# B简历：

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

### 基本信息

|  |  |  |  |
| --- | --- | --- | --- |
| 基本情况 | | | |
| **姓名** | **王帅鹏** | 性别 | 男 |
| **出生日期** | 1996-04-05 | 属相 | 牛 |
| **现住地** | 洼刘新城 |  |  |
| 大学情况 | | | |
| **学校** | 郑州科技学院 | 专业 | 计算机应用技术 |
| **入学时间** | 2013-9 | 毕业时间 | 2017-7 |
| **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 | **学士学位** |
| **学校地址** | 河南省郑州市荥阳市学院路1号 | 校长 | **徐建新** |
| **去学校的方式：** | **郑州汽车站→地铁1号线→秦岭路坐302路→到郑州科技学院** | | |
| **上小学日期：** | **2001年9月 5岁六年** | | |
| **上初中日期：** | **2007年9月 11岁三年** | | |
| **上高中日期：** | **2010年9月 14岁三年** | | |
| **上大学日期：** | **2013年9月 17岁四年** | | |
| **毕业时间：** | **2017年 7月 21岁** | | |
| **在校学生：** | **5000左右** | | |
| **有教师：** | **不太清楚** | | |

|  |  |
| --- | --- |
| 第一家公司 ： | 上海泽辛信息技术有限公司 |
| **地址：** | 上海宝山区刘行(地铁站)1018弄1号楼207室 |
| **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：** | **冯洁宜** |
| **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：** | 坐地铁7号线 到刘行站下 （富晨花苑） |

/

|  |  |
| --- | --- |
| **第二家公司情况：** | 上海华沁智能交通科技有限公司(只说公司简称) |
| **公司地址**： | 上海 杨浦区 紫荆广场(江浦路) 西座10楼 |
| **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：** | 地铁八号线 到江浦路站下 （四光村住） |
| **入职时间：** | **2019年2月** |
| **离职时间：** | **2021年4月初** |
|  | **Java工程师** |
| **项目经理名字：** | **刘鹏涛** |
| **税前：** | **15K** |
| **税后：** | **14K** |
| **缴税：** | **1K左右(具体原因不清楚当时公司就是这样扣的税)** |
| **期望薪资：** | **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。**  **2018年2月跳槽涨了3K左右 总10K**  **2018年12月涨1K总 11K左右**  **2019年9月工作一年涨了2K左右**  **2020年7月工作一年涨了2K左右**  **2021年4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？** |
| **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，那我就先说一下我对这个微服务框架的理解吧：**

服务启动向Eureka注册中心注册服务，注册中心将注册信息同步给其他注册中心，当服务消费者需要调用服务提供者，则向服务注册中心获取服务提供者地址，然后会将获取的地址缓存在本地，下次调用的则直接从缓存中取，当服务注册中心检测到服务提供者宕机、网络不通等服务不可用时，则在注册中心将服务置为down状态，并向订阅者发布当前服务提供者状态，订阅过的服务消费者更新本地缓存。

服务提供者启动后周期性（30秒）向服务注册中心发送心跳，以证明当前服务可用，服务中心在一定时间（默认90秒）未收到心跳，则认为该服务宕机，注销该实例

#### Eureka自我保护机制:

默认情况下，如果Eureka在一定时间内没有接受到某个微服务实例的心跳，Eureka注册中心将会将该实例注销。但是当网络分区故障发生时，微服务与Eureka Server之间无法正常通信，以上行为可能变得非常危险了——因为微服务本身其实是健康的，此时本不应该注销这个微服务。Eureka通过“自我保护模式”来解决这个问题——当Eureka Server节点在**短时间内丢失过多客户端时**（可能发生了网络分区故障），那么这个节点就会进入自我保护模式。一旦进入该模式，Eureka Server就会保护服务注册表中的信息，不再删除服务注册表中的数据（也就是不会注销任何微服务）。当网络故障恢复后，该Eureka Server节点会自动退出自我保护模式。

在自我保护模式中，Eureka Server会保护服务注册表中的信息，不再注销任何服务实例。当它收到的心跳数重新恢复到阈值以上时，该Eureka Server节点就会自动退出自我保护模式。它的设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。一句话讲解：**好死不如赖活着**。

**在这个微服务的搭建过程中也有可能发生微服务雪崩的问题**

在讲Hystrix之前先说一下服务雪崩，在springcloud**可以用Feign+Ribbon来进行服务之间的调用**。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用。如客户端访问A服务，而A服务需要调用B服务，B服务需要调用C服务，由于网络原因或者自身的原因，如果B服务或者C服务不能及时响应，A服务将处于阻塞状态，直到B服务C服务响应。此时若有大量的请求涌入，容器的线程资源会被消耗完毕，导致服务瘫痪。**服务与服务之间的依赖性，故障会传播，造成连锁反应，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应**

### 服务雪崩的解决方案

**l 熔断模式**：这种模式主要是参考电路熔断，如果一条线路电压过高，保险丝会熔断，防止火灾。放到我们的系统中，如果某个目标服务调用慢或者有大量超时，此时，熔断该服务的调用，对于后续调用请求，不在继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。

### 服务熔断

当下游的服务因为某种原因突然变得不可用或响应过慢，上游服务为了保证自己整体服务的可用性，不再继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。

### 服务降级

l 当下游的服务因为某种原因响应过慢，下游服务主动停掉一些不太重要的业务，释放出服务器资源，增加响应速度！(**弃车保帅，壮士断腕，刮骨疗伤**)

当下游的服务因为某种原因不可用，上游主动调用本地的一些降级逻辑，避免卡顿，迅速返回给用户

### 服务熔断和服务降级的区别和联系

服务降级有很多种降级方式！如开关降级、限流降级、熔断降级!**服务熔断属于降级方式的一种**！**因为从实现上来说，熔断和降级必定是一起出现**。因为当发生下游服务不可用的情况，这个时候为了对最终用户负责，就需要进入上游的降级逻辑了。因此，**将熔断降级视为降级方式的一种**，也是可以说的通的！

### **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**。**

**接下来我来说下这个Feign的使用吧**

我们当时在项目中有很多的模块需要互相调用，比如说这个购物车需要调用商品进行 查询，我们当时为了解决这个问题使用的是Feign来进行微服务之间的调用

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。

**--》Zuul**

Zuul是一个微服务网关，**微服务网关**，**介于客户端与服务器之间的中间层**，所有的外部请求都会先经过微服务网关。

### 为什么需要外服务网关呢

不同的微服务一般会有不同的网络地址，客户端在访问这些微服务时必须记住几十甚至几百个地址，这对于客户端方来说太复杂也难以维护。

而且可能会出现很多问题。

客户端会请求多个不同的服务，需要维护不同的请求地址，增加开发难度

后端每个微服务可能是由不同语言编写的、采用了不同的协议，比如HTTP、Dubbo、GRPC等，但是你不可能要求客户端去适配这么多种协议，这是一项非常有挑战的工作，项目会变的非常复杂且很难维护。

路由转发和**过滤器**

#### **路由转发**：接收一切外界(客户端)请求，转发到后端的微服务上去；

#### **过滤器**：在服务网关中可以完成一系列的横切功能，例如权限校验、限流以及监控等，这些都可以通过过滤器完成（其实路由转发也是通过过滤器实现的）。

过滤器的四中类型

**l PRE**：这种过滤器在请求被路由之前调用。我们可利用这种过滤器实现身份验证、在集群中选择请求的微服务、记录调试信息等。

**l ROUTING**：这种过滤器将请求路由到微服务。这种过滤器用于构建发送给微服务的请求，并使用Apache HttpClient或Netfilx Ribbon请求微服务。

**l POST**：这种过滤器在路由到微服务以后执行。这种过滤器可用来为响应添加标准的HTTPHeader、收集统计信息和指标、将响应从微服务发送给客户端等。

**ERROR**：在其他阶段发生错误时执行该过滤器。

我们可以通过将Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即**以后通过客户端(浏览器，APP)访问微服务都是通过Zuul跳转后获得**。

**至于这个springboot的运行原理的话 我也做过一些了解**

在我看来他的运行原理就是最核心的两行代码

@springbootapplication 和 springappliction.run方法

在 @SpringBootApplication 的内部包含了3个注解

@Configuration

@EnableAutoConfiguration

@ComponentScan

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

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

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

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

**功能模块这部分的话当时**

# Redis总结

**redis是一个key value型的非关系型数据库 它是基于内存的**

**它具有丰富的数据类型有**

**String ,**

**我在项目中用String 类型缓存了地区数据,因为地区数据量不算太大，但是数据库读的速度没有redis快导致用户体验度不高，**

**hash, key-filed-value类型的 可以怎么存怎么取**

**我在做电商项目的时候 购物车模块用了hash类型，因为这个模块需要频繁的存储和删除 redis 读写速度快 就用redis来存放购物车的数据了**

**filed放的是sku和非sku的id,value放的就是数据**

**在新增购物车的时候 需要先判断redis 里面有没有这个要添加的数据 如果有的话就把数据拿出来 把要添加的数据和已有的数据进行相加 如果没有的话就把这个数据给新增到redis中。**

**新增订单的时候我还用了redis的getset方法解决了接口的幂等性问题**

**在用户提交订单的时候把用户的名字作为key 去Redis中查询 判断key是否存在 存在的话就不走下去了并且从新设置失效时间 如果不存在就设置一个失效时间放入redis中。为了防止用户频繁发请求， 在新增的时候需要吧redis 中的数据取出来 把要结算的数量 取出来 用数据库的锁解决了超卖的问题每次修改返回一个值 判断这个值是不是1如果是1就可以去支付页面了 如果是0 就说明库存不住了。**

**set,**

**zset；**

**list 就是做消息队列 就是来限流,因为list是先进先出的原则，所以可以做消息队列**

**读写速度快 读每秒11万次写8万次每秒**

**reids 支持事务 所有的事务都是原子性的**

**redis有两个持久化方案**

**一种是RDB快照 快照就是在指定的时间间隔中对数据进行拍照**

**一种就是AOF基于日志的,AOF是在你保存一条数据存到日志中,**

**RDB性能高，但是在间隔时间段中，redis宕机了。那么丢失数据就是这个时间间隔中的数据，**

**其中RDB数据读取的速度比AOF快 因为AOF是以日志存储的每次启动都要执行里边的SQl语句 进行数据恢复 但是RDB丢失的数据是某个时间段的，AOF丢失的数据比较少最多丢失最近几秒的。**

**redis有三种模式 主从复制 哨兵 集群**

**主从复制就是一台主服务 多台从服务 主服务提供读和写的功能但是从只能读取数据。读写分离 故障恢复 负载均衡**

**哨兵：故障转移 监控 通知**

**就是用来监视主服务的 如果主服务宕机了会从另外的从服务里边随机指定一台当主服务，配置文件里边要尽可能的配成奇数 有一点哨兵会改我们的配置文件。**

**集群：负载均衡 数据分区 特点是所有的数据关联**

**就是部多台服务 集群有一个三主三从的模式 集群的数据是共享的**

**数据分区：数据分区(或称数据分片)是Redis集群最核心的功能。**

**Redis集群将数据分散到多个节点，一方面突破了Redis单机内存大小的限制，存储容量大大增加；另一方面每个主节点都可以对外提供读服务和写服务，大大提高了集群的响应能力。**

**高可用：集群支持主从复制和主节点的自动故障转移（与哨兵类似）；当任一节点发生故障时，集群仍然可以对外提供服务。**

**Redis集群的数据分片**

**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 。集群中的每个节点负责处理一部分哈希槽。**

**还有redis 的三个安全问题 雪崩 击穿 穿透**

**缓存穿透**

**是指查询一个一定不存在的数据，由于缓存是不命中时需要从数据库查询，**

**查不到数据则不写入缓存，这将导致业务系统每次进行查询请求的时候都会绕过缓存服务器去数据库中查询，缓存服**

**务器也就完全失去了其应有的作用。这样数据库中的压力将会变的很大我们称他**

**为缓存穿透。**

**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**

**失效时间的话，如果有值了将会导致问题。（但它的过期时间会很短，**

**最长不超过五分钟。）**

**很多时候，缓存穿透是因为有很多恶意流量的请求，这些请求可**

**能 随机生成很多Key来请求查询，这些肯定在缓存和数据库中都没**

**有，那就很容易导致缓存穿透。**

**在缓存穿透防治上常用的技术是布隆过滤器(Bloom Filter)。**

**布隆过滤器是一种比较巧妙的概率性数据结构，它可以告诉你数**

**据 一定不存在或可能存在，相比Map、Set、List等传统数据结构它**

**占 用内存少、结构更高效。**

**对于缓存穿透，我们可以将查询的数据条件都哈希到一个足够大**

**的 布隆过滤器中，用户发送的请求会先被布隆过滤器拦截，一定不**

**存 在的数据就直接拦截返回了，从而避免下一步对数据库的压力**

**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**

**异步定时更新如果提前知道某一个色号比较畅销的话，那就可以定时的咨询总**

**部是否还有存货，定时的更新库存情况就可以避免上面这种情况了。**

**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。互斥锁还有一种解决办法，那就是如果很多顾客咨询的是同一个色号的口红，那么就先处理第一个用户的咨询，其他同样请求的顾客先排队等**

**待。一直到店员从总部那里获取到最新的库存信息后，就可以安排其他**

**人继续购买了。**

**在缓存处理上，通常使用一个互斥锁来解决缓存击穿的问题。简**

**单来说就是当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。**

**redis的雪崩**

**如果缓存集中在一段时间内失效，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。**

**缓存雪崩是指当大量缓存同时过期或缓存服务宕机，所有请**

**求的都直接访问数据库，造成数据库高负载，影响性能，甚至数**

**据库宕机。**

**这个没有完美解决办法，但可以分析用户行为，尽量让失效时间点均匀分布。大多数系统计者考虑用加锁或者队列的方式保证缓存的单线程（进程）写，从而避免失效时大量的并发请求落到底层存储系统上。**

**在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对**

**某个key只允许一个线程查询数据和写缓存，其他线程等待。**

**如何避免缓存雪崩？**

**避免的方法：设置不同的过期时间或者使用缓存集群。**

**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。**

**可以通过缓存reload机制，预先去更新缓存，再即将发生大并发访问前手动触**

**发加载缓存做二级缓存，或者双缓存策略。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。**

**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.**

**Redis 数据库常用命令**

**exists 命令可以判断键值是否存在。**

**del 命令可以删除当前数据库的指定 key。**

**type 命令可以获取 key 对应的 value 值类型。**

**1. keys \* 查看所有的额key**

**2. select 0-15 切换不同redis库**

**3. move age 0-15 移动数据到其他的库**

**4. flushdb 清空当前库的数据**

**5.set key value 设置值**

**6.get key 获取key的value值**

**7.mset key value key value ....... 一次设置多个key value值**

**8.mget key key ..... 以此获取多个key的value值**

**9.expire key 秒数 ....设置key的失效时间**

**10.pexpire sex 毫秒值.... 设置key的失效时间**

**11. persist key 删除过期时间**

**12.string :**

**(1) getrange name 0 -1 [0,1] 字符串分段**

**(2) getset name new\_cxx 设置值，返回旧值**

**(3)setnx key value 不存在就插入（not exists）**

**(4)incr age 递增：**

# 定时任务

在之前的一个项目中我用定时任务做过一个在用户生日当天给用户发送邮件的功能其中用了线程池

以及第三方的接口发送邮件的 工具类 在 spring中配置了定时器的全路径 以及里边的方法 还有设置定时器的开启时间 这里我设置成了每天执行一次，然后写了一个定时器的类 再写一个执行定时器的方法 获取当天的时间根据当前时间去查询这一天的用户 新建一个线程池 因为线程池利用线程进行发送提高性能

因为线程是系统非常珍贵的资源;线程池就是一个容器;用来存放线程;并帮我们线程执行任务;以减少我们线程创建与销毁的资源消耗;

**同时每一个线程执行完的时候还要把这个线程销毁，才能在创建下一个请求的线程，那么他的创建和销毁所需要的系统资源和花费的时间是相当大的系统过度消耗导致资源不足，无法创建下面请求的线程，为了解决这一问题我们当时就提出来用线程池来处理，线程池就类似于一个容器里面可以容纳多个线程，其中的线程可以反复使用，省去了频繁创建线程对象的操作,线程池里面主要有核心线程池，任务队列，最大线程数，当有任务到达线程池的时候，会先去看核心线程数是否已满，如果未满的话就创建核心线程来执行任务，反之则去队列如果队列的线程数为达到最高值，则放入队列**

集群的情况下就不能避免已经发送过的可能会再次发送一次 这里呢可能会出现发送重复我就使用了redis 缓存了一下已经发送的用户的信息 并设置一个失效时间 如果存在 就不会在发送了避免了幂等性的问题 然后在循环调用发送邮件的线程对象。

线程池的优点：

**线程池**:

**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**

# MYSQL优化

设计：存储引擎()，字段类型

功能：索引，缓存，分表。

架构：主从复制

**从设计方面来说 就是存储引擎和字段类型的设计**

Mysql 存储引擎有两个 一个是MylSAM,一个是InnoDB

MylSAM不支持外键，不支持事务的但是他的性能高，里面有表锁(表锁就是操作一条数据的时候也会把整个表给锁住),并不适合高并发，而且缓存只能缓存索引

InnoDB是支持事务的也支持外键,里面有行锁(行锁就是操作一条数据时,只会把这一条数据锁住,不对其他数据有影响),适合高并发,不仅缓存索引还要缓存真是数据。这就导致性能不高但是安全。

字段类型的话可以设置一个字典表,两张表的话有一个相对应的关系。一个里面用int类型，另一个用varchar类型，这样就会减少主表的数据压力

**从功能上面来说就是索引，缓存，分库分表**

索引(Index)是帮助MySQL高效获取数据的数据结构；

索引的本质就是表中的字段对应二叉树，这样就可以减少一半的查询时间

索引分为单值索引(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)

复合索引(即一个索引包含多个列)

创建索引的sql语句为 create [unique ]  index indexname on 表名(表字段);

主键自动建立唯一索引,热销数据可以建立索引,以及查询中与其他表字段相关联可以建立索引

表记录太少或者创建增删改操作的表或者字段，不应该建立索引

可以通过Explain来看性能分析

查看type的值(type就是访问类型),结果值从最好到最坏依次是：

system > const > eq\_ref > ref > fulltext > ref\_or\_null > index\_merge > unique\_subquery > index\_subquery > range > index >ALL ，一般来说，得保证查询至少达到 range 级别，最好能达到 ref。

缓存就是查询第一次的时候把数据放进缓存中

缓存数据不能放动态数据比如当前时间

在linux中可以进入mysql中 通过 命令（ show variables like ‘%query\_cache%’）来查看query\_cache\_type的状态

如果是on就已经开启了。如果没有开启则在my.conf配置文件中 设置为on,也可以陪着缓存的大小

当我们一张表的数据达到上百万甚者上千万，那么即便是建立索引查询效率依然会很慢。

我们就可以进行分表，分表可以分为水平分表和垂直分表

水平分表就是创建多个结构相同的表，然后表名是动态的既可以,这样子就可以实现分表,把数据分开。查询效率就可以提高

垂直分表就是一个表中有常用数据和非常用数据，我们可以把这两个数据分开储存建立两个表，两个表是一对一的关系，

这样也可以提高查询效率

**从架构来说就是 主从复制**

Mysql服务器内部支持复制功能，仅仅需要通过配置完成下面的拓扑结构。一主多从典型结果：主服务器负责写数据。从服务器负责读数据。复制功能mysql会自带

这样就可以实现读写分离,因为从表可以复制主表，主表只负责增删改，从表只负责读.这样子就可以减少数据库的压力

# Docker

**1.什么是docker?**

docker就是用来解决运行环境和配置问题的容器引擎，其中包括了镜像，容器，仓库，

其中镜像是 Docker 运行容器的前提，

仓库是存放镜像的场所，仓库分为本地仓库，官方仓库，和私有仓库。

容器是一种轻量级、可移植的软件打包技术。

优点：

  能高效地构建应用。

    对于运维开发来说，

    能快速的交付和部署

    高效的资源利用

    轻松的迁移扩展

    简单的更新管理

**2.你会安装吗?**

Docker 运行在 CentOS 7 上，要求系统为64位、系统内核版本为 3.10 以上。

Docker 运行在 CentOS-6.5 或更高的版本的 CentOS 上，要求系统为64位、系统内核版本为 2.6.32-431 或者更高版本。

安装docker要先下载gcc，再改变yum源后可以安装docker。

**3.怎么使用**

docker images 列出 本机的所有镜像

docker search 镜像名 从Docker Hub查找相关镜像

docker pull 镜像名：tag 下载镜像

docker rmi 镜像名（id）删除镜像但是正在运行的镜像不可以删

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 创建容器并且后台运行

docker ps 查看运行中的容器

docker ps -a 查看所有的容器

docker rm 容器id（名称） 删除容器（要删运行中的容器加上 -f）

docker stop 容器id（名称） 停止容器

docker start 容器id（名称）启动容器

自定义镜像

有两种方式

commit 常用的是build 命令时：docker build -f Dockerfile的路径 -t 镜像名称:版本号 .

编写Dockerfile文件

语法

FROM 指定基础镜像

MAINTAINER 指定镜像维护者信息 maintainer

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

创建私有仓库

1、拉取私有仓库镜像

docker pull registry

2、启动私有仓库容器

docker run -id --name=registry -p 5000:5000 registry

3、打开浏览器 输入地址

http://私有仓库服务器ip:5000/v2/\_catalog，看到{"repositories":[]} 表示私有仓库搭建成功

4、修改daemon.json

vim /etc/docker/daemon.json

# 在上述文件中添加一个key，保存退出。此步用于让 docker 信任私有仓库地址；注意将私有仓库服务器ip修改为自己私有仓库服务器真实ip

{"insecure-registries":["私有仓库服务器ip:5000"]}

5、重启docker 服务

systemctl restart docker

# 多线程导出

**1.为什么需要多线程导出？**

因为需要订单需要数据迁移，而且订单量比较大。如果单线程导出的话会速度非常慢。

**2.多线程导出都用到了什么？**

用到了poi，自定义注解，java反射，多线程和线程池。

**3.具体怎么使用**

先创建一个自定义注解:其中@interface是用来声明这是个自定义注解，@target是用来规定自定义注解适用范围，@retention是用来声明注解的生命周期的。

将自定义注解标注到返回的javabean中需要导出的属性上。

刚开始我是查询出所有数据，再将数据根据每个sheet页要放多少条数据放到list集合中，再根据每个工作簿需要多少个sheet页再放入一个list集合最外的集合中的个数就是工作簿的个数，每个工作簿中的集合的个数就是sheet的个数，最里边的集合就是每个sheet页中的数据，根据工作簿的个数循环使用poi创建Excel工作簿再通过sheet页的个数创建sheet页，先创建列头，通过Java反射获取到类对象，获取所有属性，判断那个属性上被标注了注解，列头就是这些注解的value值。最后循环将数据放到每一行中。最后将所有工作簿放到zipEntry中最后关流。

这种可用性不强，需要提高可用性。所以用到了多线程和线程池。

在线程池中使用多线程查询每个工作簿的数据，将每个工作簿放到服务器当中，使CountDownLatch等待所有线程执行完之后，再将所有的工作簿放到zipEntry中最后同通过HttpServletResponse将流放到浏览器中。

# 基于token的登录

token是什么

token是一个认证令牌

为什么要用token

用token是为了确认用户是否登录，没有登录的话就不能让他访问接口，防止有人恶意访问我们的接口。

刚开始我们的项目里用的session，结果发现session因为是作用在，一个服务器的会话中，在分布式框架中不能获取到数据，然后就用了redis来解决这个问题，把数据存放到redis中，再从redis中取出数据，这样就解决了跨域问题，最后接触到了JWT技术，然后就把redis替换了。

用户在前端输入账号和密码，请求路径传到后端，会遇到一个跨域问题（跨域就是协议，端口，ip，三者之一 有一个不同就构成了跨域），在后端，我们配置了一个跨域拦截器，继承的ZuulFilter来解决跨域问题，然后我们是根据用户的账号查询数据库里是否有这个数据，没有的话说明他没有注册，然后重定向到注册页面让用户注册，反之，将传过来的密码进行MD5加密，用MD5加密密码是为了防止数据库密码泄露，然后判断密码是否正确，如果不正确返回一个状态码给前端，反之，将账号和密码放到map中，用JWT将map生成token令牌，再设置一个失效时间，然后将生成的token返回到前端。

前端我们用的是vue，在登录成功之后，我们把后端传过来的token存到sessionStorage中，然后我们在vue的main.js中设置了一个axios的全局设置，为了让每次的请求都添加上一个token参数，在axios的全局设置中判断本地的sessionStorage中是否有token，有的话就在请求上添加一个token参数，反之发请求的话会报没有token的错误。然后再配置一个axios全局设置的请求响应，判断返回回来的状态码，如果相同的话就返回到登录页面，让他登录。

session和token的区别就是session是存在一个服务器会话中，不能用在分布式框架中，相反token可以。

## 相关问题

①**sessionStorage与localStorage的区别？**

localStorage的生命周期是永久性的。假若使用localStorage存储数据，即使关闭浏览器，也不会让数据消失，除非主动的去删除数据；

sessionStorage 的生命周期是在浏览器关闭前，在整个浏览器未关闭前，其数据一直都是存在的；

sessionStorage不在不同的浏览器窗口中共享；localstorage在所有同源窗口中都是共享的；

②**cookie和session的区别？**

1. cookie数据存放在客户的浏览器上，session数据放在服务器上
2. cookie不是很安全，别人可以分析存放在本地的cookie并进行cookie欺骗，考虑到安全应当使用session
3. session会在一定时间内保存在服务器上，当访问增多，会比较占用你服务器的性能，考虑到减轻服务器性能方面，应当使用cookie
4. 单个cookie保存的数\*据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie
5. 建议将登录信息等重要信息存放为session，其他信息如果需要保留，可以放在cookie中
6. session保存在服务器，客户端不知道其中的信心；cookie保存在客户端，服务器能够知道其中的信息
7. session中保存的是对象，cookie中保存的是字符串
8. session不能区分路径，同一个用户在访问一个网站期间，所有的session在任何一个地方都可以访问到，而cookie中如果设置了路径参数，那么同一个网站中不同路径下的cookie互相是访问不到的

# 权限管理

**shor是一个安全框架 可以用来做登录认证和权限控制常用的方法有**

**login();进行认证的**

**isAuthenticated() 判断是否需要认证**

**getSubject() 获取主题**

**getInstance(); 安全管理器**

**hasRole(); 获取角色**

**isPermitted(); 获取权限**

**在项目中我们实现权限控制是由五张表来完成的一张用户表 一张权限表 一张角色表 一张角色权限表 一张用户权限表 不同的用户登上可以看到不同的页面和功能**

**在接触shor框架之前 权限框架和登录拦截都是通过拦截器来完成的 后来在网上看到了shor框架就开始学习起来 用起来还不错 在项目中我是这么使用shor框架的**

**实现这一系列功能是通过shor框架来完成的 要在web 中配置shor的过滤器 同时配置一个shor框架的配置文件 在配置文件中开启安全认证 ，以及要注入的方法，配置要过滤的方法 还要在springmvc 中开启shor的注解 新建一个类继承AuthorizingRealm这个类 得到两个方法 一个可以用来做登录认证 一个可以用来做权限控制**

**先说一下登录认证吧 在用户登录的时候 通过 SecurityUtils.getSubject(); 获取Subject对象 把用户的账号和密码放入UsernamePasswordToken()对象 通过Subject的login的方法把用户的账号和密码传到我们新建的这个类中，这doGetAuthenticationInfo的方法中的参数获取认证对象 然后获取要认证的对象 通过账号去数据查询获取信息 判断用户是否存在 如果不存在的话就返回一个状态码 存在的话新建一个SimpleAuthenticationInfo对象把 这个对象返回出去 在务层 根据返回的值 去提示用户是否登陆成功。**

**权限控制**

**创建一个SimpleAuthorizationInfo对象再去获取方法中的参数获取**

**需要认证的凭证根据凭证的信息去数据库中查询数据 同时每个方法上要加上一个@RequiresPermissions注解用来做权限控制 根据每条信息的类型来判断是按钮还是目录啥的 最后把认证的信息放入SimpleAuthorizationInfo这个对象中最后作为返回值返回回去。**

# 日志记录

当时我在公司负责的日志记录用户方面的一系列操作，经理当时就把这个业务给我负责了，当时考虑到如果用log4j做的话，一些非开发人员就看不懂了，所以当时我们这个小组就想到了使用aop+自定义注解+反射的技术

aop面向切面编程，他是一种思想，就是在不修改原来 的代码的基础上增加一些新的功能

AOP一般适用于具有横切逻辑的场合，如安全控制、事务管理、日志记录、性能统计等。

面向切面编程简单地说就是在不改变源程序的基础上为代码段增加新的功能，对代码段进行增强处理。

aop的代理通过jdk动态代理，也可以通过cglib（谁个来波）实现，默认是通过jdk动态代理实现的。JDK动态代理需要接口的支持，如果没有接口只有类，则使用cglib实现。

切面 就是要切入的哪个地方

切点 连接点

通知，前置通知，后置通知，环绕通知（joinPoint.proceed()），异常通知，

注解，就是java标注，

自定义注解 ，@inteface

java 注解包含四个元注解，

target 注解用在什么地方，mthod 方法 ,field属性 constroct用于描述构造器 paramater 用于描述参数

documented 注解是否包含在Java文档中

inherted 是否允许子类继承该注解

retation 声明注解的声明周期 runtime项目运行的时候就执行 resources 在编译阶段丢弃 class 在类加载的时候丢弃

这样就可以通过反射去获取注解的信息

在注解中声明了一个sting 类型的value 手工设置日志的信息，之后就可以在controller层的方法上加上注解并对value进行赋值，最后在日志记录的切面类中加上一个注解@component 交给ioc去管理

@aspect声明这是一个切面类，

对于类来说我能知道你的所有属性和方法，对于对象来说我能和调用你的方法

然后在类中加入通知，当时我们使用的是前置通知before

使用joinpoint.signature 获取方法的签名

signature .getMethod 获取方法对象

method.getAnnotion 获取方法上面有没有这个注解

join point.getArgs来获取参数

如果有则使用annotion.value 来获取操作的内容,最后将记录添加到数据库中即可

反射

对于类来说，我能知道你的所有属性和方法，对于对象来说我能得到你的所有属性值和调用你的方法

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。

# 什么是事务？

事务就是一组有逻辑操作单元的sql集合，要么都成功 要么都不成功

**事务的几种传播特性**

1. PROPAGATION\_REQUIRED (瑞块儿): 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: 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: 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: 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: 总是非事务地执行，并挂起任何存在的事务。

 6. PROPAGATION\_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：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

**事务的隔离级别**

 1. ISOLATION\_DEFAULT： 这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别.另外四个与JDBC的隔离级别相对应

 2. ISOLATION\_READ\_UNCOMMITTED： 这是事务最低的隔离级别，它充许令外一个事务可以看到这个事务未提交的数据。这种隔离级别会产生脏读，不可重复读和幻像读。

 3. ISOLATION\_READ\_COMMITTED： 保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据

 4. ISOLATION\_REPEATABLE\_READ： 这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。

5. ISOLATION\_SERIALIZABLE 这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读外，还避免了幻像读。

事务的四大特性

1. 原子性：事务包含的所有操作要么都发生，要么都不发生

2. 持久性：事务一旦提交成功，对数据库的改变应该是永久性的，即便是遇到了故障也不会丢失丢失提交事务的操作

3. 隔离性：事务的操作不应该被并发的其他操作所影响，多个并发事务之间应该是相互隔离的

4. 一致性：事务的所有操作必须是数据库从一个一致性状态到另一个一致性状态

# 定时任务 **发邮件 发短信 会员续费**

## 发送邮件

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类型下的它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，并且给设置一个过期时间，之后判断getset的返回值是否为空，不为空那我们就给他直接返回，并且提示一下我们的任务已经执行过了为空的话我们就先去查询我们数据库中今天过生日的用户，之后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

## 多线程是什么？

多线程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

## 创建线程的有哪些方式？

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

//创建一个线程池

ExecutorService executorService = Executors.newFixedThreadPool(3);//设置主线程3Java线程具有五中基本状态

#### 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；

#### 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；

#### 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：

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。

#### 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。

# Java

## 数据类型

基本数据类型

整数型

Byte 字节型 1个字节

short 短整型 2个字节

int 整形 4个字节

long 长整型 8个字节

浮点型

float 单精度 4个字节 有效位6-7位

double 双精度 8个字节 有效位数15-16位

字符型

char 字符型 2个字节 1个字符

布尔类型

boolean 布尔类型 1个字节 true false

引用数据类型

Class 类 interface 接口 array 数组

## 容器

### List集合

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结合的特点：

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

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

List集合的三种遍历方法

//遍历循环得到每一个元素 iterator()

Iterator<String> iterator = list.iterator();

**while**(iterator.hasNext()){

                    System.*out*.println(iterator.next());

                }

//for循环输出

**for**(**int** i=0;i<list.size();i++){//长度 list.size();

                System.*out*.println(list.get(i));

            }

//foreach循环

**for**(String s:list){

//该集合中的泛型  名：集合名  s代表get(i)

                System.*out*.println(s);

            }

List的实现类有ArrayList、linkedList、Vector

ArrayList 是基于数组的存储结构，查询效率快，删除和添加效率比较慢，线程不安全。

linkedList 是基于链式的存储结构，查询效率慢，删除和添加效率比较快，线程不安全。

Vector 线程安全，已不建议使用。

### Set集合

Set的实现类有HashSet、LinkedHashSet、TreeSet

特点：无序、无下标、不能放重复数据

常用方法：

add()

remove()

clear()

contains()

size()

### Map集合

Map的实现类有HashMap、HashTable(线程安全)

特点：不可重复放数据、无序

常用方法：

put()

get()

remove()

clear()

containsKey()

containsValue()

KeySet()

values()

size()

HashMap和HashTable的区别：

hashMap去掉了HashTable 的contains方法，但是加上了containsValue（）和containsKey（）方法。

hashTable同步的，而HashMap是非同步的，效率上比hashTable要高。

hashMap允许空键值，而hashTable不允许。

HashMap的实现原理

HashMap概述： HashMap是基于哈希表的Map接口的非同步实现。此实现提供所有可选的映射操作，并允许使用null值和null键。此类不保证映射的顺序，特别是它不保证该顺序恒久不变。

HashMap的数据结构： 在java编程语言中，最基本的结构就是两种，一个是数组，另外一个是模拟指针（引用），所有的数据结构都可以用这两个基本结构来构造的，HashMap也不例外。HashMap实际上是一个“链表散列”的数据结构，即数组和链表的结合体。

当我们往Hashmap中put元素时,首先根据key的hashcode重新计算hash值,根绝hash值得到这个元素在数组中的位置(下标),如果该数组在该位置上已经存放了其他元素,那么在这个位置上的元素将以链表的形式存放,新加入的放在链头,最先加入的放入链尾.如果数组中该位置没有元素,就直接将该元素放到数组的该位置上。

需要注意Jdk 1.8中对HashMap的实现做了优化,当链表中的节点数据超过八个之后,该链表会转为红黑树来提高查询效率。

## 包装类

### File

创建File类的三种方式

File(String path)

File(File parent,String child)

File(String parent,String child)

常用方法

file.isDirectory() 判断是否为文件夹

file.isFile() 判断是否为文件

file.exists() 文件或文件夹是否存在

file.mkdir() 创建一级文件夹

file.mkdirs() 创建多级文件夹

file.createNewFile() 创建文件

file.getName() 获取名字（文件夹、文件）

file.renameTo(File file) 重命名

file.delete() 删除文件（删除文件夹或文件，文件夹中有文件删除失败）

file.getPath() 获取当前路径

file.getParent() 获取父路径

file.getParentFile() 获取父级路径对应File对象

file.list() 获取文件夹所有子文件的名字

file.listFiles() 获取文件夹所有子文件对应的File对象构成的数组

File.separator 即 / 用于系统兼容

### Date

1.Date是java.util包中的日期类型

<1> new Date()获取当前的系统时间

<2> getTime();//将日期转化成long类型的毫秒值

（毫秒值越大年龄越小，毫秒值越小年龄越大）

2.日期格式转换类 【SimpleDateFormat】存在与java.text包中

3.将Date类型 转换成 String 类型的方法是 format（）

4.将String 类型 转换成 Date 类型的方法是 parse（）

出现受控异常，那么需要我们进行捕获处理，否则无法编译通过

### string

常用方法

charAt(下标) 返回对应下标的字符 ,最大只能是字符串长度-1

concat() 字符串拼接，不影响原来的字符串，返回新的字符串

indexOf("") 返回的是字符串对应的下标, 当字符串不存在的时候    返回-1

length() 字符串长度 在数组中length是属性，在字符串中length是方法

trim()  去掉字符串前后的空格

toUpperCase() 将该字符串中的所有小写改为大写

toLowerCase() 将字符串中所有的大写字母改为小写字母

split() 从指定字符串的位置分割，得到相应的字符串数组

substring(1) 从指定下标截取到字符串末尾  有一个参数的时候

substring(2,5) 从开始的下标，截取到结束的下标的前一个字符       包前不包后

equals() 比较字符串是否相等，相等返回true，不相等返回false

contains(“”)  包含  如果包含字符串，返回true，否则返回false

replaceAll("想要替换的字符串","替换后的字符串") 返回的是替换后的字符串

## exception

### 什么是异常

在java代码中往往在代码运行过程中出现的一些常见的问题，我们称之为异常

### 异常的分类

Throwable

Error  错误  在程序运行过程中出现的不可修复的错误（最常见的是内存溢出）

Exception 异常

Exception为RunTimeException（运行时异常）和 非运行时异常

运行时异常：

在编写代码过程中，由于逻辑判断出现的错误，这类异常不会影响代码的正常编译

常见的异常：

a.   数组越界异常

b.   数学计算异常

c.   空指针异常

IndexOutOfBoundsException索引（下标）越界异常

ArithmeticException：数学计算异常

NullPointerException：空指针异常

ArrayIndexOutOfBoundsException：数组索引越界异常

ClassNotFoundException：类文件未找到异常

ClassCastException：造型异常（类型转换异常）

NumberFormatException数字格式转化异常

非运行时异常：

在程序编译过程中就强制要求处理的异常，若不处理，那么将不能编译通过；

常见的异常

a.   IOException、Io流异常

b.   FileNotFoundException：文件未找到异常

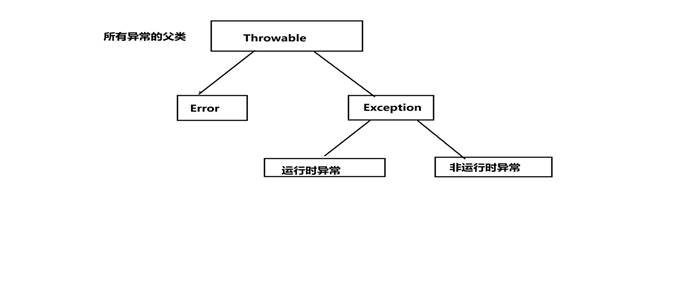
c.   EOFException：读写文件尾异常

d.   MalformedURLException：URL格式错

e.   =socketException：Socket异常

f.   SQLException sql异常

G、ParseException 日期格式转换



1、异常的解决方法

1）Throws  抛出  声明

2）Try /catch  捕获

Try{

可能出现异常的代码块

}catch{

}catch{

}

2.   finally  fianl  finalize()

1）Finally与try  catch连用，主要用于关闭连接。

2）在finally里出现的代码无论是否出现异常，都会正常执行。

3）Fianl修饰的类不能被继承，不能被重写；修饰的属性是常量，必须赋初始值，不能二次赋值。

4）Fianlize（）垃圾回收机制。

## stream

input    输入流   把本地文件输入到程序（工作空间）中（上传）

（1）InputStream   字节输入流

（2）Reader        字符输入流

 output   输出流   把系统中的文件保存到本地中（下载）

（1）OutputStream   字节输出流

（2）Writer         字符输出流

字节流   用于音乐  视频 图片的上传下载

字符流    用于文档的上传下载

缓冲流

BufferedInputStream     字节输入缓冲流

BufferedReader         字符输入缓冲流

BufferedOutputStream   字节输出缓冲流

BufferedWriter         字符输出缓冲流

## 控制语句

**if for while foreach 迭代器**

## if 判断

格式一：

if(布尔表达式){

当满足条件时执行的语句

}else{

当不满足条件时执行的语句

}

格式二：

if(布尔表达式){

当满足条件时执行的语句

}else if(布尔表达式){

当满足条件时执行的语句

}else if(.......){

......

}else{

当不满足所有条件时执行的语句

}

## 三大循环

1.**while** 当。。。的时候

格式：

初始值；

while（布尔表达式）{

当布尔表达式成立的时候，才进 入循环体 循环体； 步长；

}

2.**do while** do做 干什么 while 当。。。 的时候

格式:

do{

循环体； 步长；

}while(布尔表达式);

3.**for**

for(初始值；循环条件；步长){

循环体；

}

## while和do while的区别

while先判断循环条件，在执行循环体，do while 先执行循环体，在判断循环条件 do while来说至少执行一次循环体

break、continue、return三者的区别**：**

break 中断本次循环，只能在switch或循环体中使用

continue 跳过本次循环，继续下次循环，只能在循环体中使用

return：在方法中使用，代表有返回值，需要返 回东西

## 反射

## 什么是反射（算命先生）

对类来说 我能知道你的所有属性和方法

对于对象来说 我能得到所有的属性值和调用你的方法

得到 Class 的三种方式

1、通过对象调用 getClass() 方法来获取,通常应用在：比如你传过来一个 Object

类型的对象，而我不知道你具体是什么类，用这种方法

　 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

这说明任何一个类都有一个隐含的静态成员变量 class

　　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");

通过 Class 类获取成员变量、成员方法、接口、超类、构造方法等

getName()：获得类的完整名字。

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

getMethod(String name, Class[] parameterTypes)：获得类的特定方法，name参数指定方法的名字，parameterTypes 参数指定方法的参数类型。

getConstructors()：获得类的public类型的构造方法。

getConstructor(Class[] parameterTypes)：获得类的特定构造方法，parameterTypes 参数指定构造方法的参数类型。

newInstance()：通过类的不带参数的构造方法创建这个类的一个对象。

## 注解

关键字 @Interface

自定义注解需要注意四点：

　　1.@Target({ ElementType.METHOD })进行作用域设定(TYPE/类接口，METHOD/方法，CONSTRUCTOR/构造器，FIELD/字段)、

　　2.@Retention(RetentionPolicy.RUNTIME)进行使用范围设定，也就是说用于那种类型。

　　3.@Inherited是否进行子类继承

　　4.@Documented生成文档。

## JDK1.8新特性

### lamda表达式（接口的新实现方式）

lambda表达式的重要特征:

**可选类型声明：**不需要声明参数类型，编译器可以统一识别参数值。

**可选的参数圆括号：**一个参数无需定义圆括号，但多个参数需要定义圆括号。

**可选的大括号：**如果主体包含了一个语句，就不需要使用大括号。

**可选的返回关键字：**如果主体只有一个表达式返回值则编译器会自动返回值，大括号需要指定明表达式返回了一个数值。

接口的新实现方式

接口中的方法可以有方法体，但需要关键字static或者default来修饰，使用static来修饰的称之为静态方法，静态方法通过接口名来调用，使用default来修饰的称之为默认方法，默认方法通过实例对象来调用。

### stream流操作

### 生成流

stream() − 为集合创建串行流

parallelStream() − 为集合创建并行流

### 中间操作

**filter**：过滤，对数据源中的元素进行过滤，返回符合条件的Stream。

**map**：转换，将数据源中的元素转换成其他类型的Stream返回。（mapToInt、mapToLong、mapToDouble 返回对应数据类型int,long,double的Stream,map返回其本身的数据类Stream），主要可以对元素进行操作。

**flatMap**：合并，一个或多个Stream流合并成一个新的Stream，和转换工序类似。

**distinct**：去重，将数据源中的重复的元素去掉，和SQl中的关键字distinct一样的效果。不过在java中是根据元素的hashCode()和equals()来进行判断。

**sorted**：排序，将数据源中的元素进行排序。可以通过Comparator自定义排序器。

**peek**：调试，这个方法主要用于Debug模式调试，它对进入其中的元素进行操作后并不会返回Strean,所以对数据源不会有影响（个人见解）。

**limit**：返回数据源中前n个元素，短路操作（执行完成后并不会处理其他元素）。

**skip**:返回第n个元素之后的数据组成的Stream。配合limit操作可实现分页。

### 终止操作

**forEach**：循环执行Stream中的元素。

**toArray**：将Stream的元素返回一个对应的数组类型。

**reduce**：聚合操作，常用于统计数据。

**collect**：聚合操作，封装成目标集合类型。

**anyMatch**：或，相当于“||”，只要一个满足条件就返回true。

**allMatch**：与，相当于“&&”，全部满足条件才返回true。

**noneMatch**：全部不满足条件才返回true,与allMatch相反。

**findFirst**：获取第一个元素，短路。

**findAny**：获取任意一个元素，短路。

**min、max、count**： 聚合操作，最小值，最大值，总数量。

**forEachOrdered** 暗元素顺序执行循环操作

## 多线程

### 1 多线程的创建方式

extends Thread

implments Runnable

### 2 线程的状态

创建（new了一个线程对象）

就绪（线程对象创建之后，调用了start()方法，等待被cpu选中）

运行（获取到了cpu的使用权，然后执行程序代码）

阻塞（运行的程序执行sleep(),join(),I/O请求时）

死亡（线程run()、main() 方法执行结束，或者因异常退出了run()方法，则该线程结束生命周期）

### 3 常用操作

start:（开启线程，start是通过线程来调用run方法）

run:   此run非彼run （不是在run方法实现线程的逻辑，而是thread.run()，这个run方法是直接调用了线程中的run）

yield:（暂停当前线程，并执行其他线程）

sleep:（使当前线程由运行状态变成阻塞状态，若睡眠时其他线程调用了interrupt方法，会导致sleep抛出异常InterruptException）

join:（保证当前线程在其他线程开始时会结束）

interrupt:（中断线程）

wait/notify:（从Object类继承下来的方法）

setPriority:（设置线程优先级（只能在线程开始前设置））

stop:（强制结束线程）

### 4 安全问题

同步锁：

多个线程操作一个共享资源，会出现线程安全问题。

解决方法：在方法上加一个同步锁 synchronized

死锁：

线程A拿着线程B要访问的资源，线程B拿着线程A要访问的资源，两者互不相让，造成死锁，

解决方法：stop一个线程，让另一个线程先走完。

# javascript

## 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

js检测数据类型

typeof() 检测类型。

字符型 日期 对象**（json）**

--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

## --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

## 选择器

## 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

document的内置对象

### --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

## 常用事件 （鼠标 键盘 页面）

### 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

### jsp

18. **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

## 18. 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

19. **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

# 框架

**1.什么是mvc思想**

**m就是模型层，所有的用户数据、状态以及程序逻辑，独立于视图和控制器**

**c控制层：负责请求转发，对请求进行处理**

**v视图层：视图是用户看到并与之交互的界面**

**2.为什么要使用mvc**

**有利于代码的复用**

**有利于开发人员分工**

**有利于降低程序模块间的耦合，便于程序的维护与扩展。**

**jdbc：**

**什么是jdbc：jdbc就是一门通过java程序访问数据库的技术，连通JAVA程 序和数据库之间的桥梁**

**搭建JDBC开发环境：**

**创建项目----配置JDK-------配置工作空间的默认编码-----在项目中导入 java连接oracle数据库驱动的jar包**

**Jdbc连接数据库七大步：**

**1.加载驱动：Class.forName**

**2.创建连接： DriverManager.getConnection（“数据库连接信息”）**

**3.编写sql语句**

**4.得到到statement对象：connection.prepareStatement(sql);**

**5.执行sql 得到结果集**

6.处理结果集：statement.executeQuery();

7.关闭资源

JDBC中常用类详解：

 DriverManager类

Jdbc程序中的DriverManager用于加载驱动，并创建java程序与数据库的连接，这个类的常用方法：

DriverManager.getConnection(url, user, password) 获取数据库连接

Connection类

Jdbc程序中的Connection，它用于代表java程序和数据库的连接，Connection是jdbc中最重要的一个对象，java程序和数据库所有交互都是通过connection对象完成的，这个对象的常用方法：

prepareStatement(sql) ：创建向数据库发送预编译sql的PrepareSatement对象。

PreparedStatement类

Jdbc程序中的PreparedStatement对象用于向数据库发送SQL语句PreparedStatement对象常用方法：

executeQuery() ：用于向数据发送查询语句并返回结果集

executeUpdate()：用于向数据库发送insert、update或delete语句并返回受影响的行数

ResultSet类

一个ResultSet对象对应着一个由查询语句返回的一个表,这个表中包含所有的查询结果,实际上,我们就可以将一个ResultSet对象看成一个表。对ResultSet对象的处理必须逐行进行,而对每一行中的各个列,可以按任何顺序进行处理

**mybatis**

**原理详解：**

        MyBatis应用程序根据XML配置文件创建SqlSessionFactory，SqlSessionFactory在根据配置，配置来源于两个地方，一处是配置文件，一处是Java代码的注解，获取一个SqlSession。SqlSession包含了执行sql所需要的所有方法，可以通过SqlSession实例直接运行映射的sql语句，完成对数据的增删改查和事务提交等，用完之后关闭SqlSession。

**为什么要使用mybatis**

灵活应用：mybatis不会对应用程序或者数据库的现有设计强加任何影响。 sql写在xml里，便于统一管理和优化

解除sql与程序代码的耦合，使系统的设计更清晰，更易维护，更易单元测试。sql和代码的分离，提高了可维护性。

提供xml标签，支持编写动态sql。

但同时mybatis的优点也是他的缺点

1、编写SQL语句时工作量很大，尤其是字段多、关联表多时

2、SQL语句依赖于数据库，导致数据库移植性差，不能更换数据库。

3、虽然简化了数据绑定代码，但是整个底层数据库查询实际还是要自己写的，工作量也比较大，而且不太容易适应快速数据库修改。

**mybatis # $ 区别 ：**

**#占位符：**

**#将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号**

**将传入的数据直接显示生成在sql中**

**#是给占位符赋值防止sql注入问题**

**$拼接符：**

**.$方式一般用于传入数据库对象，例如传入表名**

**$是用来拼接的可能存在sql注入问题**

**不关键字用# 是关键字就要用$来拼接**

**mybatis常用注解**

**select查询，delete删除，update修改，insert新增**

**param加入参数，MapperScan扫描dao层，方便使用**

**jpa**

**1. JPA是什么**

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。

2**. Spring Data JPA是什么**

Spring Data JPA 是 Spring 基于**ORM 框架**、**JPA 规范**的基础上封装的一套**JPA应用框架**，可使开发者用**极简的代码**即可实现对**数据的访问和操作**。Spring Data JPA不需要过多的关心Dao层的实现，只需关注我们继承的接口，按照一定的规则去编写我们的接口即可，spring会按照规范动态生成我们接口的实现类进行注入，并且实现类里包含了一些常规操作的方法实现。如果使用JPA提供的接口来操作ORM框架，可以不写任何实现就能对数据库进行CRUD(Create创建 Read读取 Update更新 Delete删除)操作，还可以进行简单的分页，排序操作。

**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！

**JPA主键生成策略**

**Identity(爱den忒忒**):主键由数据库自动生成 （主要是自动增长型）

**Sequence(谁困死):**根据底层数据库的序列来生成主键，条件是数据库支持序列

**Table**:使用特定的数据表格来保存主键

**Auto**:由持久化引擎来决定从以上三种策略中选择最合适的

**Hibernate**

**配置hibernate.cfg.xml(数据库驱动 连接信息 方言 映射等) 新建映射文件(\*.hbm.xml放到model目录下)**

1. 读取并解析配置文件
2. 读取并解析映射信息，创建SessionFactory
3. 打开Sesssion
4. 创建事务Transation
5. 持久化操作
6. 提交事务
7. 关闭Session
8. 关闭SesstionFactory

**使用Hibernate框架就不用我们写很多繁琐的SQL语句。Hibernate实现了ORM，能够将对象映射成数据库表，从而简化我们的开发！**

**Hibernate中怎样实现类之间的关系?(如：一对多、多对多的关系)**

它们通过配置文件中的many-to-one、one-to-many、many-to-many来实现类之间的关联关系的。

**jdbc和hibernate的区别**

**hibernate是JDBC的轻量级封装，hibernate基于jdbc**

**jdbc:**

**jdbc是纯手工的原生sql语句**

**执行效率高于hibernate**

**jdbc可以直接操作数据库的表如:select \* from 表名 hibernate：**

**hibernate是全自动**

**跨数据库**

**开发效率高于jdbc**

**使用的是hql语句最终会转化为sql语句.语法是 from 对象名**

**Hibernate get和load的区别**

**访问对象时发送sql语句，访问实体不存在时返回（对象不存在异常）,如果设置了lazy=“false”,load就不再进行延迟加载**

**get立即加载 查询时立即发送sql语句，当访问实体不存在是返回null**

hibernate三种状态

*临时状态、持久化状态、托管状态/游离状态*

transient['trænzɪənt](临时状态)：

在创建对象即在new之后，在save之前的状态：

session中和数据库中都不存在该数据

persistent[pə'sɪst(ə)nt](持久化状态):

在save之后，session关闭之前，

session中和数据库中都存在对应的数据

detached[dɪ'tætʃt](托管状态/游离状态):

session关闭之后，session不存在，数据中有对应的数据

springMVC是我最常用的web控制层框架，这类框架都有共同的功能，就是通过过滤器或者拦截器来处理请求，我们的用户访问资源的路径包括协议，主机IP和端口号，资源名，我们控制层框处理的就是这个资源名和资源后边跟着的参数。具体流程是：

1. 客户端请求提交到DispatcherServlet(diss趴扯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渲染到views。

   6.将views结果显示到客户端。

**我们的springMVC用的都是注解模式其中比较常用的注解有**

（1）. @RequestMapping("/query") 配置就是请求路径。注意：他可以加到方法上，也可以加到类上面（类似于namespace）

（2）. @PathVariable 加在参数上，@RequestMapping(value = "/params4/{username}/{password}", method = RequestMethod.GET) 注入访问路径中参数

（3）. @RequestParam 加载参数上使用时，可以控制该参数是否必须传递。

（4）. @ResponseBody 加载方法上，返回的是纯数据。（注意他不要和ModelAndView连用）

同时它依赖jackson的jar包对返回值进行json化的处理

**springMVC的线程安全问题**

同时呢我们springMVC是spring的一个组件，我们在控制层上加上spring的注解@Controller,将控制层类交给交给spring管理，因为springMVC传的形参，每次调用方法都会把参数给重置，因此springMVC也是线程安全的。

**springmvc接收值的几种方式**

HttpServletRequest接收、属性接 收、对象接收，restful接收

**springmvc返回页面的几种方式以及如何跳转页面的**

使用ModelAndView中的setViewName返回。

使用@ResponseBody注解，返回的是json串。

如果是String 直接ruturn后面加上地址，有参数的话放到model中返回。

也可以放到modelMap中返回。

跳转页面默认是转发forward,也可以用重定向，redirect

**forward和redirect的区别**

forward(转发)：地址栏不会发生改变，数据共享，一次请求，执行效率高。

redirect(重定向)：地址栏会发生改变，数据不共享，多次请求，执行效率低

SpringBoot

springboot，它的主要特点是抛弃了复杂的XML文件配置，大量使用注解的方式，提升了项目的启动速度，因为xml文件在启动时需要先解析再生效，同时它可以通过maven下载依赖版本兼容的jar包。它的另一个特性是开箱即用，比如我们我们引用springboot-web这个jar框架就会自带restful风格的接口，自带springmvc和tomcat容器，如果导入了mysql的数据库连接包，就必须要配置数据连接【注：这个可以通过注解参数忽略掉】。

springboot启动是通过一个main方法启动的，这个类上需要加一个@SpringBootApplication注解，这个注解可以看作是 @Configuration、@EnableAutoConfiguration、@ComponentScan 注解的集合。这三个注解的作用分别是

（1）@EnableAutoConfiguratio：启用 SpringBoot 的自动配置机制也就是自动扫描其他注解让其生效。

（2）@ComponentScan：扫描被@Component (@Service,@Controller)注解的 bean，注解默认会扫描该类所在的包下所有的类

（3）@Configuration：允许在 Spring 上下文中注册额外的 bean 或导入其他配置类。

什么是shiro

shiro是一个安全的框架 可以帮助我们完成认证 授权等权限操作

在web.xml中配置shiro过滤器 在springmvc中 开启shiro注解 在spring中导入shiro.xml配置文件

在shiro.xml中配置拦截所有和所需要释放的请求路径 配置安全管理器 配置自定义的realm的bean

在的方法中 我们需要获取到我们的认证主体 subject对象 构建认证对象 调用subje.login方法传入认证对象

在我们自定义realm类中继承AuthorizingRealm【奥泽瑞z莹瑞来木】重写授权和认证点的方法

认证是先获取传过来的认证对象 通过用户名进行查询 判断对象是否存在 不存在就抛出认证异常交给controller全局异常处理 在我们查询到的对象中 获取密码 构建正确的认证信息 并返回

授权是先获取我们认证信息中的唯一标识 根据标识查询数据库 获取用户权限

循环用户拥有的权限 权限放进构建的授权对象中 返回这个对象

servlet （servlet是单例多线程！！！）

**Servlet 是什么？**

**Java Servlet 是运行在 Web 服务器或应用服务器上的程序，它是作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层**

为什么要用servlert

**性能更好，它是独立的因为他们是用java编写的，他是在web服务器的地址空间执行的，这样它就没有必要再创建一个单独的进程来处理每个客户端请求。**

**Servlet 创有三种方式。**

**1、实现 Servlet 接口**

**2、继承 HttpServlet 方法**

**3、继承 GenericServlet 类**

**servlet的生命周期由servlet容器控制，该容器创建servlet的实例。servlet的生命周期就是指servlet实例在创建之后响应客户端请求直至销毁的全过程。servlet实例的首次创建取决于servlet的首次调用。servlet接口定义了servlet生命周期的三个方法，这些方法就是init(),service(),destroy().**

**（1）init()方法用来把servlet导入和初始化。这个方法在servlet被预加载或在第一次请求时执行。**

**（2）servlet处理0个或多个请求。servlet对每个方法都用service()方法来处理。**

**（3）当web应用声明servlet被关闭、servlet被销毁、垃圾收集器对资源进行收集时，用destroy()方法来关闭servlet。**

# 架构

**mongodb**

**mongdb是一个基于分布式储存的非关系型数据库，是最像关系型数据库的非关系型数据库。**

主要特点有

MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易。

你可以在MongoDB记录中设置任何属性的索引来实现更快的排序。

Mongo支持丰富的查询表达式。查询指令使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。

常用命令：

use 数据库名 连接到某个数据库，如果没有直接创建

show dbs 查看所有数据库

db.createCollections 创建集合

show tables/collcetions 查看所有集合

db.集合名.drop() 删除集合

db.集合名.save() 新增文档 如果有id主键就覆盖没有就自增

**db.集合名.insert() 新增文档 有主键id会报错 没有会自增**

**redis**

**linux命令**

**ip addr 查看虚拟机ip地址**

ls 查询该目录下的所有文件

ps -ef |grep 查看进程

mkdir 创建文件夹

rm -rf 删除

cp 复制文件

mv 移动文件

cat 展示整个文件

vi 编辑文件

systemctl status firewalld（stop）关闭 （start）开启 （disable）永久关闭

查看防火墙状态

**docker**

**nginx（负载均衡）**

nignx 是一个高性能的 HTTP 和反向代理服务。用来做集成用的。

轮询策略：是默认方式每个请求会按时间顺序逐一分配到不同的后端服务器。

权重策略：在轮询策略的基础上指定轮询的几率，权重越高需要处理的请求就越多，此策略比较适合服务器的硬件配置差别比较大的情况。

ip策略：这个方法确保了相同的客户端的请求一直发送到相同的服务器，以保证session会话。这样每个访客都固定访问一个后端服务器，可以解决session不能跨服务器的问题。

**springboot +springcloud（）**

**springboot 就是一款微服务是用来接口开发，就像 Maven 整合了所有的 Jar 包，Spring Boot 整合了所有的框架。**

Springboot优点：

1. 创建独立的Spring应用程序

2. 嵌入的Tomcat，无需部署WAR文件

3. 简化Maven配置

4. 自动配置Spring

5. 提供生产就绪型功能，如指标，健康检查和外部配置

6. 绝对没有代码生成并且对XML也没有配置要求

**分布式事务 seata解决的**

seata是用来解决分布式事务的，因为一般的事务只能解决本地的接口，但是不能解决分布式框架中调用的接口，原理就是一个请求传过来之后会生成一个总事务交个seata管理，之后带着xid继续往下走，在执行每个业务的时候会生成一个分支事务交给seata管理如果这个业务存在异常会修改分支事务和总事务的状态然后直接回滚。如果没有异常就执行下个业务。