# B简历：

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

### 基本信息

|  |  |  |  |
| --- | --- | --- | --- |
| 基本情况 | | | |
| 姓名 | 刘松鹏 | 性别 | 男 |
| 出生日期 | 1996-02-17 | 属相 | 鼠 |
| 现住地 | 洼刘新城 |  |  |
| 大学情况 | | | |
| 学校 | 安徽信息工程学院 | 专业 | 计算机科学与技术（应用技术） |
| 入学时间 | 2013-9 | 毕业时间 | 2017-6 |
| 专业课 | **C语言、数据结构、数据库原理、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 | 安徽省芜湖市弋江区文津西路8号 | 校长 | **不知道** |
| 去学校的方式： | 从安徽芜湖火车站下车，坐10路公交车到马场，转37路机电学院下车 | | |
| 上小学日期： | **2001年9月 5岁** | | |
| 上初中日期： | **2007年9月 11岁** | | |
| 上高中日期： | **2010年9月 14岁** | | |
| 上大学日期： | **2013年9月 17岁** | | |
| 毕业时间： | **2017年 7月 21岁** | | |
| 入学时间： | **2013年9月** | | |
| 毕业时间： | **2017年7月** | | |
| 在校学生： | **5000左右** | | |
| 有教师： | **不太清楚** | | |

|  |  |
| --- | --- |
| 第一家公司 ： | 北京骥贤科技有限公司 |
| **地址：** | 北京昌平区国风美唐综合楼3号楼 |
| **入职时间：** | **2017年8月初** |
| **离职时间：** | **2018年9月** |
| **公司性质：** | **民营** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **李晓晓** |
| **薪资方面：** | **刚刚毕业转正以后工资是8K左右，过完年大概三月份涨到9K左右** |
| **路线：** | 地铁8号线霍营站下车 步行500米 |

|  |  |
| --- | --- |
| **第二家公司情况：** | 北京联合智唯科技有限公司 |
| **公司地址**： | 北京市 朝阳区 中骏·世界城d座 |
| **公司规模：** | **30人左右，我们项目组6人，一个项目经理，2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：** | [技术咨询](https://b2b.11467.com/search/-6280672f54a88be2.htm) , 技术服务 , 计算机系统服务 , 基础软件服务 , 应用软件服务 |
| **去公司路线：** | 地铁6号线东大桥站下车 步行600米 |
| **入职时间：** | **2018年9月** |
| **离职时间：** | **2021年4月** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **梁海超** |
| **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8K，税后大概7K左右，**  **转正9K左右**  **2018上半年涨了1K左右 ，总10K左右。**  **2018年9月跳槽涨了2K左右 总12K**  **2018年12月涨1K总 13K左右**  **2019年9月工作一年涨了1K左右**  **2020年7月工作一年涨了1K左右**  **2020年3月离职时工资为15K** |

**面试常见问题：**

|  |
| --- |
| **你为什么从上家公司离职？** |
| **上家公司待的时间长了，公司比较照顾老员工，让后把我分配去维护之前的项目，让后工作太轻松，环境太安逸，**  **然后就换个公司，想换个新的环境挑战一下自己！** |
| **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，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:**  **抗压自学能力强（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。），**  **喜欢钻研技术，**  **自学能力强** (  **1.从官网上下载需要的文件。**  **2.导入相关的文件到项目中。**  **3.在需要用到的文件中引入具体的js，css。具体在做例子的时候先做静态例子，再和后台结合做出来动态数据的例子。特别在做动态数据例子的时候，我们可以通过结合官网上提供的demo,通过f12观察这个几个问题:**  **<1>js插件怎么向后台发送请求；**  **<2>传递了哪些参数以及参数的含义;**  **<3>后台怎么接受参数;**  **<4>后台接受到参数后怎么处理,才是js组件所需的响应格式;**  **<5>如何将json数据响应给前台;**  **<6>前台如何接受到该json数据；**  **<7>接受到数据之后如何处理，才能将数据展现出来。**  **按照这个思路和流程，我们在学习第三方的js组件时，就能够快速的掌握并融入到项目中。**  **)**  **缺点: 研究技术时候，想把这个技术搞的很清楚，结果才开始的时候影响了项目开发的进度，我也意识到这个问题，所以后来对于我比较感兴趣的技术，我会利用自己的业余时间去钻研它，在工作时以项目任务为主。** |
| **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** |
| **加班在这个行业里面比较普遍，这个没问题但我平时也会去提高自己的工作效率，避免一些不必要的加班** |
| **你能接受出差吗** |
| **能** |
| 你期望薪资多少？你上家工资多少？税后拿到手的有多少？扣了多少钱的税？ |
| 期望薪资 16K 上家工资15K 税后14K左右 扣了1K左右的税 |
| 你的五年规划是啥？ |
| **近期的话我就是想先让自己的技术在得到以进一步的提升，然后对公司的业务也要尽快掌握，然后向项目经理方面发展。** |
| 是统招吗 是是本科吗 是能在学信网查到吗 我当时上的是民办大学，能在民教网查到！ |
| 你们公司还有什么福利？多少薪： 13薪  **车补**  300左右  **项目奖金**  7k-12k，做完项目，收完尾款的时候。  **年终奖**  这个就是1个月的工资，  一般是过完年之后发  **电话补助**  200  **饭/餐补**  30/天  **电脑补**  400 |

**公司内其他部门：**

**技术部（开发部/研发部），行政部，销售部，财务部**

**---------------------------------------------------------------------**

**其他问题：**

**笔试问题：我工作这么多长的时间了，理论上的知识忘得差不多了，你和你们经理联系一下直接面试吧，不行的话那么我就去直接去下一家！**

**-----------------------------------------------------------------------------------------------------------------**

坐稳再开口！！ 主动性！！ 开门见山！！大白话！！三秒钟原则！！学会反问！！

**Hr人事，leader领导，tean团队，pm项目/产品经理，cto首席技术执行官，coo首席运营官，CEO董事长、**

**介绍流程：**

**springboot的运行原理🡪微服务细讲🡪基于token登陆🡪订单**

**🡪支付🡪sql 优化🡪tomcat调优🡪java反射导出excel🡪线程池**

**SpringBoot**

**springBoot是微服务，就是用来做接口开发的。**

**优点：开箱即用**

**约定大于配置**

**内嵌服务器**

**spring boot项目的搭建前提是必须是spring boot项目，pom文件里面声明是springboot项目，jdk的版本号，引入启动器**

**Springboot项目的核心注解是@springbootApplication,它是一个组合注解，它有三个注解组成**

**@configuration spring boot 的配置，相当于xml配置文件**

**@enableConfiguration 打开自动配置的功能**

**@componentScan spring组件的扫描**

**Springcloud**

SpringCloud，**基于SpringBoot**提供了一套微服务解决方案，包括服务注册与发现，配置中心，全链路监控，服务网关，负载均衡，熔断器等组件，

**Eureka 服务注册与发现**

**它有两大组件**

**EurekaServer服务器各个节点启动后会，会再eureka中进行注册**

**EurekaClient 客户端用户与服务端的交互，客户端同时也具备一个内置的轮询的负载均衡器，在应用程序启动的时候，会向eureka服务器发送心跳连接（默认是30秒），如果eurekaServer在这个周期内没有接收到某个服务的心跳，eurekaServer将会将这个节点从服务列表中移除（默认是90秒）**

**Eureka工作原理，服务启动向eureka注册服务，注册中心将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提供者地址，将获取到的地址缓存到本地，下次调用的时候直接从本地获取，当服务注册中心检测到服务提供者宕机，网络不同等服务不可用时，则在服务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的状态，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。

默认情况下，如果Eureka Server在一定时间内没有接收到某个微服务实例的心跳，Eureka Server将会注销该实例（默认90秒）。

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。

Ribbon 负载均衡工具，

轮询，随机，过滤，响应时间，复合

restTemplate.getForObject("http://EUREKA-CLIENT1/client1/test",String.class);

**这样每次都调用 RestTemplate 的 API 是否太麻烦，而且每次都要写URL，设置返回值类型，如果是一个需要传大量参数的请求，还要构造一个复杂的请求，接着发送请求过去，最后对返回的响应结果再写一大堆代码来处理，我能不能像调用原来代码一样进行各个服务间的调用呢？答案是确定的有，Netflix已经为我们提供了一个框架Feign。Spring Cloud Feign在Netflix的Feign的基础上进行了封装，简化了咱们的使用。**

Feign

**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**

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud**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。**

**服务雪崩解决方案**

**熔断模式**

**隔离模式**

**限流模式**

**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**

**服务熔断**

**服务降级**

**服务容错**

**线程和信号隔离**

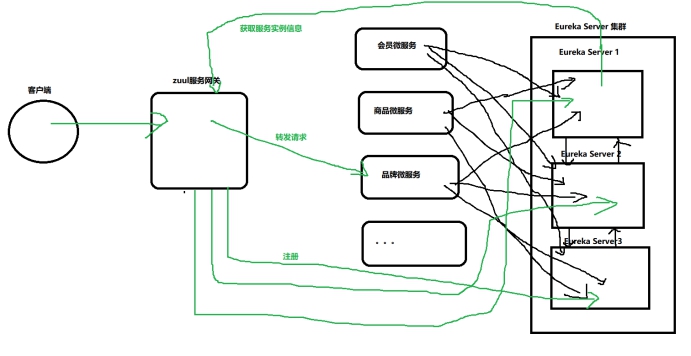
**请求缓存**

**请求合并**

**Zuul**

**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。

Zuul组件包含了对**请求的路由**和**过滤**两个最主要的功能：其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础。而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础。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**。



**Zuul Filter简介**

filter type

|  |
| --- |
| **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。  **l ERROR**：在其他阶段发生错误时执行该过滤器。 |

**Zuul提供了自定义过滤器的功能实现起来也十分简单，只需要编写一个类去实现zuul提供的接口。**

|  |
| --- |
| public abstract ZuulFilter implements IZuulFilter{  abstract public String filterType();  abstract public int filterOrder();  boolean shouldFilter();// 来自IZuulFilter  Object run() throws ZuulException;// IZuulFilter  } |

ZuulFilter是过滤器的顶级父类。在这里我们看一下其中定义的4个最重要的方法

**shouldFilter**:返回一个 Boolean 值，判断该过滤器是否需要执行。返回true执行，返回false不执行。

**run**:过滤器的具体业务逻辑。

**filterType**:返回字符串，代表过滤器的类型。包含以下4种：

**pre:请求在被路由之前执行**

routing:在路由请求时调用微服务的时候执行

post:在routing和errror过滤器之后调用

error:处理请求时发生错误调用

**l filterOrder**:通过返回的int值来定义过滤器的执行顺序，数字越小优先级越高。

## 订单模块

**当时做这个模块的时候也是出现了很多问题，像表设计，接口，接口幂等性，超卖的问题，还有订单也是用户登录才能进行的操作**

首先说下高并发情况下产生的超卖问题吧：超卖是这样产生的，同时下订单时先对商品库存进行查询、满足条件才能下单，但在高并发的情况下，可能好几个查到了同样的结果，同时下单，将数据库中的库存买到了负数。为了杜绝这种情况呢，我做了以下的处理：【超卖】

首先从redis中取出用户的购物车数据，如果为空，则响应给客户端一个用户购物车为空的提示信息，如果不为空则取出用户对应的购物车的json数据，将其转换为java对象。

然后通过商品的id作为redis的key，商品的信息作为redis的值

首先通过年月日时分的时间戳加上雪花算法生成一个订单的唯一标识，然后从购物车对应的java对象中取出购物车中的商品列表，这个商品列表其实对应的就是订单明细，循环遍历商品列表，减库存，如果减库存成功，则将该商品插入订单明细表，如果不成功就不需要插入订单明细表了，而且还需要从购物车中剔除该商品，更新购物车的总价格和总件数，并将该库存不足的商品加入一个列表，用于响应给前端进行提示。

将更新后的购物车信息中的总价格和商品总件数插入到订单表中，并设置创建时间为当前时间，当前订单状态为未支付。

最后插入支付日志表，这个表记录的是支付时候的相关信息，比如outTradeNo,payMoney等，将支付日志表中的订单id设置为刚才插入的订单id,将支付状态设置为 未支付，并将该支付日志插入到redis缓存中，方便后续在支付的时候直接从缓存中取出来进行使用。紧接着就清空该用户对应的redis中的购物车数据。因为当时购物车数据是以hash的方式存储的，所以通过hdel进行删除就行了。

在这个过程中要注意，如果整个购物车中的商品都库存不足，则直接提示前端，下单失败，库存不足；也就不需要再插入订单表和支付日志表了。

我当时采用的是基于数据库乐观锁的方式进行订单超卖的处理。核心思路就是在update语句中加入库存量是否大于或等于购买量的where条件。比如原来更新数据库中的商品库存我这样写，先查询下该商品对应的库存量，如果库存量大于等于购买量则执行update 商品表 set 库存=库存-购买量 where 商品id=购物中的商品id;否则提示库存不足。

表面看上去没有任何问题，但在遇到多线程高并发的时候，就会出现多个线程同时查询商品对应的库存量，都符合条件，所以多个线程都会依次进行update语句，最后就造成商品可能为负数，出现商品超卖的情况。

我是通过将update语句改造为 update 商品表 set 库存=库存-购买量 where 商品id=购物中的商品id and 库存量>=购买量，然后根据update的返回值，如果返回值为0，则证明没有执行该sql语句，证明库存已经不足了。这样即便在多线程的时候也不会出现超卖问题。

**【幂等性/重复下单】**

订单接口要考虑重复下单的问题，超卖的问题，以及把redis中存储的购物车数据取出来，插入订单明细表，订单表以及支付日志表。

关于重复下单这块，说白了就是要考虑接口的幂等性。幂等性我是这样理解的，就是对接口调用一次和调用多次，产生的结果都是一样的。像查询方法，就天生具有幂等性的特点。

但下订单属于添加操作，对于添加操作本身不具备幂等性的特点，所以需要我们特别处理下，如果不处理就会出现因为网络抖动或者服务器重发请求，造成虽然用户只操作了一次，但是可能最终发送了多次添加订单的请求，这样就会让用户感觉到很奇怪，明明我就下了一个订单，怎么会出现两个一模一样的订单呢。

对于接口的幂等性我是基于redis来操作的，使用的是redis的getandset

并设置一个过期时间，这样就解决了重复下单的问题。

对于上边所说的，都是一些业务逻辑，但是在项目中，我们考虑到性能方面的问题，做出了许多的优化：

比如说我们的sql优化：

**MYSQL优化主要分为以下四大方面：**

**避免索引失效**

**1.不在索引列上做任何操作（计算、函数、（自动or手动）类型转换）**

**2.尽量使用覆盖索引（只访问索引的查询（索引列和查询列一致）），减少select \***

**3. mysql 在使用不等于（!= 或者 <>）的时候无法使用索引会导致扫描全表（使用 != 和 <> 的字段索引失效( != 针对数值类型。 <> 针对字符类型）**

**4. is null,is not null也无法使用索引**

**5. like以通配符开头('%abc...')mysql索引失效会变成全表扫描的操作**

**6. 字符串不加单引号索引失效**

**7. 少用or,用它来连接时会索引失效**

**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实数据。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

(1) 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

**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；

**我们平时所说的索引，如果没有特别指明，都是指B树(多路搜索树，并不一定是二叉树)结构组织的索引**。其中聚集索引，次要索引，覆盖索引，复合索引，前缀索引，唯一索引默认都是使用B+树索引，统称索引。当然,除了B+树这种类型的索引之外，还有哈希索引(Hash Index)等。

**1. 什么情况下应该建索引**

**l 主键自动建立唯一索引；**

**l 频繁作为查询条件的字段应该创建索引**

**l 查询中与其它表关联的字段，外键关系建立索引**

**l 单键/组合索引的选择问题， 组合索引性价比更高**

**l 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**

**l 查询中统计或者分组字段**

**什么情况下不应该建索引**

**表记录太少**

**经常增删改的表或者字段**

**Where 条件里用不到的字段不创建索引**

**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%)**

可以通过Explain来看性能分析

**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

**2. MySQL锁机制**

**从数据库层面来说有悲观锁，乐观锁。**包括有表锁和行锁。

悲观锁,

每次去查询数据的时候都认为别人会修改，

所以每次在查询数据的时候都会上锁，

这样别人想拿这个数据就会阻塞直到它拿到锁。

传统的关系型数据库里边就用到了这种锁机制，

比如通过select ....for update进行数据锁定。

乐观锁,

每次去查询数据的时候都认为别人不会修改，

所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有

去更新这个数据，可以使用版本号，时间戳等机制。

乐观锁适用于多读的应用类型，这样可以提高吞吐量。

**从java代码层面来说有synchronized同步锁**

和synchronized同步块，它们都是为了

解决多线程访问时，线程安全的问题，

像java里面的hashtable和vector都是

用的synchronized来保证线程安全。

除此之外，还有读写锁

读写锁表示有两个锁，一个是读操作相关的锁，

也称为共享锁；另一个是写操作相关的锁，也叫排它锁。

多个读锁之间不互斥，读锁与写锁互斥，

写锁与写锁互斥。

也就是说多个Thread可以同时进行读取操作，

但是同一时刻只允许一个Thread进行写入操作。

通过lock()方法加锁，通过unlock()方法解锁。

除此之外还有分布式锁，普通的锁是为了解决多线程访问

产生的安全问题，分布式锁是为了解决多进程访问产生的

安全问题。我们在项目中使用Redisson来完成分布式锁。

**3. MySQL主从复制读写分离**

**我们在项目中使用了mysql的主从配置，当时是一个主数据库和两个从数据库。**

**这样做的主要目的是考虑到对数据的备份以及提高程序访问的性能。**

**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**

主从之间复制数据的流程是这样的，主数据库会将每次的操作写入到二进制日志中，从数据库通过一个IO线程读取主数据库上的二进制日志将其写入到从数据库的中继日志中，之后再通过SQL线程读取中继日志中的信息，在从数据库中重新执行一遍，这样就完成了数据的复制。

具体在配置的时候是这样操作的，首先在主数据库中会开启二进制日志，设置serverid保证唯一性；其次创建用户并授予权限；最后查看主数据库中二进制文件的文件名以及在文件中的具体位置；在从数据库中同样设置serverid保证唯一性，之后执行同步的sql语句，在这个过程中就用到了主数据库所创建的用户以及主数据库中二进制的文件名和文件中的位置；最后通过start slave启动同步进程，如果查看从数据库状态中IO和SQL都为Yes，则证明配置成功。

考虑到主从复制之间的数据延迟，我们后台的读写操作都使用的是主数据库，而前台项目和接口项目分别使用从数据库读取数据，从而进行读写分离，提高程序的访问性能。

**4. MySQL和分库分表**

**分库分表 （会就说，不会不说）**

顾名思义，即把存于一个库的数据分散到多个库中，把存于一个表的数据分散到多个表中。分库分表 的主要原因就是随着项目的不断运行，数据库中的表以及表中的数据量都会越来越多，就会出现一些问题：我们数据库的存储资源是有限的，其负载能力也是有限的，大量的数据积累肯定会导致其处理数据的能力下降,数据量越多，那么对数据的增删改查等操作的开销也会越来越大，

分库分表的方式有

垂直分表：将一个表按照字段分成多表，每个表存储其中一部分字段。

水平分表：是在同一个数据库内，把同一个表的数据按一定规则拆到多个表中，字段不变。

垂直分库：是指按照业务将表进行分类，然后分到不同的数据库上，每个库可以放在不同的服务器上，它的核心理念是专库专用

水平分库：是把同一个表的数据按一定规则拆到不同的数据库中，每个库可以放在不同的服务器上

**我们以前公司在做的一个电商管理系统的时候用到了定时任务，因为我们当时在用户注册信息的时候交用户填写了生日字段我们要在用户生日的当天给用户发送一个祝福或者给用户通知一些福利**

**当时用的是ssm框架，刚开始写的是一个后台管理系统，用户量数据并不是很大，所以这样做也是很容易的，后来就是做了个分布式框架跟这个相近的功能，做的时候也是一样的思路，测试的时候就出现问题了，系统会给用户发送重复的信息，这时候才意识到，是思路出现了问题，为了解决这个问题，当时在执行的任务中加了一个锁，但是这样还是不行的，系统还是会发送重复的数据，于是就借用了redis中的一个getAndSet方法解决重复发送的问题，并设置了一个过期时间，之后判断getAndSet中的值是否为空，为空的话直接就返回，不执行以下的一系列代码，之后又结合了线程池跟线程的技术提高我们的性能，在线程中调用我们发送通知的逻辑就可以了**

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

**我做过一个io类型的后台管理系统 用的是一个ssm框架。负责的一个功能块是用户管理。其中呢 我印象挺深刻的是做过一个这样的需求：根据用户的生日去给用户发送一条生日邮件。刚开始我直接把数据从mysql库里面直接查出来啦，因为是一个管理系统用户量没那么大 所以就直接查出来。把数据放到redis里面，从redis里面把数据查询出来，在求出当前时间的毫秒值 减去 当年的毫秒值 ，同理 用户生日的毫秒值 减去 生日那年的毫秒值，去比较得出来的毫秒值是否相等，如果相等就把这条数据的邮箱放到一个list的集合中，在把集合放到redis里面。因为每天都要发送 我就写啦一个 Timer（泰麽）TimerTask（泰麽塔式柯）他们都是java中提供定时调用的两个类  Timer是一种定时器工具，用来在一个后台线程计划执行指定任务，而TimerTask一个抽象类，它的子类代表一个可以被Timer计划的任务。在定时器中设置每天凌晨进行一次调用。一开始我在网上封装了一个邮箱的工具类，工具类是可以调用一次去发送多个邮箱，但存在一个问题 就是如果用户已经把邮箱注销啦，那么这样写程序就会报错，会报一个邮箱不存在。那么这样的写法就不成立啦  最终呢我创建一个线程池 设置线程的数量，依次去调用邮箱的封装类，我已经把邮箱的一次发送多个变成单个发送，这样也提高了程序的性能。最终在测试的时候我发现一个漏洞，就是说如果计算毫秒值的情况下，那么会有一定的差别，如果在去细致化的去计算毫秒值的话 那代码的复杂性就又提高，最后我就直接在sql数据库查询数据的时候，写啦一个MONTH（慢）函数，我做过一个io类型的后台管理系统 用的是一个ssm框架。负责的一个功能块是用户管理。其中呢 我印象挺深刻的是做过一个这样的需求：根据用户的生日去给用户发送一条生日邮件。刚开始我直接把数据从mysql库里面直接查出来啦，因为是一个管理系统用户量没那么大 所以就直接查出来。把数据放到redis里面，从redis里面把数据查询出来，在求出当前时间的毫秒值 减去 当年的毫秒值 ，同理 用户生日的毫秒值 减去 生日那年的毫秒值，去比较得出来的毫秒值是否相等，如果相等就把这条数据的邮箱放到一个list的集合中，在把集合放到redis里面。因为每天都要发送 我就写啦一个 Timer（泰麽）TimerTask（泰麽塔式柯）他们都是java中提供定时调用的两个类  Timer是一种定时器工具，用来在一个后台线程计划执行指定任务，而TimerTask一个抽象类，它的子类代表一个可以被Timer计划的任务。在定时器中设置每天凌晨进行一次调用。一开始我在网上封装了一个邮箱的工具类，工具类是可以调用一次去发送多个邮箱，但存在一个问题 就是如果用户已经把邮箱注销啦，那么这样写程序就会报错，会报一个邮箱不存在。那么这样的写法就不成立啦  最终呢我创建一个线程池 设置线程的数量，依次去调用邮箱的封装类，我已经把邮箱的一次发送多个变成单个发送，这样也提高了程序的性能。最终在测试的时候我发现一个漏洞，就是说如果计算毫秒值的情况下，那么会有一定的差别，如果在去细致化的去计算毫秒值的话 那代码的复杂性就又提高，最后我就直接在sql数据库查询数据的时候，写啦一个MONTH（慢）函数，**

**我们当时是分布式框架spring+springboot+springCloud**

**定时器我们用的是springboot的**@Scheduled 定时器

我们只需要创建一个类加上三个注解@Component 交给ioc进行管理

@Configuration声明这是一个文件

@EnableScheduling **//启用调度器**

**然后规定一个定时时间定时扫描就行了**

**当时我们在做项目的时候结合了业务场景实际情况考虑到如果注册用户过多的话，如果每次发送请求一到达的话就创建一个新线程，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，甚至可能比处理实际请求的时间花费的要多的多，这样就可能会使，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

**用多线程的方式来提高系统性能**

1. java中实现线程的方式

在java中实现线程有两种方式：继承Thread类，实现Runnable接口，一个java main程序默认会开启两个线程一个是主线程，一个垃圾回收线程。

2. 为啥会产生线程安全问题?

当多个线程操作同一个资源才会出现线程安全问题

多个线程访问同一个资源，导致结果和期望值不同，我们就说它是 非线程安全的（线程不安 全），反之我们就说它是 线程安全的。

多个线程访问同一个资源如果每次运行结果 和单线程运行的结果是一样的，就是线程安全的。

若有多个线程同时执行写操作，一般都需要考虑线程同步，否则的话就可能影响线程安全。

3. 线程的状态

(1)新建状态(New)：新创建了一个线程对象。

(2)就绪状态(Runnable)：线程对象创建后，其他线程调用了该对象的start()方法。该状态 的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

(3)运行状态(Running)：就绪状态的线程获取了CPU，执行程序代码。

(4)阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直 到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

a、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。

b、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则 JVM会把该线程放入锁池中。

c、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会 把该线程置为阻塞状态。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

## 线程池概念

## 对于高并发、多线程的问题，我们还通过线程池的方式提高程序的性能。

接下来咱说说这个线程池吧，我们也都知道如果没有线程池，则就会频繁的去创建以及销毁线程，这就造成了我们的性能大量的损耗，所以我们在项目中也会通过线程池的方式提高程序的性能。首先我们也都知道了线程池可以避免频繁的创建和销毁线程所造成的性能损耗，原理就和数据库连接池差不多，说白了就是项目启动的时候在线程池中就已经创建好了指定数量的线程，需要的时候直接去用，用完后再放回线程池供其他程序进行使用。

比如我要将数据库中的图片加水印或者在硬盘上批量生成文件，这时候都可以用线程池。就像要洗100个碗，你可以让一个人去干，这一个人就像是一个线程，你也可以让10个人一块去干，这就是线程池中的多线程。相比而言多线程执行的时间更短，效率更高。在项目中我们是用**ThreadPoolExecutor**来创建线程池的，它里面有几个核心的参数信息，线程池的**核心大小**，**队列**以及**线程池的最大值**。线程池的工作原理是这样的，默认情况下，创建线程池之后，线程池中是没有线程的，需要提交任务之后才会创建线程。如果当前线程池中的线程数目小于核心线程数，则每来一个任务，就会创建一个线程去执行这个任务；如果当前线程池中的线程数目>=核心线程数，则每来一个任务，会尝试将其添加到队列中，若添加成功，则该任务会等待空闲线程将其取出去执行；当队列已满，添加失败，就会尝试创建新的线程去执行这个任务；这时候创建新的线程是根据线程池时的最大线程数为依据的，如果当前线程池中的线程数目达到最大线程数，则会采取任务**拒绝策略**进行处理；说的简单点就是先把核心线程数给占满了，不够用就开始往队列里面放，如果队列也占满了，就往创建的最大线程池里面放，如果也占满了就可以根据策略进行拒绝处理。

如果线程池中的线程数量大于 **corePoolSize**【核心线程】时，某线程的空闲时间超过**keepAliveTime**【存活时间】，线程将被终止，直至线程池中的线程数目不大于corePoolSize【核心线程】。

线程池中的队列，一般常用的有**ArrayBlockingQueue**【有界阻塞队列】：基于数组的队列，创建时必须指定大小；还有**LinkedBlockingQueue**【无界阻塞队列】：基于链表的先进先出队列，如果创建时没有指定此队列大小，则默认为Integer.MAX\_VALUE；

被动说：一般需要根据我们会根据任务的类型来配置**线程池大小**，如果是CPU密集型任务，就需要尽量压榨CPU，可以设为 CPU个数\*CPU核数+1，如果是IO密集型任务，参考值可以设置为2\*CPU个数\*CPU核数。

### Jvm虚拟机调优

我们的linux服务器,安装centos6.5, 64位的操作系统和64位的软件，

配置了8个cpu,64G内存。

我们在jvm优化的时候是这样做的

首先JVM将内存划分为：

年轻代

年老代

永久代（方法区）

其中年轻代和年老代属于堆内存，永久代不属于堆内存，有虚拟机直接分配。

年轻代：年轻代用来存放JVM刚分配的Java对象

年老代：年轻代中经过垃圾回收没有回收掉的对象将被放到年老代

永久代：永久代存放Class类、Method方法元信息，其大小跟项目的规模、类、方法的量有关，一般设置为256M就足够，预留30%的空间。通过

-XX:PermSize=128M -XX:MaxPermSize=128M 来设置永久代（方法区）的大小。

jvm的垃圾回收算法有[GC]

串行算法（单线程）

并行算法

并发算法

吞吐量优先的并行收集器

响应时间优先的并发收集

我们的项目设置的是响应时间优先的并发收集器，

将堆大小通过 -Xms -Xmx设置为5G,将年轻代通过 -Xmn 设置为3g,

设置年老代为并发收集，当时设置的是运行8次GC以后对 内存空间进行压缩、整理。打开对年老代的压缩，可以消除碎片。

被动说：

在linux服务器上部署项目的时候，通常会使用的一些命令

1. 通过netstat -anp | grep 端口号 可以查看端口对应的进程号，通过netstat -anp | grep 进程号 可以查看进程对应的端口号，之后可以结合ps -ef | grep 进程号， 来查看进程信息。

2.通过wget下载相关的文件

3.通过yum在线安装软件，比如我在linux上

安装mysql时

4.通过ps -ef | grep 进程名来查看进程，

比如我通过 ps -ef | grep java来查看

tomcat是否启动,通过 ps -ef | grep mysql

来查看mysql是否启动。

5.通过service 服务名 start来启动服务，如

通过service mysqld start来启动mysql

6.通过vi /etc/sysconfig/iptables来编辑文件

开启防火墙的端口号，最后通过service iptables restart

来重启防火墙使其配置生效

7.通过 kill -9 进程号来强制杀死进程

8.通过 cat /proc/cpuinfo查看cpu

9.通过 cat /proc/meminfo查看内存

10.通过 df -h 查看硬盘信息

1. 通过 top命令,可以查看正在运行的进程和系统负载信息

=============（补充点/被动说）=============

日志管理模块

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。

之前做其他项目的时候，日志处理模块通常都是在控制层结合log4j进行日志的 控制台打印以及日志文件的存储，除此之外还会把日志信息插入到mysql数据库中存储起来，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，以后维护起来也特别麻烦。

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的处理上。

具体在做的时候我是这样写的

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。在这个普通的java类中我会自定义一个横切逻辑，说白了就是一个普通的方法，但这个方法中需要特别注意几点，因为当时在项目中我使用的是环绕通知，

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，之所以采用value，是因为value这个字段有特殊的含义，它可以在使用自定义注解给日志信息赋值的时候省略不写，用起来更加方便之后就可以在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，横切的controller层

在方法上添加参数joint point

使用joinpoint.signature来 获取方法的签名然后再根据方法的签名获取到所有的方法，

signature .getMethod 获取方法对象

之后可以通过 getDeclaredFields()获取类的所有属性，getDeclaredField(属性名)获取类中指定 名字的 属性,getDeclaredMethods()获取类的所有方法，获取属性后还可以通过getType()获取属性的类型，通过getName()获取属性名。

这时候自定义注解就有作用了，一般来说，反射和自定义注解是一起使用的

什么是反射

method.getAnnotion 获取方法上面有没有这个注解

join point.getArgs来获取参数

通过request.getParmter来获取token,token里面我们是将用户信息加密的，再解密，将用户的登录信息拿出来

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。

**Redis是一个key-value类型的非关系型存储系统。它支持丰富的存储类型最为常用的数据类型主要由五种string(字符串)、 list(链表)、set(集合)和zset(有序集合) hash。**

**其中**

**string(地区（三级联动），品牌,历史订单)**

**我们在单体架构项目的用户展示地区模块存储数据就是用的这个redis String类型的数据,因为地区数据量不是太大，redis是基于内存的，使用数据库读的速度没有redis快，用户的体验度将会不高，所以我们这一点是用redis做的**

（1）**2.分布式框架中使用redis完成分布式session存储**

分布式框架的核心作用就是来完成负载均衡的，因此应用的服务器至少在两台和以上,

当时我们用的nginx来做我们web服务的反向代理完成负责均衡。这样的话传统办法的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。

我们公司当时解决方案是，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

（2）计数器功能，可以按照自己设置的步长（就是每次加1还是加2）。可以用于

在线人数统计，在新闻网站中可以用户点赞，投票,微博数，粉丝数。

（3）接口限流，就是在一分钟内只允许同一IP的客户端访问10次。

我们的分布式框架中会发布一些接口，这些接口为了防止被恶意访问，我们不仅要

做接口安全，还要做接口限流这块我们是结合redis进行实现的。

redis的getAndSet 并设置过期时间

我们当时是对客户端的访问进行限流，允许客户端每分钟访问该接口十次，超过十次就会提示该用户操作频繁，一分钟该限制在redis中自动失效，

我们当时在springcloub框架中使用rateLimit组件进行限流，具体操作就是在路由网关中进行配置，首先配置

如果我们不使用rateLimite，可以直接使用redis配合过滤器实现接口限流，具体实现

就是获取客户端访问的IP地址和访问的服务名组成key，然后存入到redis中，并从配置文件

中获取失效时间。并且通过redis的计数器功能给这个IP地址设置访问次数，如果达到次数

就开始限流。

**list(链表)、**

2.list：**他的操作分别是push/pop.**简单点说就是上车需要排队，后来的push在后面，当然老人孕妇可以插个队push的队列的前面，然后大家通过pop一个个上车。

（1）.**多用于消息队列应用的场景，比如秒杀，日志收集。**

**（2）.取最新N个数据的操作（比如微博和新闻网站会缓存前五千条品论信息，超**

**过五千条才会去查数据库）**

**set(集合)**

**4.set:概念和java中的set差不多，无序集合，元素唯一性，不可重**

**复。**

在微博应用中，可以将一个用户所有的关注人存在一个集合中，将其所有粉丝存在

一个集合。Redis还为集合提供了求交集、并集、差集等操作，可以非常方便的实现如共同关注、共同喜好、二度好友等功能，对上面的所有集合操作，你还可以使用不同的命令选择将结果返回给客户端还是存集到一个新的集合中

**hash。**

1.存储一些部分变更的数据，如用户信息中只修改了生日只需要把该用户对应的生

日字段修改。

2.他的存储模式为set(key,key,value)

这种模式可以用来存储购物车，一个用户只有一个购物车，每个商品单独存

储

**我们在做电商项目的时候使用hash做的是购物车数据**

**因为购物车的数据经常存和删除，这样直接作用于数据库的时候，**

**我们就是把用户名加一个唯一标识符当做表名**

**filed放的是sku和非sku的id,value放的就是数据**

**和zset(有序集合)**

**5.zset:是有序集合，元素唯一性，不可重复。**

应用场景：粉丝充值贡献榜。

游戏排行榜（LOL中的各个大区的玩家段位排名，查询自己在该大区的排

名）。

**实现缓存数据分页**

**Redis数据库具有以下特征：**

**多样的数据模型**

**持久化**

**Redis通常将数据存储于内存中，或被配置为使用虚拟内存。**

**Redis有一个很重要的特点就是它可以实现持久化数据**

**，通过两种方式可以实现数据持久化：**

**一种是RDB快照的方式，将内存中的数据不断写入磁盘；**

RDB持久化是指在指定的时间间隔内将内存中当前进程中的数据生成快照保存到

硬盘(因此也称作快照持久化)，用二进制压缩存储，保存的文件后缀是rdb；当

Redis重新启动时，可以读取快照文件恢复数据**性能较高，如果说redis 出现故障的情况下，redis还没来得及持久化数据，就可能会引起一定程度的数据丢失**

**或使用类似MySQL的AOF日志方式，记录每次更新的日志。**

RDB持久化是将进程数据写入文件，而AOF持久化，则是将Redis执行的每次

写、删除命令记录到单独的日志文件中，查询操作不会记录； 当Redis重启

时再次执行AOF文件中的命令来恢复数据。所以说他是不容易丢失数据的，因此也成为主流的持久胡方案

**将Redis作为缓存服务器时使用，也会造成很多问题，比如 缓存穿透：（不命中就是查询不到）**

**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

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，

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。（会使代码难度加大）

**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。大多数系统设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。

一般来说，在企业中实际使用Redis的时候肯定不会只有一台Redis服务器的，因为只有一台Redis服务器的话会存在以下两个问题：

|  |
| --- |
| l 从结构上，单个Redis服务器会发生单点故障，并且一台服务器需要处理所有的请求负载，压力较大。  l 从容量上看，单个Redis服务器内存容量有限，就算一台Redis服务器内容容量为256G，也不能所有内容用作Redis存储内存，一般来说，单台Redis最大使用内存不应该超过20G。 |

那么我们如何去解决上述的两个问题呢？我们可以通过**部署Redis集群来解决**。Redis集群部署有三种模式：**主从复制模式、哨兵模式、集群模式**。

**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。**

**1.1 主从复制的作用**

**(1) 数据冗余**：主从复制实现了数据的热备份，是持久化之外的一种数据冗余方式。

**(2) 故障恢复**：当主节点出现问题时，可以由从节点提供服务，实现快速的故障恢复；实际上是一种服务的冗余。

**(3) 负载均衡**：在主从复制的基础上，配合读写分离，可以由主节点提供写服务，由从节点提供读服务（即写Redis数据时应用连接主节点，读Redis数据时应用连接从节点），分担服务器负载；尤其是在写少读多的场景下，通过多个从节点分担读负载，可以大大提高Redis服务器的并发量。

**(4) 读写分离**：可以用于实现读写分离，主库写、从库读，读写分离不仅可以提高服务器的负载能力，同时可根据需求的变化，改变从库的数量。

**(5) 高可用基石**：除了上述作用以外，主从复制还是哨兵和集群能够实施的基础，因此**说主从复制是Redis高可用的基础**。

**1.1 主从复制模式存在的问题**

如果现在主节点挂掉了怎么办？**一旦主节点出现故障无法提供服务，这时候需要人工介入手工将从节点调整为主节点，同时应用端还需要修改新的主节点地址，这种故障转移的方式对于很多应用场景是不能容忍的。**正是由于这个问题，Redis提供了Sentinel(哨兵)架构来解决这个问题。

**1.1 什么是哨兵**

Redis Sentinel是一个分布式的架构，**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**。

Redis Sentinel 包含若干个 Sentinel 节点和 Redis 数据节点，每个 Sentinel 节点会对数据节点和其他 Sentinel 节点进行监控，当发现节点异常时，会对节点做下线标识，如果被标识的是主节点，此时会与其他Sentinel 节点进行协商，当大多数Sentinel 节点都认为主节点不可达时候，会发起选举，选出一个 Sentinel 节点来完成自动故障转移的工作，同时会将这个变化通知给 Redis 的应用方。这个过程是完全自动化的，无需人工干预。

**1.2 哨兵的功能**

**l 监控**（Monitoring）：哨兵会不断地检查主节点和从节点是否运作正常。

**l 自动故障转移**（Automatic failover）：当主节点不能正常工作时，哨兵会开始自动故障转移操作，它会将失效主节点的其中一个从节点升级为新的主节点，并让其他从节点改为复制新的主节点。

l 配置提供者（Configuration provider）：客户端在初始化时，通过连接哨兵来获得当前Redis服务的主节点地址。

l 通知（Notification）：哨兵可以将故障转移的结果发送给客户端。

**1.1 哨兵模式总结**

哨兵系统的搭建过程，有几点需要注意：

(1) 哨兵系统中的主从节点，与普通的主从节点并没有什么区别，故障发现和转移是由哨兵来控制和完成的。

(2) 哨兵节点本质上是redis节点，只不过不能存储数据。

(3) 每个哨兵节点，只需要配置监控主节点，便可以自动发现其他的哨兵节点和从节点。

(4) 在哨兵节点启动和故障转移阶段，各个节点的配置文件会被重写(config rewrite)。

(5) 刚才的例子一个哨兵只监控了一个主节点；实际上，一个哨兵可以监控多个主节点，通过配置多条sentinel monitor即可实现。

**1.2 哨兵模式的基本原理**

**l 定时任务**：每个哨兵节点维护了3个定时任务。定时任务的功能分别如下：通过向主从节点发送info命令获取最新的主从结构；通过发布订阅功能获取其他哨兵节点的信息；通过向其他节点发送ping命令进行心跳检测，判断是否下线。

**l 主观下线**：在心跳检测的定时任务中，如果其他节点超过一定时间没有回复，哨兵节点就会将其进行主观下线。顾名思义，主观下线的意思是**一个哨兵节点“主观地”判断下线**；与主观下线相对应的是客观下线。

**l 客观下线**：哨兵节点在对主节点进行主观下线后，会通过sentinel is-master-down-by-addr命令询问其他哨兵节点该主节点的状态；如果判断主节点下线的哨兵数量达到一定数值，则对该主节点进行客观下线。

需要特别注意的是，客观下线是主节点才有的概念；如果从节点和哨兵节点发生故障，被哨兵主观下线后，不会再有后续的客观下线和故障转移操作。

**l 选举领导者哨兵节点**：当主节点被判断客观下线以后，各个哨兵节点会进行协商，选举出一个领导者哨兵节点，并由该领导者节点对其进行故障转移操作。

监视该主节点的所有哨兵都有可能被选为领导者，选举使用的算法是Raft算法；Raft算法的基本思路是先到先得：即在一轮选举中，哨兵A向B发送成为领导者的申请，如果B没有同意过其他哨兵，则会同意A成为领导者。选举的具体过程这里不做详细描述，一般来说，哨兵选择的过程很快，谁先完成客观下线，一般就能成为领导者。

**l 故障转移**：选举出的领导者哨兵，开始进行故障转移操作，该操作大体可以分为3个步骤：

在从节点中选择新的主节点：选择的原则是，首先过滤掉不健康的从节点；然后选择优先级最高的从节点(由slave-priority指定)；如果优先级无法区分，则选择复制偏移量最大的从节点；如果仍无法区分，则选择runid最小的从节点。

更新主从状态：通过**slaveof no one**命令，让选出来的从节点成为主节点；并通过slaveof命令让其他节点成为其从节点。

将已经下线的主节点(即6379)设置为新的主节点的从节点，当6379重新上线后，它会成为新的主节点的从节点。

**2. 集群模式(Redis Cluster)**

Sentinel模式基本可以满足一般生产的需求，具备高可用性。**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**。

**2.1 什么是Redis集群？**

Redis Cluster是Redis官方提供的分布式存储解决方案，是在3.0版本后推出的。

Redis集群采用**无中心结构**，集群由**多个节点(Node)组成**，Redis的数据分布在这些节点中。每个节点都保存着数据和整个集群的状态，每个节点都和其它所有节点连接。

集群中的节点分为主节点和从节点：只有主节点负责读写请求和集群信息的维护；从节点只进行主节点数据和状态信息的复制。

**1.1 Redis集群的作用**

**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。

**1.2 Redis集群特点**

1. 所有的Redis节点彼此互联(PING-PONG机制)，内部使用二进制协议优化传输速度和带宽。

2. 节点的fail是通过集群中超过半数的节点检测失效时才生效。

3. 客户端与Redis节点直连，不需要中间proxy层。客户端不需要连接集群所有节点，连接集群中任何一个可用节点即可。

4. Redis集群把所有的物理节点映射到[0-16383]slot上（不一定是平均分配）,集群负责维护node<->slot<->value。

5. Redis集群预分将所有的存储空间划分为好16384个哈希槽，集群中的每个节点负责处理一部分哈希槽。当需要在 Redis 集群中放置一个 key-value 时，根据 CRC16(key) mod 16384的值，决定将一个key放到哪个哈希槽中。然后将key放到哈希槽对应的节点上。

**1.1 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校验和 。集群中的每个节点负责处理一部分哈希槽。

举个例子，一个集群可以有三个节点，其中：

节点 A 负责处理 0 号至 5500 号哈希槽。

节点 B 负责处理 5501 号至 11000 号哈希槽。

节点 C 负责处理 11001 号至 16383 号哈希槽。

这种将哈希槽分布到不同节点的做法使得用户可以很容易地向集群中添加或者删除节点。 比如说：

如果用户将新节点D添加到集群中，那么集群只需要将节点A、B、C 中的某些槽移动到节点D就可以了。

如果用户要从集群中移除节点A， 那么集群只需要将节点A中的所有哈希槽移动到节点B 和节点C，然后再移除空白（不包含任何哈希槽）的节点A就可以了。

因为将一个哈希槽从一个节点移动到另一个节点不会造成节点阻塞，所以无论是添加新节点还是移除已存在节点，又或者改变某个节点包含的哈希槽数量，都不会造成集群下线。

**1.2 Redis集群的主从复制模型**

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型,每个节点都会有N-1个复制品.

在我们例子中具有A，B，C三个节点的集群,在没有复制模型的情况下,如果节点B失败了，那么整个集群就会以为缺少5501-11000这个范围的槽而不可用.整个集群就不可用了，就下线了。

然而如果在集群创建的时候（或者过一段时间）我们为每个节点添加一个从节点A1，B1，C1,那么整个集群便有三个master节点和三个slave节点组成，这样在节点B失败后，集群便会选举B1为新的主节点继续服务，整个集群便不会因为槽找不到而不可用了

不过当B和B1 都失败后，集群是不可用的

Redis启动时会自动读取RDB快照文件，将数据从硬盘载入到内存，根据数量的不同，这个过程持续的时间也不尽相同，通常来讲，一个记录1000万个字符串类型键，大小为1GB的快照文件载入到内存需要20-30秒的时间。

**开启AOF**

默认情况下，Redis没有开启AOF（append only file）持久化功能，可以通过在配置文件中作如下配置启用：

|  |  |
| --- | --- |
| 1 | appendonly yes |

**同步硬盘数据**

虽然每次执行更改数据库的内容时，AOF都会记录执行的命令，但是由于操作系统本身的硬盘缓存的缘故，AOF文件的内容并没有真正地写入硬盘，在默认情况下，操作系统会每隔30s将硬盘缓存中的数据同步到硬盘，但是为了防止系统异常退出而导致丢数据的情况发生，我们还可以在Redis的配置文件中配置这个同步的频率：

|  |  |
| --- | --- |
| 1  2  3 | # appendfsync always  appendfsync everysec  # appendfsync no |

第一行表示每次AOF写入一个命令都会执行同步操作，这是最安全也是最慢的方式；

第二行表示每秒钟进行一次同步操作，一般来说使用这种方式已经足够；

第三行表示不主动进行同步操作，这是最不安全的方式。

l **Redis的使用场景**

n **计数器**

Redis是一个很好的计数器，能够实现原子递增，可以使用reids的get和set命令重置计数器，还能够让计数器过期

n **缓存服务器**

Reids的读写速度比较高，而且易于集群，吞吐量大（存数据和读数据的量）,使用KEY-VALUE存储,适合于做缓存服务器，当数据的数据热度比较高并且不经常更新时，就可以把它存redis中，减少数据库io操作，提升系统性能

n **消息服务器（Pub/Sub）**

Reids可以作为一个消息服务器使用，redis可以设定某一个KEY进行消息发布和消息订阅，当一个KEY值进行了消息发布之后，订阅它的客户端都会收到相应的消息

**reids在项目中哪些模块用到，代码怎么写的？**

l **秒杀业务**

把你要卖出的商品比如10个商品放到缓存中；然后在redis里设置一个计数器来记录请求数，这个请求数你可以以你要秒杀卖出的商品数为基数，比如你想卖出10个商品，只允许100个请求进来。那当计数器达到100的时候，后面进来的就显示秒杀结束，这样可以减轻你的服务器的压力。然后根据这100个请求，先付款的先得后付款的提示商品已秒杀完。

l **网站首页排行榜**

网站首页的排行榜每个用户登录都会看到，访问量较大，而且会定时更新，我们把用户的浏览量存如mongoDB中，每天晚上凌晨两点时启动定时器，然后通过后台算法计算出首页的排行榜信息，把计算出来的排行榜信息存入redis中，前台通过ajax异步加载把数据加载出来。

总结：redis的持久化

Snapshotting（快照）RDB

此时在 src目录下生成文件appendonly.aof，这个文件记录了每一步的操作，效率稍慢但是更安全

数据类型，持久化，问题，集群【主从，哨兵，集群】

为啥要配置redis集群：

1.提高效率

2.备份替换。

redis集群服务器至少需要六台

三主三从

**查看进程**

**说明：**

**redis集群至少需要3个主节点，每个主节点有一个从节点总共6个节点**

**replicas指定为1表示每个主节点有一个从节点**

Redis服务器程序是单进程模型，也就是在一台服务器上可以同时启动多个

Redis进程，Redis的实际处理速度则是完全依靠于主进程的执行效率。若在

服务器上只运行一个Redis进程，当多个客户端同时访问时，服务器的处理

能力是会有一定程度的下降；若在同一台服务器上开启多个Redis进程，

Redis在提高并发处理能力的同时会给服务器的CPU造成很大压力。即：在实

际生产环境中，需要根据实际的需求来决定开启多少个Redis进程。若对高

并发要求更高一些，可能会考虑在同一台服务器上开启多个进程。若 CPU

资源比较紧张，采用单进程即可。

在Redis中，实现高可用的技术主要包括持久化、主从复制、哨兵和集群，下面

分别说明它们的作用，以及解决了什么样的问题。

●持久化：持久化是最简单的高可用方法(有时甚至不被归为高可用的手段)，主

要作用是数据备份，即将数据存储在硬盘，保证数据不会因进程退出而丢失。

●主从复制：主从复制是高可用Redis的基础，哨兵和集群都是在主从复制基础

上实现高可用的。主从复制主要实现了数据的多机备份，以及对于读操作的负载均衡和简单的故障恢复。缺陷：故障恢复无法自动化；写操作无法负载均衡；存储能力受到单机的限制。

●哨兵：在主从复制的基础上，哨兵实现了自动化的故障恢复。缺陷：写操作无法负载均衡；存储能力受到单机的限制。

●集群：通过集群，Redis解决了写操作无法负载均衡，以及存储能力受到单机

限制的问题，实现了较为完善的高可用方案。

为了减少对数据库的访问量，可以借助我们的缓存，我一般会选择redis；

当时我为了避免redis服务器产生单点故障，也为了提高redis服务器的存储数据的容量；我们将redis配置为集群模式，为了提高集群的高可用性，我们使用了3台redis作为master;

并且为每台master都配置了对应的slave；这样就可以达到数据扩容同时提高集群的可用性。

整个集群是由不同的node【节点】组成的，而每个node【节点】上面被分配了slot也就是槽，数据是被放到了对应的slot【槽】中。

redis集群中共有**16384个槽**，集群会把这16384个槽**平均分配到不同的节点上**；当向集群中存数据的时候，会根据key采用CRC16算法，算出来一个值，让这个值和*16384求余数*，这样就把数据分片的存储到不同的槽中，也就落到了不同的节点上，实现了数据的分片，同时也起到了扩容的目的。

如果没有配置对应的slave服务器，则一台master宕机就会导致整个集群不可用，这就大大削弱了集群的功能，所以我们可以为每台master配置一个slave，这样当master宕机时，对应的slave就会充当master，使集群继续提供服务。如果宕掉的那台master又被重启了，则该master会变为slave。

在我们构建的redis集群中，有3个主，3个从，当存活的 主 节点的数量小于总节点的一半时，集群不可用。即当存活的主节点数量为2时，集群不可用。

在具体创建集群的时候我们使用的是***redis-trib.rb create  --replicas 1 后面跟上各个redis服务器的ip地址和端口号，表明对每个master服务器都创建一个对应的slave服务器。***

配置完集群后可以通过./redis-cli加上-c参数连接到集群，并且通过cluster nodes查看

集群中各个节点的信息，通过这个信息可以看到谁是主，谁是从，以及主从之间的关联关系。

在项目中我们会通过jedisCluster连接redis集群，进行数据的操作，我们将集群中各个节点的ip地址和端口号构建成一个HostAndPort并把它们放到一个set集合中传给jedisCluster。

Redis 有一个redisTemplate：是spring 封装了 RedisTemplate 对象来进行对redis的各种操作，它支持所有的 redis 原生的 api。

我用redisTemplate里面string中的一个get set 方法就是opsForValue（ops噢弗valus）

这方法的作用就是 用来判断接口的幂等性 前台生成一个随机数

**3、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：