# Redis总结

redis 是一个基于缓存的非关系型数据库（nosql）

有丰富的数据类型，其中的数据类型有

String ,hash,zset,hset,list

我们用String 常用于存的是什么 拿出来的就是什么

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

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

下来就是redis的持久化方案

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一条数据到日志中,

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

AOF比较安全，是AOF即便redis宕机了，丢数据也是只丢这一条的

这两种方案是看项目中的需要那种的

redis的数据恢复流程,第一redis先判断日志文件是否存在，如果存在就扫描日志文件，然后启动redis

如果不存在就看看RDB快照，如果RDB快照存在的话就直接扫描快照，如果不存在就启动redis

redis的主从复制,就是redis主机负责写，从机负责读，从机没有写的权利。可以有两种方式启动主从

一种是命令启动，就是启动两台，一台是主机，从机启动里面写个命令slaveof 主机ip和主机端口号，这种的弊端就是如果从机挂掉,再次启动就不是从机了

另一种就是在reids.conf文件中配置slaveof 主机ip和主机端口号

redis的哨兵(至少4台)，通俗来讲就是监督主从复制的。如果主机挂掉的30秒之后，哨兵就会把一个从机变成主机，另一个从机变成这个主机的从机

redis的集群最少就是启动6台，因为有一个奇数原则所以一般是三主三从

因为redis是有事务是原子性。所以我们可以用redis的getAndset来解决redis接口的幂等性（幂等性是什么）在我们OA项目部署nginx的时候，用户的登录的验证是放在session中的，我们部署了两台tomcat相当于两台服务器，里面的session是不共享,这就导致在轮训访问的时候，用户会重新登录，这个时候我们就可以借助redis来实现数据共享

说了这么redis这么多得优点，说一下他的缺点

缓存雪崩，缓存穿透，缓存击穿

# 定时任务

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吗我们要在用户生日的当天给用户发送一个生日快乐

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，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，甚至可能在比处理实际请求的时间要多的多，就算cpu是个在厉害，也不能一直干活不叫他休息吧，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，如果队列也达到最高线程数则去判断这个线程池是否已经达到这个线程池的最大核心数，如果达到了则直接执行拒绝策略拒绝任务，多线程核心实现为ThreadPoolExecutor类,我们用的线程池是FixedThreadPool，他就是创建一个固定工作线程数量的线程池。核心线程数和最大线程数是一样的，任务如果过多会在队列中阻塞。如果某个线程因为执行异常而结束，那么线程池会补充一个新线程，然后发送一个 邮件给用户的话，首先引入pom依赖还要知道你要以什么邮件类型发送的，配置他的邮件服务器主机地址，发送方的账号，授权码， 获取用户的账号，用户的邮箱，同时也要判断数据库里面的生日字段与当前时间的是否相等，就获取当前的时间是几月几号，去进行匹配就行了；但这样的话可能会出现重复发送问题也就是接口幂等性问题，发送多个请求只会返回一个结果，为了解决接口幂等性，我们当时用的是redis，因为redis是基于内存的读取速度快，有丰富的数据类型，同时他也是原子性的支持事务他的操作要么都发生要么都不发生，基于这一店可以使用redis提供的getandset方法来解决接口幂等性问题，当一个线程给一个用户发送过请求之后就把用户名当作key存进去，value可以随意定义，设置一个归定的失效时间,判断这个规定时间内这个key存在不存在就行了如果存在直接返回

# 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。

# 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