# B简历：

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

### 基本信息

|  |  |  |  |
| --- | --- | --- | --- |
| 基本情况 | | | |
| 姓名 | 王龙 | 性别 | 男 |
| 出生日期 | 1995-01-09 | 属相 | 猪 |
| 现住地 | 郑州升龙又一城 |  |  |
| 大学情况 | | | |
| 学校 | **成都东软学院** | 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 | [四川省成都市都江堰市青城山镇东软大道1号](https://api.map.baidu.com/marker?output=html&location=30.894435,103.60298&title=%E6%88%90%E9%83%BD%E4%B8%9C%E8%BD%AF%E5%AD%A6%E9%99%A2&content=%E5%9B%9B%E5%B7%9D%E7%9C%81%E6%88%90%E9%83%BD%E5%B8%82%E9%83%BD%E6%B1%9F%E5%A0%B0%E5%B8%82%E9%9D%92%E5%9F%8E%E5%B1%B1%E9%95%87%E4%B8%9C%E8%BD%AF%E5%A4%A7%E9%81%931%E5%8F%B7&src=vmpZixin) | 校长 | **学校领导都没见过，所以也不知道校长的名字** |
| 去学校的方式： | 公交线路：地铁7号线 → 地铁4号线 → w23路 → 758路 → 都江堰102路，全程约71.4公里 | | |
| 上小学日期： | **2001年9月 6岁六年** | | |
| 上初中日期： | **2007年9月 12岁三年** | | |
| 上高中日期： | **2010年9月 15岁三年** | | |
| 上大学日期： | **2013年9月 18岁四年** | | |
| 毕业时间： | **2017年 7月 22岁** | | |
| 在校学生： | **5000左右** | | |
| 有教师： | **不太清楚** | | |

|  |  |
| --- | --- |
| 第一家公司 ： | 郑州扬宸信息技术有限公司 |
| **地址：** | 郑州市 金水区 中国人保大厦 19楼 |
| **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：** | **崔岩** |
| **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：** | 五号线内环->中央商务区 地铁口 A2口 |

|  |  |
| --- | --- |
| **第二家公司情况：** | 北京思莱维恩信息技术有限公司(只说公司简称) |
| **公司地址**： | 北京昌平区中关村生命科学园生命科学院 （北京 - 昌平区 - 泰康商学院东门） |
| **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：** | 5号线->15号线 国展 地铁站D口出来  8号->朱辛庄站 ->昌平线->生命科学院做公交 417路->科学园路 |
| **入职时间：** | **2019年2月** |
| **离职时间：** | **2021年1月初** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **席亚歌** |
| **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(10K)** |

|  |
| --- |
| 四年涨薪： **2015年工作的时候是5.5K，税后大概5.4K左右，**  **转正6K左右**  **2016上半年涨了1K左右 ，总7K左右。**  **2017年2月跳槽涨了3K左右 总10K**  **2017年12月涨1K总 11K左右**  **2018年9月工作一年涨了2K左右**  **2019年7月工作一年涨了2K左右**  **2019年11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/核心技术点）【主动说】========最近做的是基于springboot微服务的电商项目，再之前一点做的是ssm项目，负责了其中的几个模块，我先给你说下框架方面的东西吧

**框架篇**

**首先讲下对springboot的理解**

SpringBoot是Spring开发的一个框架，SpringBoot项目就是让你尽可能快的跑起来Spring项目，SpringBoot其实不是什么新的框架，只是里面包含了很多框架的使用方式，需要用的时候导入pom文件即可，SpringBoot的优点就是开箱即用，约定大于配置，内置web容器。开发springboot项目的时候，我发现springboot的核心就是启动类上的两行代码，[在启动类上面加上@SpringBootApplication，在main方法中加入SpringApplication .run](mailto:在启动类上面加上@SpringBootApplication，在main方法中加入SpringApplication.run)方法即可。@SpringBootApplication注解由三个注解组成@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。

快速创建一个SpringBoot项目

　（1）创建一个maven工程。

　　（2）导入SpringBoot的依赖

　　（3）建一个类，在类上标注@SpringBootApplication,来告诉SpringBoot这是一个主程序类，说明这是一个SpringBoot应用

　　（4）再主程序类里添加main方法（idea快捷键：psvm），然后在main方法里添加SpringApplication.run(主程序类.class,args);

　　（5）编写相关的Controller、Service等。

　　（6）运行测试：主程序里直接运行main方法即可开启应用。tomcat8080端口（内置了tomcat）。

　　（7）部署：导入SpingBoot的Maven插件

**讲述对springcloud微服务的理解**

分布式服务框架就是不同的模块放在不同的机器上面 微服务架构是分布式服务的一种，微服务架构就是在这种基础上进行了更细微的拆分，把还在一个系统中的不同业务拆分为单独的服务，对于这个服务可以交给一个团队进行独立的技术选型，独立的部署独立的运维。

在具体开发一个微服务的时候，用的是**springcloud**，这个框架比较重要的组件Eureka ,ribbon ,feign,hystrix,zuul。

首先介绍一下Eureka

在项目中我们用Eureka建立注册中心项目，在pom文件中引入Eureka的依赖，之后在启动类中加入 @EnableEurekaServer的注解证明该项目的作用是作为注册中心使用，并且在对应的application.properties中指明Eureka注册中心服务器的地址，方便提供者和消费者对其进行访问，为了增强注册中心的高可用性，我们做了Eureka集群，防止因为单台Eureka导致的单点故障问题。

如果没有注册中心，客户端通过ip地址访问服务端，如果服务越来越多，调用也会越来越复杂，一旦ip地址发生变动，则所有用到的地方都需要发生改变，维护起来会特别麻烦；如果服务端宕机了，所有访问的客户端也都会受到影响。通过注册中心可以让服务端注册到注册中心上去，并且可以进行负载均衡；这样就可以达到高可用以及提高并发的 目的。客户端 也不需要 直接 调用服务端，而是通过注册中心获取地址列表，从地址列表中选一个服务端接口进行调用，后续的维护各方面也会比较方便。

Eureka自我保护机制

Eureka 自我保护机制是为了防止误杀服务而提供的一个机制。当个别服务出现心跳检测失败时，则认为是服务的问题，剔除服务；当 Eureka 捕获到大量服务的心跳失败时，则认为可能是网络问题，进入自我保护机制，一旦进入保护机制， Eureka会尝试保护其服务列表中的信息，不会删除服务列表中的任何服务，并且在页面给出提示，

Eureka 仍然能够接受新服务的注册和查询请求，但是不会被同步到其它节点上(即保证当前节点依然可用)

当网络稳定时，再将当前实例新的注册信息同步到其它节点中，当服务心跳恢复时，Eureka 会自动退出自我保护机制。

然后就是ribbion组件

Ribbon 是一套实现客户端负载均衡的工具。SpringCloud里的客户端负载均衡, 都是使用的Ribbon

还有就是feign组件

feign也是一个消费者调用服务的一个组件， Feign集成了Ribbon。利用Ribbon维护了服务列表信息，并且通过轮询实现了客户端的负载均衡。而与Ribbon不同的是，调用服务的方式不同

Feign的话调用服务接口就像调用本地方法一样，避免了需要不断的解析/封装json 数据的繁琐，使用方法就是将我们发布的restful风格接口抽象化成一个方法，然后在该接口类上加上@FeignClient，并配置对应服务名，同时在启动类上加上@EnableFeignClients让feign注解生效，在调用接口是需要注意的是get请求传参时必须要加上@RequestParam注解，如果post请求传递的参数是一个实体对象还要加上@RequestBody，使用feign的好处就是调用接口比较简单。

还有hystrix组件

Hystrix 负责监控服务之间的调用情况，连续多次失败的情况进行熔断保护。保护的方法就是使用 Fallback，当调用的服务出现故障时，就可以用 Fallback 方法的返回值；Hystrix 间隔时间会再次检查故障的服务，如果故障服务恢复，将继续使用服务。

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。

还有zuul组件

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。

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。

**服务网关两个核心**

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）。**

我们做分布式框架肯定会有分布式事务的问题

  所谓事务是指一组逻辑操作单元即一组sql语句。当这个单元中的一部分操作失败的时候，整个事务回滚，只有全部正确才完成提交

      事务的四大特性ACID

  原子性（Atomicity）

原子性是指事务是一个不可分割的工作单位，事务的操作要么都发生要么都不发生

  一致性（Consistentcy）

事务必须使数据库从一个一致性状态到另一个一致性状态

  隔离性（Isolation）

一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

  持久性（Durability）

指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

事务的几种传播特性

 1. PROPAGATION\_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: 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。

分布式事务，简单的说，就是一次大的操作由不同的小操作组成，这些小的操作属于不同的应用，分布在不同的服务器上，分布式事务需要保证这些小操作要么全部成功，要么全部失败。本质上来说，分布式事务就是为了保证不同数据库的数据一致性。

最经典的场景就是支付了，一笔支付，是对买家账户进行扣款，同时对卖家账户进行加钱， 这些操作必须在一个事务里执行，要么全部成功，要么全部失败。而对于买家账户属于买家 中心，对应的是买家数据库，而卖家账户属于卖家中心，对应的是卖家数据库，对不同数据库的操作必然需要引入分布式事务。

seata解决分布式事务

@GlobalTransactional 全局事务

在Seata中，涉及到TC,TM,RM这几个关键组件，其中TC叫做

事务协调器，TM叫做事务管理器，RM叫做资源管理器。在使用分布式事务的方法上面加上@GlobalTransactional，seate检测到这个注解之后，就会向事务协调器申请一个全局事务id，并在数据库全局事务表生成一条数据，之后在这个方法里产生的所有事务，都会在分支事务表中生成一条分支事务数据并且对全局事务进行关联，同时还会将事务要修改的表进行上锁，防止其他事务同时修改表，在创建完分支事务后，就会在undo\_log表中根据执行的SQL语句自动生成回滚的SQL语句，并且绑定 分支事务Id，和全局事务Id，当业务执行完成之后，如果出现异常，就根据这个sql语句自动进行回滚，如果没有出现异常，协调器就会通知所有分支事务，将创建的所有数据给删除；

# 1.多线程

1 多线程的创建方式

方式一：继承Thread类的方式

方式二：实现Runnable接口的方式

重写run方法 （run方法为多线程的方法体） 启动用start方法

2 线程通常都有五种状态，创建、就绪、运行、阻塞和死亡。

* 创建状态。在生成线程对象，并没有调用该对象的start方法，这是线程处于创建状态。
* 就绪状态。当调用了线程对象的start方法之后，该线程就进入了就绪状态，但是此时线程调度程序还没有把该线程设置为当前线程，此时处于就绪状态。在线程运行之后，从等待或者睡眠中回来之后，也会处于就绪状态。
* 运行状态。线程调度程序将处于就绪状态的线程设置为当前线程，此时线程就进入了运行状态，开始运行run函数当中的代码。
* 阻塞状态。线程正在运行的时候，被暂停，通常是为了等待某个时间的发生(比如说某项资源就绪)之后再继续运行。sleep,suspend，wait等方法都可以导致线程阻塞。
* 死亡状态。如果一个线程的run方法执行结束或者调用stop方法后，该线程就会死亡。对于已经死亡的线程，无法再使用start方法令其进入就绪

3 安全问题

当多个线程同时执行，它们之间也是相互抢占资源来执行，抢占资源的过程中会导致出现非法数据，这种现象会出现多线程的并发安全问题。

通过同步代码块的方式将对象锁起来。用synchronized来将代码限制起来

4 什么是死锁？

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系 统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。是操作系统层面的一个错误，是进程死锁的简称

5 线程和进程的区别？

简而言之，进程是程序运行和资源分配的基本单位，一个程序至少有一个进程，一个进程至少有一个线程。进程在执行过程中拥有独立的内存单元，而多个线程共享内存资源，减少切换次数，从而效率更高。线程是进程的一个实体，是cpu调度和分派的基本单位，是比程序更小的能独立运行的基本单位。同一进程中的多个线程之间可以并发执行。

你们项目中哪些地方用过 （不用行不行）

多线程定时发送邮件 多线程poi导出

做poi导出，我们同样需要使用反射和自定义注解，但不同的是，这时候自定义注解添加的位置成了属性，作用是为了得到需要导出的字段以及相应的数据。

最初，我只是做了一个简单的Excel表，使用XSSFWorkbook，生成sheet页、行、列，随之放入数据，最后写入Excel中，关流即可。这样写并没有什么问题，开发和测试都一切顺利。但是到了上线阶段数据量上去之后就不适用了。

后来我想到了生成多个excel文件多个sheet页并将其做成一个压缩包，这样再多的数据量也不会有事了。我先是尝试了使用流的方式进行操作。首先创建了ServletOutputStrea流和ZipOutputStream流，在此，我们需要确定每个sheet页数据的条数和sheet页的个数，以此来确定excel文件的个数，然后使用HSSFWorkbook通过循环，生成sheet页、行、列，随后放入数据，生成excel，依次写入zip压缩流下载完成。

   再之后，需求发生了变化，需要做成多线程的poi导出，（此时先介绍多线程）这时，我之前做的那种思路的zip压缩流就不能再用了，因为它无法加到多线程中去。我换了一个思路，先在服务器上创建了一个临时文件夹，通过多线程去生成多个的excel文件，将其放到临时文件夹中。这些都完成之后，通过FileInputStream流依次读取临时文件夹中的数据，再通过ZipEntry来进行写入，最后下载zip压缩包，关流。至此，多线程poi导出便写完了。

（springboot创建定时器 只需要在启动类上加@EnableScheduling 然后在需要定时的方法上加 @Scheduled(cron = "0 00 08 ? \* \*")）

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，创建我们的多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）启动用statr方法

设置发送人，接收人，标题，内容，这就实现了我们的邮箱发送

 线程池是怎么理解的？

像平时我在工作中呢也会用到一些线程池，比如批量生成文件，批量给图片添加水印等，这都用到过线程池，我对线程池的理解是这样的，可以提高程序的性能，提高最大效率。

我给您打个比方啊，比如说一个快递站对吧，有100件快递需要送，让一个人负责去送，这一个人我们可以把它当作成一个线程，那100个快递就相当于100个任务，这时候用一个人去送的话，可能会送一天，把这些快递送完，影响工作效率，但是我多用几个人,用十个人一起去送这100个快递，这样节省时间，这十个人就相当于我把线程变成了十个，所以说通过这个线程池，能提高程序的性能，而且当我们频繁的去创建、销毁线程是比较耗费性能的，线程池还有个好处就是在线程池中创建好的线程是可以重复使用的，不用频繁的去创建、销毁线程，这也能提高了性能。

我在项目中用线程池用的是那个 ThreadPoolExcutor来创建线程池的，我具体在用的时候我还记得，它里面有几个核心参数，有那个核心线程大小，还有这个这个最大线程数，然后还有存活时间，还有队列，队列的话我们常用的是无界阻塞队列（LinkedBlockingQueue）它是基于链表的和有界阻塞队列（ArrayBlockingQueue）它是基于数组的，创建有界阻塞队列的话，必须指定大小，而无界阻塞队列可以不指定队列大小，默认为Integer的最大值，还有就是拒绝策略。

我对这几参数信息是这么理解的，我给您打个比方吧，这个核心线程大小是3个，最大线程数为5个，存活时间设置为3秒，用有界阻塞队列我们要指定大小为10个，那它在具体的工作原理呢大概是这样的，在最开始的时候，线程池里面是没有任何线程的，等到有个任务提交以后，线程池里面会创建一个线程，当第二个任务过来的时候，如果第一个线程处理完成之后，那么还是有第一个线程来处理第二个任务，这就是上面说的重复使用，假如说第一个线程还在处理，没有处理完成，那么这时候会创建第二个线程来处理第二个任务，同理第三个任务过来，要创建第三个线程，那么当这三个线程都在处理任务的时候，又有新的任务过来了，那这时候就要把任务放在队列里面，我们用的有界阻塞队列，设置大小最多是10个，所以只能放10个任务，当核心线程也就是那3个线程都在运行，并且队列也都被占满了的时候，那么再有新的任务过来，还能被进行处理，因为我们定义的最大线程数有5个，这时候会创建第四个线程去处理第十四个任务，同理，第十五个任务过来也是能被处理的，当这个时候核心线程都在运行，队列也都被占满，最大线程也达到5了，这时候再来新的任务，就走拒绝策略了，这就是这几个参数的原理，对这些的认识理解。

还有这个存活时间，存活时间我是这样理解的，我们定义的存活时间是3秒，当核心线程处理完之后，队列里的任务也都处理完，所有的都处理完成之后，等待了3秒没有任何新的任务过来，这时候就会把线程池里面的线程从5个减少为3个，只保留了核心线程，我们也可以这样理解，

之前我也举了列子快递站嘛，这个核心线程当成是我们快递站的正式工，队列呢就相当于放快递的小仓库，刚开始三个正式工在干活送快递，这时候仓库满了忙不过来了，刚好三个正式工也都在忙，那我们可以招聘两个临时工来干活送快递，当所有的快递都被送完之后，那么等个一个小时、半天时间，没有新的快递到了，那么就会把那两个临时工给辞掉，保留那3个正式工就行了，这就是我对存活时间的理解。

 Nginx 和 tomcat 负载均衡你了解吗？你是怎么做负载均衡的？单点故障怎么解决？

我在之前的项目中也做过负载均衡，我是这样理解的，最基本的服务器架构就是浏览器到tomcat到数据库，这就是最最基本的服务器架构，并且不靠谱，为什么这么说，一个tomcat的什么，就叫吞吐量吧，是在1500左右，但是如果有10000个请求同时访问，Tomcat运行不过来，导致性能慢。用户体验度差，在如果说这一个tomcat宕机了，那这个项目就停止了？不用了？所以一个tomcat不行，我们就配置多个tomcat来进行高并发的访问，比如说配置了3太tomcat.你不能让这个地区的用户访问这个地址，那个地区的用户访问那个地址，用户肯定也不愿意，那么这时候我们需要用Nginx来请求分发，把用户的请求分发到配置的tomcat上，浏览器和tomcat之间用nginx来连接，然后经过分发策略来分给tomcat,nginx中的分发策略有好几种，我常用的也就是那三种，第一种是轮询，这样来说，比如我们配置了两台tomcat，分为t1和t2，当发送过来一个请求，它会走t1上，第二条请求会走t2上，第三条请求会走t1上，第四条请求就走t2上，同理会均衡的分发到tomcat上，第二种权重策略，通过一个关键词weight,来分发请求，因为公司每个服务器配置不一定完全一样，有的配置高，能力大，有的配置低能力小，这样在使用轮询策略就浪费资源了，所以我们可以这样来使用权重，两台tomcat,t1给weight设置成3，t2给weight设置成1，那么当有四条请求进来时，t1会走3条请求，而t2只走1条请求，这就是权重（weight）策略，第三种 ip\_hash就是将发送请求的客户端的ip地址进行hash化，并将其绑定到对应的tomcat中。这样就能保证以后只要是这个客户端发送的请求都被固定的tomcat所处理，就能解决session漂移的问题。但是还有一个问题就是，传统办法的sesion存储在对应的web服务器上，一旦请求访问的服务器发生改变，就会造成session获取不到要重新登录，这样用户的体验度极差，导致用户可能不会再用这个软件。

我当时解决方案是，用redis存储和管理session，在用户登录时，首先验证用户的登录信息，验证完成后我们需要将用户的登录信息存储到redis中，使用的是redis的string类型。这一块的重要的是key的生成，当时我们考虑到用户同时在线问题，于是使用的登录业务关键字+用户ID+时间戳组成key，这样的用户在另一台电脑登录成功，就会产生一个新的key，并且将老的session信息给删除。在将存储session的同时设置session的失效时间，可以使用redisTemplate.expire() .这里我们公司要求如果用户的session没有失效，关闭浏览器后可以不用登陆访问，于是我们生成的sessionKey放到cookie中，并且设置和session一样的失效时间，对cookie进行在浏览器上的持久化，这样在拦截器或者过滤器中可以获取cookie中的sessionID，通过sessionID从redis中获取对应的session信息，如果有就放过，同时给session延长失效时间，若果没有就返回到登陆页面让用重新登陆。

单点故障我理解的就是，在服务器架构中，只要是单个的服务器都会出现单点故障，比如配的一个nginx,这时候我们解决nginx的单点故障有两种方式，一种是用F5，F5性能极高，好用，但是价格非常昂贵，一般在刚起步的公司上一般是不用的，还有一种方式就是通过LVS来解决，LVS好像是一个虚拟软件吧，这个也不是我负责的，所以也只是了解一点。

**redis相关**

项目中的 业务 （redis 存数据）

Redis是一个以Key value形式储存的，高性能的非关系型数据库，用来做缓存

因为基于内存所以读写的速度非常快

Redis 与其他 key - value 缓存产品有以下三个特点

1.Redis支持事务，同时还支持数据的持久化 可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用 有两种持久化方案 一种基于日志 AOF 一种是快照 rdb 默认是rdb

AOF就是你每次操作redis中的数据时，对redis中的数据进行增删改操作之后都会保存在磁盘中，这样要比rdb安全，但效率没有rdb高

rdb就是每隔一段时间对redis中的数据保存在磁盘中，redis一旦宕机就会失去一片的数据，相对于不安全，但效率高于AOF

2.Redis 拥有丰富的数据类型string，list，set，zset，hash

3.Redis支持数据的备份，即master-slave（主从）模式的数据备份

项目中把访问比较多的数据，或者不经常改变的数据存储到redis中，比如地区，购物车数据

Redis的三大问题 雪崩，击穿，穿透

雪崩

就是请求数据过大 或者就是本地缓存没有了 这时就会去访问mysql 导致mysql宕机

解决方法是 自动降级

穿透

访问一个不存在的key

解决方法是 当受到恶意访问一个不存在的key时，将这个key存储再redis中，并设置一个失效时间

Redis高可用，集群方式 主从复制，哨兵，集群

**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。**

**哨兵它本身也是一个独立的 Redis 节点**，**只不过它不存储数据**，**只支持部分命令**，**它能够自动完成故障发现和故障转移，并通知应用方，从而实现高可用**。

**但是当遇到单机内存，并发和流量等瓶颈的时候，主从模式或Sentinel模式就不能满足需求了**，这个时候需要**对存储的数据进行分片**，**将数据存储到多个Redis实例中**。

分布式锁

加锁实际上就是在redis中，给Key键设置一个值，为避免死锁，并给定一个过期时间。

**购物车是怎么做的？**

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。