|  |  |  |  |
| --- | --- | --- | --- |
| 基本情况 | | | |
| **姓名** | 宋晓方 | 性别 | 男 |
| **出生日期** | 1996-7-19 | 属相 | 鼠 |
| **现住地** | 丰台区 东营里 2号院 |  |  |
| 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** | 2011-9 | 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** | **郑州市二七区学院路1号** | 校长 | 秦 |
| **去学校的方式：** | **在火车站乘302 到学院路** | | |

|  |  |
| --- | --- |
| 第一家公司 ： | 蓝信科技（郑州）有限公司 |
| **地址：** | 郑州中原区杜兰街63号 |
| **入职时间：** | **2017年7月** |
| **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：** | **陈洋** |
| **薪资方面：** | **转正以后是6千 到18年4月的时候长到8千** |
| **路线：** | **公司附近住的 骑个电车就到啦** |

|  |  |
| --- | --- |
| **第二家公司情况：** | 上海炫体信息科技有限公司(只说公司简称) |
| **公司地址**： | 上海市 闵行区 万科·七宝国际 21栋60号楼（地铁9号线中春路6号口） |
| **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" \o "技术咨询公司) , 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9号线 中春路号口** |
| **入职时间：** | **2019年1月** |
| **离职时间：** | **2021年4月底** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **于少鹏** |
| **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：** | **17-18k** |
| **薪资的构成:** | **岗位工资(5K)+绩效工资(10K)** |

|  |
| --- |
|  |
|  |

先做一个自我介绍：后续填充

把话题引到电商项目 购物车一块：用到的知识点就是springclod redis springboot mybatis mysql 占到主动权 。

要么单说springclod 各大组件 细致话聊 ，（如果是当面的话就拿一张纸）

要么聊 redis +在购物车的几个应用场景 （可以转消息队列）

SpringCloud 理论知识

Springcloud是一个分布式微服务架构的一站式解决方案，可以理解为是一个技术栈，包含了很多组件，其中最核心的几个就是Eureka（犹瑞卡）,Ribbon,Feign（芬）,hystrix（黑丝特瑞）,zuul（如）这几个组件。

Eureka 你可以把它理解为一个注册中心 所有的微服务都会在EurekaSercer中心进行注册，从而可以达到一个数据的交互。Eureka分为服务端和一个客户端 服务端提供服务注册功能，在各个节点启动 会在服务中心进行注册 这样就会在服务注册表中存储所有可用服务节点信息，服务节点的信息可以在界面中直观的看到。客户端是一个java客户端，用于简化EurEka的交互。同时也具备一个内置的使用轮询负载算法的负载均衡器，在应用启动后，将会向Eureka发送心跳（默认周期30秒）。如果Eureka注册中心在多个心跳周期内没有接收到某个节点的心跳，EurekaServer将会从服务注册表中把这个服务节点移除（默认90秒）。Eureka也我们提供了可视话的监控页面，可以直观地看到各个 Eureka Server 当前的运行状态和所有已注册服务的情况。

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**。

综上，自我保护模式是一种应对网络异常的安全保护措施。它的架构哲学是宁可同时保留所有微服务（健康的微服务和不健康的微服务都会保留），也不盲目注销任何健康的微服务。使用自我保护模式，可以让Eureka集群更加的健壮、稳定。

在Spring Cloud中，可以使用eureka.server.enable-self-preservation = false 禁用自我保护模式。

# Eureka工作原理

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

Feign的理论知识

Feign是一个声明式的Web服务客户端，使得**编写Web服务客户端**变得非常容易，

**只需要创建一个接口，然后在上面添加注解即可**。**声明式调用就像调用本地方法一样调用远程方法**。Spring Cloud 的声明式调用, 可以做到使用 HTTP 请求远程服务时能就像调用本地方法一样的体验，开发者完全感知不到这是远程方法，更感知不到这是个 HTTP 请求。**它解决了让开发者调用远程接口就跟调用本地方法一样**，无需关注与远程的交互细节，更无需关注分布式环境开发。

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

# Feign的工作原理

主程序入口添加了@EnableFeignClients注解开启对FeignClient扫描加载处理。根据Feign Client的开发规范，定义接口并加@FeignClient注解。

当程序启动时，会进行包扫描，扫描所有@FeignClients的注解的类，并且将这些信息注入Spring IOC容器中，当定义的的Feign接口中的方法被调用时，通过JDK的代理方式，来生成具体的RequestTemplate.当生成代理时，Feign会为每个接口方法创建一个RequestTemplate。当生成代理时，Feign会为每个接口方法创建一个RequestTemplate对象，该对象封装HTTP请求需要的全部信息，如请求参数名，请求方法等信息都是在这个过程中确定的。

然后RequestTemplate生成Request,然后把Request交给Client去处理，这里指的是Client可以是JDK原生的URLConnection,Apache的HttpClient,也可以是OKhttp，最后Client被封装到LoadBalanceClient类，这个类结合Ribbon负载均衡发送服务器之间的调用。

# 服务之间调用如何传递token

1. **在公共模块下新建一个Feign的拦截器类并实现RequestInterceptor的接口**
2. **在服务消费者调用服务提供者需要传递token时，服务消费者项目下新建一个配置类， 让咱们在公共模块下编写的feign拦截器生效**

Hystrix理论知识

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**

* **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

* 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

* **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).

* **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**。**

* **服务容错**
* **线程和信号隔离**

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

* **请求缓存**

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

* **请求合并**

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

* **近乎实时的服务监控**

Zuul的理论知识

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

在某些场景下存在跨域请求的问题

每个微服务都会需要鉴权、限流、权限校验等逻辑，如果每个业务都各自为战，自己造轮子实现一遍，会很头疼，完全可以抽出来，放到一个统一的地方去做。

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**。

# Zuul默认的路由规则

最直观的理解：“路由”是指根据请求URL，将请求分配到对应的处理程序。在微服务体系中，Zuul负责接收客户端所有的请求。根据不同的URL匹配规则，将不同的请求转发到不同的微服务处理。通过Zuul和Eureka进行整合，Zuul会根据服务名自动的从注册中心中获取服务地址并转发请求，这样做的好处不仅可以通过单个端点来访问应用的所有服务，而且在添加或移除服务实例的时候不用修改Zuul的路由配置

# Zuul中的过滤器

通过之前的学习，我们得知Zuul它包含了两个核心功能：对**请求的路由和过滤**。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。其实，路由功能在真正运行时，它的路由映射和请求转发同样也由几个不同的过滤器完成的。所以，**过滤器可以说是Zuul实现微服务网关功能最为核心的部件**，每一个进入Zuul的HTTP请求都会经过一系列的过滤器处理链得到请求响应并返回给客户端。

拦截器的四中类型

* **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。
* **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。
* **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。

Redis 是一个非关系形的数据库 他是基于内存的，我们为什么要用redis的呢 redis的优势就是他的读写非常快 读每秒110000次 写每秒81000次

1 redis不仅支持k/v类型的数据，同时还提供了string list，set，zset，hash数据结构的存储。  
2、redis支持集群化，即master-slave模式的主从复制  
3、redis支持数据的持久化，可将内存数据保存在磁盘中，重启的时候可以再次加载进行使用

使用Redis就是为了要减少请求和数据库之间的交互

String 数据类型的使用场景

我们在OA项目的用户展示地区的时候就是用的这个

用的redis的String类型,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，所以我们这一点是用redis做的

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getand 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

hash这个类型和数据库的表很像，key-filed-value

key就相当于表名,filed就相当于id,value就是其他字段

我们在做电商项目的时候用于做购物车数据

因为购物车数据经常存和删除，这样直接作用于数据库的时候，数据会炸

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

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，

这个我只是了解 我没有具体的用做过 消息队列的话 我用的是rabiitmq

此时要么面试官打断你问消息队列 要么继续redis的持久化 集群

Redis 两种持久话方案 默认的是rdb

Rdb ：rdb是对redis中的数据进行一个快照的模式，就是以时间间隔的形式将内存中的数据写入的硬盘中。实际操作过程是fork一个子进程，先将数据集写入到一个临时文件，临时文件写完以后在替换之前的文件，用二进制 压缩存储。

配置的话就是在redis.conf中，找到save的关键字，来指定每隔多长时间有多少个key发生变化就将其持久化。

Aof：aof是将每次写操作对应的指令 保存到后缀名为aof的文本文件中。

Rdb的性能要比aof高

Rdb的文件中直接存储的是数据 也就是说在redis服务启动的时候直接加载rdb文件中数据就可以啦。

而aof文件中存储的是指令 那么redis服务启动的时候先加载aof中指令，在依次去执行这些指令。

Aof的实用性 和可靠性要比rdb高

Rdb呢就是说如果redis发生故障 那么还没来的急持久化的数据就是被丢失，因为他是按时间间隔来进行持久化的 而aof他能够做的数据不容易丢失，你也可以理解为他是实时更新，因为他是redis每发生一次变化就以指令的方式保存在文件中啦。

Redis 的三种状况

雪蹦： 大量的key 同一时间失效

穿透：恶意访问一个你缓存 ，数据库一定不存在的数据

击穿：一个热门的key 突然失效 大量的并发请求直接击穿你的数据库啦

解决方案就是：将热门的key设置永不过期

占空间，内存消耗大，并且不能保持数据最新

我可以去做一个定时器 定时去更新热门的key

1. 设置一个互斥锁 不过不建议这样做

1. 代码复杂度增大

2. 存在死锁的风险

Redis 的集群化

主从复制 哨兵 集群部署

主从复制就是：单台的redis可能会发生单点故障，一旦单点故障 数据就会丢失。

主从复制就将一台redis的数据去复制给多台redis服务器 实现一主多从 他还数据的复制是单项的 只能由主节点到从节点。

主从复制的几个优点吧

数据沉于：实现了数据的备份

故障恢复：如果我的主节点出现了问题 那么我的从节点就提供服务 实现快速的故障恢复。

**负载均衡 ： 主从模式 是读写分离的 如果在写少读多的情况下 ，读redis数据的时候可以从多个redis服务分担读取 可以提高了redis服务的并发量。**

哨兵 ：就是主从复制存在一个问题，故障恢复无法自动化 。哨兵就是解决这个问题的 哨兵就是一个监听 ,哨兵也是一个独立的redis节点，不过他不存储数据，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**。

哨兵的基本原理的：

定时任务 ：每个哨兵节点维护了三个定时任务。分别是：

通过向主从节点发送info命令来获取最新的主从结构

通过发布订阅功能获取其他节点的信息

通过向其他节点发送ping命令进行心跳检查 判断是否下线

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

**客观下线：**哨兵节点在对主节点进行主观下线后，就是我们在进行配置哨兵的时候会配置一个多少节点同意就对主节点下线 注意是（主节点没有心跳 会询问其他的从节点 主节点的状态 当从节点到达一个数值的时候就下线啦）

**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。

如果当原来的主节点在启动以后他会变成新主节点的从节点

集群模式：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**。

集群是redis官方提供的分布式存储方案 是在3.0版本推出的

Redis就是把多台的redis服务部署在一起 Redis集群采用**无中心结构**，集群由多个节点组成的

Redis集群是两个方面 高可用 数据分片

**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集群的主从复制模型**

**Redis集群的数据分片**

准备一个redis介绍完 另外一个mongdb

以前也没怎么用过mongdb 但私下也研究过

**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**

# RabbitMQ

削峰 异步 解耦

面对我们一些高并发场景 ;我们的项目可能会扛不住;

响应慢;

请求量大;

消息队列是什么 就是可以暂时存储生产者的数据;暂存 等待消费者来消费;

削峰 就是我们有3000个请求 生产者将数据交给消息队列 就可以返回了; 我们消费者可以先拿1000个数据消费;

解耦 我们有 ABCD 四个服务 a 生产出一个id ;

放入 消息队列中 其他服务调用与否 我们a是不用关心的

异步 就业务逻辑一条直线往下进行 所谓异步就是把他进行了分支 共同执行

我们的消息队列 有两大核心 交换机 和队列组成的

有三种推送模式

直连模式 扇形模式 主题模式

直连模式 ：就是 交换机和队列是点对点的关系

扇形模式： 那么扇形就是 一对多的关系 。

主题交换机：就是可以是点对点关系 也可以是一对多的关系 他们之间通过规则来进行的规范

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。

\*  **(星号) 用来表示一个单词 (必须出现的)**

#  **(井号) 用来表示任意数量（零个或多个）单词**

通配的绑定键是跟队列进行绑定的，举个小例子

队列Q1 绑定键为 \*.TT.\*          队列Q2绑定键为  TT.#

如果一条消息携带的路由键为 A.TT.B，那么队列Q1将会收到；

如果一条消息携带的路由键为TT.AA.BB，那么队列Q2将会收到；

消息队列的信息确认机制 ：

首先为什么要有消息确认机制 如果服务出现异常 不能服务啦那是不是数据就丢失啦 所以就有了消息确认机制

RabbitMq 是把消息分为四种情况 ：

生产者找不到交换机和队列 返回一个信息

生产者对交换的时候要给一个信息，

交换机给队列也要返回一个信息

队列给消费者要给一个信息

消费者本身也是在监听消息 消费者的确认机制就是 ：

自动确认：消费段只要接收到消息就认为消息已经成功 ，那么如果消息处理的时候失败啦 那么这条消息就会丢失 一般解决这种的话 就只能捕捉着异常在根据日志，做后续的处理。

手动确认 ：里面有三个参数

第一个ack 是用于肯定确认 没什么好说的

第二个 nack 这个可以理解设置一些条件不消费某条数据

第三个 reject 是指不确认的消息是否丢回队列里 重新入队 这个模式的话要谨慎 如果考虑不周全的会出现消息一直重复在调用 最终导致一个挤压的

死信交换器 ：在队列上指定一个交换器，如果队列上发生啦消息被拒绝，消息数量超过队列最大限制就被删除 ，，，就会把这条消息转的这个队列上

延迟队列 ：顾名思义嘛

Springclod 的时候扯到分布式事务

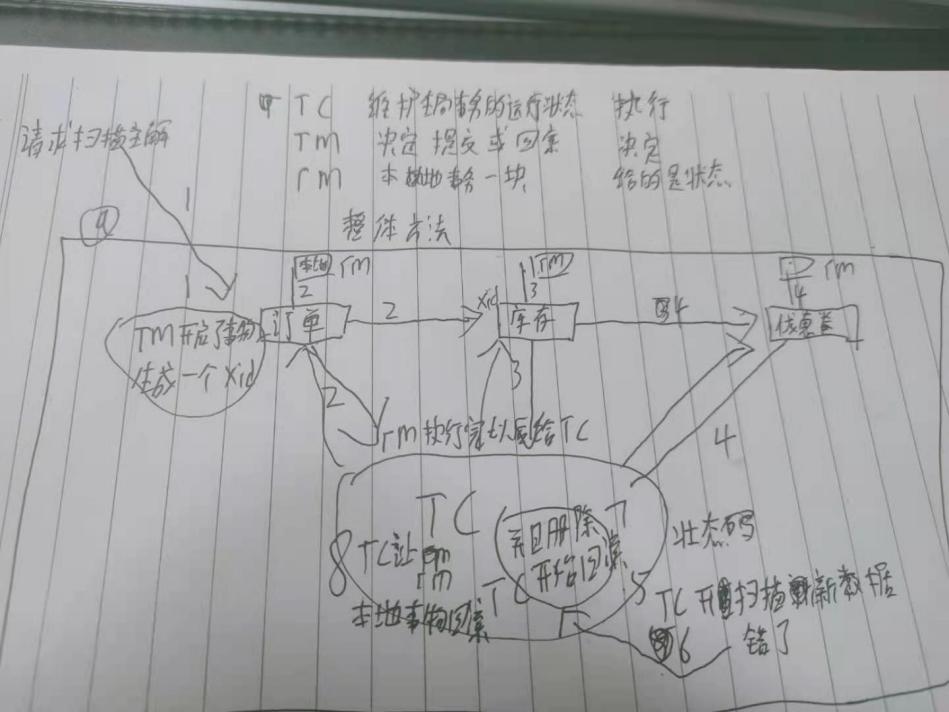
在分布式事务中为了要解决 数据的一致性就必须用到了分布式事务。

而我们是用的 Seate 是阿里开发出来的解决分布式事务的一个插件。

Seata框架中一个分布式事务包含3中角色：

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

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



第一步： 在需要加事务的方法上加@GlobalTransactional（格搂博崔傻事坎捞） 生成全局事务并创建全局唯一的xid。

第二步 ：我们在做订单生成的时候 需要改动三个不同的数据库，订单 ，库存，优惠卷。

当第一个订单数据改动以后 ，订单会生成一个RM也就事本地数据 并且会吧信息提交给TC，

第三步 ：订单携带xid去做库存的修改 完成以后 生成本地事务 RM 并将信息给TC 。

第四步： 携带xid来到优惠卷 生成一个本地事务 但优惠卷 报错啦 。那么本地事务RM记录信息 并发送到TC上面 TC开始检查数据 发现状态码不正确 TC告诉TM 那么 TM就决定回滚 ， TC开始让当前事务产生的数据开始回归 TC 开始删除数据 RM 通过xid去找log日志表的数据 看做啦那些改动 已经提交的数据全部回到本地事务上开始回滚 ，为什们RM本地事务 会回滚 是在本地事务提交前，申请一个记录的全局锁，因此他1直接提交本地事务，并向TC去汇报本地事务执行成功 但他的全局锁没有释放 ，全局锁的释放取决于二阶段是提交命令还是回滚命令

如果事务全部执行成功 那么TM 就提交事务 释放全局锁 并让TC去删除log表里的信息。

Dubbo 的运行过程

首先服务提供者 Provider 启动然后向注册中心注册自己所能提供的服务。

服务消费者 Consumer 启动向注册中心订阅自己所需的服务。然后注册中心将提供者元信息通知给 Consumer， 之后 Consumer 因为已经从注册中心获取提供者的地址，因此可以通过负载均衡选择一个 Provider 直接调用 。

之后服务提供方元数据变更的话注册中心会把变更推送给服务消费者。

服务提供者和消费者都会在内存中记录着调用的次数和时间，然后定时的发送统计数据到监控中心。

### springboot

微服务 进行接口开发的

Spring Boot是由 Pivotal 团队提供的全新的框架，spring boot 整合了所有的框架让spring应用程序尽快跑起来并且尽可能减少你的配置文件

（1）优点：

1）为所有 Spring 开发提供一个更快更广泛的入门体验

2）零配置。无冗余代码生成和XML 强制配置，遵循“约定大于配置”（我们可以直接用它里面默认的配置，不需要去修改）

3）集成了大量常用的第三方库的配置， Spring Boot 应用为这些第三方库提供了几乎可以零配置的开箱即用的能力

4）提供一系列大型项目常用的非功能性特征，如嵌入式服务器web、安全性、度量、运行状况检查、外部化配置等

5）Spring Boot 不是Spring 的替代者，Spring 框架是通过 IOC 机制来管理 Bean 的。Spring Boot 依赖 Spring 框架来管理对象的依赖。Spring Boot 并不是Spring 的精简版本，而是为使用 Spring 做好各种产品级准备

（2）常用注解：

@restcontroller @getmapping @PathVariable @MapperScan

@springbootapplication（他是由三个注解组成的分别是@springbootConfiguration读取配置文件，配置文件的路径是当前根目录@EnableAutoConfiguration开启自动配置，扫描当前的所有依赖的jar包，发现新的依赖出现将会根据依赖完成各种自动配置@ComponentScan属于spring 框架(@Component,@Service,@Controller,@Repository,@Entity)，扫描范围默认情况下是启动类所在的同名包及其子孙包）

说完以后你可以和他去扯io项目的事务

**事务的传播特性: 一共七个 你随便说几个 其他的就说忘啦**

1. PROPAGATION\_REQUIRED(瑞快润特)

如果事务不存在则创建，如果存在使用存在的事务

1. PROPAGATION\_SUPPORTS(死普特死)

如果当前存在事务，则使用存在的事务，如果不存在则使用非事务方式处理。

1. PROPAGATION\_MANDATORY(慢的脆)

如果当前存在事务，则使用存在的事务，如果不存在则抛出异常。

4. PROPAGATION\_REQUIRES\_NEW(瑞快润)

永远创建新的事务，如果当前存在事务则挂起

5. PROPAGATION\_NOT\_SUPPORTED

永远使用非事务方式执行，如果当前存在事务则挂起

6. PROPAGATION\_NEVER

不支持事务执行方式，如果存在事务则抛出异常

7. PROPAGATION\_NESTED

如果存在一个活动的事务，则运行在一个嵌套的事务中. 如果没有活动事务, 安按照PROPAGATION\_REQUIRED 事务属性执行

事务的隔离级别

：1：读未提交

一个事务开始写数据，则另一个事务不允许同时进行写操作。但可以读数据。但可能读事务会把写事务未提交的事务读取到。

2读提交

如果式读事务 那么允许其他事务读写 ，如果是写事务 那么就禁止其他事务操作该数据 。这种隔离级别 产生了一个问题 A事务刚读完数据 B事务就写完了数据并提交了数据。

3 可重复读

可重复读取是指在一个事务内，多次读同一个数据，在这个事务还没结束时，其他事务不能访问该数据(包括了读写)，这样就可以在同一个事务内两次读到的数据是一样的，因此称为是可重复读隔离级别，读取数据的事务将会禁止写事务(但允许读事务)，写事务则禁止任何其他事务(包括了读写)，这样避免了不可重复读和脏读，但是有时可能会出现幻读。(读取数据的事务)可以通过“共享读镜”和“排他写锁”实现。

解决了更新丢失、脏读、不可重复读、但是还会出现幻读

4 (可序化)

提供严格的事务隔离，它要求事务序列化执行，事务只能一个接着一个地执行，但不能并发执行，如果仅仅通过“行级锁”是无法实现序列化的，必须通过其他机制保证新插入的数据不会被执行查询操作的事务访问到。序列化是最高的事务隔离级别，同时代价也是最高的，性能很低，一般很少使用，在该级别下，事务顺序执行，不仅可以避免脏读、不可重复读，还避免了幻读

解决了更新丢失、脏读、不可重复读、幻读(虚读)

分布式就扯那么多 下面换一个方向 多线程的导出

Java反射 自定义注解 多线程 线程池

当时在公司做一个报表,需要使用excel导出,做了好几个版本,进行优化,第一版 就选择了poi导出,通过反射+注解+zip来做,

但是反应慢,我们加上线程池;

整体的一个思路：

考虑到程序的灵活性 需要用户传过来两个参数一个是sheet的数据量 和每个excel中多少sheet页。

1. 先查出要导出多少数据量 sheet=总数据量/每页数据量 excel=sheet总数/一个excel的数量。循环excel总数 每次循环 相当于创建一个excel的数据 ，创建workbox 在循环我们当前excel中的每个sheet页 ，然后求出我的起始下标与条数 条数已经传过来啦。起始下标就是 根据（循环一个excle的总sheel页+1 循环是从0 开始的 ）\*sheet的页数+1 就得到起始下标啦。往数据库里取出数据 得到一页的数据。 下面通过反射得到我们的javabean的属性，设置上自定义注解,value为我们的表头信息, getDeclaredFields获取所有字段的集合;循环判断加了注解;将注解的value设为表头;

Createcell.setcellvalue();

循换每个sheet的数据; 创建row 在循环我们反射的bean属性;如果加了注解就创建一个cell;并判断类型;放入对应cell中;

写入我们服务器文件中

为了导出速度 我们加人了线程池 并使用contdownlach（抗得拉赤）

等所有线程执行完成;我们通过java提供的zip进行压锁并响应; 将响应流加入我们的zip输出流;读写我们excel文件夹; 完成多线程导出;

线程池：

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;

可以减少我们线程创建与销毁的资源消耗;

创建方式 自定义型

Threadpoolexctor t=new Thredddpoolexctor

参数 :

1.核心线程数 即空闲也不会销毁的线程 核心数不足 提交任务会创建

2.最大线程数;核心数满了 就创建非核心 ;想家不能超过最大线程数

3.非核心超时时间: 非核心线程限制多久被回收;

4.线程队列:队列怎么处理线程 是直接创建新线程还是等待还是直接交给核心执行

5,饱和策略:丢弃任务 丢弃其他任务 新建线程 报异常

常用得线程

Fixedthredpool（谁科室度瑞炮）=Excutors.newfixedthredpool（咦柯吃丢死）.(牛弗式度瑞炮)

固定大小的线程;核心与最大一致 控制并发量 异常 建新的

Cachedthredpool（台柯）可缓存的线程池; 核心0 最大 max 有任务创建线程; 闲置回收 节约资源

Singlethredexecutor

一个单线程的线程池; 保证顺序

**ScheduledThreadPool（式待吐瑞破）**

创建一个可以定时 或 延时的线程池

还有一个单线程的定时延时线程

Zip 就是java提供的一个压缩包工具

Zipoutputstream zip输出流

Zipentry 相当于zip 中的一个位置

说一些框架 下面围绕spring springmvc 说 springboot

### springmvc

（1）springMVC的运行原理

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

（2）接参传参

接参：request 基本数据类型 对象（注意：时间和文件）

传参：转发传值 json传值 artreebute

（3） 页面跳转

页面跳转的方式有 转发forward 重定向redirect

Redirect和forward的区别：

转发(forward)和重定向(redirect)的区别：

转发地址栏不发生变化，数据共享，一次请求，执行效率高；

重定向地址栏发生变化，数据不共享，多次请求，执行效率低。

1. 常用的注解有：@controller @RequestMapping @ResponseBody

M 持久层框架

### mybatis+plus

半ORM思想使用sql语句进行crud操作 数据库的 持久层框架

它支持自定义 SQL、存储过程以及高级映射。

MyBatis 免除了几乎所有的 JDBC 代码以及设置参数和获取结果集的工作。

MyBatis 可以通过简单的 XML 或注解来配置和映射原始类型、接口和 Java POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

使用select delete insert update 标签做增删改查

（1）优点：

① 基于SQL语句编程，相当灵活，不会对应用程序或者数据库的现有设计造成任何影响，SQL写在XML里，解除sql与程序代码的耦合，便于统一管理；提供XML标签，支持编写动态SQL语句，并可重用。

② 与JDBC相比，减少了50%以上的代码量，消除了JDBC大量冗余的代码，不需要手动开关连接；

③ 很好的与各种数据库兼容（因为MyBatis使用JDBC来连接数据库，所以只要JDBC支持的数据库MyBatis都支持）。

④ 能够与Spring很好的集成；

⑤ 提供映射标签，支持对象与数据库的ORM字段关系映射；提供对象关系映射标签，支持对象关系组件维护。

（2）缺点：

① SQL语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写SQL语句的功底有一定要求。

② SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

（3）#{}和${}的区别是什么？

${}是字符串拼接，#{}是占位符赋值；

Mybatis在处理${}时，就是把${}直接替换成变量的值。而Mybatis在处理#{}时，会对sql语句进行预处理，将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

使用#{}可以有效的防止SQL注入，提高系统安全性。

（4）,使用MyBatis的mapper接口调用时有哪些要求？

Mapper接口方法名和mapper.xml中定义的每个sql的id相同；

Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同；

Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同；

Mapper.xml文件中的namespace即是mapper接口的类路径。

（5）.Mybatis动态sql有什么用？执行原理？有哪些动态sql？

Mybatis动态sql可以在Xml映射文件内，以标签的形式编写动态sql，执行原理是根据表达式的值 完成逻辑判断 并动态拼接sql的功能。

Mybatis提供了9种动态sql标签：trim | where | set | foreach | if | choose | when | otherwise | bind。

1. 常用的方法

selectById() 根据id查询

selectList(); 查List<对象>

selectPage() 分页

selectOne()按条件查

insert()新增

updateById()根据id查询

deleteById()根据id删除

### 四，hibernate

hibernate是全自动跨数据库开发效率高于jdbc的持久成框架

使用的是hql语句最终会转化为sql语句.语法是 from 对象名

（1），hibernate运行原理

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

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

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

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

（2），十．hibernate的五大核心接口

1.Configuration(类):

加载配置文件hibernate.cfg.xml文件中的配置信息，从而得到：

1）.hibernate的底层信息：

数据库连接，jdbc驱动，方言，用户名，密码。

2）.hibernate的映射文件(\*.hbm.xml)

2.sessionFactory（接口）

通过configuration创建sessionFactory。

可以用来获得session。

sessionFactory是线程安全的，

里边加载的数据信息有数据库的配置信息和映射关系.

3.Session(接口)

线程不安全。

相当于jbdc的connection.

用来操作数据库.

4.Transaction（接口）

进行事物的控制实现事物的commit（提交)和rollback（回滚）

5.Query(接口)

（3），hibernate主键生成策略

1》increment(银科瑞门特) :先选出当前表的最大ID，在此基础上加1，hibernate会自行处理，适用于int short,long类型的主键

2》sequence:oracle数据库会自行处理，会增加一个序列配置

<generator class="sequence">

<param name="sequence">sequence\_id</param>

</generator>

3》native（通用多个数据库，数据库的本地生成策略）

4》uuid：生成一个32位的，不重复的字符串，可以达到夸数据库

（4）．hibernate中session中的常用方法

save():保存

saveOrUpdate()保存或者修改

delete()删除

update()修改

get()根据唯一标示获取对象 立即加载

load()根据唯一标示获得对象 ，延迟加载

# Mysql优化

## Mysql优化哪些方面？

## 表设计上 存储引擎和字段类型

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

## 功能上 创建索引

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

提高数据检索的效率，降低数据库的IO成本。

通过索引列对数据进行排序，降低数据排序的成本，降低了CPU的消耗

1. 索引的缺点：

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

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

## SQL语句上 合理的sql

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

# Docker容器引擎

## 1、docker是什么？

Docker是一款可以解决运行环境和配置问题的软件容器引擎；

Docker的组成部分（容器，镜像（image），仓库）

## 2、docker的相关命令

docker version 查看docker版本

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

镜像的操作命令：

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

docker search 镜像名 搜索镜像 收藏数

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

docker rmi 镜像名（id） 注意正在被容器使用的镜像是无法删除

容器的操作命令 （有镜像才能创建容器）

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

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

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

退出容器 Ctrl+P+Q

查看运行中的容器 、docker ps

查看所有的容器docker ps -a

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

停止容器 docker stop 容器id（名称）

启动容器docker start 容器id（名称）

拷贝 宿主机向容器中拷贝文件

docker cp 宿主机的文件 容器id:目录

拷贝 容器中的文件 拷贝到宿主机上

docker cp 容器id:目录 宿主机的目录

## 3、数据卷怎么创建 (创建一个容器 并和宿主机共享目录)

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

数据卷容器 （多个容器共享一个目录 （修改一个目录 多个容器跟着改变））

先创建一个共享的容器

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

创建容器指定共享的容器 实现共享目录

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

## 4、制作镜像

commit

build（Dockerfile）

编写Dockerfile文件

语法 ：

FROM 指定基础镜像

MAINTAINER 指定镜像维护者信息

RUN 用于执行指定脚本命令

CMD 指定启动容器时执行的命令

EXPOSE 指定容器暴露的端口

ENV 指定环境变量

ADD 将文件从宿主机复制到容器指定位置，同时对压缩文件有自动解压功能

COPY 将文件从宿主机复制到容器指定位置

ENTRPOINT 设置容器启动时需要运行的命令

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