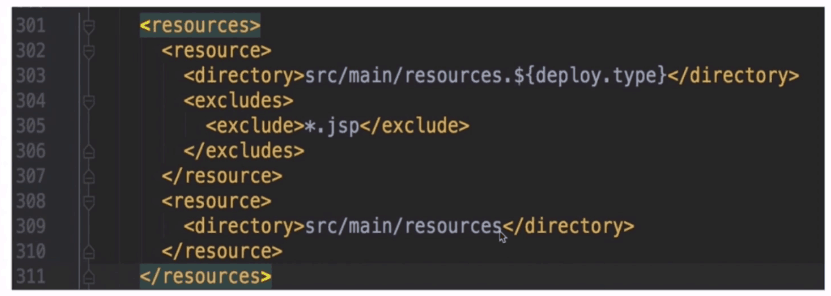
**1.本地项目使用cmd命令行打包，打包命令为：**

首先cd到项目所在目录，在执行命令

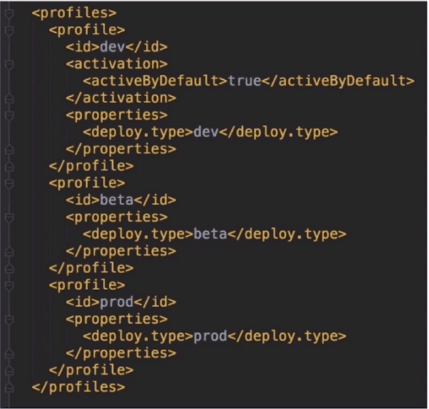
mvn clean package –Dmaven.test.skip=true (后面表示的是跳过测试)

**2.maven环境隔离配置**

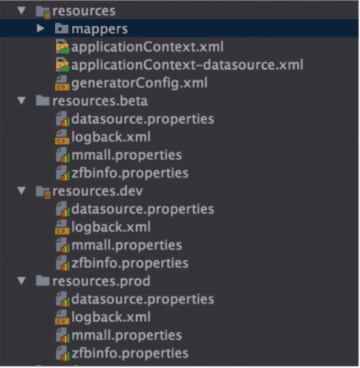
在pom.xml中build节点增加



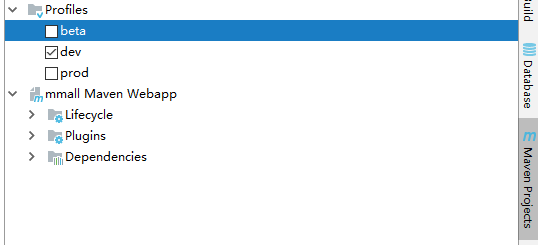
在pom.xml中增加profiles节点（profiles节点和build节点同级）



新建对应的文件夹，并把要隔离的文件分开，公共的留下



**3.maven环境隔离IDEA中设置默认环境**



**4.tomcat集群**

通过Nginx负载均衡进行请求转发。

Tomcat集群的问题：

（1）Session登录信息存储及读取的问题

（2）服务器定时任务并发的问题

解决方案：

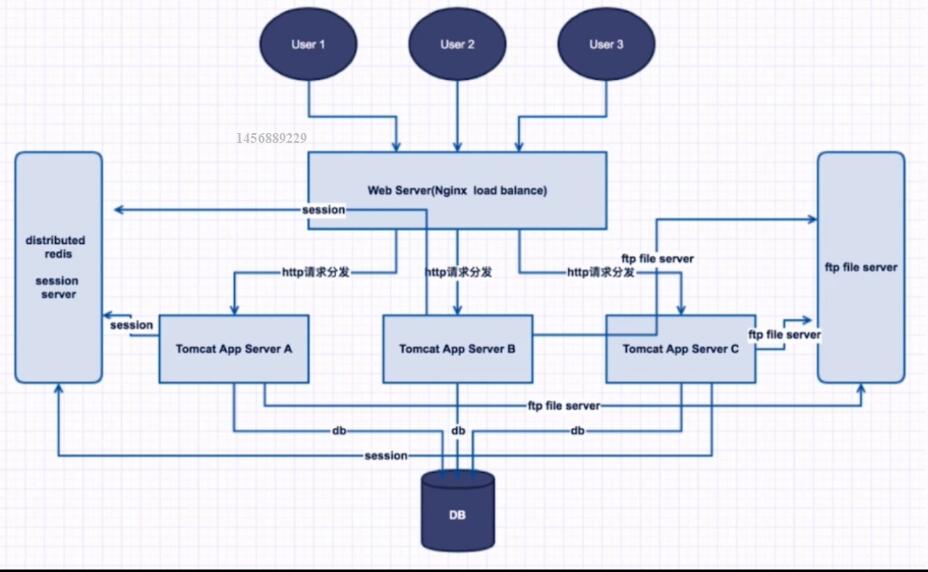
1. 采用nginx ip hash policy

优点：可以不改变现有技术架构，直接实现横向扩展（省事）

缺点：1. 导致服务器请求（负载）不平均（完全依赖ip hash）的结果

2. 在IP变化的环境下无法服务

二期真架构



**5.Nginx负载均衡配置。常用策略。**

① 轮询

优点：实现简单

缺点：不考虑每台服务器处理能力



1. 权重

优点：考虑了每台服务器处理能力的不同



1. ip hash

优点：能实现同一个用户访问同一个服务器

缺点：根据ip hash不一定平均



④url hash（第三方）

优点：能实现同一个用户访问同一个服务器

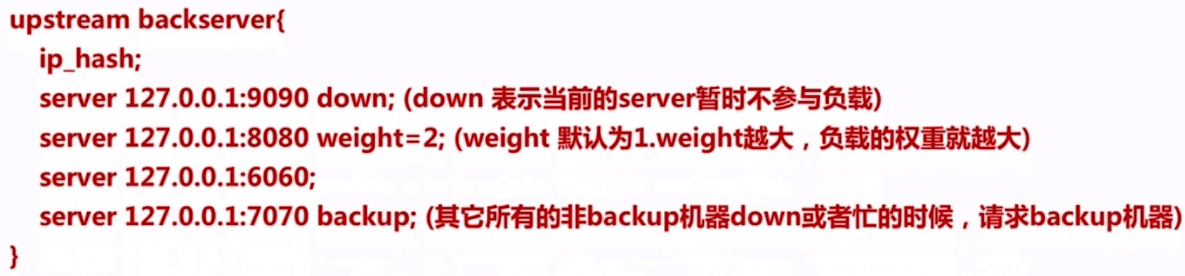
缺点：根据url hash分配请求会不平均，请求频繁的url会请求到同一个服务器上



⑤fair（第三方）

优点：按后端服务器的响应时间来分配请求，响应时间短的优先分配

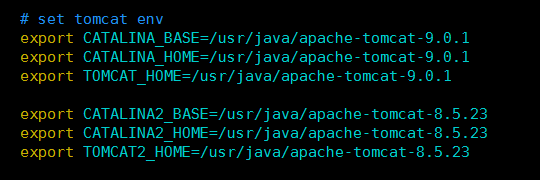




**Tomcat集群配置**

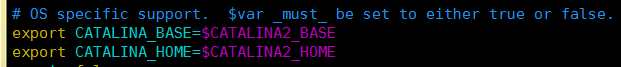
首先 /etc/profile增加tomcat环境变量

Tomcat 解压后

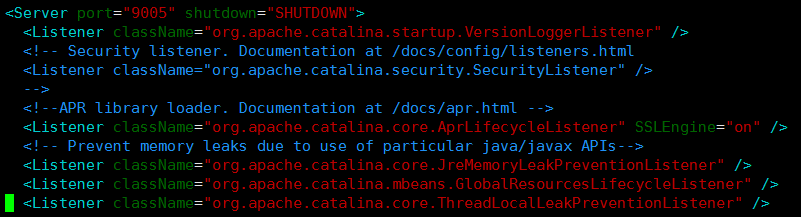


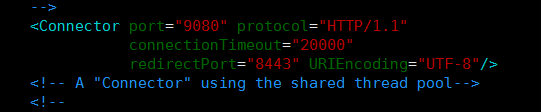
并要source /etc/profile使配置文件生效

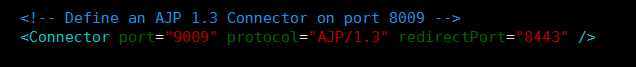
打开第二个tomcat目录bin下catalina.sh并找到 #OS specific support. $var\_must\_be set to either true or false. 在这行下面编辑（我的第二个tomcat是8.5.23）



打开第二个tomcat conf文件下的server.xml，修改三个地方的端口。



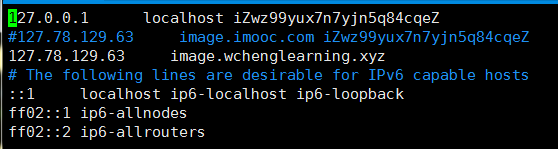


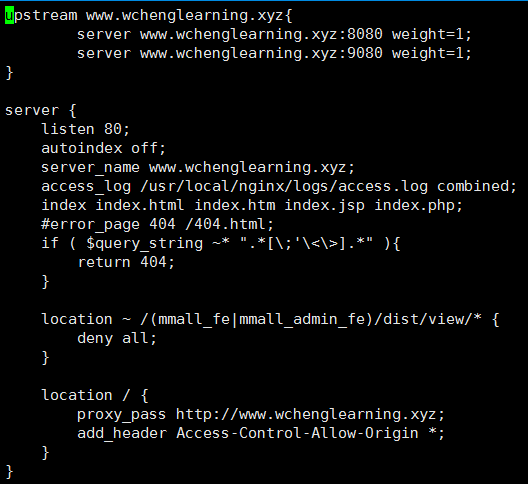


**Nginx + tomcat搭载集群配置：**

在Nginx下的conf文件下新建一个loadBalance.conf文件，编辑配置即可

首先cd /etc 编辑 vim hosts文件。添加自己的域名解析，然后配置负载均衡





**Redis的安装**

**1.linux下的安装**

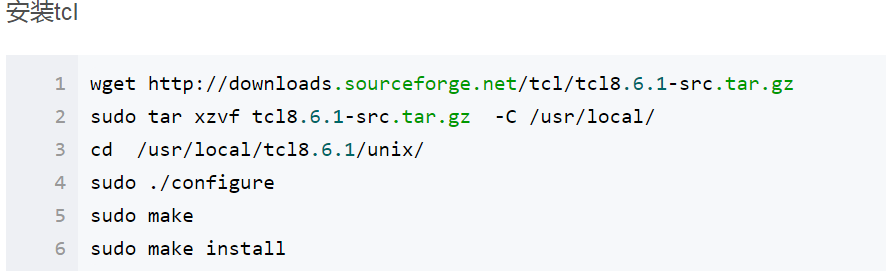
首先下载redis的压缩包：<http://download.redis.io/releases/>

解压到相应的文件夹 tar –zxvf redis-2.8.24 –C /usr

然后 cd /usr/redis-2.8.24 输入命令make

make完后会生成一个src目录，这时测试有没有编译成功，输入命令make test

如果出现报错You need tcl 8.5 or newer in order to run the Redis test，则按照如下步骤操作：



安装成功后启动redis ： ./redis-server 这时redis等待连接，新开一个窗口进入redis/src

输入./redis-cli就可连接成功。但是此时窗口是被占用的。按下Ctrl+C即可退出窗口占用。让redis后台启动 输入命令 ./redis-server &

Redis的关闭命令: ./redis-cli shutdown

2.windows下的安装

因为redis官方没有提供windows版本的，windows版本都是由微软团队进行维护的。所以，先下载redis windows版本：<https://github.com/MicrosoftArchive/redis/releases>

下载完解压缩即可用命令和linux下一样

Linux下redis启动：

默认启动 ./redis-server

默认连接 ./redis-cli

默认关闭 ./redis-cli shutdown

在指定端口启动 ./redis-server --port 端口号

在指定端口连接 ./redis-cli –p 端口号

在指定端口关闭 ./redis-cli –p 端口号

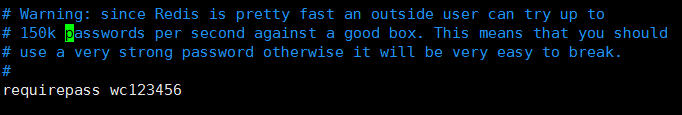
通过修改配置文件启动：./redis-server ../redis.conf

指定IP连接：./redis-cli –p 6379 –h 127.0.0.1 （如果远程连接的话，可以把ip改成远程ip）

指定ip关闭：./redis-cli –p 6379 –h 127.0.0.1 shutdown

配置redis密码：

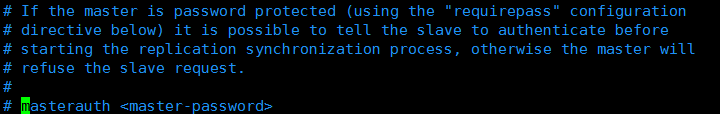
在redis.conf文件夹下找到requirepass 后面即为自己的密码



启动时就要用配置文件方式启动：./redis-server ../redis.conf

连接时就要用密码连接：./redis-cli –p 端口号 –a wc123456

当使用分布式时，有主从服务器，需要主从服务器的密码一致。先注释requirepass，找到masterauth 后面的即为自己的密码



Windows下的这些启动方式一样

在使用redis时，由于自己操作有问题，从github上拉取项目时出错了，导致项目运行时报错，即使不报错了，再登录时会报redis pool初始化失败，不合法的Integer.parseInt()…真的是头都到了，各种找错误都没用，最后发现要target下的mmall properties是最开始的，重新clean package下就好了……

小结：

首先因为本地、测试、线上的环境不同，所以配置了maven环境隔离，方便开发。其次，当项目越来越大时，肯定需要多台服务器(多个tomcat)，于是配置了nginx的负载均衡，然后，一些常用的数据不需要每次都从数据库里取出来，因此采用了redis，首先封装一个jedis对象，重写其中的方法。调用时，起初只用redis缓存用户信息，但缓存信息时采用的是sessionId作为键，每次刷新或重启服务器或者访问到其他服务器时，sessionId都是会变的，所以redis里的值获取不到。除非不刷新，总是请求到这台服务器。这时的解决办法：是采用cookie存值，然后redis从cookie里取值。但是新的问题又会出现，怎么让用户访问新的网页时刷新redis里的过期时间，因此新增了一个sessionExipreFilter.每次访问新的.do方法时都会刷新过期时间。（因为我们取数据是从redis里取的，虽然cookie也有过期时间，但是只要redis第一次从cookie获取到值后，之后的cookie就可以不用管它了，因为之后的时间我们都是从redis获取数据。）这样的问题是，当用户多了，访问量多了，cookie越来越大，占用磁盘空间？

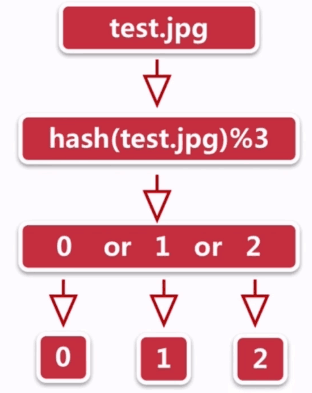
3、从hash→consistant hash→consistant hash + 虚拟节点

首先hash

什么是hash：hash（散列、杂凑）函数，是将任意长度的数据映射到有限长度的域上。直观解释起来，就是对一串数据m进行杂糅，输出另一段固定长度的数据h，作为这段数据的特征（指纹）。

取余：

对于一个对象，hash取余后（3 表示redis服务器的个数）存在取余后的值对应的服务器上



优点：数据分布均匀

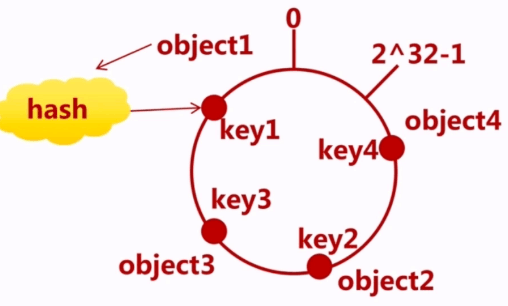
缺点：当服务器动态的添加、删除节点或者某台服务器宕机，会导致命中率超大幅度下降，甚至导致服务不可用。

Consistant hash：它是一种分布式哈希（DHT）实现算法，设计目标是为了解决因特网中的热点问题，初衷和CARP十分类似。

它按照常用的hash算法将对应的key哈希到一个2^32次方个值的环形空间中，即0 ~ （2^32-1）的环形数字空间中，这些数字头尾相连。

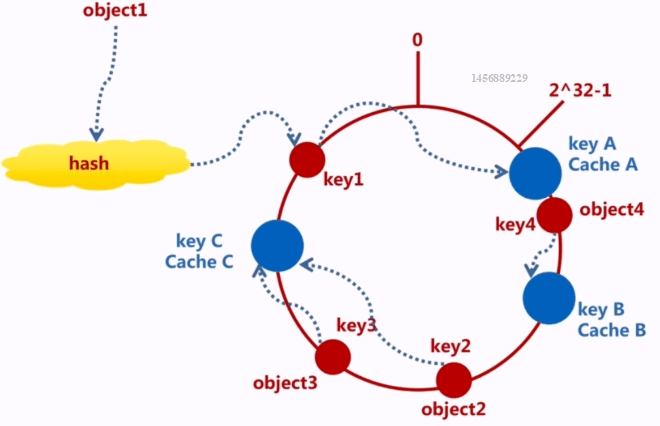


然后把对象映射到hash空间



然后把服务器通过hash算法映射到hash空间

按照顺时针的顺序把把key放到最近的cache



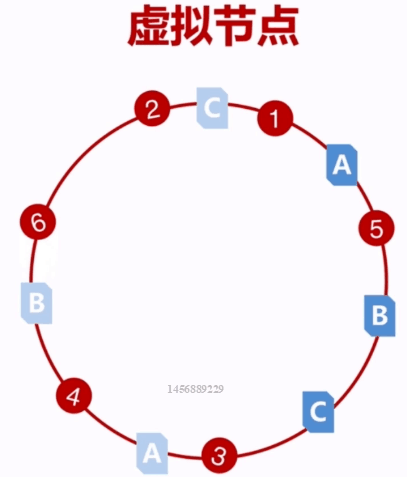
优点：动态的增删节点，服务器宕机时，影响的只是顺时针的下一个节点。

缺点：因为存在hash倾斜性，会导致服务器分布不均，进而导致key的分布、服务器的压力不均匀。若中间某一权重较大的服务器宕机后，命中率下降明显。



Consistant hash + 虚拟节点

引入虚拟节点解决consistant hash分布不均导致负载不均的问题。一个真实节点对应若干个虚拟节点，当key被映射到虚拟节点上时，则被认为映射到虚拟节点所对应的真实节点上。



命中率公式：（1-n/(n+m)）\*100%，其中服务器台数是n，新增的服务器台数是m。

4、Redis分布式 ：

新建了一个RedisShardedPool 初始化redis分布式池，然后新建工具类封装其中的一些方法，包括getResource和returnResource等等。然后之前调用RedispoolUtil的地方都替换为RedisShardedPoolUtil

分布式和集群的区别：

集群：是个物理形态。同一个业务，部署在多个服务器上。

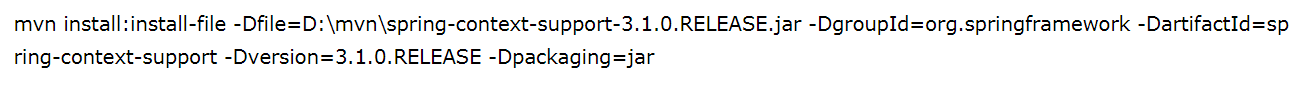
分布式：是个工作方式。一个业务拆分为多个子业务，部署在不同的服务器上。

集群的每一台服务器实现的功能没什么区别。分布式每一台实现的功能是有差异的。

遇到一个问题 怎么把本地jar 打包到maven仓库。

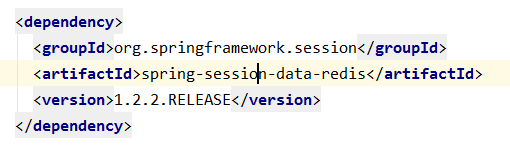
mvn install:install-file –Dfile=jar包的位置 –DgroupId=自己命名的Id –DartifactId=自己命名的Id –Dversion=自己的version –Dpackaging=jar

例如：

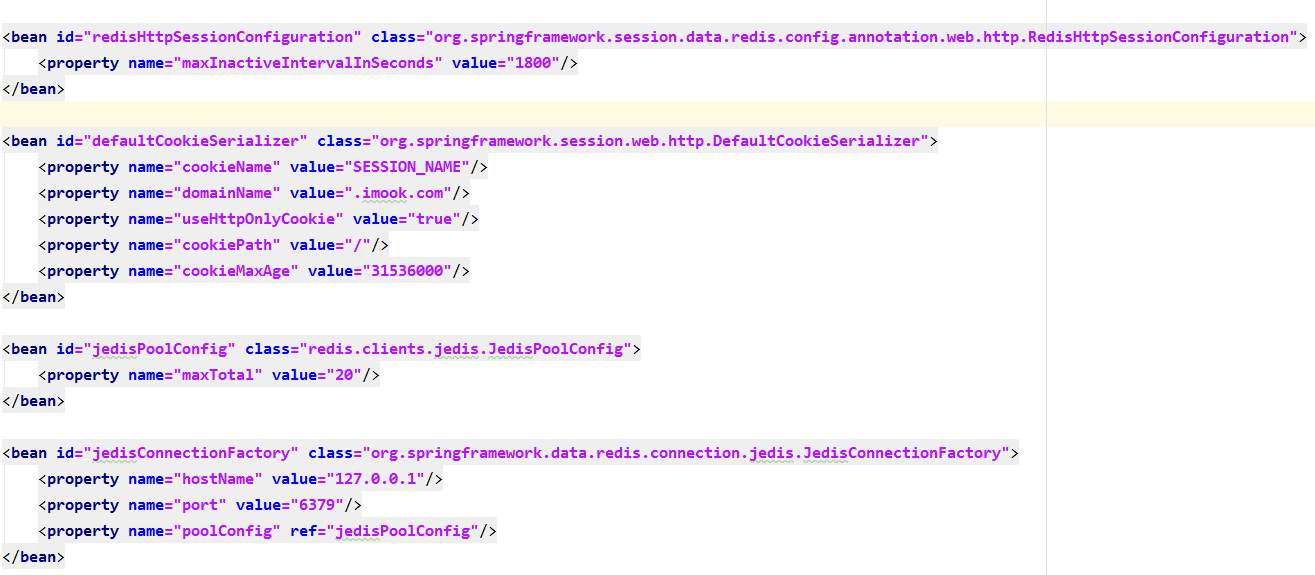


5、SpringSession的全局实现单点登录

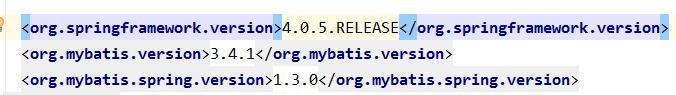
首先需要引入spring-session-data-redis依赖：

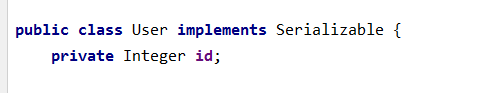


然后配置spring-session，新建一个applocation-context-spring-session.xml，配置相应的Bean对象：

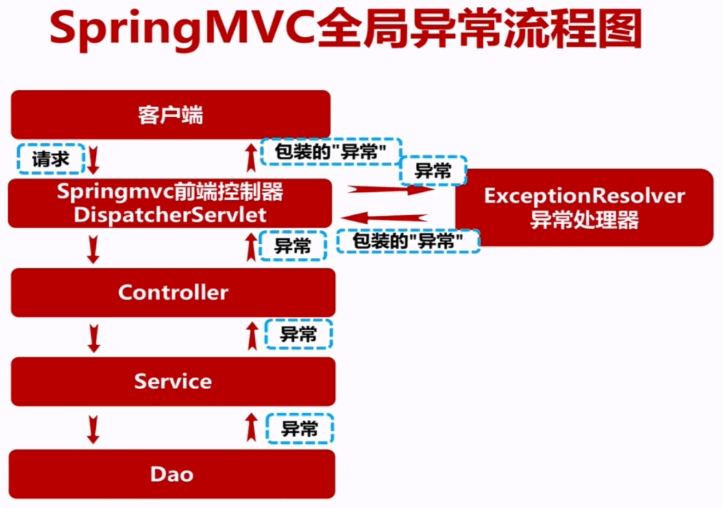


需要注意的是org.springframework.version需要配置到4.0.3及以上，否则会报异常，并且在实体类里要继承Serializable，否则无法存储session





6、SpringMVC全局异常流程图

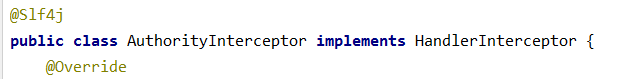


<context:annotation-config/> 对已经注册的bean进行操作的配置

自定义异常时，一定要在自定义的类上加上@Component注解（表明这是一个bean对象），这个注解和@Repository,@Service一样。习惯在特定的层才加上的@Repository,@Service等，其余的地方用@Component注解。让我想到了自己自定义工具类时，要用到其实，比如需要引用Repository，但是@Repository时注入的却是空，是不是需要在自定义的工具类上加上@Component注解？

7、Springmvc拦截器

首先需要新建一个类继承HandlerInterceptor：



重写它的三个方法，然后在dispatcher-servlet.xml配置拦截器：



此时拦截器就生效了，然后在新建的类里编写相应的逻辑，需要注意的几点：

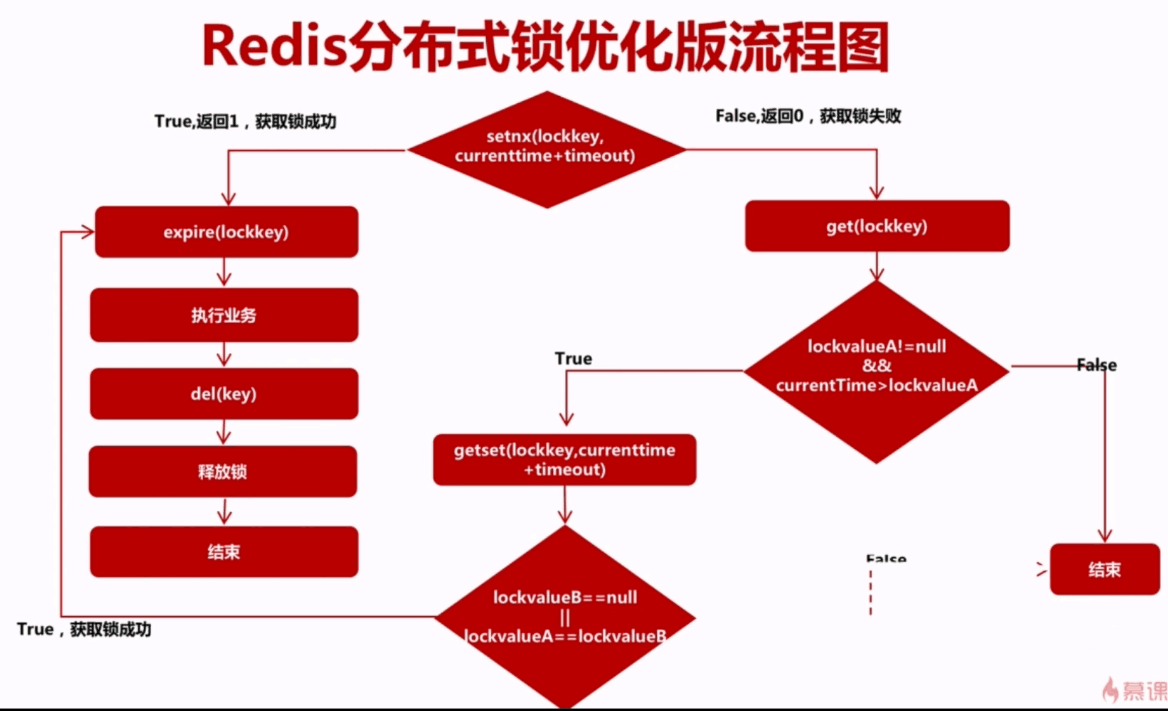
①需要重置HttpServletResponse，否则会报异常

②解决拦截登录循环问题，即登录的接口不需要被拦截，否则登录不上

1. 富文本上传拦截器处理，因为不同的返回类型需要不同处理。

④拦截器相关代码重构，把所有的验证的代码都用拦截器替代。

8、spring schedule + redis 分布式锁



在使用分布式锁的时候,首先通过setnx获取锁，如果获取到锁则执行相应的逻辑，执行完后，就释放锁。如果没有获取到锁，则继续判断，之前的锁存不存在，不存在则重置锁，如果存在但是它的有效期已经小于当前时间，也重置锁，然后执行相应的逻辑。否则获取不到锁。

其实在程序中是开启一个线程来执行这个定时任务的，当这个定时任务在执行时，redis就会setnx一个键值对(setnx（set if not exists）表示如果存在则返回0，如果不存在则设置键值对)。正好对应着：如果该任务在执行中（就要设置这个键值对），则表示这个锁正在被使用，其他线程不能调用。如果该键值对不存在，则说明有任务已经结束，另一个线程可以获取锁执行这个任务了。（锁是自动获取的）。

9、Redisson

首先引入redisson及相关的jar包,注意版本（不知道版本怎么适配,当我用到了2.10.0时，会报错）

<**dependency**>  
 <**groupId**>org.redisson</**groupId**>  
 <**artifactId**>redisson</**artifactId**>  
 <**version**>2.9.2</**version**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>com.fasterxml.jackson.dataformat</**groupId**>  
 <**artifactId**>jackson-dataformat-avro</**artifactId**>  
 <**version**>2.9.2</**version**>  
</**dependency**>

引用redisson，当然就要初始化一个redisson对象，新建一个RedissonManager类，配置相应的config，通过@PostConstruct注解来初始化，也可以通过一个静态块初始化这个配置。然后通过@Component注解注入bean,以便其他类调用。然后就可以通过定时任务关单了。Lock.trylock()方法相当于把之前通过redis.setnx()这一系列方法封装了。Redisson的官方文档：

<https://github.com//redisson/redisson/wiki/>目录

https://github.com//redisson/