# 一、Lombok框架

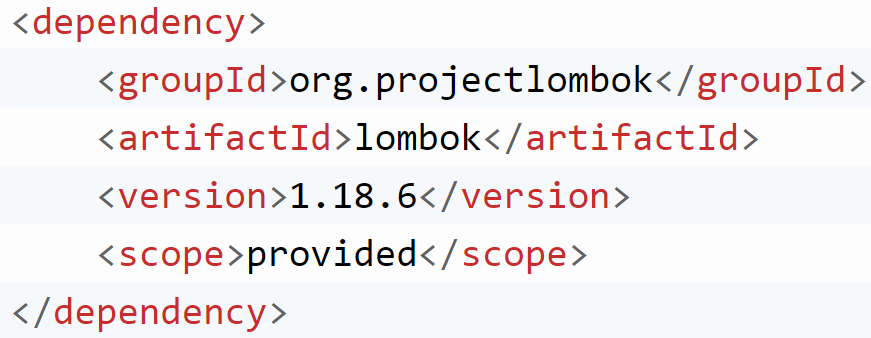
## 1. Lombok的作用

如果用一句话来介绍lombok的话，那么lombok的作用就是通过简单的注解来精简代码以达到消除冗长代码的目的。

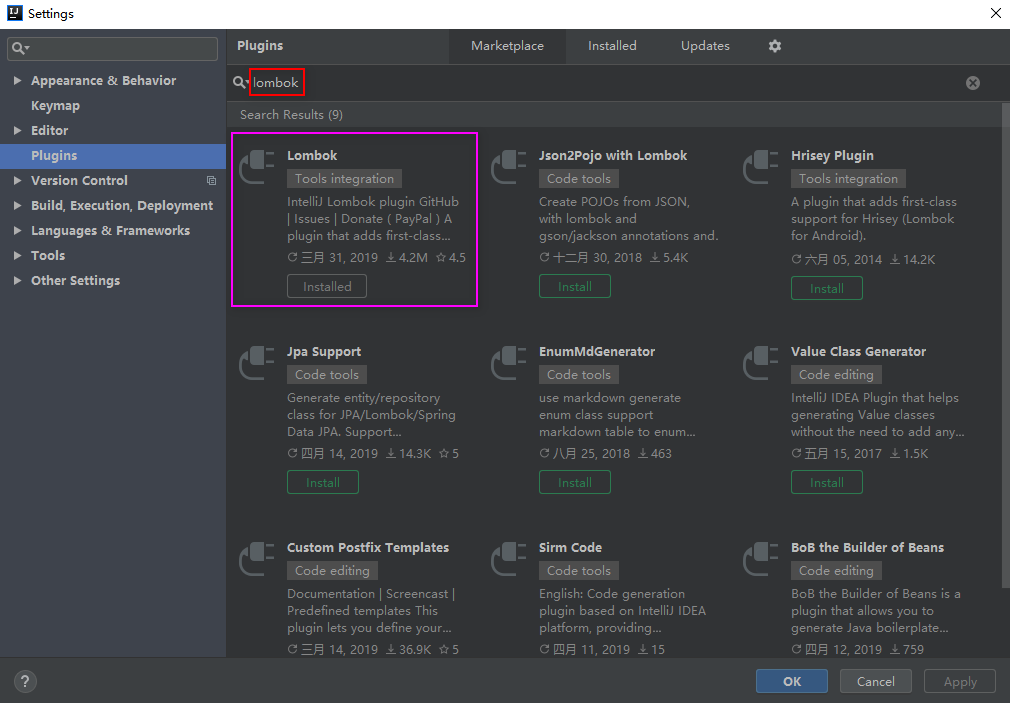
## 2. Lombok的优点

* 提高编码效率；
* 使代码更简洁；
* 消除冗长代码；
* 避免修改字段名字时忘记修改方法名。

## 3. pom.xml中关于Lombok的依赖

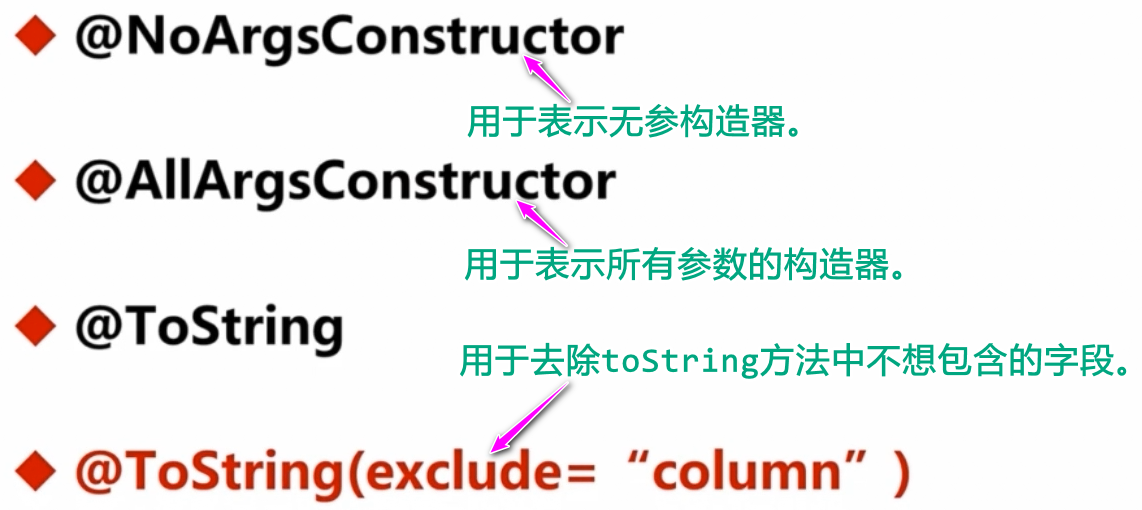


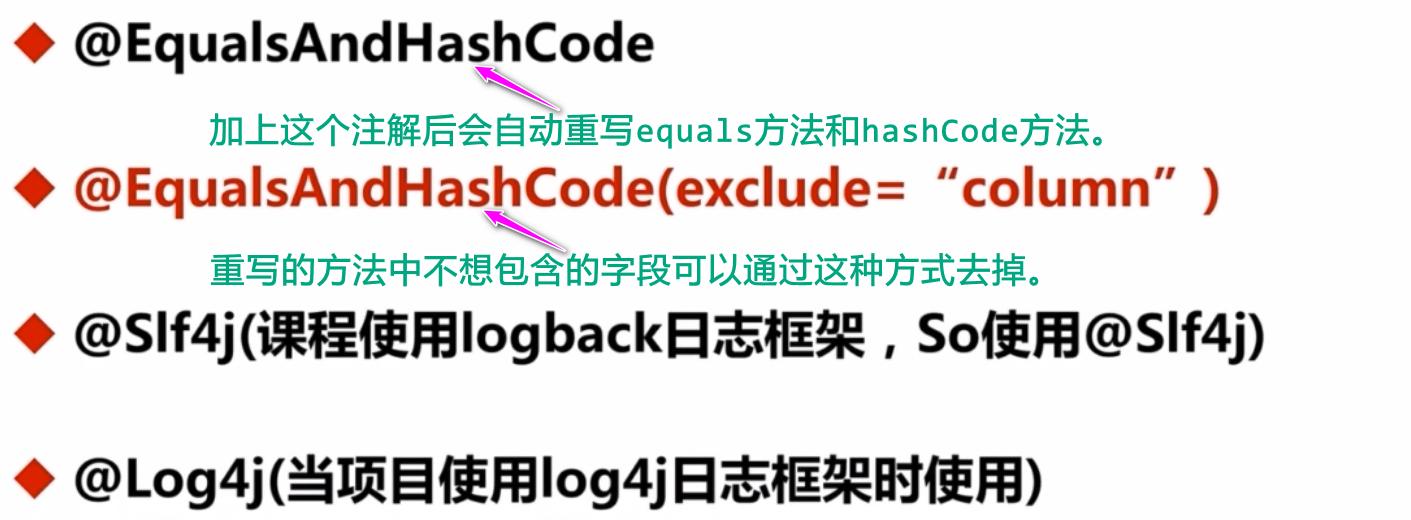
上面的scope标签有没有都可以，都不影响。另外，如果是在IDEA中使用lombok框架的话，对于maven项目，除了要在pom.xml中引入依赖外，还要在IEDA中安装相应的插件，如下图所示。

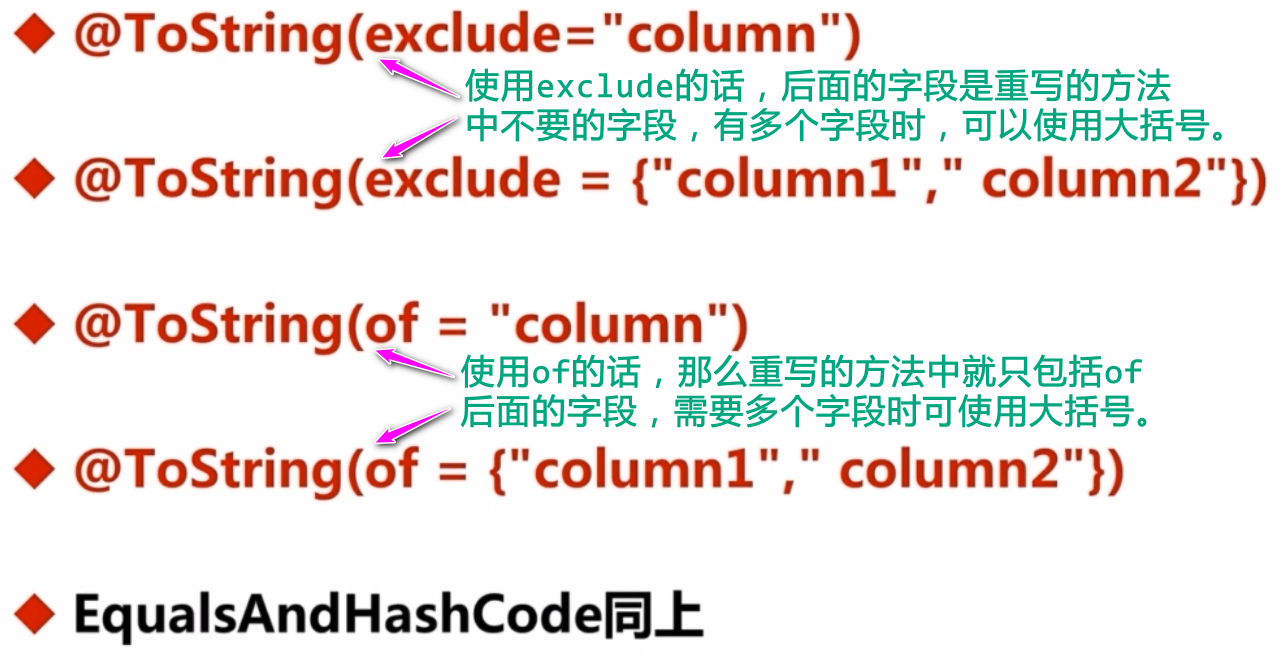


## 4. Lombok的常用注解





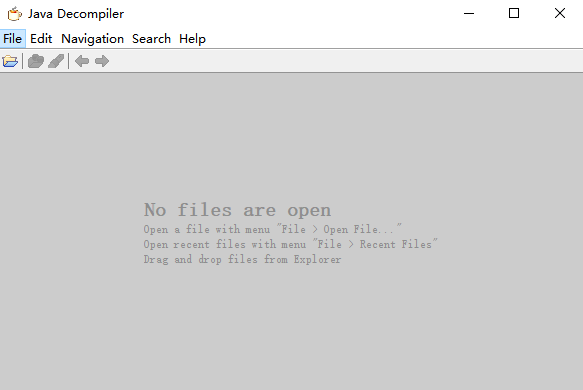






# 二、反编译工具

这里要说的反编译的工具就是java Decompiler，我们可以使用这个工具把class文件进行反编译，从而查看到对应的源码文件，该工具的界面如下。



# 三、maven的环境隔离

## 1. 使用maven环境隔离的原因

自己在本地开发时几乎是不需要使用到环境隔离的，但是在实际的项目开发中进行环境隔离是很有必要的。因为实际的项目开发中是会有多个环境的，不同的环境是会存在一定差异的，为了让这些环境之间互不影响，这时就需要用到Maven的环境隔离了。

实际的项目环境一般包括本地开发环境（Local），开发环境（Dev），测试环境（Beta）以及线上环境（Prod）。由于我是在本地开发，本地开发环境和开发环境是一样的，所以下面在配置的时候，本地开发环境就不配置了。

|  |
| --- |
| 隔离的各个环境之间的各种配置可能存在的一些常见差异：   * FTP服务器相关的配置信息不一样； * 数据库配置不一样； * 支付接口不一样，例如支付宝分沙箱环境和真实环境，我们在开发的时候可以用沙箱环境，但是上线后就要用真实环境了； * 框架的配置文件不一样，例如日志的输出路径等等。 |

以上提到的只是一部分的差异，我们在本地开发时，明明项目是可以正常运行的，但是一到线上就是各种报错，这就是线下的开发环境和线上环境不同导致的。由于环境不同，所以在本地开发的项目，即便在本地能正常运行，但是在发布到线上后，很多配置的内容其实还是需要进行修改的，否则就会报错。

|  |
| --- |
| maven环境隔离解决的实际问题：   * 避免人工修改的弊端，因为人为修改容易犯错； * 轻松分环境编译、打包、部署等等； |

我们把由于环境不同而可能内容不同的配置文件拷贝成多份，比如开发环境一份、测试环境一份、线上环境一份。然后在开发环境下需要进行打包等操作时，就把开发环境中的那份打包进来；如果是线上环境需要进行打包等操作时，就把线上环境的那份打包进来，这样一来就完成了分环境进行打包等操作。如果不是拷贝多份的话，就需要根据环境的不同来修改配置文件的内容，这样就会很容易出错。

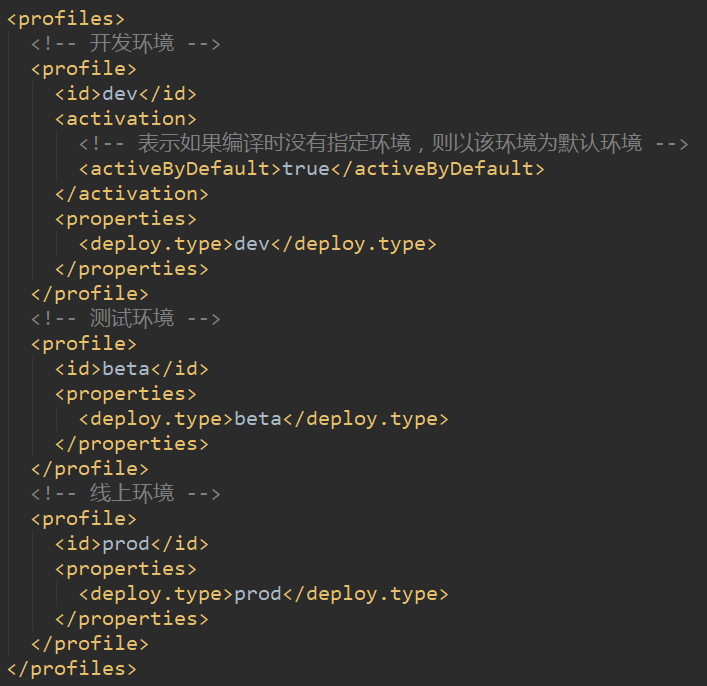
## 2. 配置maven环境隔离的步骤

### 2.1 pom.xml中的配置

①首先在build节点中增加如下配置：

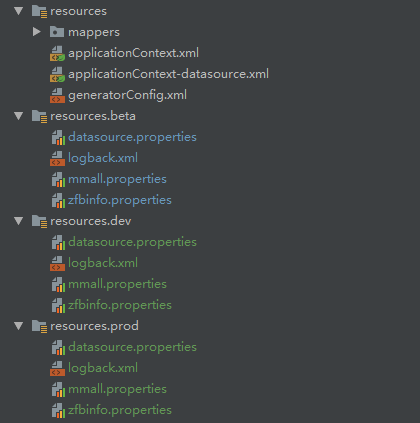


②然后在pom.xml中增加profiles节点，并作如下配置：

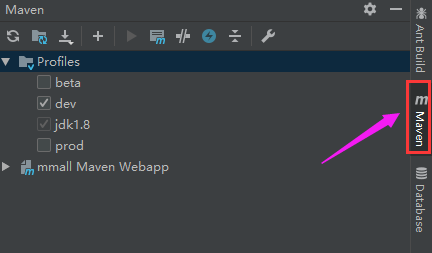


### 2.2 新建针对不同环境的文件夹

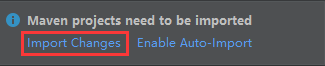
这里把可能会因为环境不同而内容不同的配置文件分别拷贝到不同环境的文件夹下，如下图所示。拷贝好之后，再根据环境来修改对应该环境的目录下相应的配置文件的内容。



### 2.3 在IDEA中设置默认环境



目前使用的环境是开发环境（dev），需要注意的是，这里的对勾分实心的对勾，还有半实心的对勾，像下面jdk1.8前面的对勾就是半实心的，而我们在设置默认环境的时候一定要打实心的对勾。如果我们想要更改默认环境的话，就在这里更改就可以了。每次更改的时候，上图的下面会出现下图中所示的内容。这时要点击我标记的地点，以便能够保存更改。



### 2.4 执行打包等命令

如果要对项目执行打包操作的话，就在命令提示符中输入以下命令：

|  |
| --- |
| mvn clean package -Dmaven.test.skip=true -Pdev |

这个命令是针对开发环境的，如果是测试环境的话，以上命令其他部分不变，最后面的部分变成-Pbeta，同理，线上环境的话，就变成-Pprod。如果是使用以下命令的话，就代表使用默认环境进行打包，而之前在pom.xml中配置的默认环境是开发环境（Dev）。

|  |
| --- |
| mvn clean package -Dmaven.test.skip=true |

# 四、Tomcat集群与nginx负载均衡

## 1. 基本概念

我之前在作总结的时候，有专门讲过集群、分布式以及负载均衡的区别。我对nginx负载均衡的理解就是，当浏览器中的请求到达nginx的时候，由于与nginx相关联的tomcat服务器有很多，所以nginx会把用户的请求随机分发给其中的某个服务器来处理，由于请求是随机分发给任意一个服务器来处理的，所以不会出现过多请求都交给某一个服务器来处理的情况，这一行为就是nginx的负载均衡。这样做的好处是不会使某一个服务器的负载过重，从而就达到了负载均衡的目的。不过由于每个服务器的性能可能不一样，所以我们也可以通过权重的方式，把更多的请求分发给性能较好的服务器来处理。

与nginx相关联的多个tomcat服务器就叫做tomcat集群，其实集群可以理解成是很多人在干同一件事。就是由于请求太多了，一个服务器不够用，所以才会用多个服务器一起来处理请求，这其实就是集群。下面就主要讲一些关于集群相关的知识。

## 2. Tomcat集群的作用和实现原理

### 2.1 Tomcat集群的作用

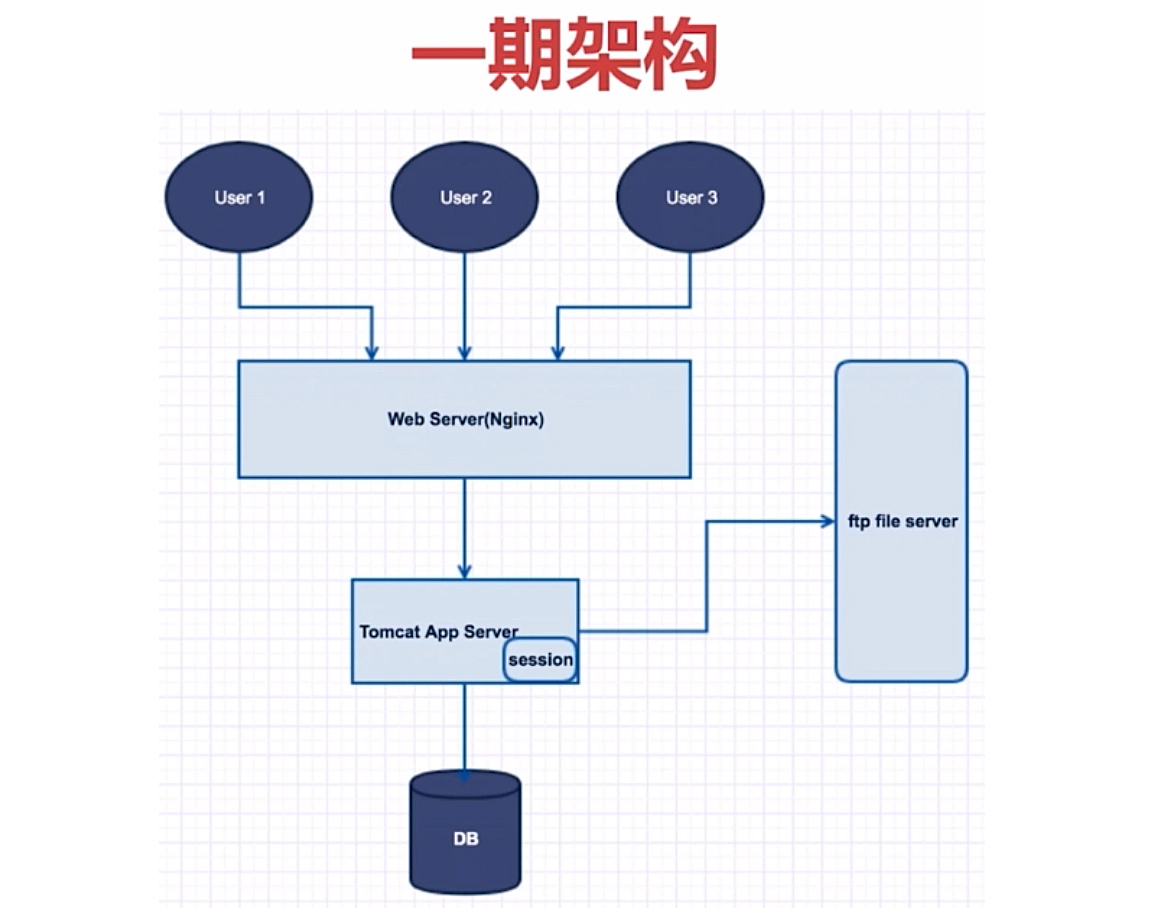
* 提高服务的性能，并发能力以及高可用性；
* 提供项目架构的横向扩展能力。

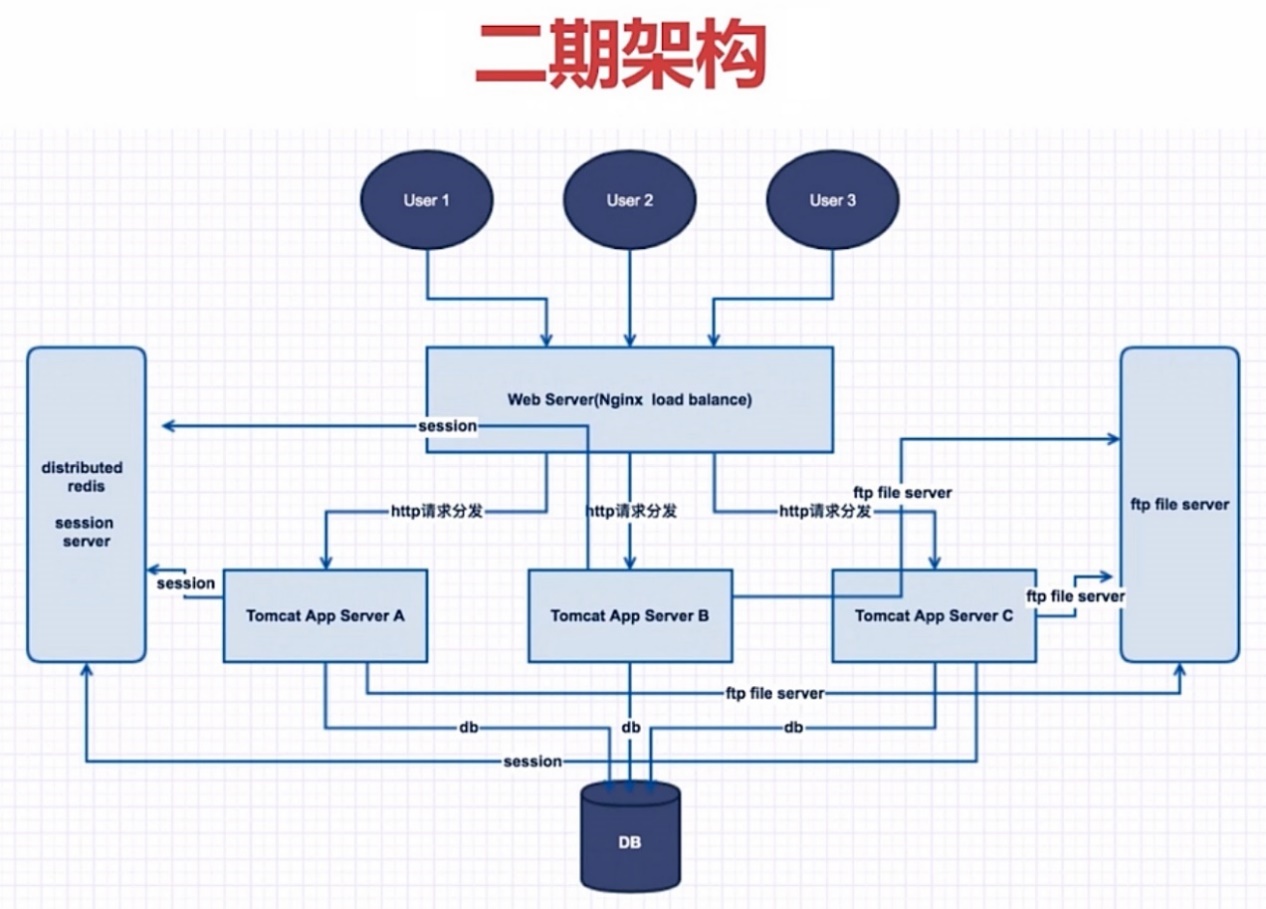
### 2.2 Tomcat集群的实现原理

* 通过nginx负载均衡进行请求转发。

注：tomcat集群不一样就只有以上提到的两个作用，也不一定就上面提到的一个实现原理，我这里记录的都只是老师在课上提到的。

## 3. 一期架构和二期架构对比





## 4. nginx负载均衡的常见策略







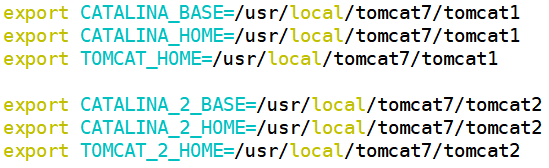
以上关于权重这一策略，我这里特别讲一下，该项目中使用的就是这种策略。上图中在使用weight的时候，分别使其等于15和10。这里需要说明的是，权重表示的只是概率问题，weight=15表示请求分到该服务器上的概率是weight=10时的1.5倍。并不是说有25个请求就一定有15个被分到weight=15所在服务器上。

## 5. 集群和负载均衡的具体步骤

在叙述详细步骤前先说一下，我下面步骤中演示的tomcat和nginx都是在linux环境下，不过在访问的时候，是在windows下的浏览器中进行访问的。

### 5.1 配置环境变量

首先在linux的/usr/local/tomcat7目录下增加两个tomcat，分别命名为tomcat1和tomcat2，然后执行vim /etc/profile打开profile文件，打开后在该文件中增加下图中这些内容。



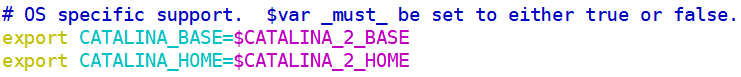
以上配置的意思是在集群中设置两个tomcat服务器，如果还想配置更多的话，就按照图中的方式依次加上就可以了。保存退出后执行source /etc/profile使配置生效。如果不执行这个命令，而是重启虚拟机的话，也可以生效。

注：如果之前在profile文件中配置过tomcat，记得给注释掉，以免对这里配置tomcat集群产生影响。

### 5.2 修改tomcat2中的catalina.sh文件

这里tomcat1的catalina.sh文件不进行修改，只修改tomcat2的catalina.sh文件，具体位置为：/usr/local/tomcat7/tomcat2/bin/catalina.sh。

打开tomcat2的bin目录下的catalina.sh文件后，在下图的注释下面新增两行内容，新增的具体内容如下图所示。新增成功后，保存退出。

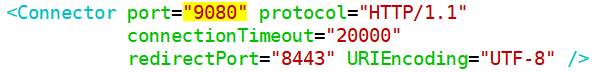


### 5.3 修改tomcat2中的server.xml文件

修改完catalina.sh后，来到tomcat2的conf目录下，修改该目录下的server.xml文件，主要是修改三个端口。首先是把下图中的8005端口改成9005。



然后是把下图中的8080端口改成9080端口。



最后是把下图中的8009端口改成9009端口。



修改完成之后，保存退出即可。需要提一下的是，以上三个端口不一定非要改成图中的那三个端口，改成其他的也行，但是必须要保证端口没有被占用。

### 5.4 分别进行访问测试

启动tomcat1的bin目录下的startup.sh以及tomcat2的bin目录下的startup.sh。在启动之前不要忘记在linux的防火墙中开放这里修改的9080端口。另外，为了看出两个tomcat的区别，可以把tomcat2的webapps/ROOT目录下的tomcat.png图片换成其他的图片，但是名字还叫这个，以便观察区别。修改好之后，就可以在windows下的浏览器中分别进行访问了。

由于我linux系统的ip是192.168.31.18，所以访问地址为192.168.31.18:8080和192.168.31.18:9080。

### 5.5 修改windows以及linux中的hosts

由于tomcat以及nginx都安装在了linux中，而我是在windows的浏览器中进行访问的，所以必须要修改windows的hosts文件，在windows的hosts文件中添加下图中所示的内容即可，下图中的192.168.31.18是linux系统的ip地址。



至于为什么要修改linux中的hosts文件，这个我在下面会讲到，其实不修改linux中的hosts文件也是可以的。如果要修改的话，步骤如下。首先通过vim /etc/hosts命令打开hosts文件，然后在hosts文件中加上下图中内容，保存退出即可。



### 5.6 配置nginx的conf文件

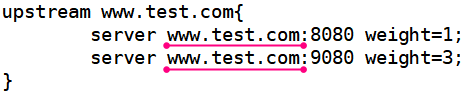
由于在一期的项目中，已经在/usr/local/nginx/conf目录下创建了vhost目录，且已经在/usr/local/nginx/conf目录下的nginx.conf文件中使用include包含了vhost目录中的所有文件，所以这里直接操作vhost目录中的文件即可。首先在vhost目录下新建一个www.test.com.conf文件，然后给该文件添加内容，添加的内容如下。



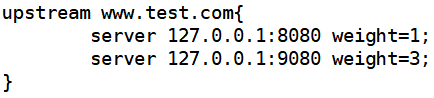
添加完成之后，保存退出。如果之前没有启动nginx的话，启动nginx。如果已经启动了的话，就在/usr/local/nginx/sbin目录下通过./nginx -s reload命令重启nginx。然后还可以通过./nginx -t命令测试是否重启成功。

Nginx启动成功后，由于之前已经启动了tomcat1和tomcat2，所以可以在windows的浏览器中访问www.test.com，然后不停刷新页面，如果之前对tomcat2的tomcat.png文件进行过修改的话，会发现出现的页面是不固定的。因为通过nginx会把请求随机分发给我们配置的对应8080端口的tomcat1以及对应9080端口的tomcat2。而tomcat2显示的页面中的图片又进行了修改，所以一直刷新的话，页面其实是在两个不同的页面之间进行切换。哪个页面出现的概率大，取决于上面conf文件中配置的权重。

这里说一下为什么要修改linux中的hosts文件。www.test.com.conf文件内容的部分截图如下。

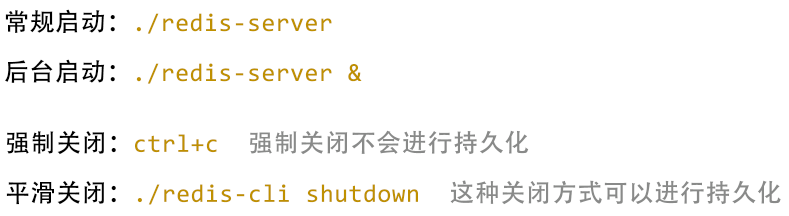


该图中的server后面之所以可以使用www.test.com:8080或www.test.com:9080，就是因为在linux中修改了hosts文件。如果没有修改hosts文件的话，这里这样写是访问不到内容的。如果不想修改linux中的hosts文件也行，不过就不能再像上图中那么写了，而是要写成下图中这样。



# 五、Redis快速入门

## 1. redis的启动和关闭



使用带&的方式启动redis服务后，即便使用ctrl+c进行强制关闭，后台还是处于启动状态，通过查看进程可以看出来。



以上启动redis服务和客户端都是使用默认的6379端口启动的，如果不想使用默认端口的话，可以使用如下启动方式。



除了以上提到的启动和关闭的方式外，还可以通过修改配置文件的方式启动。

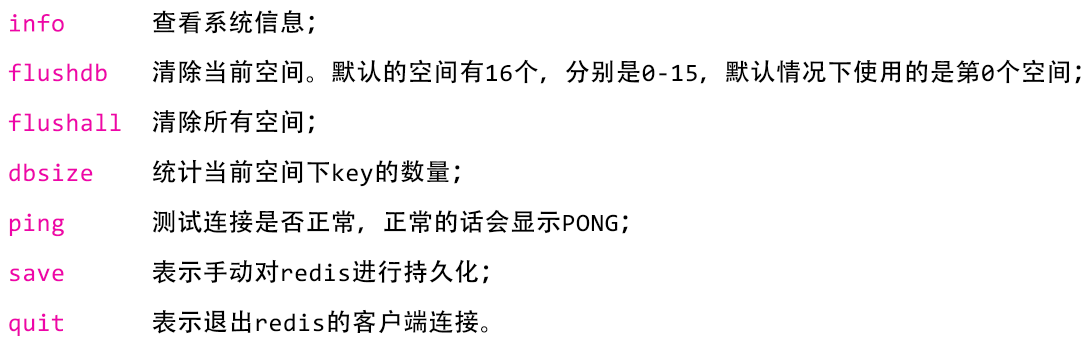


很多时候，我们启动客户端其实是要输入密码了，下面这种方式就是带密码启动。

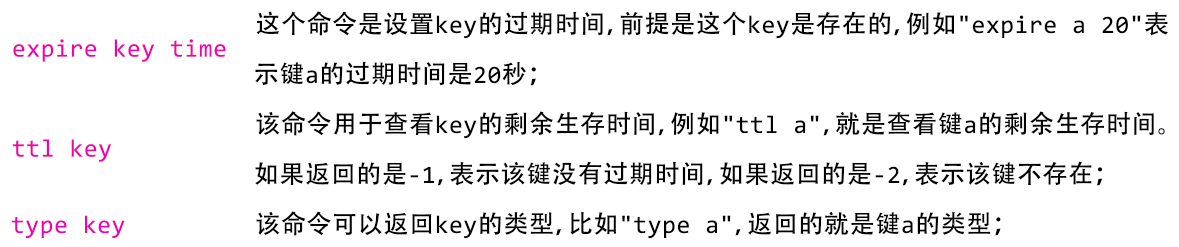


## 2. redis常用的命令

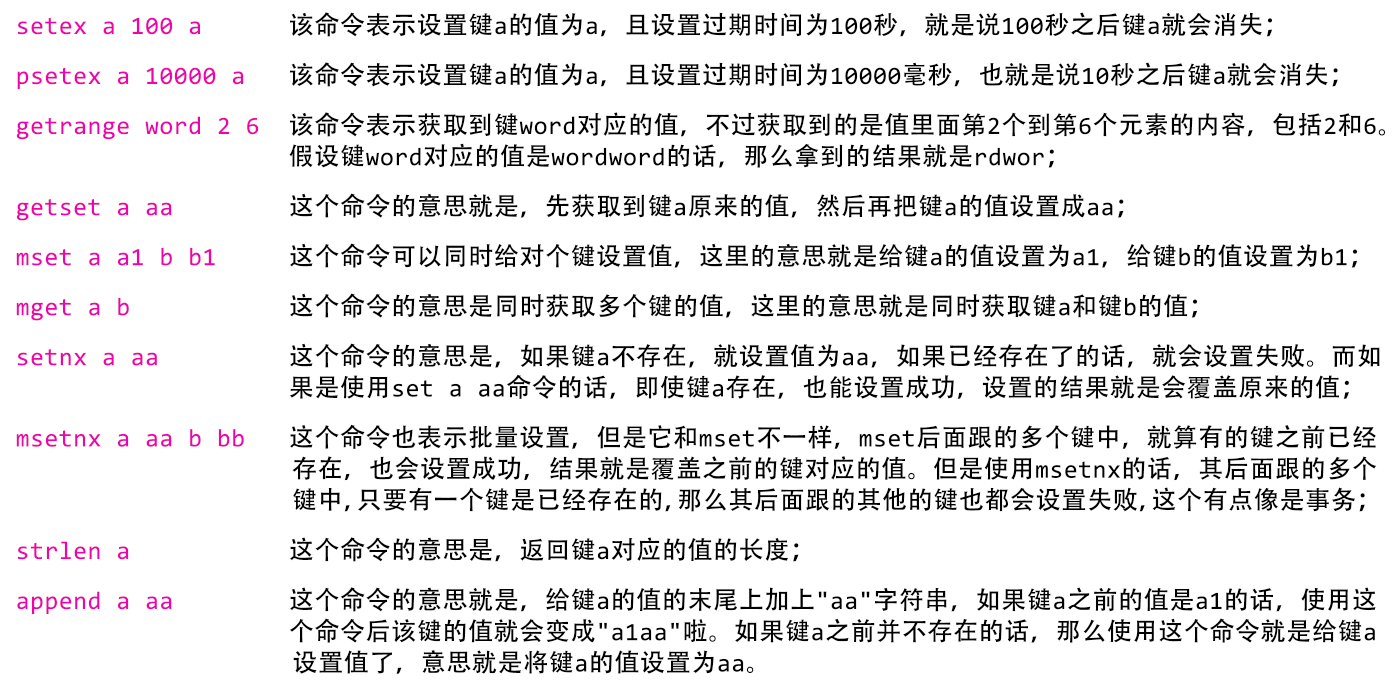
### 2.1 常用的系统命令



### 2.2 redis常用的键命令



### 2.3 redis数据结构之String字符串



关于redis的其他类型的数据结构这里就不再讲啦，因为在这个项目之前学redis的时候我作过总结，上面都有讲过，具体可以参考那时候作的总结。

# 六、单点登录及其相关知识点

## 1. 使用jedis连接redis

### 1.1 连接池的构建

该项目中创建了一个RedisPool类，通过这个类完成对连接池的构建，在构建jedis连接池的时候，主要涉及JedisPool和Jedis这两个类。在RedisPool类中还设置了连接池的一些参数，如最大连接数、最大空闲数、最小空闲数、连接redis的ip地址、端口号以及超时时间等。另外，RedisPool类中还设置了一个公开的获取Jedis的方法。

### 1.2 jedis常用API的封装

连接池构建好之后，我们就可以通过jedis连接池连接redis了。在RedisPool类中通过JedisPool类是可以获取到Jedis类的，而如果想要对redis进行操作，就需要用到Jedis类的一些API，这里把常用的一些方法都封装在了RedisPoolUtil类中了。比如使用expire方法设置redis中key的有效期、使用set方法添加一个键值对、使用setex方法添加一个带有效期的键值对、使用get方法获取key的value值、使用del方法删除一个键值对等。

由于RedisPool类已经被应用在了RedisPoolUtil类的所有方法中，所以直接调用RedisPoolUtil类中的相关方法就可以连接上redis并对其进行相应的操作。

## 2. JsonUtil类的封装

这里要说的JsonUtil类是一个相当强大的工具类，也是可以直接应用到工作中的一个类。这个类中主要封装的是java对象和json字符串之间相互转换的方法，比如java对象转成json字符串的方法、json字符串转成java对象的方法等。由于老师把各种可能性都考虑到了，并且都封装进了这个类中，所以这个方法的复用性很强。

在这个工具类中有一个相当重要的类，它就是ObjectMapper类。ObjectMapper是Jackson提供的一个类，作用是实现java对象与json格式之间的相互转化。它可以将对象转换成json字符串，反之亦然。JsonUtil这个工具类就是靠着ObjectMapper类才完成java对象与json字符串之间的相互转换的。

我大致说一下这个工具类在该项目中主要的应用场景。在集群状态下，如果要实现单点登录的话，用户登陆的session信息就不能放在tomcat服务器中，而是要放在redis中。但是在redis中，是没法储存一个对象的，而用户的登录信息又是一个user对象，这时候如果想把这个登录信息储存到redis中，最好的办法就是把这个对象转成json字符串，然后再进行存储。这时候这个JsonUtil工具类就派上用场啦，可以在储存用户登陆信息的时候，先使用工具类把这个user对象转成json字符串储存在redis中，然后当想获取这个用户的登录信息的时候，再使用这个工具类把json字符串转成对象返回给前端即可。

## 3. CookieUtil类的封装

这个工具类中主要封装了三个方法，它们的作用分别是新增一个cookie信息并响应给浏览器、获取浏览器的请求中的cookie信息、删除浏览器中指定的cookie信息。

如果想实现用户登陆信息的共享，离不开这个工具类。由于在tomcat集群环境下，会有多个tomcat服务器，而我们又不知道用户的登录请求会被nginx分发给哪个服务器，而不同的服务器中的信息又不能共享。如果想要解决这个问题，就需要把用户的登录信息储存到redis中。

当用户登录的时候，我们可以获取到用户的登录信息以及这次会话的session的id值，这时候我们可以把session的id值作为键值对的key，把用户的登录信息转json字符串后的内容作为键值对的value储存到redis中。与此同时，还要将session的id值传入到CookieUtil类中要响应给浏览器的方法中。在这个方法中，首先要创建一个cookie对象，这个cookie的name值可以自己定义一个，而value值就是传入的session的id值，然后给这个cookie对象的一些常用属性设置好值之后就返回给浏览器。

当用户再次访问的时候，不管用户的请求被分发到哪个服务器中都没关系。因为这个工具类的另一个方法可以获取到请求中所有的cookie，然后这个方法会对这些cookie进行判断，看有没有哪个cookie的name值是我们之前自己定义的那个name值，如果找到了这样的cookie，那么就返回这个cookie的value值，而这个value值就是我们之前存的session的id值。然后再把这个value值作为键值对中的key值到redis中获取这个键对应的value值，再把这个字符串形式的value值转成对象，如果这个对象不为空的话，就说明该用户已经登录啦，这样一来，就在集群环境下获取到了用户的登录信息啦，同时也解决了用户登陆信息共享的问题。

如果用户执行了退出登录的操作，那么我们就调用CookieUtil工具类中另一个方法删除浏览器中的cookie信息，同时也删除redis中该用户的登录信息即可。

## 4. 重置session有效期

这里是使用一个过滤器类来完成对session有效期的重置的，这里说一下为什么要重置session的有效期。由于session的有效期是30分钟，所以这就导致一个结果，那就是即使实现了session共享，但是当用户登陆30分钟后，登陆信息就会失效，就又要重新登录了，这样用户体验就很不好了。这时候可以引入一个可以拦截所有以"\*.do"结尾的请求的过滤器类，而使用过滤器类的作用就是，在用户登陆后，只要用户和网站进行交互，就会拦截到交互的请求，然后把session的过期时间重置成30分钟。这里说的交互分很多种，比如用户查看了商品详情，比如用户把商品加入购物车等等。

至于为什么非要进行下重置，我这里再举个例子。假设用户现在登录了，这时候session还有30分钟就会过期。假设用户登陆后过了5分钟才与网站进行交互，比如查看了商品详情什么的。如果不重置session的过期时间，那么再过25分钟，这个用户的登录信息就会失效，就要重新登录了。但是如果重置了，那么session的有效期就又会变成30分钟，也就是说，重置后的session就又变成是30分钟后失效，而不是25分钟。如果用户再与网站进行交互，就又会重置session的过期时间。这样一来，只要用户在30分钟内至少和网站交互一次，那么该用户的登录信息就永不会失效。

## 5. 总结

关于单点登录这一块，主要讲到了jedis的一些相关知识，还有就是一些工具类的封装。其实我觉得只要解决了在集群环境下session共享的问题，那么就能够实现单点登录了。由于是集群环境，所以肯定不止一个tomcat服务器，当用户的登录请求到达nginx的时候，我们并不知道nginx把请求转发到哪个tomcat服务器中啦。假设用户的登录请求被转发到tomcat1中啦，而请求中是携带有用户的session信息的，而用户的登录信息都在这里面。如果不能实现session共享，那么万一用户又查看了下个人信息，而查看个人信息的这个请求正好是由tomcat2来处理的，而tomcat2中是没有用户的登录信息的，这时候就会认为该用户没有登录，然后提示用户进行登录。可是用户刚登录过，然后查看一下个人信息居然还要再登录一下，这样用户体验就很不好啦。

而实现session共享的方法就是，在用户登录的时候，不把用户的session信息保存在任何一台tomcat服务器中，而是把它保存在redis中。假设用户的登录请求是交由tomcat1来处理的，这时候把请求中携带的用户的session信息保存到redis。假设用户查看个人信息的请求是由tomcat2来处理的，这时首先会在redis中寻找看有没有该用户对应的session信息，如果能找到，就说明该用户已经登录，就不用再登录啦；如果没找到，就说明该用户还没有登录，就提示用户登录。

# 七、XXXXXXXXXXXXXXXXXX