# 定时任务

我们项目中有些需求需要特定发送，比如用户生日祝福发送，发短信等，我们这时候就要用到定时器。

拿用户生日祝福来说吧，我们首先导入mail包用来发送发送邮件，在配置文件中简单配置就可以使用，要定时发送邮件的话，我们可以创造一个定时器。

1.创造

然后新建我们的定时器类TimerTask

在需要定时执行的方法上添加我们的注解 来设置我们的定时时间

@Scheduled(cron = "0 30 22 ? \* \*")

设置上之后 这个方法的方法体就变成了我们定时任务的执行体 等定时时间达到之后 就会去执行方法体里面的内容

注意类上需要添加我们的component注解 component注解是吧我们这个类 交给aop进行管理

1. 发送邮件

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

3.分布式锁

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 她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

1. **多线程**

为什么使用多线程？要用这个疑问的话首先要知道什么是多线程:

我举例来说：我们打开腾讯管家，腾讯管家本身就是一个程序，也就是说它就是一个进程，它里面有很多的功能能查杀病毒、清理垃圾、电脑加速等众多功能。

按照单线程来说，无论你想要清理垃圾、还是要病毒查杀，那么你必须先做完其中的一件事，才能做下一件事，这里面是有一个执行顺序的。

如果是多线程的话，我们其实在清理垃圾的时候，还可以进行查杀病毒、电脑加速等等其他的操作，这个是严格意义上的同一时刻发生的，没有执行上的先后顺序。

我们生日发送用多线程原因是：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3

#### Java线程具有五中基本状态：

五个状态：新建状态，就绪状态，运行状态，阻塞状态，死亡状态

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；

3）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；

4）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，此时进入阻塞状态，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。

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。

#### 5线程安全问题

在我们运行过程中可能会遇到线程安全问题

什么是线程安全问题？

（1）运行结果错误

我们多线程成一起对变量进行操作 进行i++ 一千个线程同时执行 得到的结果不一定是1000 会有得到重复数据的 是因为同时读到变量的值 （200）同时加一 最终的结果都是201

我们可以加一个synchronized锁 目的是为了让线程走到有锁的地方单个顺序执行

synchronized是加在容易产生线程安全问题的地方

但是枷锁也会碰到逻辑安全问题

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

# Redis

1 什么是redis (数据库 关系型数据库 mysql orale db2 sqlserver ）

redis 是基于key-value存储系统的非关系型数据库（缓存） redis mogodb ) 内存快 数据类型丰富 可持久化

##### Redis 优势

1. 读写快

因为redis是基于内存读写的所以速度快

性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s

（2）数据类型丰富5种

丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作

1. 可持久化

(rdb 按照一定时间的key改变 进行持久化 丢失数据会丢失一片

aof 日志 写操作(增删改操作)的指令会被记录 丢失只会丢失很少的数据)

1. 原子性

原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。

基于内存 读写快 数据类型丰富5种 持久化方案2种

（2-1）数据类型丰富5种

string（字符串）

key value

命令 set 和get

设置值

set key value

get key

hash（哈希）

key filed value

设置值

hmset key filed value

取值

hget key filed

list（列表）

设置值 lpush key value，value

取值 lrange key 下标，下标

set（集合）

设置值

sadd key value ,value

取值

smembers key

zset(sorted set：有序集合)。

设置值

zadd key score value

zadd key 分 值

取值 zrange by score key 小分 大分

zrange key 小分 大分

##### 2.使用场景

经常访问但不会轻易改变的数据 都可以使用来减轻数据库的压力

(购物车( 电商项目使用aof方案 ) 地区 热点数据 缓存)

1. 三大问题

穿透 击穿 雪崩

**（3-1）穿透**

缓存穿透 根本不存在的key 是穿透

把传过来的key 设置个失效时间

**（3-2）击穿**

缓存击穿（热点数据集中失效）

缓存的key时间到期 直接去查数据库了

**解决方案：**

互斥锁 (mutex key)：这种方案思路比较简单，但是存在一定的隐患，如果构建缓存过程出现问题或者时间较长，可能会存在死锁和线程池阻塞的风险，但是这种方法能够较好的降低后端存储负载并在一致性上做的比较好。

” 永远不过期 “：这种方案由于没有设置真正的过期时间，实际上已经不存在热点 key 产生的一系列危害，但是会存在数据不一致的情况，同时代码复杂度会增大。

**（3-3）雪崩**

redid大量的key同时失效使得大量请求进入到数据库造成雪崩

**如何避免缓存雪崩？**

如果缓存因某种原因发生了宕机，那么原本被缓存抵挡的海量查询请求就会像疯狗一样涌向数据库。此时数据库如果抵挡不了这巨大的压力，它就会崩溃。

##### 4使用缓存集群，保证缓存高可用

和飞机都有多个引擎一样，如果缓存层设计成高可用的，即使个别节点、个别机器、甚至是机房宕掉，依然可以提供服务，例如前面介绍过的 Redis Sentinel 和 Redis Cluster 都实现了高可用。

**使用缓存集群 3种 主从复制 哨兵 集群**

**（4-1）主从复制**

**优点：**

主从 1主多从 读写分离 主服务提供写服务 从服务提供读功能

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。

修改配置文件

slaveof 主服务的IP 主服务端口

***（4-2）哨兵***

监测主服务 主服务如果死亡 将从 从服务中选择一个作为主服务并告知其他从服务 主服务进行了变更

**（4-3）集群 (分片)**

数据分区(或称数据分片)是Redis集群最核心的功能

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。

一般是3主 3从 主服务死亡 从服务顶上 如果都死亡 其他的主服务将 死亡的服务占据的哈希槽 在进行分配

# docker

1. 什么是docker（虚拟机VMware）

Docker是一个开源的应用容器引擎，Docker可以让开发者可以打包他们的应用以及依 赖包到一个轻量级、可移植的容器中，然后发布到任何流行的Linux机器上，也可以实 现虚拟化，容器是完全使用沙箱机制，相互之间不会有任何接口，重要的是容器性能开 销极低。

Docker实现了一次安装到处运行；方便做持续集成并有助于整体发布的容器虚拟化技 术。

docker的优点：

l 更快速的启动时间

l 更快速的应用交付和部署

l 更便捷的升级和扩缩容

l 更简单的系统运维

l 更高效的计算资源利用

Docker环境搭建

centOS6.9：

1. yum install -y epel-release

Docker使用EPEL发布，RHEL系的OS首先要确保已经持有EPEL仓库，否则先检 查OS的版本，然后安装相应的EPEL包。

2. yum install -y docker-io

3. 安装后的配置文件：/etc/sysconfig/docker

4. 启动Docker后台服务：service docker start

docker version验证

l CentOS7安装Docker

1. 安装需要的软件包

yum install -y yum-utils device-mapper-persistent-data lvm2

2. 设置Docker的yum源

yum-config-manager --add-repo <http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

3. 更新yum软件包索引

yum makecache fast

4. 安装Docker CE

yum -y install docker-ce

5. 启动Docker

systemctl start docker

6. 测试

docker version

7. 卸载

systemctl stop docker

yum -y remove docker-ce

rm -rf /var/lib/docker

Docker总结 (面试就是linux的常用命令，docker的相关命令)

帮助命令：docker info

查看镜像：docker images

删除镜像：docker rmi 镜像

搜索镜像：docker search 镜像名

拉取镜像：docker pull 镜像名

按照镜像名称搜索镜像docker search

创建容器：docker run -id –name 容器名 -p 8080:8080 镜像名:版本号

删除容器：docker rm -f 容器ID

查容器：docker ps 查询当前运行 -a 查询运行和不运行

# 事务

1. 什么是事务？

事务就是一组有逻辑的操作单元，即一组sql语句，事务是访问数据库的一个操作序列，数据库应用系统通过事务集来完成对数据库的存取

1. 事务的四大特性

ACID

原子性：指事务是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

隔离性：指并发执行的事务之间不能互相干扰，事务与事务之间是相互隔离的

一致性：指事务必须从一个一致性状态到另一个一致性状态

持久性：指事务操作的结果不轻易发生改变，事务的操作是永久性的即使后来的故障或异常也不会改变结果。

1. 事务的传播特性

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

6. PROPAGATION\_NEVER(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(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

1. 事务的隔离级别

并发事务导致的问题：

在许多事务处理同一个数据时，由于没有采取有效的隔离机制，并发处理数据时，会带来一些问题。

①、第一类丢失更新：撤销一个事务时，把其他事务已提交的更新数据覆盖。

②、脏读：一个事务读取到另一个事务未提交的更新数据。

③、幻读也叫虚读：一个事务执行两次查询，第二次结果集包含第一次中没有或某些行已经被删除的数据，造成两次结果不一致，只是另一个事务在这两次查询中间插入或删除了数据造成的。

④、不可重复读：一个事务两次读取同一行的数据，结果得到不同状态的结果，中间正好另一个事务更新了该数据，两次结果相异，不可被信任。

⑤、第二类丢失更新：是不可重复读的特殊情况。如果两个事物都读取同一行，然后两个都进行写操作，并提交，第一个事物所做的改变就会丢失。

**1、DEFAULT（default）**

默认隔离级别，每种数据库支持的事务隔离级别不一样，如果Spring配置事务时将isolation设置为这个值的话，那么将使用底层数据库的默认事务隔离级别。顺便说一句，如果使用的MySQL，可以使用"**select @@tx\_isolation**"来查看默认的事务隔离级别

**2、READ\_UNCOMMITTED(read\_uncommitted)**

读未提交，即能够读取到没有被提交的数据，所以很明显这个级别的隔离机制无法解决脏读、不可重复读、幻读中的任何一种，因此很少使用

**3、READ\_COMMITED(read\_commited)**

读已提交，即能够读到那些已经提交的数据，自然能够防止脏读，但是无法限制不可重复读和幻读

**4、REPEATABLE\_READ(repeatable\_read)**

重复读取，即在数据读出来之后加锁，类似"select \* from XXX for update"，明确数据读取出来就是为了更新用的，所以要加一把锁，防止别人修改它。REPEATABLE\_READ的意思也类似，读取了一条数据，这个事务不结束，别的事务就不可以改这条记录，这样就解决了脏读、不可重复读的问题，但是幻读的问题还是无法解决

**5、SERLALIZABLE(serlalizable)**

串行化，最高的事务隔离级别，不管多少事务，挨个运行完一个事务的所有子事务之后才可以执行另外一个事务里面的所有子事务，这样就解决了脏读、不可重复读和幻读的问题了

## Mysql （优化）

创建索引

什么情况下创建索引：

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**

**什么情况下不应该创建索引：**

**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%)**

索引分类：

* **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/组合索引**

即一个索引包含多个列

创建索引命令：

|  |  |
| --- | --- |
| 创建 | create [unique ] index indexname on table\_name(column)); |
| 删除 | drop index indexname on mytable; |
| 查看 | show index from table\_name\g; |
| 使用alter命令 | alter table tbl\_name add primary key (column\_list) : 该语句添加一个主键，这意味着索引值必须是唯一的，且不能为 null。 |
| alter table tbl\_name add primary key (column\_list) |
| alter table tbl\_name add index index\_name (column\_list): 添加普通索引，索引值可出现多次。 |
| alter table tbl\_name add fulltext index\_name (column\_list):该语句指定了索引为 fulltext ，用于全文索引。 |

分表,分库

Aop做日志记录

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。

之前在做其他项目的时候，日志模块通常都是在控制层结合log4j进行日志的控制台打印及日志文件的存储，除此之外还会把日志信息插入到mysql数据当中存储起来，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，以后维护起来也比较麻烦。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处理上。

具体在做的时候我是这样想的：首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。在这个普通的切面类当中需要注意几点，因为当时在项目中我使用的是环绕通知，所以就会让这个方法返回一个Object的类型值，再者就是在方法上有一个ProceedingJoinPoint类型的参数。

具体在写方法的时候有几个关键的地方，首先因为要获取当前登录的用户，用户的信息是存储在session中的，我但是采用的解决方案是使用，=ThreadLocal+filter来完成在一个普通类中获取当前请求的request对象，进而获取存储在session中的用户信息，ThreadLocal可以把它理解成一个Map，但它的特殊的地方就是它用当前线程充当key，所以在使用的时候，存储信息用set就行了，之所以没有写key，就是因为当前访问的线程就是默认的key，同理取数据get（），当时我封装了一个工具类，工具类中有个setRequest方法，就是将request对象存储到ThreadLocal中，同样还 有个getRequest方法，就是获取当前前程对应的request对象。

之后我会在自定义的Filter中的doFilter方法中，调用工具类的setRequest方法，将之前请求存储在ThreadLocal中，当然还得在web.xml中配置Filter使其生效。接下来我会在日志切面类中通过调用工具类的getRequest方法来获取request，进而通过getSession来获取session，这样就可以取到存在session中的相关用户信息了。

获取了用户信息之后就要记录用户做了什么事请，在这块我们当时的项目是这么规定格式的，就是要记录用户执行了那个类的哪个方法，并且要把执行这个方法时候对应的参数信息也给获取到，比如用户添加了一个商品，那记录的信息就应该是调用了ProductController类的addProduct方法，并且也要获取到添加的商品信息参数，这样才能看得明白。

获取类名和方法名这块我们是通过反射机制，调用ProceedingJoinPoint方法获取的，获取参数信息是通过request.getParameterMap()之后对其进行循环遍历，这样就获取到了提交时候的参数详情。

在这个切面类中还有个最重要的方法就是ProceedingJoinPoint.proceed()。它代表的就是实际要执行的核心业务逻辑，它的返回值就是实际执行方法的返回值，比如刚才说的ProductController类中的addProduct方法，这个proceed()代表的就是addProduct方法，而它的返回值就是控制层中的addProduct方法的返回值。之所以它的返回值类型为Object就是因为不同方法的返回值不一样，但他们都属于Object对象。

接着我会把proceed方法进行try。。Catch。。捕获，然后在这个方法执行完后记录操作成功的日志信息插入到数据库，在catch捕获异常的时候记录错误日志，同时会把异常信息记录数据库。

最后需要在spring的配置文件中配置aop：config以及配置切入点表达式来对控制层中的增删改查方法进行拦截，这里就用到了切点表达式中特殊符号的双竖杠。

在上交任务完成后，我们经理给我说这个东西，做的整体上还不错，但是有个问题就是日志虽然记录了操作哪个类的哪个方法，程序员可以读懂，但业务员根本就不明白，不够人性化，让我的日志记录再改进下。

我通过跟我们团队人员的讨论，最终决定通过自定义注解的方式来完成这个改进。

在自定义日志注解的时候，通过@target设置为Method指明该注解只能用在方法上，通过将@Retention设置为Runtime指明将注解保留至运行时。这样就可以通过反射去获取注解信息。在注解中声明了一个String类型的value来让程序员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Controlelr中的方法上加入自定义注解并且对value进行日志信息的赋值，如@Log（”增加了一条商品“）。

最后在AOP的日志切面类中通过获取方法签名得到Method，通过Method的isAnnotationPresent判断该方法上面是否加入了自定义日志注解，如果是则再通过Method的getAnnotation来获取自定义的日志注解，最后再通过.value()方法获取自定义日志注解中日志信息的值，这样在记录日志信息的时候就可以显示更加人性化的信息。

# 基于token的登录注册

在坐登录注册的业务时，我决定用redis作为注册时发送验证码的缓存，在业务中首先是验证手机号格式

从redis缓存获取用户输入的手机号是否已经发送过验证码，验证手机号是不是已经注册过，防止手机号重复，然后用邮箱的方式给用户发送验证码，在redis的缓存中设置验证码的存储时间为一分钟，

当用户输入验证码传到后台后，从redis获取验证码是否一致，如果一致则为注册成功，当用户点击注册按钮时，发送到后台的用户名则判断是否唯一或者不可以是空值，密码使用MD5加密方式可以保证一定的安全度，在用户注册成功之后会发送注册成功的邮件到用户的邮箱使用了多线程可以，提高了用户体验度，

用户登录时后台根据用户名获取用户数据，判断用户的密码是否正确，用JWT令牌生成token代表用户的身份信息存储到sessionStore中，因为JWT是基于json的一种公开规范，非常方便解在用户退出登录时删除存储的token信息的一套登录注册业务；

# 多线程导出

我们项目中的用户数据是使用的多线程导出压缩包

场景分析（为什么使用多线程)

应为我们的用户数据量庞大，为了防止一次性导出导致内存溢出，和单个excel数据量巨大无法读取的错误，所以我考虑使用多线程导出

将数据进行分批，通过不同的线程，下载到不同的excel下的多个sheet页中

最后统一打包到zip文件中进行下载

通过用户传递的count,size定制用户需要的每个excel需要多少sheet页（count）每个sheet需要多少条数据（size）

首先我们查询数据库用户的总数确定我们需要生成多少个sheet页，这些sheet页需要生成几个excel，为了保证我们的cpu资源的高利用率，提升文件的下载速度，我使用了多线程的方式进行下载，但我们最后要统一的返回一个zip压缩包，我们的主线程需要等待分支线程全部完成才能进行打包操作，这里我使用了countDownLatch，它就是声明一个变量，主线程中使用await()方法等待分支线程，每当我们一个支线程完成，通过countDown()方法进行-1，当countDownLatch为0时，在进行主线程操作

我也创建了线程池，来回收线程资源，节约线程频繁创建和销毁的资源损失

在分支线程中，我需要sheet页个数（size），几个sheet页（count），第几个excel（num），countDownLatch这些参数，这里我的线程工具类没有交给ioc去管理无法注入dao层，所以我们页通过参数传递dao层接口进去

首先创建workbook对象

根据count和num我们可以计算出页数下标，循环count次，创建count个sheet页

然后查询不同sheet页的数据

遍历数据填补到对应的row行中，全部循环结束则线程对应的一个excel生成完毕

new一个FileOutputStream流将workbook写入我们项目中的file临时文件中，关流

通过countDownLatch.countDown();告诉主线程完成

全部分支线程完毕，主线程将所有临时文件生成一个zip下载，并删除所有的临时文件

# 权限管理 5表

原理:（可说可不说）

设计权限管理系统的时候，会创建一张t\_permission权限表。表的字段有name,id,pid,type,url。其中核心字段pid为了实现一个权限树的结构（递归）。type用来识别权限级别是菜单还是按钮。url是控制层请求路径，是控制权限的核心。

实现步骤：

1. 首先我们在用户登录成功之后获取当前登录用户所拥有的权限集合，并放入session对象中。然后取出后台管理系统中所有的权限集合也放入session对象中。

2. 然后我们需要新建一个登录拦截器类，继承HandlerInterceptorAdapter(憨德乐 因特塞铺特 额戴普特)类，并重写preHandle(铺rei翰得)方法。

3. PreHandle方法中我们首先从session对象中获取当前用户所拥有的权限集合和所有的权限集合。

4. 接下来我们要拿请求的url和所有权限集合中的每一个权限的url去比较，从而去判断当前这个请求url是否需要去拦截，如果当前请求的url没有匹配上所有权限集合里的每一个权限的url，则说明当前请求的url不需要被拦截接下来就可以放行了。 如果当前请求的url匹配到所有权限集合中的任意一个权限的url，则说明当前请求的url需要被拦截。

5. 接下来我们会判断当前用户是否拥有该url的权限，拿当前请求的url去和用户权限集合中的每一个权限url去比较，如果匹配到则说明该用户有该请求url的访问权限那么我们就放行，否则的话我们又判断了当前请求是否为ajax请求，如果是ajax请求，则通过resposne对象返回给客户端一个状态码2000，然后在页面的ajax全局设置请求完成回调函数里判断服务器返回的状态码是否为2000，如果为2000则通过bootbox弹出一个警告框”您没有操作权限!”。如果不是ajax请求则直接重定向到我们自己建的无权限页面。