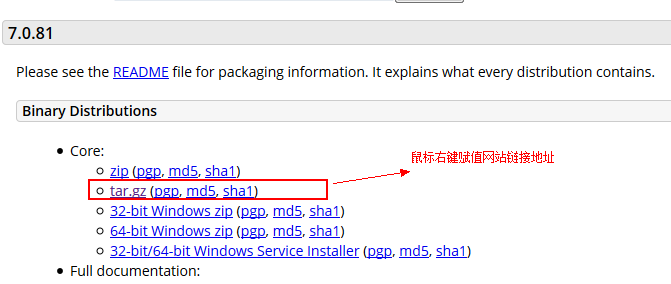
1：安装好JDK

2：进入tomcat的官网下载：

<http://tomcat.apache.org/download-70.cgi>



3：在Xshell中下载

根目录下创建了tomcat目录：mkdir /usr/local/tomcat

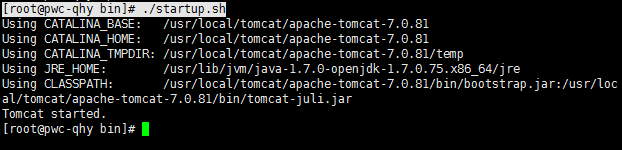
备注：我先创建了tomcat目录。那么接下来的下载和安装都是在自己创建的目录中。

下载：[root@pwc-qhy tomcat]# wget -c <http://mirror.bit.edu.cn/apache/tomcat/tomcat-7/v7.0.81/bin/apache-tomcat-7.0.81.tar.gz>

解压：tar -zxvf apache-tomcat-7.0.81.tar.gz

4：进入tomcat的bin目录，启动tomcat

[root@pwc-qhy bin]# ./startup.sh



关闭：  ./shutdown.sh

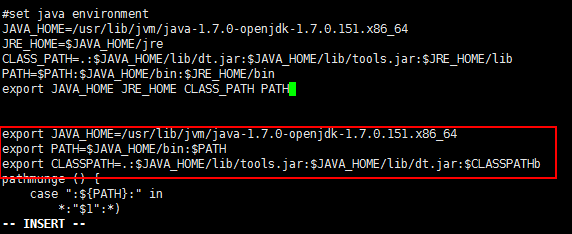
相关网站：

<http://www.cnblogs.com/xdp-gacl/p/4097608.html>

配置环境变量：vi .bash\_profile

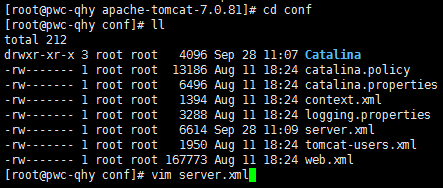
export CATALINA\_HOME="/home/apache-tomcat-7.0.85"

export CATALINA\_BASE="/home/apache-tomcat-7.0.85"

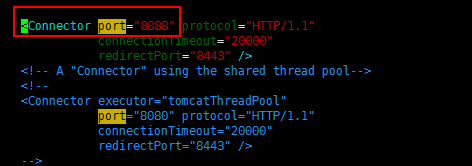


备注1：JDK的版本填写自己的实际版本，不要照抄。否则和实际版本不符合，tomcat启动的时候找不到jdk

备注2：tomcat默认端口是8080，如果有需要改动的，可以修改conf下的server.xml文件



/port



按i键 进入insert 可编辑状态

方向键上下左右移动到目标，删除默认的8080，改为自己想要的端口。

备注：如果在编辑过程中出现了未知操作，可先ESC键退出编辑状态，然后直接输入:q! 回车退出，这样就不会保存乱操作。

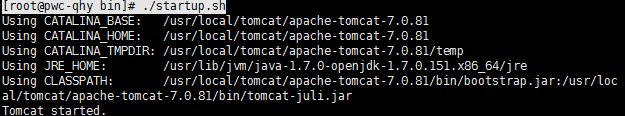
保存退出： :wq! 回车

备注3：报错

启动tomcat：[root@pwc-qhy apache-tomcat-7.0.81]# cd bin

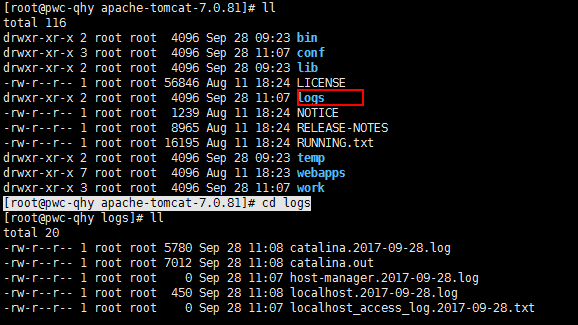
[root@pwc-qhy bin]# ./startup.sh

效果：



检查是否启动成功：进入tomcat目录下的logs目录

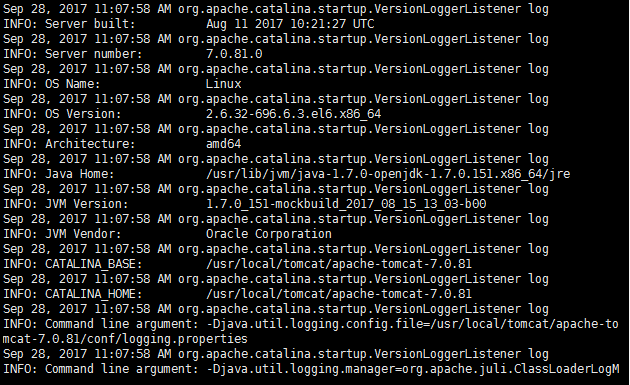
[root@pwc-qhy apache-tomcat-7.0.81]# cd logs



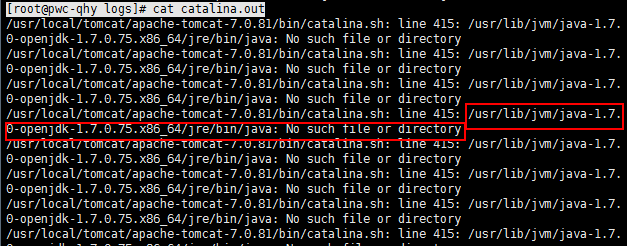
从图中可以看出logs中出现了一些文件

[root@pwc-qhy logs]# cat catalina.out

正常：会出现tomcat启动时的信息



报错：



错误原因：说找不到目标文件

分析原因：tomcat是否配置了环境变量

环境变量是否配置错误

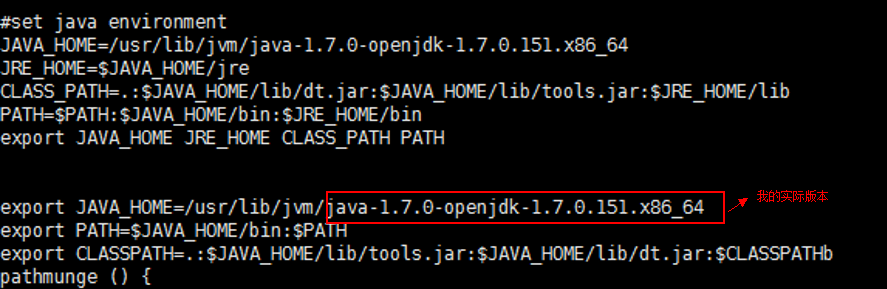
解决办法：vi /etc/profile 找到环境变量

发现export JAVA\_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.75.x86\_64

这是直接从网上抄下来jdk版本。

而我jdk的实际安装版本是

java-1.7.0-openjdk-1.7.0.151.x86\_64



注意：在使用java –version查看jdk版本的时候还是要进入jdk的实际安装目录中去查看，不然会出现安装的是151版本，配置的时候直接抄下的75版本，java –version命令查看又是75版本，导致出现错误后很难排查。

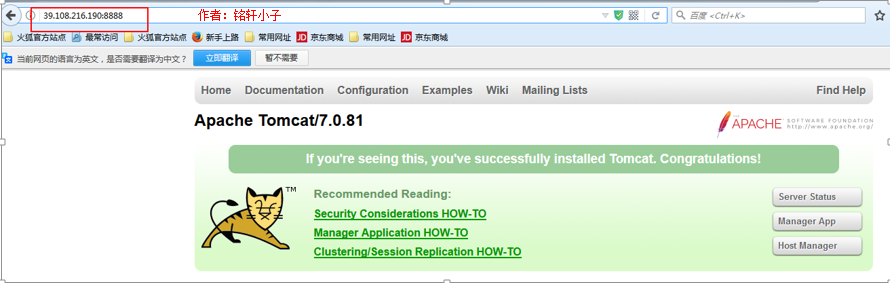
立即执行修改的文件：source /etc/profile 这样就不用重新启动虚拟机了

[root@pwc-qhy jvm]# source /etc/profile

5：启动tomcat，页面测试

因为我是在阿里云上的虚拟机，那么在访问的时候就是ip+端口

注意不要写成localhost+端口了，这是本地访问



**Linux环境下清理Tomcat缓存**

1.进入tomcat/bin目录下，执行sh shutdown.sh命令关闭tomcat。

2.然后执行命令 ps -aux | grep tomcat 命令​，查看tomcat命令是否真的关闭

3.如果没有关闭执行 kill -9 #pid 来彻底关闭tomcat（#pid是tomcat的进程ID）​

root    17505    0.0        0.0    63344  768    pts/1    S+  10:22  0:00 grep tomcat​

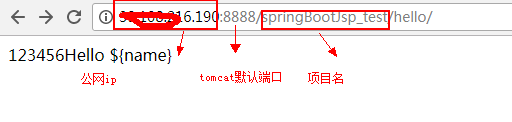
USER    PID    %CPU  %MEM  VSZ  RSS    TTY  STAT  START TIME COMMAND​

4.最后进入tomcat/work目录下​，执行 rm -rf Catalina 来删除缓存文件

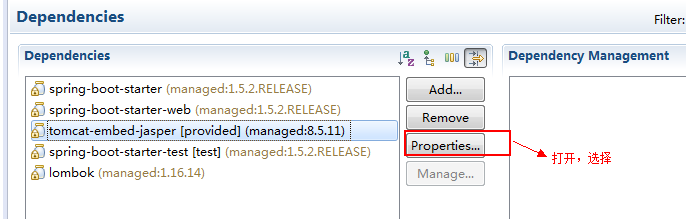
5.执行ls ​确认是否删除，如果work目录下没有Catalina文件夹说明删除成功。

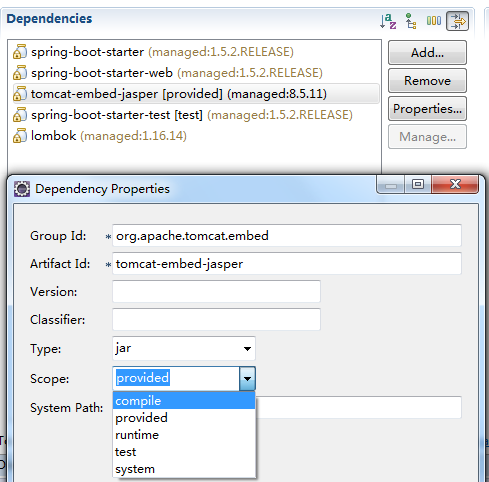
6.在进入tomcat/bin目录下，执行sh startup.sh命令来重启tomcat。

**部署项目**



**注意：部署springBoot项目的时候注意，因为springboot内置了tomcat，所以将springboot再部署到tomcat下面会冲突。所以在maven打包前需要先将tomcat的依赖设置成provided**





备注：compile：默认的scope。任何定义在compile scope下的依赖将会在所有的class paths下可用。maven工程会将其打包到最终的arifact中。如果你构建一个WAR类型的artefact，那么在compile scope下引用的JAR文件将会被集成到WAR文件内。

provided：这个scope假定对应的依赖会由运行这个应用的JDK或者容器来提供。最好的例子就是servlet API。任何在provided scope下定义的依赖在构建时的类路径里是可用的，但是不会被打包到最终的artifact中。如果是一个WAR的文件，servlet API在构建时的类路径里是可用的，但是并不会被打包到WAR文件中。

runtime：在runtime scope下定义的依赖只会在运行期可用，而在构建期的类路径下不可用。这些依赖将会被打包到最终的artifact中。比如你有一个基于web的应用需要在运行时访问MySQL数据库。你的代码没有任何MySQL数据库驱动的硬依赖。你的代码仅仅是基于JDBC API来编写，在构建期并不需要MySQL数据库驱动。然而，在运行期，就需要相应的驱动来操作MySQL数据库了。因此，这个驱动应该被打包到最终的artifact中。

test：只用于测试变异的依赖（比如JUnit），execution必须定义在test scope下。这些依赖不会被打包到最终的artefact中。

system：于provided scope很像。唯一的区别在于，在system scope中，你需要告诉Mave如何去找到这个依赖。如果你要引用的依赖在Maven仓库中不存在时，就可以用这个scope。不推荐使用system依赖。

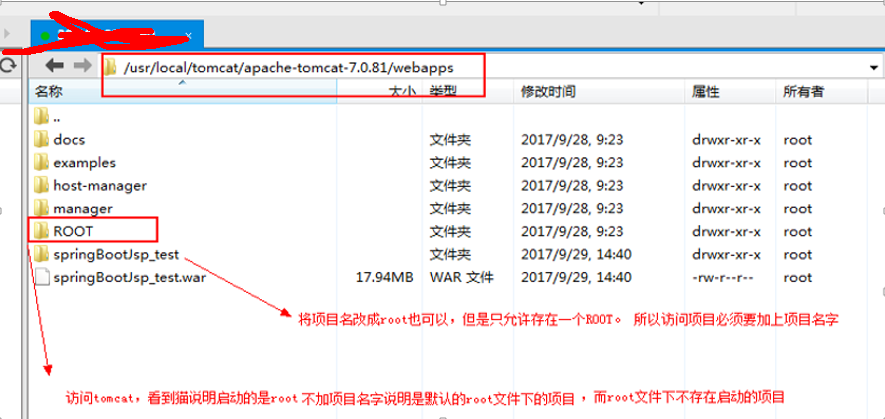
import：从其它的pom文件中导入依赖设置。

注意：在win环境下启动springboot项目，可以不需要写项目名直接访问，而linux环境必须完整路径

Windows： <http://localhost:8080/hello>

Linux：<http://39.108.235.190:8888/springBootJsp_test/hello/>

使用Xshell 可以看到tomcat下的webapps目录



添加：

1：Tomcat的webapps下的所有文件不重要可以删除，直接rm-rf webapps 删除文件

2：mkdir webapps 创建webapps文件夹

3：将项目发布到tomcat 使用xshell 直接将打好的war包 拖进来

注意：此时因为是新建的webapps文件，所以war包丢进来后，默认运行项目是ROOT 目录下，tomcat运行项目的时候会根据项目名字默认创建同名文件夹。所以将war包更名ROOT.war后再放到webapps下更方便。

4：启动tomcat 进入bin目录 ./startup.sh

5:验证war包是否打开 cd webapps ll 会显示ROOT文件夹和ROOT.war 说明解压war包成功

6：tomcat打开三个端口

[root@pwc-qhy ROOT]# /sbin/iptables -I INPUT -p tcp --dport 8080 -j ACCEPT

[root@pwc-qhy ROOT]# /sbin/iptables -I INPUT -p tcp --dport 8081 -j ACCEPT

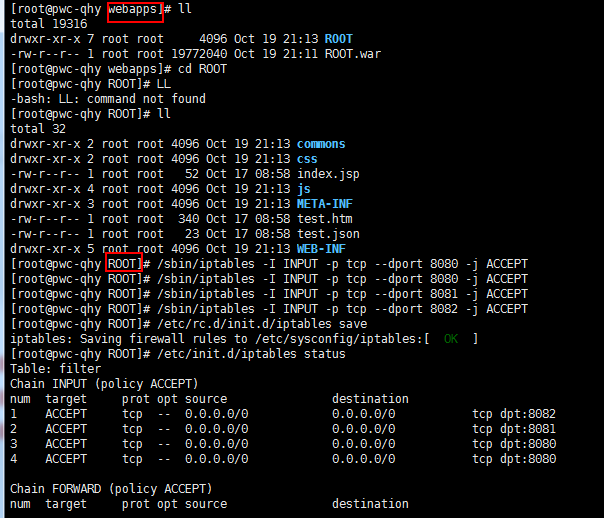
[root@pwc-qhy ROOT]# /sbin/iptables -I INPUT -p tcp --dport 8082 -j ACCEPT

7：保存端口

[root@pwc-qhy ROOT]# /etc/rc.d/init.d/iptables save

8：查看是否更改成功

[root@pwc-qhy ROOT]# /etc/init.d/iptables status



9:重启tomcat 需要删除缓存

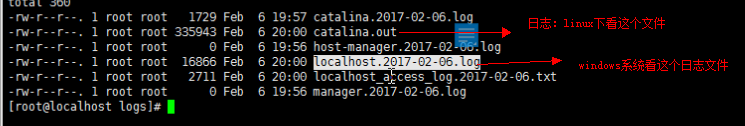
work/cataLina/localhost 下 直接删除catalina目录也行，会自动生成

建议删除localhost目录

注意：work目录删除会自动生成

Temp：临时文件 目录删除了不会自动生成，删除了会报警 一般不用删除

删除当前目录下的文件 ：rm –rf \*.\*



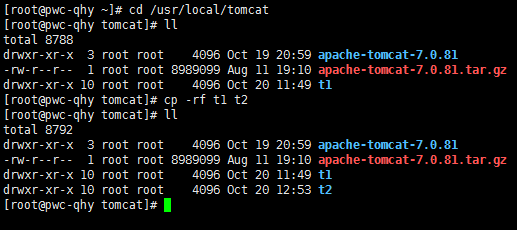
10：查看日志是否启动项目，是否报错

两种方式查看：

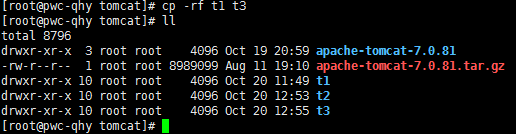
Logs目录下：cat catalina.out 会逐步显示日志 直到结束

tail -f catalina.out 直接显示最终结果

11：装三台tomcat



[root@pwc-qhy tomcat]# cp -rf t1 t2



注意：一台服务器上如果开启多个tomcat实例需要修改配置文件

Conf/server.xml

修改三处：8005,8080,8009 习惯上加10 8015.。。。。

t 1: 8005 8080 8009

t 2: 8015 8081 8019

t3: 8025 8082 8029

开启端口的时候，最好去阿里云的安全组查看是否支持这些端口

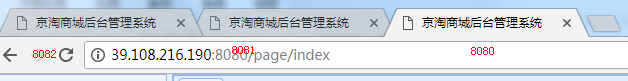
因为在t1台上已经运行了项目，那么复制过来的时候，t2和t3中已经有了ROOT目录有了项目了。如果复制过来后里面有t1的日志后缓存文件，先删除，再启动t2和t3 过程和t1是一样的效果。

例如启动t1：<http://39.108.216.190:8080/page/index>

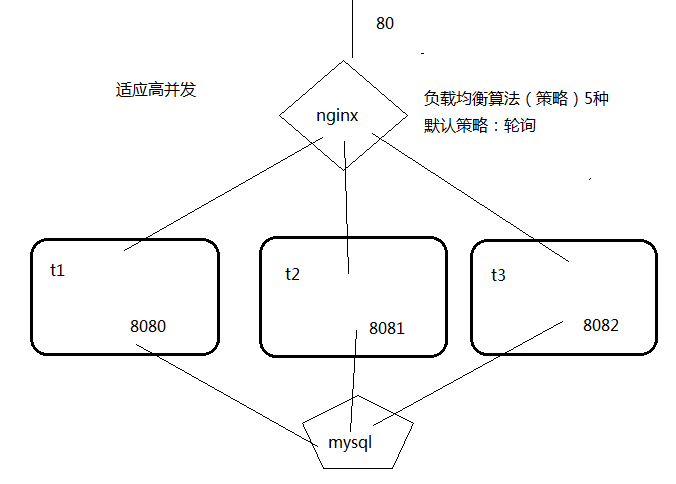
t 2：<http://39.108.216.190:8081/page/index>

t 3：<http://39.108.216.190:8082/page/index>

到此就开启了三台tomcat单独运行同一个项目

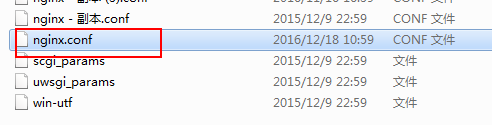


Nginx负载均衡：



因为Nginx和tomcat处在不同的服务器上，这里使用windows上的Nginx转发到linux上的tomcat。

1：Nginx中修改配置文件

:22

2： 原理：Nginx会侦听80端口，侦听manage.jt.com域名，然后拦截所有请求，将请求转发到<http://jt_ts>，然后通过jt\_ts这个名字找到了三台tomcat，第一次请求访问第一台，第二次访问第二台，第三次访问第三台，这样就做到了轮询。

2.1 #负载均衡配置，添加三台服务器，默认是轮询

upstream jt\_ts{ jt\_ts:名字随便取

server 39.108.216.190:8080; 就是启动项目的路径

server 39.108.216.190:8081;

server 39.108.216.190:8082;

}

2.2:

#后台服务器

server {

listen 80;

server\_name manage.jt.com;

location / {

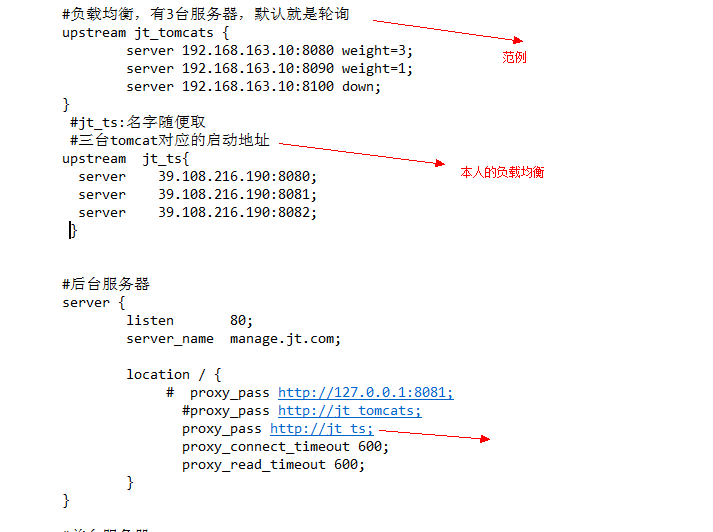
proxy\_pass <http://jt_ts> 对应了上面的名字

proxy\_connect\_timeout 600;

proxy\_read\_timeout 600;

}

}

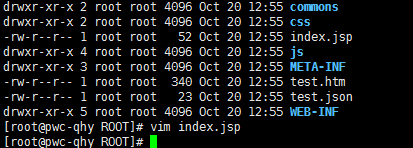


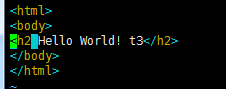
3：重启Nginx stop—start 可以通过查看进程，是否关闭



4：启动项目的时候，三台tomcat工作，怎么知道当前是哪台tomcat在 处理请求，可以通过编辑当前项目的index.jsp 欢迎页面来知道

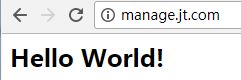






再次访问：

第一次：



第二次：



第三次：



到此说明Nginx上负载均衡的轮询功能就实现了。

1台服务器不能承受用户高并发访问，引入多个tomcat，并发数增加，必须利用负载均衡算法来实现3个台变成一个整体，用户的请求，平均分配在各个服务器上。3台服务器就形成一个集群。

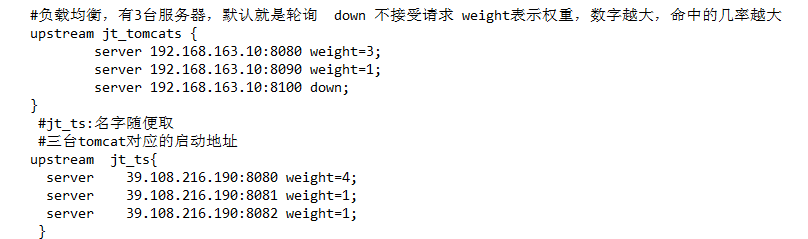
用nginx +tomcats实现业务系统负载均衡，提高业务系统并发能力。

如果3台服务器性能不同，

如果按默认负载均衡算法（轮询）不合理。怎么能让不同性能的机器平均分配？？

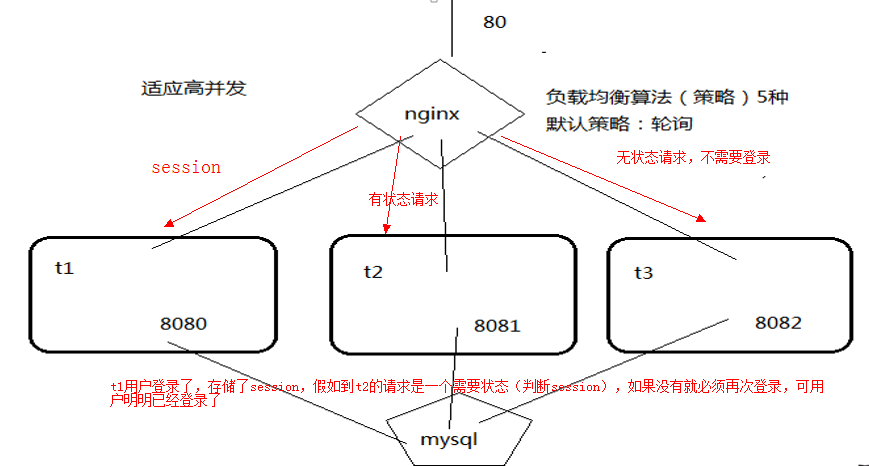
权重：

继续修改Nginx的配置文件,域名后+weight=数字 数字越大获取请求的几率越大



down ：如果不需要使用哪台服务器了，可以停掉。请求就不会命中了

知识扩展：登录状态问题



1. 无状态（前台首页，商品详情）最好（大数据集群可以动态扩展，可以增加或者减少）
2. 有状态（登录，购物车，订单） scala 函数式编程+面向对象编程

由此引出来 session 共享问题

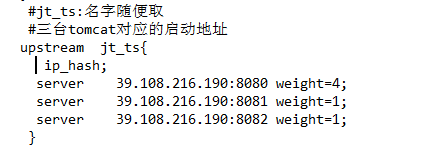
解决办法： SSO单点登录

nginx IPhash策略、urlhash策略，第三方支持（单独安装nginx插件）

ip hash（tomcat session黏着）（ ip+port） % n机器数

缺点：hash算法，数据倾斜（大数据），资源不能平均充分利用

数据倾斜根源在于key，url\_hash，key=ip+port，url=业务，量比key变化量大，缓解



总结：

ip\_hash和url\_hash，nginx解决session共享；

url\_hash key的变化比ip\_hash算法多，缓解数据倾斜问题。