；

②重载要求参数的个数、类型、顺序至少有一个不同

重写要求方法名和返回值类型与父类完全一致；

③重载与访问修饰符无关,与异常无关

重写的访问修饰符权限不能小于父类；不能报出比父类更多的异常。

**12、抽象类 抽象方法的特点abstract 抽象的**

①abstract修饰的类叫做抽象类 修饰的方法叫做抽象方法

②抽象类不能被实例化，使用抽象类中的属性与方法时，要去创建一个类

去继承抽象类；

③抽象方法没有方法体；

④抽象方法要在抽象类中；

⑤抽象类中可以有抽象方法，也可以有普通方法；

⑥一个类继承抽象类，要重写抽象类中所有的抽象方法；

**13、接口的特点interface 接口** implements 实现

①接口中所有的方法都是抽象方法；

②final修饰的属性成为常量，必须赋初始值，不能二次赋值；

③接口不能被实例化，使用接口中的属性和方法时，要创建一个类实现接口

④实现接口必须重写接口中所有的抽象方法；

⑤一个类可以实现多个接口（多实现）

14、**接口和抽象类的区别（异同点）**

相同点：

①都可以有抽象方法

②都不能被实例化

③一个类要继承抽象类 必须重写抽象类中所有的抽象方法

一个类实现接口 也必须重写接口中所有的抽象方法

不同点：

①接口不是类，用interface表示，

抽象类是一个用abstract修饰符修饰的类

②接口里面只能有抽象方法

抽象类中可以有抽象方法和普通方法

③一个类可以实现多个接口

但是只能继承一个类

④接口中的变量都是final修饰的常量，必须赋初始值不能二次赋值

而抽象类没有限制

1. **final的特点**

1）修饰的类不能被继承，修饰的方法不能被重写；

2）修饰的属性是常量，必须赋初始值，不能二次赋值；

**16、BigDecimal（数学类5）**

常用方法

1）add（） 加法

2）Subtract（） 减法

3）Multiply（） 乘积

4）Divide（） 除法

5）Remainder（） 求余数

6）Min（） 最小值

7）Max（） 最大值

8）Abs（） 绝对值

9）Negate（） 相反数

**17、Random类**

nextBoolean()在boolean取值范围内随机取值；

nextDouble()：0-1之间的小数 有效位15-16位；

nextFloat()：0-1之间的小数 有效位7-8位

nextInt()：int取值范围内的所有数的随机

nextInt(101)：[0-100]

nextInt(参数) 取值范围 是 0到参数-1

随机取出0--参数之前的书（不包含本身）

nextInt(参数)+数 取出指定范围内的随机一个数

括号内的数为终止数，括号外为从多少开始

**18、String和StringBuilder和StringBuffer的区别**

1）String是一个字符串常量，长度固定

2）StringBuffer是一个长度可变的量，线程安全，运行效率低高于String

3）StringBuilder是一个长度可变的量，线程不安全，运行效率高于StringBuffer高于String

**19、集合**

1）collection （顶层接口） 开来可深

作用：进行数据储存

collection是所有集合的所有集合接口

·创建对象

Collection集合名 = new 实现类();

Collection<泛型> 集合名 = new 实现类<泛型>();

·注意：泛型只能是类，如果要使用基本数据类型，请使用基本数据类型的包装类

·常用方法:

1.add(); 添加

2. size(); 集合的元素的个数

3. remove() 移除指定元素

4. iterator(); 迭代器

a. hasNext() 判断是否有下一个元素（true为有下一个元素。false为没有 下一个元素）

b. next() 取得迭代器中的下一个元素

2）List集合是Java.util包中的类

·创建对象的方法：

List 集合名 = new 实现类();

List<泛型> 集合名 = new 实现类<泛型>();

·List集合的特点：

List集合有下标，数据是有序的，可以存放重复数据

·List集合的常用方法：

add() 添加元素

remove() 移除元素 瑞姆

size() 获得集合的长度

get(index) 获得指定下标的元素

iterators() 获得迭代器 艾特瑞特

3）Set集合

①HashSet ②TreeSet

特点：无序 无下标 不能放重复数据

·ArrayList 和 LinkedList，Vector的区别：

·ArrayList基于数组的存储结构，查询效率快，删除和添加数据比较慢，线程不安全

· LinkedList基于链式存储结构，查询效率慢，删除和添加效率快， 线程不安全

·Vector线程安全的，已经不建议使用

**20、Map映射(不能被实例化，通过实现类创建Map对象）**

Map接口的实现类 HashMap，HashTable

1.对象的创建

Map<泛型，泛型> 变量名 = new 实现类<泛型，泛型>();

存值得方式：以键值对的形式存在，并且，键对象不允许重复，值对象 允许重复

map.put("键对象"，“值对象”);

常用方法

取值方式: get（） 通过键对象获取值对象 map.get(键对象);

得到所有的值对象：map.values();

得到所有的键对象：map.keySet();

2.HashMap和HashTable的区别

HashMap 线程不安全的，效率高，允许键对象和值对象为null

HashTable线程安全的，效率低，不允许键对象和值对象为空，否则会报异常

Map是无序的，它的存储结构是哈希表<key,value>键值对，map中插入元素是根据key计算出的哈希值来存储元素的，因此他不是按照元素的添加顺序来存储对象的，所以Map是无序的

1. hashMap的底层原理

HashMap存储方式在jdk8之前用的是数组加链表，在之后用的是数组+链表+红黑树。当我们往hashMap里放值时，首先计算key的hashcode，再%（取模）数组的长度，从而获得该值在数组上的位置下标，若该位置上没有放置元素，则直接存放在该位置上；若已有元素，则插入该位置下的链表头部。Jdk1.8之后在此基础上增加了红黑树的存储方式，当链表上的key达到8个的时候将链表变为红黑树的存储方式，小于6个时候会再转换成链表的存储方式。（补充：我后来在网上查阅资料的时候看到个问题，就是说这个头插法有可能会在扩容时候出现问题，扩容的时候会重新放置数据，并且它会将链表中的数据倒序插入到一个新的链表，这时候当有一个新的元素存储过来的时候，会造成新元素和从链表尾部过来的值相互指向，造成死锁，jdk8之后采用了尾插法就避免了这个问题）

**21、get和post区别**

1）get是从服务器上获取数据，post是向服务器传送数据。

2）get传送的数据量较小，不能大于2KB。

post传送的数据量较大，一般被默认为不受限制。

3）get安全性非常低，post安全性较高。但是执行效率却比post方法好。

4）在进行文件上传时只能使用post而不是get。

22、**反射**

1. 通过类对象获取类的所有属性和方法；
2. 通过对象可以获取到属性的值和调用它的方法；

反射中常用的几个对象：

Class 类（可以获取所有的属性（getDeclaredFields()、getFields()、获取所有方法 getDeclaredMethods()））

Field 属性（获取属性名getName()、获取属性的数据类型getType() ））

Method 方法（get、获取返回值类型 getReturnType() 、执 行方法invoke() ）

Constructors 构造方法 （获取构造方法 getConstructors()）

Annotation 注解 （获取注解 getAnnotations()）

1. 自定义注解
   1. 自定义注解的关键字@Interface
   2. 自定义注解的四个元注解
      1. Target 声明注解的使用范围
      2. Retention 声明注解的生命周期
2. Lambda表达式

Lambda表达式是jdk1.8的新特性，是一种简单快捷实现接口的新写法，但是不是所有的接口都适合lambda表达式，需要以函数接口为前提（函数接口也就是有且只有一个抽象方法，都可以有多个普通方法），使用如下：

//接口

public interface EnumDemo {  
 public void query(String a);  
}

//实现方法及调用

String abb = "888";  
EnumDemo enumDemo = (a)->{  
 System.*out*.println(a);  
};  
enumDemo.query(abb);

1. Stream流
   1. Stream就是用来操作集合的；
   2. 创建Stream流的方法
      1. 集合名.stream();
   3. Stream的常用方法
      1. forEach()用来遍历集合
      2. filter()条件过滤
      3. Limit() 取定量数据（取几条）
      4. Skip()跳过几条
      5. Sorted()对集合进行排序
2. 多线程
   1. 什么是进程？什么是线程？

进程：进程指正在运行的程序。确切的来说，当一个程序进入内存运行，即变成一个进程，进程是处于运行过程中的程序，并且具有一定独立功能，进程主要用于获取CPU资源。

线程：线程是程序中最小的执行单元，一个进程至少有一个线程；

多线程：一个进程中有多个线程同时执行任务；

* 1. 线程的状态
     1. 创建（new之后）
     2. 就绪（执行start之后）
     3. 运行（执行run之后）
     4. 阻塞（执行sleep或wait之后，sleep不释放锁，wait释放锁）
     5. 死亡（run方法执行结束后）
  2. 创建线程的方法
     1. 实现Runable类
     2. 继承Thread类
     3. 实现callable类以及Future；(此线程可以有返回值)
  3. 线程安全问题
     1. 当多个线程同时操作同一个资源时会造成线程不安全问题；此时我们通过synchronized给代码块加锁
     2. 还有就是死锁，A线程拿着B线程需要资源不释放，B线程拿着A线程需要的资源也不释放，造成死锁，导致程序无法往下进行；尽量避免死锁的情况，或者线程礼让（线程礼让也不完全能解决死锁的问题）

1. 线程池
   1. 什么是线程池？
      1. 线程池就是事先创建若干个可执行的线程放入一个容器中，当有任务需要执行时，直接从线程池拿空闲线程使用，执行完任务后将线程归还给线程池，这样就实现了线程的复用，从而减少了创建和销毁线程对象的开销。
   2. 线程池的优点？
      1. 降低系统消耗；
      2. 提升响应速度；
      3. 提高线程的管理性；
   3. 线程池运行原理
      1. 提交多线程任务，判断线程池核心线程数是否已经满，若没有满足则创建线程执任务，
      2. 若核心线程都在占用，判断等待队列是否已满，未满则进入队列；
      3. 队列已满则判断线程数是否达到最大线程，若为达到则直接创建线程执行任务；
      4. 未达到最大线程数则进入拒绝策略；
   4. 线程池的核心类ThreadPoolExecutor的主要参数
      1. c**orePoolSize：线程池中核心线程数**
      2. **maximumPoolSize：线程池最大线程数**
      3. **keepAliveTime：非核心线程闲置超时时间**
      4. Unit**：keepAliveTime的时间单位**
      5. **workQueue：线程池中的任务队列**
      6. **threadFactory：线程池创建线程时使用的线程工厂**
      7. **handler：线程池的饱和策略**
   5. 常用的线程池：

FixedThreadPool

创建一个固定工作线程数量的线程池。核心线程数和最大线程数是一样的，任务如果过多会在队列中阻塞。如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

|  |
| --- |
| 线程池中的线程处于一定的量，可以很好的控制线程的并发量  线程可以重复被使用，在显式关闭之前，都将一直存在  超出一定量的任务被提交时候需在队列中等待 |

CachedThreadPool

创建一个可缓存的线程池。核心线程数是0，最大线程数是 Integer.MAX\_VALUE。当线程池中的线程都处于活动状态的时候，线程池会创建新的线程来处理新任务，否则会用空闲的线程来处理新任务。这类线程池的空闲线程都是有超时机制的，keepAliveTime在这里是有效的，时长为60秒，超过60秒的空闲线程就会被回收，当线程池都处于闲置状态时，线程池中的线程都会因为超时而被回收，所以几乎不会占用什么系统资源。

|  |
| --- |
| 线程池中线程的数量没有固定大小，可达到最大值（Interger. MAX\_VALUE）  线程池中的线程可进行缓存重复利用和回收（回收默认时间为1分钟）  当线程池中没有可用线程，会重新创建一个线程 |

SingleThreadExecutor

创建一个单线程的线程池。这个线程池的核心线程数和最大线程数都是1，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

|  |
| --- |
| 线程池中最多有1个线程，之后提交的任务将会排在队列中依次执行 |

ScheduledThreadPool

创建一个支持定时及周期性的任务执行的线程池。

|  |
| --- |
| 可定时或者延迟执行线程活动 |

SingleThreadScheduledExecutor

创建一个单线程的支持定时及周期性的任务执行的线程池。

|  |
| --- |
| 可定时或者延迟执行线程活动  线程池中最多有1个线程 |

1. js基础

**1、js概念**

Js全称javascript，是一种基于对象和事件驱动的具有安全性能的脚本语言

**2、js数据类型**

基本数据类型（3种）

字符串(string) "hello"、"学习" 'javascript'

数值(number) 18, 365.25, -12.7

布尔(boolean) true, false

复合数据类型（2种）

数组Array

对象 object

特殊数据类型（2种）

空 null

未定义 undefined

**3、js检测数据类型**

typeof() 检测类型。

**4、请写出js的5个内置函数并说明其作用**

isNaN() 判断一个值是否为数值,如果是数值返回false,否则返回true

Number() 把指定值转换为数字,如果无法转换返回 NaN

parseInt() 将指定值转为整数,如果无法转换返回 NaN

parseFloat() 将指定值转为小数,如果无法转换返回 NaN

eval() 将传入的字符串解析为JavaScript 代码并执行

**5、js中的内置对象**

--math对象(掌握5个)：

ceil（）向上取整 floor（）向下取整 abs（）绝对值

round（）四舍五入 random（）随机数 min() 最小值 max() 最大值

--string对象(掌握5个)：

indexOf（）查找子字符串在字符串中首次出现的位置

lastIndexOf() 查找子字符串在字符串中最后一次出现的位置

split（）将字符串分割为数组

replace（）替换字符串

substr（）按下标和个数截取字符串

substring（）按下标截取字符串

toLowerCase() 将字符串转为小写

toUpperCase() 将字符串转为大写

Replace()替换

--date对象(掌握5个)：

getFullyear（）返回指定日期的年份

getMonth（）返回指定日期的月份

getDay（）返回指定日期星期中的第几天（0-6）

getDate（）返回指定日期月份中的第几天（1-31）

getHours() 返回指定日期对象的小时数

getMinutes() 返回指定日期对象的分钟数

getSeconds() 返回指定日期对象的秒数

getMilliseconds() 返回指定日期对象的毫秒数

toLocalestring（）根据当前区域设置并转化为字符串返回

toString（）将日期转化为字符串返回

--array对象(掌握5个)：

concat（）将两个数组或多个数组拼接成一个新的数组

shift() 移除数组中第一个元素并返回该元素

unshifit() 向数组头部添加一个或多个元素，并返回数组的新长度

pop（）移除数组中最后一个元素并返回该元素

push（）向数组中添加一个元素并返回数组的长度

splice() 在任意位置添加或删除任意个元素

toString（）将数组以字符串的形式返回

join（）返回一个字符串，字符串是以指定的符号分隔

**6、DOM和BOM 对象**

--document:

getElementById() 返回对拥有指定 id 的第一个对象的引用。

getElementsByName() 返回带有指定名称的对象集合。

getElementsByTagName() 返回带有指定标签名的对象集合。

--window:

alert() 弹出警告框

confirm() 弹出确认框

prompt()弹出输入框

open() 打开新窗口

close() 关闭窗口

--history

forward() 加载历史列表中的下一个url，相当于浏览器前进按钮功能

back() 加载历史列表中的前一个url，相当于浏览器后退按钮的功能

go() 加载历史列表中某个具体页面

--location

href 设置或返回浏览器地址栏

reload() 重新加载当前页面，相当于浏览器刷新按钮的功能

**7、dom常用操作(掌握5个)**

getElementById() 通过id获取html元素

getElementsByName() 通过name获取html元素数组

getElementsByTagName() 通过标签名获取html元素数组

body 获取页面中body元素

innerHTML 设置或获取html元素html内容

childNodes 返回html元素的子节点数组

 firstChild 返回html元素的首个子节点

 lastChild 返回html元素的最后一个子节点。

 nextSibling 返回节点的下一个兄弟节点

 parentNode 返回节点的父节点

 previousSibling 返回节点的上一个兄弟节点

 appendChild() 向元素添加新的子节点，并作为最后一个子节点。

 cloneNode(是否深度克隆) 克隆元素

 insertBefore(新节点,已有子节点) 在指定的已有的子节点之前插入新节点。

 removeChild(已有子节点) 从元素中移除子节点

 replaceChild(新节点,旧节点) 替换元素中的子节点

**8、js全局变量和局部变量**

全局变量：声明在函数外(var可用可不用)一般为了规范加上var

局部变量：声明在函数内部(加上var是局部变量，不加var是全局变量)

**9、js事件**

**事件就是用户对网页或浏览器做出的动作.通常来说,鼠标和键盘的动作称之为事件。**

页面事件：

载入页面 onload

离开页面 onunload

鼠标事件:

单击事件 onclick

双击事件 ondblclick

鼠标按下事件 onmousedown

鼠标抬起事件 onmouseup

鼠标移上事件 onmouseover【实战】

鼠标移开事件 onmouseout 【实战】

键盘事件：

键盘按下事件 onkeydown

键盘抬起事件 onkeyup

键盘按下并抬起事件 onkeypress

表单事件：

表单提交事件 onsubmit

表单重置事件 onreset

获得焦点事件 onfocus

失去焦点事件 onblur

内容改变事件 onchange

**10、JS正则表达式**

元字符:元字符是具有特殊含义的字符

\d 匹配一个数字字符，等价于[0-9]

\w 匹配一个大小写字母、数字、下划线，等价于[0-9a-zA-Z\_]。

\s 匹配一个空白字符(空格、制表符(tab)、换行符(\n))

.匹配除\n之外的任意一个字符

\D 匹配一个非数字字符,等价于[^0-9]

\S 匹配一个非空白字符(非空格、非制表符(tab)、非换行符(\n))

\W 匹配一个非大小写字母、非数字、非下划线，等价于[^0-9a-zA-Z\_]

边界符：

^ 匹配字符串的起始位置

$ 匹配字符串的结束位置

量词

? 匹配前面的子表达式0次或1次

+ 匹配前面的子表达式1次或多次

\* 匹配前面的子表达式0次或多次

{n} 匹配确定的n(非负整数)次

{n,} 至少匹配n(非负整数)次，最多匹配无数次

{n,m} 至少匹配n次，最多匹配m次(n必须小于等于m)

修饰符

g 全局匹配

m 多行匹配

i 忽略大小写匹配

**11、oracle数据类型**

字符类型：

varchar2:最多4000字节,可变长度,最为常用

nvarchar2:最多2000个字符，可变长度，内部实现存在问题不常用

char:最多2000字节，固定长度，长度不够以空格填充 大文本数据类型

数值类型：

number(总长度,小数点后的位数) 最大精度38位

(小数点后的位数:小数位置四舍五入)

日期类型：

date (年月日 时分秒)

通常情况下如果是GBK编码则一个汉字对应2个字节，如果是UTF-8编码则一个汉字对应3个字节。

**12、Oracle的5个聚合函数 ，并说明用途**

-oracle的聚合函数

--count（）统计数量

--max（）最大值

--min（）最小值

--avg（）平均值

--sum（）总和

**13、主键外键**

主键--唯一标识一条记录，不能有重复的，不允许为空

primary key

constraint pk\_id primary key(字段名)

外键--表的外键是另一表的主键, 外键可以有重复的, 可以是空值

constraint 外键名 foreign key(字段名) references 主表(主键字段名)

**14、oracle常用函数函数**

 --数学函数

abs(字段名|值) 绝对值

ceil(字段名|值) 向上取整

floor(字段名|值 ) 向下取整

trunc(字段名|值) 截断

round(字段名|值) 四舍五入

 --字符串函数

concat(字段名|值, 字段名|值) 字符串拼接

initcap(字段名|值) 将首字母转为大写，剩余的全转为小写

instr(字段名|值,子字符串[,起始位置[,第几次出现]]) 按照起始位置和指定次数查找子字符串在字符串中出现的位置

length(字段名|值) 字符串长度

lower(字段名|值) 转为小写

upper(字段名|值) 转为大写

rpad|lpad(字段名|值,字符串总长度,填充的字符串) 左填充和右填充

rtrim|ltrim (字段名|值,要删除的字符串) 左去除|右去除

substr(字段名|值,起始位置,个数) 按照起始位置和个数截取字符串

replace(字段名|值,s1,s2) 替换字符串

reverse(字段名|值) 字符串反转，这里反转的是字节不是字符

--转换函数

to\_char(值|字段名[,格式] ) 将日期或数字转为字符串

to\_date(值|字段[,格式] ) 将字符串转为日期

 --日期和时间函数:

add\_months 增加或减去月份

last\_day 返回日期所在月的最后一天

next\_day(date,day) 给出日期date和星期day之后计算下一个星期的日期

 --通用函数:

nvl(字段名|值,默认值) 如果指定值为空则显示默认值，否则显示指定值

nvl2(字段名|值,默认值1,默认值2) 如果指定值为空则显示默认值2，否则显示默认值1

 decode(字段名|值,条件1,值1,条件2,值2,…,默认值)

如果指定值等于条件1则显示值1，

如果指定值等于条件2则显示值2，

…

如果指定值不等于任何条件则显示默认值

**15、Oracle分页：**

--假设现在数据库里有839条记录，页面上每页显示37条，现在查出第13页要显示的数据

select \* from (

select rownum rn,t.\* from (

select \* from t\_student

) t where rownum <= (13\*37)--页数\*每页显示条数

) where rn > (13-1)\*37--(页数-1)\*每页显示条数;

**16、DML数据操作语言**

--增加

insert into 表名 (字段1,字段2,字段3,...) values (值1,值2,值3,...);

insert into 表名 values (值1,值2,值3....);

--删除

delete from 表名 [where子句];

--修改

update 表名 set 字段1=值1,字段2=值2,… [where子句];

**17、序列**

--创建序列语法

create sequence 序列名

minvalue 最小值

maxvalue 最大值

start with 起始值

increment by 增量(每次递增的值)

cache 缓存的序列值的个数

order;

--序列名.nextval 获取下一个值

--序列名.currval 获取当前值

**18、两表联查**

内连接

内连接 inner join 只显示两张表有关联的数据

select 字段名,字段名 from 表1 inner join 表2 on 表1和表2的关联条件

select \* from t\_student s inner join t\_class c on s.classid=c.id;

左连接

左连接 left join 会将左边表的全部数据进行展示，另外一张表有和它关联的就展示，没有就展示为null

select 字段名,字段名 from 表1 left join 表2 on 表1和表2的关联条件

select \* from t\_student left join t\_class on t\_class.id=t\_student.classid;

右连接

右连接 right join 会将右边表的全部数据进行展示，另外一张表有和它关联的就展示，没有就展示为null

select 字段名,字段名 from 表1 right join 表2 on 表1和表2的关联条件

**19、JDBC操作数据库的实现步骤及其主要代码**

--导jar包

--加载驱动为数据库连接做准备 关键字：Class.forName

--连接数据库 关键字：DriverManager.getConnection()

--写sql语句

--创建数据库操作对象 关键字：preparestatement

--执行sql语句 关键字：executeupdate()/executequery()

--关闭连接释放资源 关键字：.close()

**20、端口**

tomcat默认端口8080

oracle默认端口 1521

**21、Servlet的生命周期**

1.实例化：由web服务器进行servlet的实例化

2.初始化：使用init()方法进行初始化

3.提供服务：使用service()方法提供服务。（具体使用doGet()和doPost()方法提供服务）

4.销毁：使用destroy()方法进行销毁

5.不可用：标记为垃圾回收

**22、jsp运行原理**

--jsp的运行原理:

1、用户发送一个请求---->服务器(tomcat)

2、服务器根据请求的url(地址)----->找到相应的jsp文件

3、将找到的jsp文件转译成java源文件(servlet)

4、将转译出来的java源文件进行编译，得到一个可执行文件(.class)

5、执行.class文件并且告诉服务器(tomcat)

6、服务器返回一个响应信息------>用户

**23、Jsp 9大隐式对象。**

**输入输出对象**：

request 对应的实体类是(HttpServletRequest)

获取请求的信息(这是用户提交上来的信息)

同时还是作用域通讯对象

在当前请求中，一直有效

response 对应的实体类是(HttpServletResponse)

响应用户的信息(这是要返回给用户的信息)

out 对应的实体类是(JspWriter)

输出对象，一般输出一些调试信息

out.println(); 向页面输出一些信息

System.out.println(); 向控制台输出一些信息

**作用域通讯对象**：

application 对应的实体类是(ServletContext)

只要服务器不关闭，一直有效

session 对应的实体类是(HttpSession)

只要当前会话不关，一直有效

pageContext 对应的实体类是(PageContext)

只在本页面上有用(离开里本页面就没用了)

request

在当前请求中，一直有效

**servlet对象：**

page 对应的实体类是(Page)

相当于new了一个实例，类似于java中的this关键字

config 对应的实体类是(ServletConfig)

可以获取servlet初始化的一些信息，不太常用

**错误对象：**

exception 对应的实体类是(Throwable)

专门处理错误

**24、Jsp的四个作用域、作用域范围、以及对应的作用域对象**

page 当前页面 pageContext

request 当前请求 可以跨越多个页面 request

session 当前会话 多次请求 session

application 整个项目 整个服务应用 application

**25、Redirect和forward的区别**

转发和重定向的区别；

redirect重定向 特点：

地址栏会发生变化

数据不共享(用户请求信息不会带到新页面)

执行效率低，好几次请求

写法：

response.sendRedirect(目标文件地址);

forward转发 特点：

地址栏不会发生变化

数据共享(用户请求信息会被带到新页面)

执行效率高，只有一次请求

写法：

request.getRequestDispatcher(目标文件地址).forward(request, response);

26、**get和post的区别：**

1.get是一般用于从服务器上获取数据， post一般用于向服务器传送数据。

2.get提交的数据会存放在URL之后,以？分割url和传输数据，参数之间以&相连。get对数据不安全，post相对安全

3.get提交的数据大小有限制，最多只能有1024字节(因为浏览器对url长度有限制)而post提交没有限制。

4.通常用post做文件上传而get是不支持的。