**生产环境Session解决方案**

# Session简介

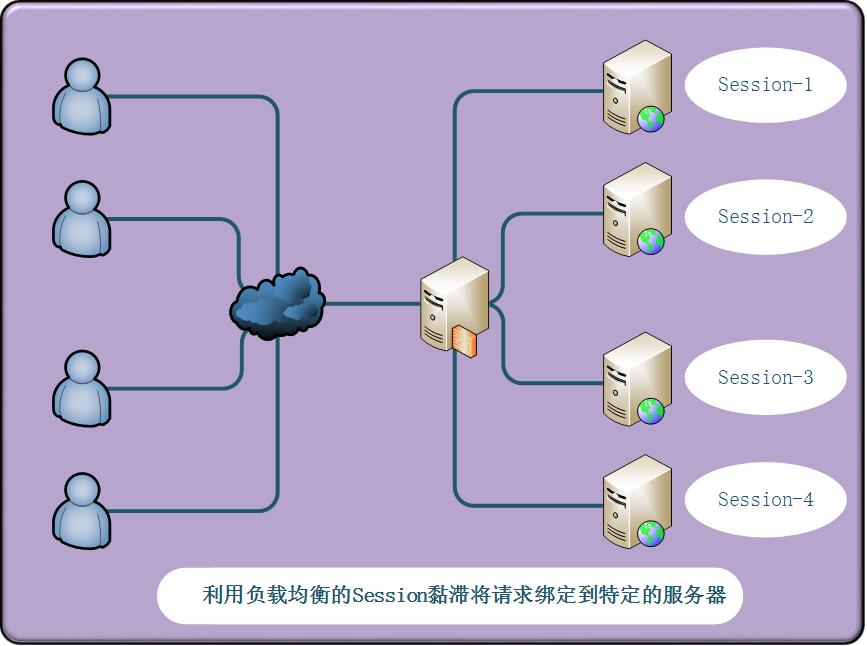
应用服务器（Tomcat）的高可用负载均衡架构设计主要服务于服务无状态这一特性（静态页面），但是事实上，绝大多数生产业务总是有状态的，例如：交易类的电子商务网站，需要有购物车记录用户的购买信息，每次购买请求都是向购物车中增加新的商品；社交网站中，需要记录用户当前的登录状态，获取用户的个人信息、最新发布的消息及好友请求状态等，用户每次刷新页面都需要更新这些信息。

Session是由应用服务器维持的一个服务器端的存储空间（内存中），用户在连接服务器时，会由服务器生成一个唯一的SessionID，客户端使用该Session ID 为标识符来存取服务器端的Session存储空间。而Session ID则保存到客户端，使用浏览器Cookie保存的，用户提交页面请求时，也会将 Session ID提交到服务器端，来存取Session空间的数据。服务器也通过URL重写的方式来传递Session ID的值，因此不是完全依赖Cookie。如果客户端Cookie禁用，则服务器可以自动通过重写URL的方式来保存Session的值，并且这个过程对程序员透明。

WEB应用中将这些多次请求修改使用的上下文对象称作会话(Session)，单机情况下，Session可由部署在服务器上的WEB容器（Tomcat、Resin）进行管理。但在使用高可用负载均衡的集群环境中，由于负载均衡服务器可能会将每次请求分发到集群任何一台应用服务器（Tomcat）上，所以保证每次请求依然能够获得正确的Session比在单机上实现要复杂的多。

## 解决方案一：Session绑定

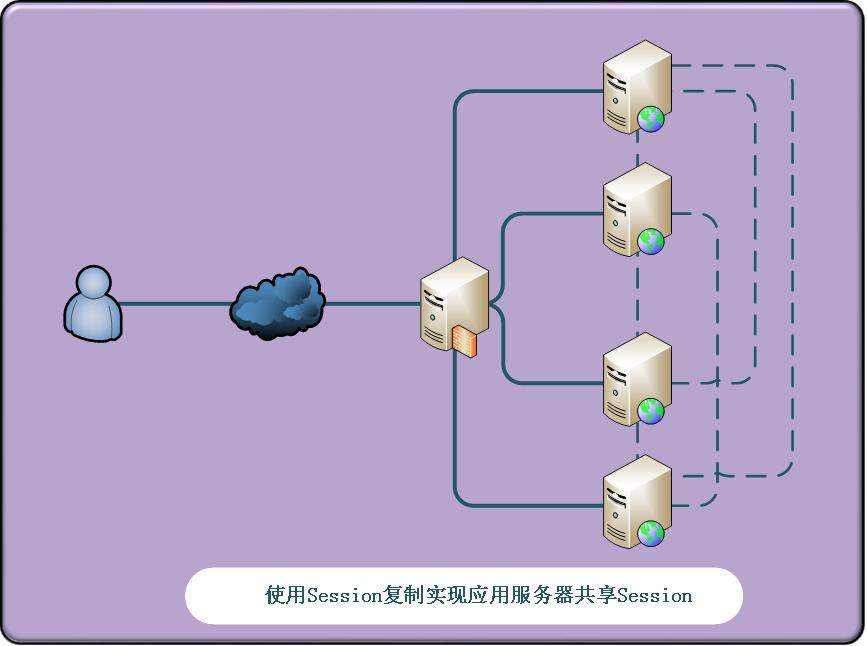
Session绑定可以利用负载均衡的源地址Hash（ip\_hash）算法实现。负载均衡服务器总是将来源于同一个IP的请求分发到同一台服务器上，也可以根据Cookie信息将同一个用户的请求总是分发到同一台服务器上。当然这时负载均衡服务器必须工作在HTTP协议层上。这样整个会话期间，同一个用户所有的请求都在同一台服务器上处理，即Session绑定在某台特定服务器上，保证Session总能在这台服务器上获取。这种方法又被称为会话黏滞。如图所示。



但是Session绑定的方案显然不符合我们对系统高可用的需求，因为一旦某台服务器宕机，那么该机器上的Session也就不复存在了，用户请求切换到其他机器后因为没有Session而无法完成业务处理。因此虽然大部分负载均衡服务器都能提供源地址负载均衡算法，但很少用网站利用这个算法进行Session管理。

## 解决方案二：Session复制

Session复制是小型架构使用较多的一种服务器集群Session管理机制。应用服务器开启Web容器的Session复制功能，在集群中的几台服务器之间同步Session对象，使每台服务器上都保存了所有用户的Session信息，这样任何一台机器宕机都不会导致Session数据的丢失，而服务器使用Session时，也只需要在本机获取即可。如图所示。



这种方案实现简单，从本机读取Session信息也很快速，但只能应用在集群规模比较小的环境下。当集群规模较大时，集群服务器间需要大量的通信进行Session复制，占用服务器和网络的大量资源，系统不堪负担。而且由于所有用户的Session信息在每台服务器上都有备份，在大量用户访问的情况下，甚至会出现服务器内存不够Session使用的情况。

而大型网站的核心应用集群就是数千台服务器，同时在线用户可达千万，因此并不适用这种方案。

## 解决方案三：Session服务器之Memcached

## 解决方案四：Session服务器之Redis

## 环境描述：

|  |  |  |  |
| --- | --- | --- | --- |
| 操作系统 | IP地址 | 主机名 | 软件包列表 |
| CentOS7.x-x86\_64 | **192.168.200.111** | **nginx** | **nginx** |
| CentOS7.x-x86\_64 | **192.168.200.112** | **node1** | **JDK Tomcat** |
| CentOS7.x-x86\_64 | **192.168.200.113** | **node2** | **JDK Tomcat** |

## 环境配置：

安装前准备配置：

**配置所有机器：**

**[root@localhost ~]# cat /etc/hosts**

**192.168.200.111 nginx**

**192.168.200.112 node1**

**192.168.200.113 node2**

**[root@localhost ~]# iptables -F**

**[root@localhost ~]# systemctl stop firewalld**

**[root@localhost ~]# setenforce 0**

### Nginx 服务器配置：

配置主机名：

**[root@localhost ~]# hostname nginx**

**[root@localhost ~]# bash**

安装nginx软件包并修改：

**[root@nginx ~]# yum -y install pcre-devel zlib-devel openssl-devel**

**[root@nginx ~]# useradd -s /sbin/nologin -M nginx**

**[root@nginx ~]# tar xf nginx-1.15.9.tar.gz -C /usr/src/**

**[root@nginx ~]# cd /usr/src/nginx-1.15.9/**

**[root@nginx ~]# ./configure --prefix=/usr/local/nginx --user=nginx --group=nginx && make && make install**

**[root@nginx ~]# ln -s /usr/local/nginx/sbin/nginx /sbin/**

**[root@nginx ~]# vim /usr/local/nginx/conf/nginx.conf**

**upstream tomcat\_pool {**

**server 192.168.200.112:8080 weight=1 max\_fails=1 fail\_timeout=10s;**

**server 192.168.200.113:8080 weight=1 max\_fails=1 fail\_timeout=10s;**

**}**

**server {**

**listen 80;**

**server\_name localhost;**

**charset utf-8;**

**location / {**

**root html;**

**index index.html index.htm;**

**proxy\_pass http://tomcat\_pool;**

**}**

**[root@nginx ~]# killall -HUP nginx**

### 安装配置Tomcat：

使用系统默认的jdk环境，由于redis和memcached服务器有版本同步限制，本实验提供的都是tomcat7匹配的包，因此需要安装tomcat7版本

配置主机名：

**[root@localhost ~]# hostname node1 另外一台机器配置为node2**

**[root@localhost ~]# bash**

安装配置Tomcat

解压apache-tomcat-7.0.54.tar.gz 包

**[root@tomcat1 ~]# tar xf apache-tomcat-7.0.54.tar.gz**

解压后生成apache-tomcat-7.0.54文件夹，将该文件夹移动到/usr/local下，并改名为tomcat

**[root@tomcat1 ~]# mv apache-tomcat-7.0.54 /usr/local/tomcat**

**[root@node1 lib]# ln -s /usr/local/tomcat/bin/\* /usr/local/bin**

启动Tomcat

**[root@tomcat1 ~]# startup.sh**

**Using CATALINA\_BASE: /usr/local/tomcat**

**Using CATALINA\_HOME: /usr/local/tomcat**

**Using CATALINA\_TMPDIR: /usr/local/tomcat/temp**

**Using JRE\_HOME: /usr/local/java**

**Using CLASSPATH: /usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar**

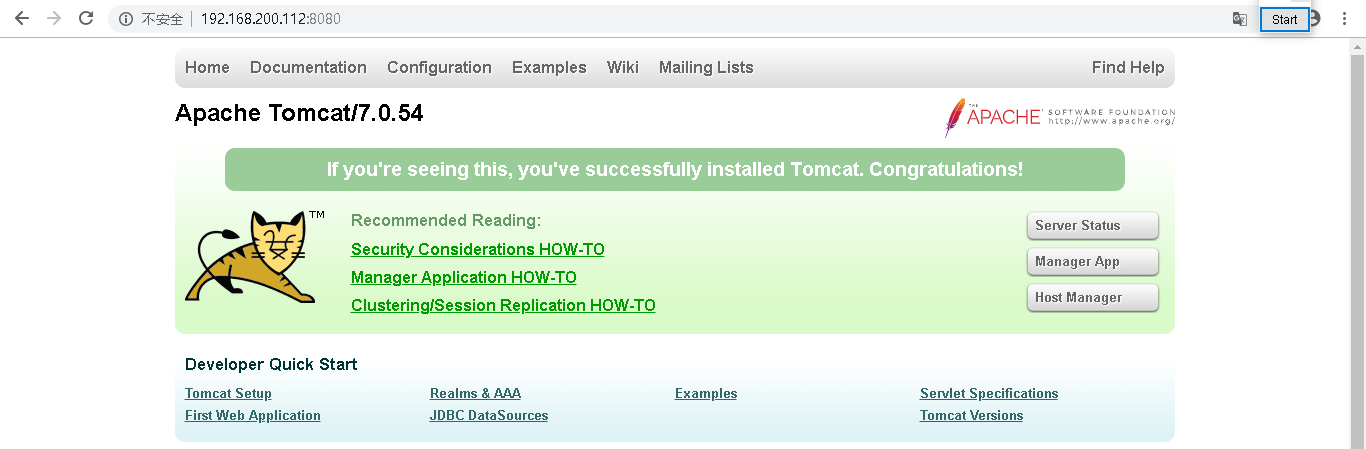
**Tomcat started.**

Tomcat 默认运行在8080端口

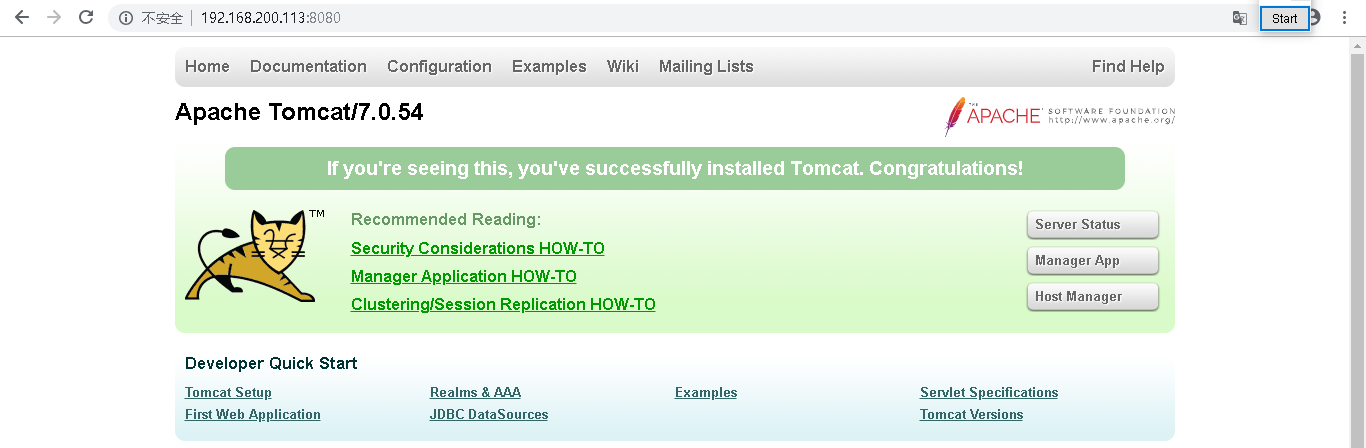
**[root@tomcat1 ~]# netstat -lnpt | grep java**

**tcp 0 0 :::8080 :::\* LISTEN 3318/java**

浏览器访问测试 <http://192.168.200.112:8080>



浏览器访问测试 http://192.168.200.113:8080



#### 建立session.jsp的测试页面

**[root@node1 ~]# vim /usr/local/tomcat/webapps/ROOT/session.jsp**

**Session ID:<%= session.getId() %><BR>**

**SessionPort:<%= request.getServerPort() %><br>**

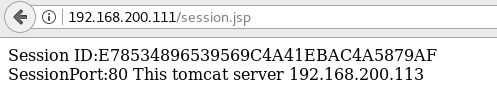
**<% out.println("This tomcat server 192.168.200.112");%>**

**[root@node2 ~]# vim /usr/local/tomcat/webapps/ROOT/session.jsp**

**Session ID:<%= session.getId() %><BR>**

**SessionPort:<%= request.getServerPort() %><br>**

**<% out.println("This tomcat server 192.168.200.113");%>**

测试结果，每次的session ID都不一样，哪怕访问在同一台服务器都会发生变化，如果ie切换效果不明显，建议使用firefox浏览器，由于后面多个实验要恢复到初始tomcat环境，因此实验成功后保存好快照以方便后续实验还原到初始状态

## Session绑定：

**[root@nginx ~]# vim /usr/local/nginx/conf/nginx.conf**

**34 upstream tomcat\_pool {**

**35 ip\_hash;**

**36 server 192.168.200.112:8080 weight=1 max\_fails=1 fail\_timeout=10s;**

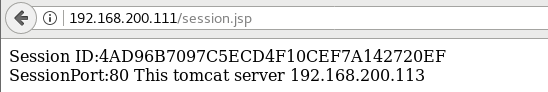
**37 server 192.168.200.113:8080 weight=1 max\_fails=1 fail\_timeout=10s;**

**38 }**

**[root@nginx ~]# killall -HUP nginx**

用户访问测试同一客户机只被分配到一台机器上，并且Session ID不再发生变化

浏览器访问测试 [http://192.168.200.111/session.jsp](http://192.168.200.101/session.jsp)



**成功后删除ip\_hash;以方便后续实验**

## Session复制：

Tomcat支持Session集群，可在各Tomcat服务器间复制全部session信息，当后端一台Tomcat服务器宕机后，Nginx重新调度用户请求分配到另外一台服务器，客户端可从另一台Tomcat服务上获取用户的session信息。

Session集群可在Tomcat服务器规模(一般10台以下)不大时使用，否则会导致Session复制时性能代价过高；

修改Tomcat配置文件

**[root@node1 tomcat]# vim /usr/local/tomcat/conf/server.xml**

**将Engine这一行**

**<****Engine name="Catalina" defaultHost="localhost">**

**修改为：**

**<Engine name="Catalina" defaultHost="localhost" jvmRoute="node1"> #tomcat2 配置为jvmRoute="node2"**

**<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/> #去掉注释**

**[root@node1 ~]# tail -2****/usr/local/tomcat/webapps/ROOT/WEB-INF/web.xml**

**<distributable/> #添加内容**

**</web-app>**

**所有tomcat服务器需要添加组播地址才能互相通信**

**[root@node1 ~]# route add -net 224.0.0.0 netmask 240.0.0.0 dev ens33**

**[root@node2 ~]# route add -net 224.0.0.0 netmask 240.0.0.0 dev ens33**

重新启动Tomcat

**[root@node1 ~]# shutdown.sh**

**Using CATALINA\_BASE: /usr/local/tomcat**

**Using CATALINA\_HOME: /usr/local/tomcat**

**Using CATALINA\_TMPDIR: /usr/local/tomcat/temp**

**Using JRE\_HOME: /usr/local/java**

**Using CLASSPATH: /usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar**

**[root@node1 ~]# startup.sh**

**Using CATALINA\_BASE: /usr/local/tomcat**

**Using CATALINA\_HOME: /usr/local/tomcat**

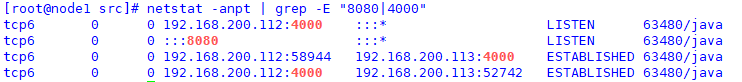
**Using CATALINA\_TMPDIR: /usr/local/tomcat/temp**

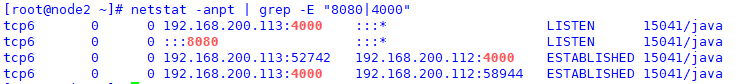
**Using JRE\_HOME: /usr/local/java**

**Using CLASSPATH: /usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar**

**Tomcat started.**

默认的监听复制消息的 TCP 端口是在 4000-4100 范围内第一个可用的server socket



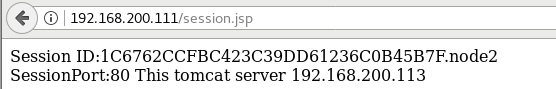


Session测试：

浏览器访问测试 [http://192.168.200.111/session.jsp](http://192.168.200.101/session.jsp)

刷新后session ID未发生变化，但是node标签及IP地址在变化





## Session服务器之Memcached

还原两台tomcat到初始状态后在两台tomcat上安装并启动Memcached，使用系统自带的memcached包

**[root@node1 ~]# yum -y install libevent memcached**

**[root@node1 ~]# memcached -u root -m 512M -d**

**[root@node1 ~]# netstat -antp| grep :11211 #memcacehd 端口为11211**

**tcp 0 0 0.0.0.0:11211 0.0.0.0:\* LISTEN 73902/memcached**

**tcp6 0 0 :::11211 :::\* LISTEN 73902/Memcached**

测试memcached 能否存取数据

**[root@node1 ~]# yum -y install telnet**

**[root@node1 ~]# telnet 192.168.200.112 11211**

**set username 0 0 8**

**zhangsan**

**STORED**

**get username**

**VALUE username 0 8**

**zhangsan**

**END**

**quit**

**Connection closed by foreign host.**

将提供好的“Tomcat7-Memcached JAR包”目录中的.jar包复制到两台tomcat的/usr/local/tomcat/lib/ 下

**[root@node1 ~]#cd /usr/local/tomcat/lib/**

**[root@node1 lib]# rz**

编辑tomcat 配置文件连接指定的 memcached服务器，两台tomcat配置一样

**[root@node1 ~]# vim /usr/local/tomcat/conf/context.xml**

**<Context>**

**<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"**

**memcachedNodes="memA:192.168.200.112:11211 memB:192.168.200.113:11211"**

**requestUrilgnorePattern=".\*\(ico|png|gif|jpg|css|js)$"**

**transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"**

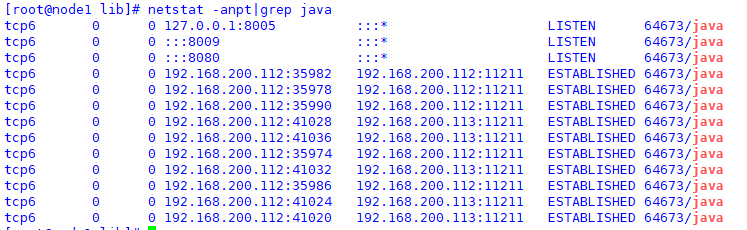
**/>**

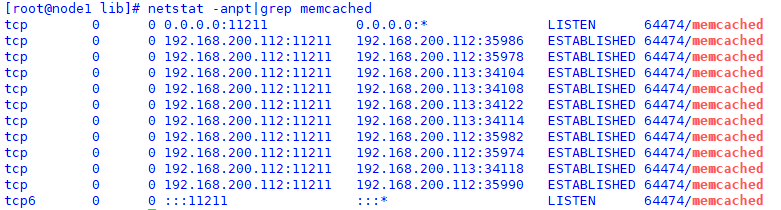
**</Context>**

**[root@node1 ~]# shutdown.sh**

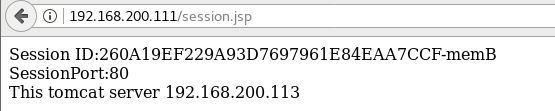
**[root@node1 ~]# startup.sh**

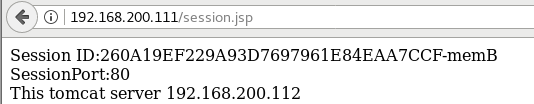
如果成功，tomcat与Memcached 端口会连在一起，前后有变化





浏览器访问测试 [http://192.168.200.111/session.jsp](http://192.168.200.101/session.jsp)

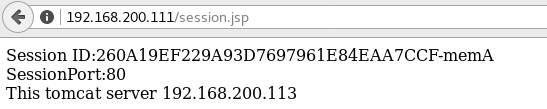


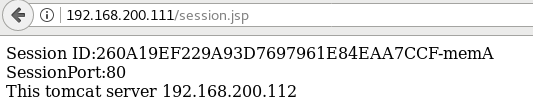


关闭memB：192.168.200.113的memcached服务器，在进行测试发现memcached服务器发生了变化，但是SessionID并没有变化

**[root@node2 ~]# killall -9 memcached**

浏览器访问测试 [http://192.168.200.111/session.jsp](http://192.168.200.101/session.jsp)





## Session服务器之Redis

Redis与Memcached的区别

* 内存利用率：使用简单的key-value（键值对）存储的话，Memcached的内存利用率更高，而如果Redis采用hash结构来做key-value存储，由于其组合式的压缩，其内存利用率会高于Memcached。
* CPU性能对比：由于Redis只使用单核，而Memcached可以使用多核，所以平均每一个核上Redis在存储小数据时比Memcached性能更高。而在100k以上的数据中Memcached性能要高于Redis，虽然Redis最近也在存储大数据的性能上进行优化，但是比起Memcached还是稍有逊色。
* Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。
* Redis支持数据的备份，即master-slave模式的数据备份。

还原两台tomcat快照到tomcat初始状态

安装部署redis,本实验在node1上部署redis，实际环境中需要用专门的服务器部署redis

**[root@node1 ~]# tar xf redis-3.2.5.tar.gz -C /usr/src/**

**[root@node1 ~]# cd /usr/src/redis-3.2.5/**

**[root@node1 redis-3.2.5]# make**

这里如果报错：tclsh8.5: not found 发现少了tcl 报错，则安装tcl

**wget http://downloads.sourceforge.net/tcl/tcl8.5.9-src.tar.gz**

**cd /tcl8.5.9-src/unix**

**./configure**

**make && make install**

接着make就没报错了

**[root@node1 redis-3.2.5]# mkdir -p /usr/local/redis/{bin,etc,var}**

**[root@node1 redis-3.2.5]# cd src/**

src目录下这些文件作用如下

* redis-server：Redis服务器的daemon启动程序
* redis-cli：Redis命令行操作工具.你也可以用telnet根据其纯文本协议来操作
* redis-benchmark：Redis性能测试工具，测试Redis在你的系统及你的配置下的读写性能
* redis-stat：Redis状态检测工具，可以检测Redis当前状态参数及延迟状况

**[root@node1 src]# cp redis-benchmark redis-check-aof redis-cli redis-server /usr/local/redis/bin/**

**[root@node1 src]# cp ../redis.conf /usr/local/redis/etc**

**[root@node1 src]# vim /usr/local/redis/etc/redis.conf //修改配置文件**

**daemonize no 改为 daemonize yes //是否把redis-server启动在后台，默认是“否”。若改成yes，会生成一个pid文件**

**bind 127.0.0.1 改为 bind 0.0.0.0 //任意主机都可访问**

**其他的看需要修改**

关闭redis：（如果启动过的话，没有启动过就直接启动）

**[root@node1 redis-3.2.5]# killall -9 redis-server**

启动redis：

**[root@node1 redis-3.2.5]# /usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf**

查看是否启动：

**[root@node1 redis-3.2.5]# netstat -anpt |grep redis**

**tcp 0 0 0.0.0.0:6379 0.0.0.0:\* LISTEN 77526/redis-server**

将tomcat需要调用redis的jar包（在提供好的” Tomcat7-Redis JAR包”中）放入tomcat/lib

两台tomcat都要配置

[root@node1 ~]# cd /usr/local/tomcat/lib/

[root@node1 lib]# rz

修改context.xml文件以支持调用redis

**[root@node1 redis-3.2.5]# vim /usr/local/tomcat/conf/context.xml**

**在Context段中加入以下内容**

**<Context>**

**<Valve className="com.orangefunction.tomcat.redissessions.RedisSessionHandlerValve" />**

**<Manager className="com.orangefunction.tomcat.redissessions.RedisSessionManager"**

**host="192.168.200.112"**

**port="6379"**

**database="0"**

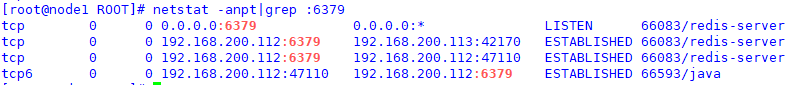
**maxInactiveInterval="60" />**

**</Context>**

**[root@node1 ~]# shutdown.sh**

**[root@node1 ~]# startup.sh**

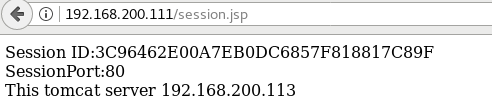
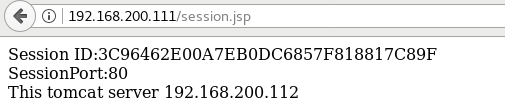
查看两台tomcat是否连接到redis服务器

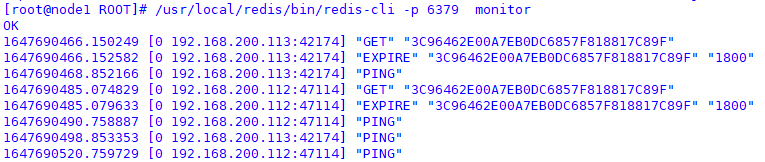


监控redis共享session：

[root@node1 ROOT]# /usr/local/redis/bin/redis-cli -p 6379 monitor

浏览器访问测试 [http://192.168.200.111/session.jsp](http://192.168.200.101/session.jsp)





监控到的Redis服务器的值与浏览器访问的session值是一样的