

一、Tomcat线程池

1.线程池

2.线程池的配置

1.开启并使用

2.更改完重启服务效果

3.参数说明

4.最佳实践

二、Tomcat连接器

1.通用属性

2.最佳实践

三、禁用Tomcat AJP连接器

四、Tomcat热部署与热加载

五、Tomcat JVM参数优化

1.Java栈和Java堆

2.JVM空间

1.Young新生代

2.Old Generation老年代

3.Perm永久代

4.这三个区会占用一定的内存空间，要针对其进行分配

3.常用参数说明

4.在tomcat中设置JVM参数

生产环境 Session 解决方案

1.Session 简介

解决方案一：Session 绑定 基本不用

解决方案二: Session 复制 LC

解决方案三 : Session服务器之 Memcached

解决方案四 : Session 服务器之 Redis

2.实验

1.环境描述

2.环境配置

1.安装前准备配置:

2.Nginx 服务器配置 192.168.200.108

3.安装配置 Tomcat 192.168.200.109 192.168.200.110

建立session.jsp 的测试页面

3.Session 绑定

4.Session 复制 (方法一, 相较于官方更简便)

5.Session 服务器之 Memcached

6.Session 服务器之Redis

Redis 与 Memcached 的区别

内存利用率:

性能对比:

作业: nginx+openssl实现HTTPS

附录:

LVS 源地址 Hash 配置:

Haproxy 源地址 Hash 配置

Tomcat 官方 Session 复制代码 (方法二)

一、Tomcat线程池

在tomcat服务中每一个用户请求都是一个线程,所以可以使用线程池(也叫连接器)来提高性能

1.线程池

线程池是一种多线程处理形式，处理过程中将任务添加到队列，
然后创建线程后自动启动这些任务，线程池线程都是后台线程。
每个线程都使用默认的堆栈大小

它由线程池管理器，工作线程， 任务接口，任务队列组成
线程池的应用及优缺点

单个任务处理的时间短

将需处理的任务的数量大

优势 ? ? ? ? ? ?

减少在创建和销毁线程上所花的时间以及系统资源的开销

缺点 ? ? ? ? ? ? ?

如不使用线程池，有可能造成系统创建大量线程而导致消耗完系统内存以及”度切换? ? ”

2.线程池的配置

1.开启并使用

使用系统自带jdk，部署Tomcat

```
[root@localhost ~]# rz
[root@localhost ~]# tar xf apache-tomcat-8.5.40.tar.gz
[root@localhost ~]# mv apache-tomcat-8.5.40 /usr/local/tomcat8
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
```

修改配置：用户

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/tomcat-users.xml
```

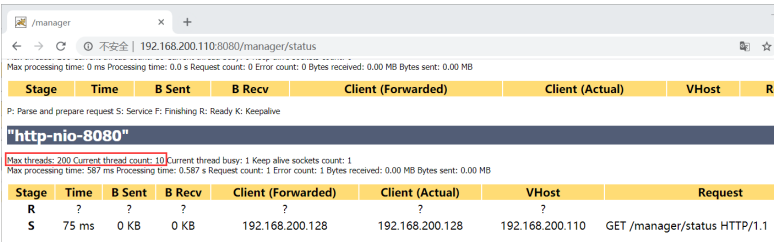
#末尾添加

```
<role rolename="manager-gui"/>
<user username="tomcat" password="tomcat" roles="manager-gui"/>
```

```
[root@localhost ~]# vim /usr/local/tomcat8/webapps/manager/META-INF/context.xml
```

```
<!-- <Valve className="org.apache.catalina.valves.RemoteAddrValve"
      allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />-->
```

访问并查看：



Stage	Time	B Sent	B Recv	Client (Forwarded)	Client (Actual)	VHost	R
S	75 ms	0 KB	0 KB	192.168.200.128	192.168.200.128	192.168.200.110	GET /manager/status HTTP/1.1

针对于修改上面图片中红框部分：

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
```

```
57 <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
```

```
58         maxThreads="400" minSpareThreads="100" prestartminSpareThreads="true"
maxQueueSize="100"/>
```

2.更改完重启服务效果

不容易看到，那个-1的，但是配置是没有问题的

红框中200的位置显示-1

3.参数说明

maxThreads(最大线程数)	默认值是200(可适当调整)如果配置了一个Executor,则该属性的任何值集将被正确记录,但是它将被显示为-1
minSpareThreads(最小活跃线程数)	默认是25(调整活跃线程数的时候必须开启下面的参数)
prestartminSpareThreads(是否在启动时就生成minSpareThreads个线程)	默认是false,改为true则开启
MaxQueueSize(最大的等待队列数，超过则请求拒绝)	基本是无上限，假如你超过最大线程数，就可以给你设置100的等待队列数

4.最佳实践

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
57     <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
58         maxThreads="400" minSpareThreads="100" prestartminSpareThreads="true"
maxQueueSize="100"/>
```

二、Tomcat连接器

Connector是Tomcat接受请求的入口，每个Connector有自己专属的监听端口

Connector有两种：HTTP Connector（8080）和AJP Connector（8009）

1.通用属性

属性	描述
enableLookups	是否进行DNS查找，以IP地址的方式返回，建议关闭，设置为False；起到加速作用
maxPostSize	最大的通过表单提交文件的大小，如果此属性设置为2097152（2兆字节），则大于2兆不能提交
port	端口
protocol	协议：可以指定模式 BIO NIO APR org.apache.coyote.http11.Http11Protocol-阻止Java连接器 org.apache.coyote.http11.Http11NioProtocol-非阻塞Java连接器 org.apache.coyote.http11.Http11AprProtocol- APR /本机连接器
URIEncoding	解决乱码（定义字符集）

除了上面列出的常见的Connector属性之外，

标准HTTP连接器（BIO, NIO, 和APR/native）都支持以下属性

属性	描述
acceptCount	传入连接请求最大队列的长度默认100

acceptorThreadCount	能够接收连接的线程数
compression	压缩功能，一般关闭，Tomcat前端一般会有nginx，而当nginx打开完之后，Tomcat就不用打开了
connectionUploadTimeout	上传过程中，连接的超时时间
disableUploadTimeout	关闭的超时时间
executor	配置线程池用到的
maxConnections	最大连接数
maxThreads	最大线程数，默认200可适当调整
minSpareThreads	最小活跃线程数默认20
SSLEnabled	是否要启动SSL通信

2.最佳实践

```
<Connector port="8080" protocol="org.apache.coyote.http11.Http11NioProtocol"
    connectionTimeout="20000"
    redirectPort="8443"
    enableLookups="false"
    maxPostSize="10485760"
    URIEncoding="UTF-8"
    acceptCount="100"
    acceptorThreadCount="2"
    connectionUploadTimeout="true"
    maxConnections="10000"
    SSLEnabled="false"/>
```

三、禁用Tomcat AJP连接器

AJP（Apache JServer Protocol）给apache提供的代理端口

我们一般使用Nginx+tomcat的架构，用不到AJP协议可以注释掉

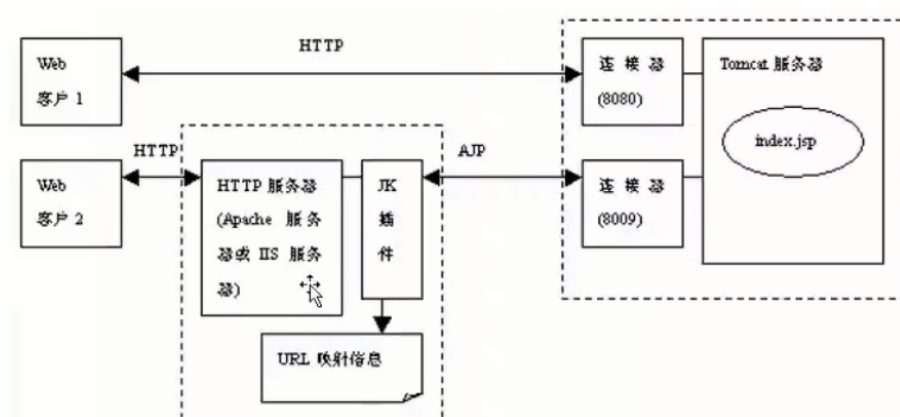


图22-1 Web客户访问Tomcat服务器上的JSP组件的两种方式

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
<!-- Define an AJP 1.3 Connector on port 8009
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
-->

[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
重启将查看不到8009端口
[root@localhost ~]# netstat -lnpt | grep java
tcp6      0      0 127.0.0.1:8005          :::*           LISTEN
58771/java
tcp6      0      0 :::8080                :::*           LISTEN
58771/java
```

四、Tomcat热部署与热加载

解决面试相关问题；生产经验用的多
项目开发过程中，经常要改动tomcat的JSP文件

不想重新启动服务的原因：

1. 服务器从新启动需要花费很长时间
2. 重启后报错率高
3. 线上业务不能中断，无法实现重启

开发运行中想看到结果，想直接获得debug结果

这时有两种方式：热部署和热加载

都是在 server.xml -> context属性中设置的

热加载：reloadable= "true"

热部署：autoDeploy= "true" unpackWARS="true"

两者区别：

热加载：服务器会监听class（类）文件改变，包括web-inf/class, wen-inf/lib, web-inf/web.xml等文件，若发生更改，则局部进行加载，**不清空session，不释放内存。开发中用的多**，但是要考虑内存溢出的情况。

热部署：整个项目重新部署，包括你重新打上.war文件。**会清空session，释放内存，项目打包的时候用的多**。

五、Tomcat JVM参数优化

作为一名运维工程师来说，会调整JVM的值就行

Tomcat基于Java开发，在安装之前要安装jdk, 不管是系统自带，还是自己安装

jdk核心就是JVM，即java Virtual Machine Java虚拟机

Tomcat中的所有代码都是在JVM中跑的

所以对虚拟机的优化也是对其中的代码的优化

1.Java栈和Java堆

适当调整tomcat的运行jvm参数可以提升整体性能

1. Java栈

Java栈与每一个线程关联的，jvm在创建每个线程的时候，

会分配一定的栈空间给线程使用

它主要用来存储 线程执行过程中的局部变量和方法的返回值，

以及方便调用上下文，栈空间随着线程的终止而释放

2. Java堆

Java中堆是由所有的线程共享的一块内存区域，

堆用来保存各种java对象，比如数组，线程对象等



2.JVM空间

1.Young新生代

新生成的对象都是放在新生代的

新生代的目标就是尽可能快速的回收掉那些生命周期短的对象

新生代分为三个区：

一个Eden区, 两个survivor区(一般情况)

大部分对象是在Eden区中生成，某一个时刻只有其中一个是被使用的，当Eden区满了，GC（Java中的垃圾回收器）就会将存活的对象转移到空闲的survivor区间中，根据jvm的策略，在经过几次垃圾收集后，仍然存活于survivor的对象将被移动到old generation区间。（新生成的对象都在新生代，java的GC会进行垃圾回收你这个对象，但是它发现你这个内存对象还在应用，根据几次的垃圾回收，仍然还有些对象没有被回收的话就给他放“老年代里，放到老年代里不一定不会被GC回收）

2.Old Generation老年代

Oldgeneration区主要保存生命周期长的对象，一般是一些老的对象，当一些对象在Young复制转移到一定次数以后，内存对象就会被转移到old generation区，一般如果系统中用application级别的缓存，缓存中的对象往往会被转移到这一区间，放到老年代里不一定不会被GC回收

3.Perm永久代

Perm主要保存class, method, filed对象，并不会被GC回收

4.这三个区会占用一定的内存空间，要针对其进行分配

Total heap

- -Xms: 指定了jvm初始启动以后初始化内存；JVM刚启动占用内存大小
- -Xmx: 指定jvm堆的最大内存，在jvm启动以后，会分配-Xmx参数指定大小的内存给JVM,但是 不一定全都使用，jvm 会根据- Xms参数来调节真正用于jvm的内存
- Xmx- Xms=virtual大小(之差就是三个virtual空间大小)

年轻代

- -XX:NewRatio=8 意味着oidgeneration和young的比值8: 1,这样eden+2*survivr=1/9堆内存
- -XX:SurvivorRatio=32意味着eden和一个survivor的比值是32: 1,这样一个Survivor就占young区的1/34
- -Xmn参数设置了年轻代的大小

永久代

- -XX:PermSize=16M -XX : MaxPermSize=64M

Thread Stack

- -XX:Xss=128k

3.常用参数说明

1. file. encoding 默认文件编码
2. -Xmx1024m 设置JVM最大可用内存为1024MB

3. -Xms1024m 置JVM最小内存为1024m, 此值可以设置与-Xmx相同, 以避免每次垃圾回收完成后JVM重新分配内存
4. XX:NewSize 设置年轻代
5. XX:MaxNewSize 设置最大的年轻代大小
6. -XX:PermSize 设置永久代大小
7. -XX:MaxPermSize 设置最大永久代大小

4.在tomcat中设置JVM参数

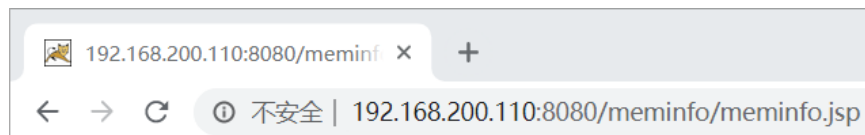
Tomcat性能取决于内存大小

- **上策：优化代码（需要开发经验足够丰富，对开发人员要求较高）**
- **中策：jvm优化机制 垃圾回收机制 把不需要的内存回收（运维的工作）**
- **下策：加足够大的内存（内存条）**
- **下下策：每天0点定时重启tomcat（使用较为广泛），写个计划任务**

中策：优化catalina.sh配置文件

```
[root@localhost local]# cd /usr/local/tomcat8/webapps/
[root@localhost webapps]# ls
docs  examples  host-manager  manager  ROOT
[root@localhost webapps]# rz
[root@localhost webapps]# ls
docs  examples  host-manager  manager  meminfo  meminfo.war  ROOT
```

访问测试 “

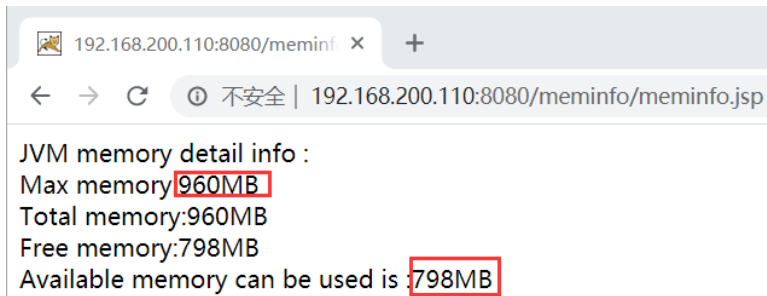


```
JVM memory detail info :
Max memory:441MB
Total memory:98MB
Free memory:67MB
Available memory can be used is :410MB
```

```
[root@localhost ~]# vim /usr/local/tomcat8/bin/catalina.sh
```

#在开头添加下面这一行，在公司就不是按照这个大小设置了，配4G或8G

```
JAVA_OPTS="-Djava.awt.headless=true -Dfile.encoding=UTF-8 -server -Xms1024m -Xmx1024m
-XX:NewSize=512m
-XX:MaxNewSize=512m -XX:PermSize=512m -XX:MaxPermSize=512m"
```



生产环境 Session 解决方案

1.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 to 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多。

解决方案一：Session 绑定 基本不用

10台以内还好，超过10台效果就不好了

Session 绑定可以利用负载均衡的源地址 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。

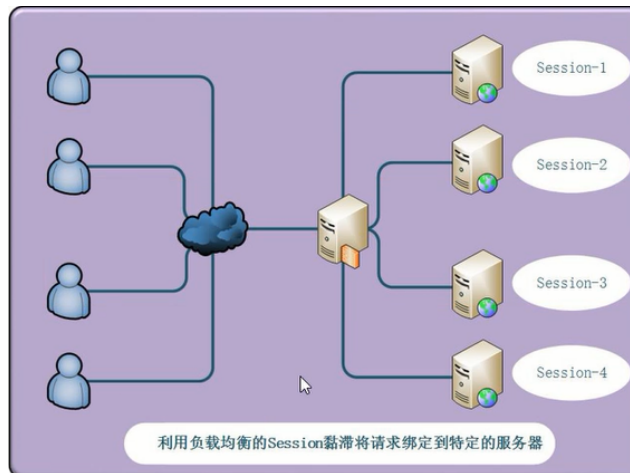
（负载均衡集群 软件 Nginx LVS 传输层 Haproxy）

这样整个会话期间，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

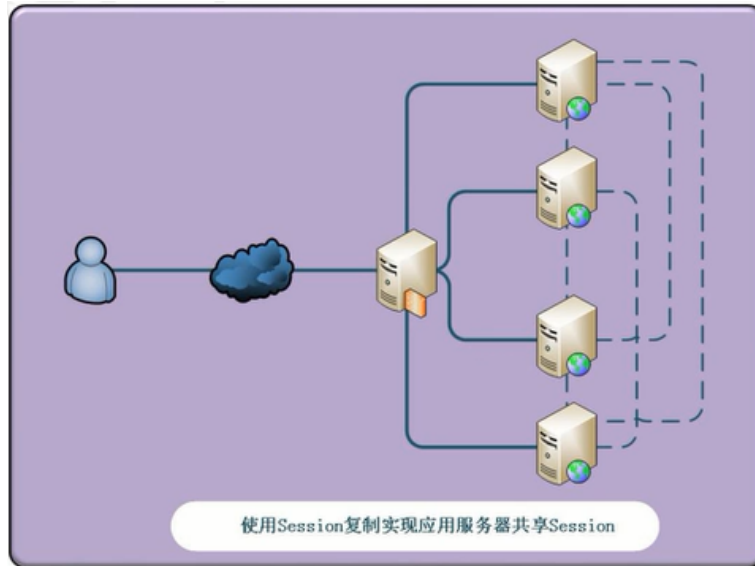
解决方案二: Session 复制 LC

Session复制是小型架构（与公司大小没关系，与机器数量有关系）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（用的是组播），
（进行大量组播通信的时候）占用服务器和网络的大量资源（占用大量内存），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

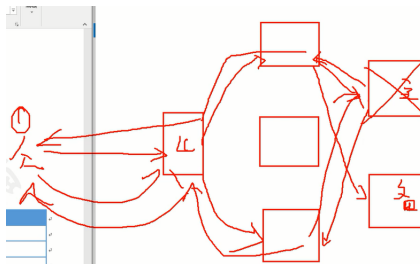
解决方案三：Session服务器之 Memcached

内存型服务器

缺点：

当内存重启的时候，会将服务器中的数据全都释放

使用这种方案需要开启多台Memcached机器，做好备份，-----分布式服务
非关系型数据库



解决方案四：Session 服务器之 Redis

内存型数据库：将session信息存储到数据库中

支持存储数据的方式都支持key / value

redis支持持久化：
平时用内存数据存储在，但是会定期将数据同步到磁盘上
当机器中断再次启动的时候，会从磁盘中将数据加载到内存中，从而保证内存不会丢

ps：
mq消息队列，是Tomcat的另一个机制
对于开发来说，分为生产者模型，消费者模型，
例如：淘宝中的卖家和买家；卖家是生产者，买家是消费者
发朋友圈；发布者是生产者，观看者是消费者
消息队列的作用是在两者之间消息传递的一种排队机制，并非是用来解决session问题的
但是对很多很复杂的业务来说，带有交易性质，用户状态，都会用到消息队列

2.实验

1.环境描述

操作系统	IP 地址	主机名	软件包列表
CentOS7.5-x86_64	192.168.200.108	nginx	nginx
CentOS7.5-x86_64	192.168.200.109	Node1	JDK Tomcat
CentOS7.5-x86_64	192.168.200.110	Node2	JDK Tomcat

2.环境配置

1.安装前准备配置：

配置所有机器：

```
[root@localhost ~]# cat /etc/hosts
192.168.200.108 nginx
192.168.200.109 node1
192.168.200.110 node2
[root@localhost ~] scp /etc/hosts192.168.200.109:/etc/hosts
[root@localhost ~] scp /etc/hosts192.168.200.110:/etc/hosts
[root@localhost ~]# iptables -F
[root@localhost ~]# systemctl stop firewalld
[root@localhost ~]# setenforce 0
```

2.Nginx 服务器配置 192.168.200.108

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 --
with-file-aio
--with-http_stub_status_module --with-http_ssl_module --with-http_flv_module --with-
http_gzip_static_module && make && make install
```

```
--prefix=/usr/local/nginx          # 指定安装目录
--user=nginx --group=nginx          # 指定运行的用户和组
--with-file-aio                    # 启用文件修改支持
--with-http_stub_status_module      # 启用状态统计
--with-http_ssl_module              # 启用 ssl 模块
--with-http_flv_module              # 启用 flv 模块，提供寻求内存使用基于时间的偏移量文
件
--with-http_gzip_static_module      # 启用 gzip 静态压缩
```

```
[root@nginx ~]# ln -s /usr/local/nginx/sbin/nginx /sbin/
[root@nginx ~]# vim /usr/local/nginx/conf/nginx.conf
34 upstream tomcat_pool {
35     server 192.168.200.109:8080 weight=1 max_fails=1 fail_timeout=10s;
36     server 192.168.200.110:8080 weight=1 max_fails=1 fail_timeout=10s;
37 }
38
39 server {
40     listen 80;
41     server_name localhost;
42     charset utf-8;
43
44     location / {
45         root html;
46         index index.html index.htm;
47         proxy_pass http://tomcat_pool;
48 }
[root@nginx ~]# nginx
```

3.安装配置 Tomcat 192.168.200.109 192.168.200.110

安装 JDK 方法



软件包格式	描述
jdk-*.tar.gz	源代码包，解压后配置环境变量
jdk-*.bin	通用二进制，给执行权限，chmod +x ， ./ 来执行 执行后配置环境变量
jdk-*.rpm	1、装直接安装 rpm 软件包 2、先安装 bin 软件包，生成.rpm 格式的（通用二进制）
jdk-*.exe	Windows平台使用，直接安装使用

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
```

启动 Tomcat

```
[root@tomcat1 ~]# /usr/local/tomcat/bin/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

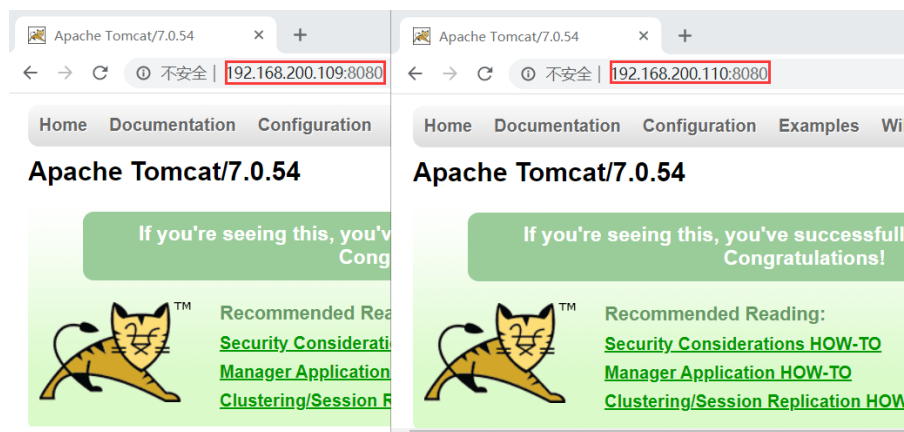
Tomcat started.

Tomcat 默认运行在 8080 端口

```
[root@tomcat1 ~]# netstat -anpt |grep :8080
```

浏览器访问测试 http://192.168.200.109:8080

浏览器访问测试 http://192.168.200.110:8080



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() %>
```

```
<% out.println("This tomcat server 192.168.200.109");%>
```

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

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

```
SessionPort:<%= request.getServerPort() %>
```

```
<% out.println("This tomcat server 192.168.200.110");%>
```

☐ 重新启动 Tomcat

```
☐ [root@node1 ~]# /usr/local/tomcat/bin/shutdown.sh && /usr/local/tomcat/bin/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
```

```
☐ 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.
```

192.168.200.109:8080/session.jsp

Session ID:9A655D6AF8D14EF160E7CF3215B20137
SessionPort:8080 This tomcat server 192.168.200.109

192.168.200.110:8080/session.jsp

Session ID:4D0C9F4328DCED7803EEA6958685595F
SessionPort:8080 This tomcat server 192.168.200.110

192.168.200.109:8080/session.jsp

Session ID:0C6171092663CDED8F5368655EBA64B1
SessionPort:8080 This tomcat server 192.168.200.109

192.168.200.110:8080/session.jsp

Session ID:82BEE5A4342DFA3895D09DB9AB7F542C
SessionPort:8080 This tomcat server 192.168.200.110

192.168.200.108/session.jsp

Session ID:15781B99262A041667195D9B58D7DD09
SessionPort:80 This tomcat server 192.168.200.110

192.168.200.108/session.jsp

Session ID:14EC5F560DDC62E9C25D4C9E244995CE
SessionPort:80 This tomcat server 192.168.200.109

nginx调度-轮询

3.Session 绑定

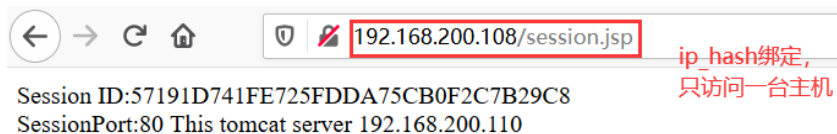
```
[root@nginx ~]# vim /usr/local/nginx/conf/nginx.conf
```

```
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
```

用户访问测试只被分配到一台机器上

浏览器访问测试 <http://192.168.200.108/session.jsp>



4.Session 复制 (方法一, 相较于官方更简便)

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;消耗服务器的资源。

修改 Tomcat 配置文件

```
[root@node1 conf]# cp server.xml server.xml.bak
```

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

将 Engine 这一行修改为:

旧: <Engine name="Catalina" defaultHost="localhost">

新: <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 ~]# vim /usr/local/tomcat/webapps/ROOT/WEB-INF/web.xml
```

末尾添加 倒数第二行 <distributable/>

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

<distributable/> # 添加内容

</web-app>

重新启动 Tomcat

```
[root@node1 ~]# /usr/local/tomcat/bin/shutdown.sh && /usr/local/tomcat/bin/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

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.

查看端口状态:

```
[root@node1 conf]# netstat -anpt | grep -E "8080|4000"
```

```
[root@node1 ~]# netstat -anpt | grep -E "8080|4000"
```

```
tcp6      0      0 192.168.200.109:4000 :::*          LISTEN
59658/java
```

```
tcp6      0      0 :::8080      :::*          LISTEN
59658/java
```

```
tcp6      0      0 192.168.200.109:4000 192.168.200.110:44300 ESTABLISHED
59658/java
```

```
tcp6      0      0 192.168.200.109:41108 192.168.200.110:4000 ESTABLISHED
59658/java
```

```
[root@node1 ~]# grep "4000" /usr/local/tomcat/logs/catalina.out
```

信息: Receiver Server Socket bound to:/192.168.200.109:4000

信息: Replication member

added:org.apache.catalina.tribes.membership.MemberImpl[tcp://{192, 168, 200,

110}:4000,{192, 168, 200, 110},4000, alive=22104, securePort=-1, UDP Port=-1, id={76

11 -36 55 1 76 74 123 -127 -16 76 -70 57 3 -117 7 }, payload={}, command={}, domain=

{},]信息: Manager [localhost#], requesting session state from

org.apache.catalina.tribes.membership.MemberImpl[tcp://{192, 168, 200, 110}:4000,{192,

168, 200, 110},4000, alive=25633, securePort=-1, UDP Port=-1, id

={76 11 -36 55 1 76 74 123 -127 -16 76 -70 57 3 -117 7 }, payload={}, command={},

domain={},]. This operation will timeout if no session state has been received within

60 seconds.

```
[root@node2 ~]# netstat -anpt | grep -E ":(8080|4000)"
```

```
tcp6      0      0 192.168.200.110:4000 :::*          LISTEN
59380/java
```

```

tcp6      0      0 :::8080          :::*              LISTEN
59380/java
tcp6      0      0 192.168.200.110:4000 192.168.200.109:41108 ESTABLISHED
59380/java
tcp6      0      0 192.168.200.110:44300 192.168.200.109:4000 ESTABLISHED
59380/java

```

[root@node2 ~]# grep "4000" /usr/local/tomcat/logs/catalina.out

信息: Receiver Server Socket bound to:/192.168.200.110:4000

信息: Replication member

added:org.apache.catalina.tribes.membership.MemberImpl[tcp://{192, 168, 200, 109}:4000,{192, 168, 200, 109},4000, alive=1040, securePort=-1, UDP Port=-1, id={29 -87 -62 76 90 -21 68 105 -85 101 -65 -18 1 -99 47 -97 }, payload={}, command={}, domain={},]

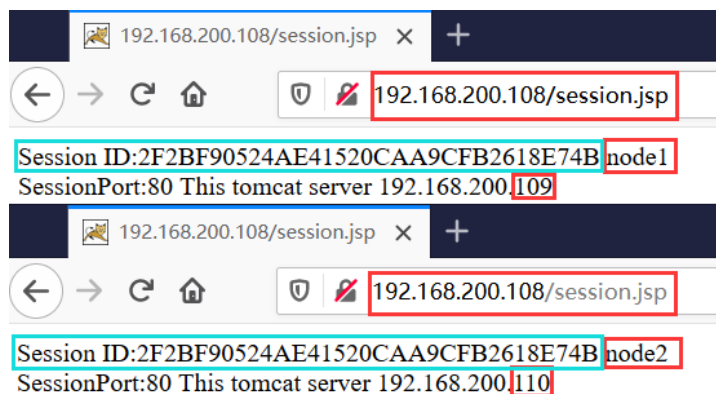
Session 测试: (记得将ip_hash取消)

浏览器访问测试 http://192.168.200.112:8080/session.jsp 刷新后 session ID 未发生变化

浏览器访问测试 http://192.168.200.113:8080/session.jsp 刷新后 session ID 未发生变化

浏览器访问测试 http://192.168.200.111/session.jsp

刷新后 session ID 未发生变化, 但是 node 标签及 IP 地址在变化



[root@node1 ~]# netstat -anpt | grep -E "8080|4000"

```

tcp6      0      0 192.168.200.109:4000 :::*              LISTEN
59658/java
tcp6      0      0 :::8080          :::*              LISTEN
59658/java
tcp6      0      0 192.168.200.109:41110 192.168.200.110:4000 ESTABLISHED
59658/java
tcp6      0      0 192.168.200.109:4000 192.168.200.110:44300 ESTABLISHED
59658/java
tcp6      0      0 192.168.200.109:41108 192.168.200.110:4000 ESTABLISHED
59658/java

```

[root@node2 ~]# netstat -anpt | grep -E "8080|4000"

```

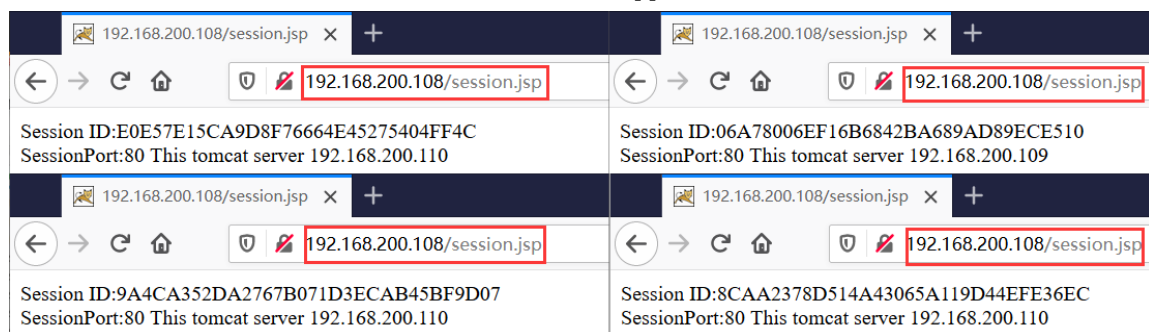
tcp6      0      0 192.168.200.110:4000 :::*          LISTEN
59380/java
tcp6      0      0 :::8080      :::*          LISTEN
59380/java
tcp6      0      0 192.168.200.110:4000 192.168.200.109:41108 ESTABLISHED
59380/java
tcp6      0      0 192.168.200.110:4000 192.168.200.109:41110 ESTABLISHED
59380/java
tcp6      0      0 192.168.200.110:44300 192.168.200.109:4000  ESTABLISHED
59380/java

```

还原环境，取消之前几个文件的设置

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

```
[root@node1 ~]# vim /usr/local/tomcat/webapps/ROOT/WEB-INF/web.xml
```



5.Session 服务器之 Memcached

两台 tomcat 上安装 Memcached

```
[root@node1 ~]# netstat -anpt | grep :11211
```

```

tcp      0      0 0.0.0.0:11211      0.0.0.0:*          LISTEN
60622/memcached
tcp6     0      0 :::11211          :::*          LISTEN
60622/memcached

```

选项:

- h #查看帮助信息
- p #指定 memcached 监听的端口号默认 11211
- l #memcached 服务器的 ip 地址
- u #memcached 程序运行时使用的用户身份必须是 root 用户
- m #指定使用本机的多少物理内存存数据默认 64M
- c #memcached 服务的最大链接数
- vvv #显示详细信息
- n #chunk size 的最小空间是多少单位字节
- f #chunk size 大小增长的倍数默认 1.25 倍

-d #在后台启动

```
[root@node1 ~]# netstat -antp | grep :11211
```

(检测 memcached 是否存活, memcached 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 # 描述符 过期时间 长度
```

```
zhangsan
```

```
STORED
```

```
get username
```

```
VALUE username 0 8
```

```
zhangsan
```

```
END
```

```
quit
```

Connection closed by foreign host.

最后执行让 Tomcat-1 Tomcat-2 通过 (msm 表示一堆文件) 连接到 Memcached

将 session 包中的 "*.jar 复制到/usr/local/tomcat/lib/ 下面

```
[root@node1 ~]# mkdir mem
```

```
[root@node1 ~]# cd mem
```

```
[root@node2 mem]# cp *.jar /usr/local/tomcat/lib/
```

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:192.168.200.109:11211 memB:192.168.200.110:11211"
```

```
# memcached 节点
```

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

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

```
/>
```

```
</Context>
```

```
[root@node1 ~]# /usr/local/tomcat/bin/shutdown.sh
```

```
[root@node1 ~]# /usr/local/tomcat/bin/startup.sh
```

重启时会有memcache输出信息, 可以另开一个窗口

如果成功, tomcat 与 Memcached 端口会连在一起, 前后有变化

Tomcat-1 与 Tomcat-2:

```
[root@node1 ~]# netstat -anpt | grep memcached
```

tcp	0	0 0.0.0.0:11211	0.0.0.0:*	LISTEN
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.109:41162	ESTABLISHED
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.109:41158	ESTABLISHED
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.109:41166	ESTABLISHED
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.110:47982	ESTABLISHED
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.110:47994	ESTABLISHED
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.109:41174	ESTABLISHED
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.110:47990	ESTABLISHED
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.109:41170	ESTABLISHED
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.110:47978	ESTABLISHED
60622/memcached				
tcp	0	0 192.168.200.109:11211	192.168.200.110:47986	ESTABLISHED
60622/memcached				
tcp6	0	0 :::11211	:::*	LISTEN
60622/memcached				
[root@node1 ~]# netstat -anpt grep java				
tcp6	0	0 127.0.0.1:8005	:::*	LISTEN
61962/java				
tcp6	0	0 :::8009	:::*	LISTEN
61962/java				
tcp6	0	0 :::8080	:::*	LISTEN
61962/java				
tcp6	0	0 192.168.200.109:41174	192.168.200.109:11211	ESTABLISHED
61962/java				
tcp6	0	0 192.168.200.109:41158	192.168.200.109:11211	ESTABLISHED
61962/java				
tcp6	0	0 192.168.200.109:59870	192.168.200.110:11211	ESTABLISHED
61962/java				

```

tcp6      0      0 192.168.200.109:41162 192.168.200.109:11211 ESTABLISHED
61962/java
tcp6      0      0 192.168.200.109:59882 192.168.200.110:11211 ESTABLISHED
61962/java
tcp6      0      0 192.168.200.109:59866 192.168.200.110:11211 ESTABLISHED
61962/java
tcp6      0      0 192.168.200.109:41166 192.168.200.109:11211 ESTABLISHED
61962/java
tcp6      0      0 192.168.200.109:59878 192.168.200.110:11211 ESTABLISHED
61962/java
tcp6      0      0 192.168.200.109:59874 192.168.200.110:11211 ESTABLISHED
61962/java
tcp6      0      0 192.168.200.109:41170 192.168.200.109:11211 ESTABLISHED
61962/java
[root@node2 ~]# netstat -anpt | grep memcached
tcp      0      0 0.0.0.0:11211      0.0.0.0:*          LISTEN
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.109:59874 ESTABLISHED
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.110:38740 ESTABLISHED
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.110:38748 ESTABLISHED
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.109:59870 ESTABLISHED
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.110:38752 ESTABLISHED
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.109:59882 ESTABLISHED
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.110:38744 ESTABLISHED
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.109:59866 ESTABLISHED
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.110:38756 ESTABLISHED
60416/memcached
tcp      0      0 192.168.200.110:11211 192.168.200.109:59878 ESTABLISHED
60416/memcached

```

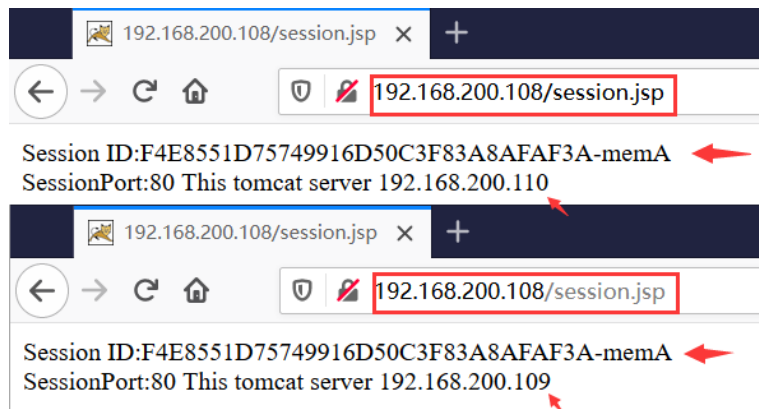
```

tcp6      0      0 :::11211          :::*               LISTEN
60416/memcached

[root@node2 ~]# netstat -anpt | grep java
tcp6      0      0 127.0.0.1:8005    :::*               LISTEN
61824/java
tcp6      0      0 :::8009           :::*               LISTEN
61824/java
tcp6      0      0 :::8080           :::*               LISTEN
61824/java
tcp6      0      0 192.168.200.110:38748 192.168.200.110:11211 ESTABLISHED
61824/java
tcp6      0      0 192.168.200.110:47986 192.168.200.109:11211 ESTABLISHED
61824/java
tcp6      0      0 192.168.200.110:47982 192.168.200.109:11211 ESTABLISHED
61824/java
tcp6      0      0 192.168.200.110:47978 192.168.200.109:11211 ESTABLISHED
61824/java
tcp6      0      0 192.168.200.110:47994 192.168.200.109:11211 ESTABLISHED
61824/java
tcp6      0      0 192.168.200.110:47990 192.168.200.109:11211 ESTABLISHED
61824/java
tcp6      0      0 192.168.200.110:38744 192.168.200.110:11211 ESTABLISHED
61824/java
tcp6      0      0 192.168.200.110:38752 192.168.200.110:11211 ESTABLISHED
61824/java
tcp6      0      0 192.168.200.110:38756 192.168.200.110:11211 ESTABLISHED
61824/java
tcp6      0      0 192.168.200.110:38740 192.168.200.110:11211 ESTABLISHED
61824/java

```

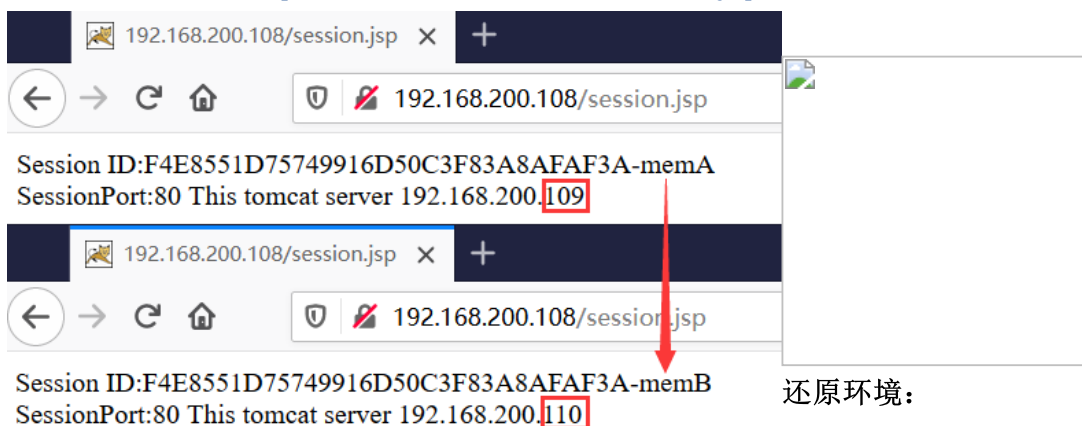
浏览器访问测试 <http://192.168.200.108/session.jsp>



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://192.168.200.108/session.jsp>



还原环境:

```
[root@node1 ~]# killall -9
```

memcached

```
[root@node1 ~]# netstat -lnpt | grep 11211
```

```
[root@node1 ~]# vim /usr/local/tomcat/conf/context.xml #删除添加内容
```

删除添加的jar包

```
[root@node1 ~]# ls mem/ | wc -l
```

11

```
[root@node1 ~]# ls /usr/local/tomcat/lib | wc -l
```

32

用脚本: 基于一个目录删除另一个目录重复的内容

```
[root@node2 ~]# ls mem/ | while read line
```

```
> do
```

```
> rm -rf /usr/local/tomcat/lib/$line
```

```
> done
```

```
[root@node2 ~]# ls /usr/local/tomcat/lib | wc -l
```

21

6.Session 服务器之Redis

内存型更支持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 Memcached。

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，而且memcached的几台机器之间不能互相传送是数据，独立运行虽然 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。针对上面这个问题：

解决办法：在一台机器上跑多实例，从而将性能充分发挥

redis ：数据的可靠性非常好

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。

Redis 不仅仅支持简单的 key-Value 类型的数据，同时还提供 list，set，zset，hash 等数据结构的存储。

安装部署Redis 192.168.200.109

```
[root@node1 ~]# mkdir redis
[root@node1 ~]# cd redis
[root@node1 redis]# rz
[root@node1 redis]# ls
commons-logging-1.1.3.jar  jedis-2.5.2.jar  tomcat-redis-session-manage-tomcat7.jar
commons-pool2-2.2.jar      tomcat-juli.jar
[root@node1 redis]# cd
[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 test
```

这里如果报错：tclsh8.5: not found 发现少了 [tcl 报错](#)，则安装 tcl

```
[root@node1 redis-3.2.5]# wget http://downloads.sourceforge.net/tcl/tcl8.5.9-src.tar.gz
[root@node1 redis-3.2.5]# tar xf tcl8.5.9-src.tar.gz
[root@node1 redis-3.2.5]# cd tcl8.5.9/unix
[root@node1 unix]# ./configure
[root@node1 unix]# 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;服务端程序（服务端软件）

- 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; 客户端连接工具
- 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
- redis-check-aof 持久化的

```
[root@node1 src]# cd ../
[root@node1 redis-3.2.5]# ls
redis.conf    # 主配置文件
[root@node1 src]# cp redis-benchmark redis-check-aof redis-cli redis-server
/usr/local/redis/bin/
[root@node1 src]# cd ../
[root@node1 redis-3.2.5]# cp redis.conf /usr/local/redis/etc
[root@node1 redis-3.2.5]# vim /usr/local/redis/etc/redis.conf      // 修改配置文件
128 daemonize yes
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
61 bind 0.0.0.0
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:

```
[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      (为啥要加配置文件的路径)
```

查看是否启动:

```
[root@node1 redis-3.2.5]# netstat -anpt |grep redis
[root@node1 redis-3.2.5]# netstat -anpt |grep redis
tcp        0      0 0.0.0.0:6379          0.0.0.0:*            LISTEN
77496/redis-server
tcp        0      0 127.0.0.1:21231      0.0.0.0:*            LISTEN
71573/src/redis-ser
```

监控 redis 共享 session:

```
[root@node1 redis-3.2.5]# /usr/local/redis/bin/redis-cli -p 6379 monitor      # 这是实时监控的命令
1555299973.187326 [0 192.168.200.113:44980] "EXPIRE"
"29C6E0F087C504F4C90BBEAE924F20DD" "1800"
1555299973.440488 [0 192.168.200.112:64702] "GET"
```

```
"29C6E0F087C504F4C90BBEAE924F20DD"
```

```
1555299973.441966 [0 192.168.200.112:64702] "EXPIRE"
```

```
"29C6E0F087C504F4C90BBEAE924F20DD" "1800"
```

```
1555299976.826974 [0 192.168.200.112:64702] "PING"
```

复制jar包（库文件）-----修改配置文件

将 tomcat 需要调用 redis 的 jar 包放入 tomcat/lib

```
[root@node1 ~]# ls redis
```

```
[root@node1 ~]# cp redis/* /usr/local/tomcat/lib/
```

```
[root@node1 ~]# ls /usr/local/tomcat/lib/ | wc -l
```

```
37
```

```
[root@node1 ~]# scp redis/* 192.168.200.110:/usr/local/tomcat/lib/
```

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(同时修改192.168.200.110)

```
<Context>
```

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

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

```
host="192.168.200.109"           //redis的IP地址
```

```
port="6379"                     //redis的端口
```

```
database="0"
```

```
maxInactiveInterval="60" />
```

```
</Context>
```

```
[root@node1 ~]# /usr/local/tomcat/bin/shutdown.sh
```

```
[root@node1 ~]# /usr/local/tomcat/bin/startup.sh
```

```
[root@node1 ~]# netstat -anpt | grep :6379
```

```
tcp        0      0 0.0.0.0:6379          0.0.0.0:*             LISTEN
77496/redis-server
```

```
tcp        0      0 127.0.0.1:6379        127.0.0.1:39866        ESTABLISHED
77496/redis-server
```

```
tcp        0      0 127.0.0.1:39866       127.0.0.1:6379        ESTABLISHED
77916/redis-cli
```

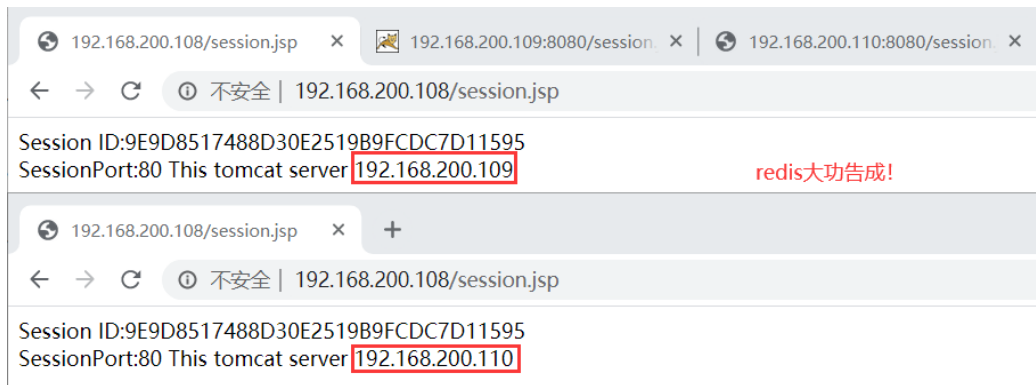
```
[root@node1 ~]# netstat -anpt | grep :6379    # 访问刷新的次数越多，这里的内容就越多
```

```
tcp        0      0 0.0.0.0:6379          0.0.0.0:*             LISTEN
77496/redis-server
```

```
tcp        0      0 127.0.0.1:6379        127.0.0.1:39870        ESTABLISHED
77496/redis-server
```

```
tcp        0      0 127.0.0.1:39870       127.0.0.1:6379        ESTABLISHED
78703/redis-cli
```

浏览器访问测试 <http://192.168.200.108/session.jsp>



```
[root@node2 ~]# netstat -anpt | grep :6379    # 只有访问了，才显示连接端口
tcp6      0      0 192.168.200.110:60762    192.168.200.109:6379    ESTABLISHED
59919/java
```

作业: [nginx+openssl实现HTTPS](#)

附录:

LVS 源地址 Hash 配置:

```
[root@lvs ~]# rpm -ivh /media/cdrom/Packages/ipvsadm-1.26-4.el6.x86_64.rpm
Preparing... ##### [100%]
1:ipvsadm ##### [100%]
[root@lvs ~]# ifconfig eth0:0 192.168.200.254 broadcast 192.168.200.254 netmask
255.255.255.255 up
[root@lvs ~]# route add -host 192.168.200.254 dev eth0:0
[root@lvs ~]# ipvsadm -A -t 192.168.200.254:8080 -s sh
[root@lvs ~]# ipvsadm -a -t 192.168.200.254:8080 -r 192.168.200.112:8080 -g
[root@lvs ~]# ipvsadm -a -t 192.168.200.254:8080 -r 192.168.200.113:8080 -g
[root@node1 ~]# vim realserver.sh
#!/bin/bash
VIP="192.168.200.254"
/sbin/ifconfig lo:0 $VIP broadcast $VIP netmask 255.255.255.255 up
/sbin/route add -host $VIP dev lo:0
echo "1" > /proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" > /proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" > /proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" > /proc/sys/net/ipv4/conf/all/arp_announce
```

```
[root@node1 ~]# bash realserver.sh
[root@lvs ~]# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 192.168.200.254:8080 sh
-> 192.168.200.112:8080 Route 1 0 0
-> 192.168.200.113:8080 Route 1 0 0
```

测试:

用浏览器打开 <http://192.168.200.254:8080/session.jsp>

```
[root@lvs ~]# ipvsadm -Lnc
IPVS connection entries
pro expire state source virtual destination
TCP 00:31 FIN_WAIT 192.168.200.2:53487 192.168.200.254:8080 192.168.200.113:8080
TCP 00:52 SYN_RECV 192.168.200.2:53527 192.168.200.254:8080 192.168.200.113:8080
TCP 00:52 SYN_RECV 192.168.200.2:53526 192.168.200.254:8080 192.168.200.113:8080
TCP 00:27 FIN_WAIT 192.168.200.2:53486 192.168.200.254:8080 192.168.200.113:8080
TCP 00:31 FIN_WAIT 192.168.200.2:53488 192.168.200.254:8080 192.168.200.113:8080
TCP 00:31 FIN_WAIT 192.168.200.2:53485 192.168.200.254:8080 192.168.200.113:8080
[root@lvs ~]# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 192.168.200.254:8080 sh
-> 192.168.200.112:8080 Route 1 0 0
-> 192.168.200.113:8080 Route 1 0 8
[root@lvs ~]# ipvsadm -d -t 192.168.200.254:8080 -r 192.168.200.113:8080
```

测试:

用浏览器打开 <http://192.168.200.254:8080/session.jsp>

Haproxy 源地址 Hash 配置

编译安装 Haproxy

```
[root@haproxy ~]# yum -y install pcre-devel bzip2-devel
[root@haproxy ~]# tar xf haproxy-1.4.24.tar.gz -C /usr/src/
[root@haproxy ~]# cd /usr/src/haproxy-1.4.24/
[root@haproxy haproxy-1.4.24]# make TARGET=linux26 && make install
```

建立 haproxy 的配置目录及文件

```
[root@haproxy haproxy-1.4.24]# mkdir /etc/haproxy
```

```
[root@haproxy haproxy-1.4.24]# cp examples/haproxy.cfg /etc/haproxy/
```

haproxy 配置项的介绍

haproxy 的配置文件通常分为三部分:

global (全局配置部分)

defaults (默认配置部分)

listen(应用组件部分)

```
[root@haproxy ~]# egrep -v "#|^$" /etc/haproxy/haproxy.cfg
```

global

log 127.0.0.1 local0 # 配置日志记录, local0 为日志设备, 默认存放到系统日志

log 127.0.0.1 local1 notice #notice 为日志级别, 通常有 24 个级别

maxconn 4096 # 最大连接数

uid 99 # 用户 uid

gid 99 # 用户 gid

daemon # 守护进程模式

defaults

log global # 定义日志为 global 配置中的日志定义

mode http # 模式为 http

option httplog # 采用 http 日志格式记录日志

option dontlognull

retries 3 # 检查节点服务器失败次数

maxconn 2000 # 最大连接数

timeout 5000 # 连接超时时间

clitimeout 50000 # 客户端超时时间

srvtimeout 50000 # 服务器超时时间

listen webcluster 0.0.0.0:80 # 定义一个 webcluster 的应用

balance source # 负载均衡器调度使用的算法

server inst_1 192.168.200.112:8080 check port 80 rise 3 inter 2000 fall 3

server inst_1 192.168.200.113:8080 check port 80 rise 3 inter 2000 fall 3

```
[root@haproxy ~]# vim /etc/haproxy/haproxy.cfg
```

this config needs haproxy-1.1.28 or haproxy-1.2.1

global

log 127.0.0.1 local0

log 127.0.0.1 local1 notice

#log loghost local0 info

maxconn 4096

#chroot /usr/share/haproxy

```
uid 99
gid 99
daemon
#debug
#quiet
defaults
log global
mode http
option httplog
option dontlognull
retries 3
#redispatch
maxconn 2000
conntimeout 5000
clitimeout 50000
srvtimeout 50000
listen web-cluster 0.0.0.0:80
option httpchk GET /session.jsp
mode http
balance source
server inst1 192.168.200.112:8080 cookie app1 check inter 2000 rise 2 fall 5
server inst2 192.168.200.113:8080 cookie app1 check inter 2000 rise 2 fall 5
```

创建自启动脚本

```
[root@haproxy ~]# cp /usr/src/haproxy-1.4.24/examples/haproxy.init /etc/init.d/haproxy
[root@haproxy ~]# ln -s /usr/local/sbin/haproxy /usr/sbin/haproxy
[root@haproxy ~]# chmod +x /etc/init.d/haproxy
[root@haproxy ~]# /etc/init.d/haproxy start
Starting haproxy: [ 确定]
```

测试:

用浏览器打开 <http://192.168.200.111/session.jsp>

停止 tomcat2 服务器:

```
[root@node2 ~]# /usr/local/tomcat/bin/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

浏览器访问测试 <http://192.168.200.111/session.jsp>

haproxy 保持 Session 亲缘性的三种方式

用户 IP 识别

haproxy 将用户 IP 经过 hash 计算后指定到固定的真实服务器上（类似于 nginx 的 ip hash）

配置指令 `balance source`

```
listen web-cluster 0.0.0.0:80
```

```
option httpchk GET /session.jsp
```

```
mode http
```

```
balance source
```

```
server inst1 192.168.200.112:8080 cookie app1 check inter 2000 rise 2 fall 5
```

```
server inst2 192.168.200.113:8080 cookie app1 check inter 2000 rise 2 fall 5
```

Cookie 识别

企业应用系统多数使用 C/S（客户端/服务器）架构，

Session 的管理方式可将 Session 记录在客户端，

每次请求服务器的时候，将 Session ID 放在请求中发送给服务器，

服务器处理完请求后再将修改过的 Session 响应给客户端。

网站没有客户端，但是可以利用浏览器支持的 Cookie 记录 Session。

利用 Cookie 记录 Session 也有缺点，比如受 Cookie 大小限制，能记录的信息有限，

每次请求响应都需要传输 Cookie，影响性能，如果用户关闭 Cookie，访问就会不正常。

但是用于 Cookie 的简单易用，可用性高，支持应用服务器的线性伸缩，

而大部分应用需要记录的 Session 信息又比较小。

因此事实上，许多网站都或多或少地使用 Cookie 记录 Session。

haproxy 将 WEB 服务端发送给客户端的 cookie 中插入(或添加前缀)haproxy 定义的后端的服务器 COOKIE ID。

配置指令 `cookie SESSION_COOKIE insert indirect nocache`

用 Fiddler 工具抓包，可以看到用户的请求头的 cookie 里有类似”Cookie

`jsessionId=0bc588656ca05ecf7588c65f9be214f5; SESSION_COOKIE=app1”`

`SESSION_COOKIE=app1` 就是 haproxy 添加的内容

```
listen web-cluster 0.0.0.0:80
```

```
mode http
```

```
cookie SESSION_COOKIE insert indirect nocache
```

```
server inst1 192.168.200.112:8080 cookie app1 check inter 2000 rise 2 fall 5
```

```
server inst2 192.168.200.113:8080 cookie app1 check inter 2000 rise 2 fall 5
```

Session 识别

haproxy 将后端服务器产生的 session 和后端服务器标识存在 haproxy 中的一张表里。
客户端请求时先查询这张表。

```
配置指令  appsession JSESSIONID len 64 timeout 5h request-learn
listen web-cluster 0.0.0.0:80
mode http
appsession JSESSIONID len 64 timeout 5h request-learn
server inst1 192.168.200.112:8080 cookie appl check inter 2000 rise 2 fall 5
server inst2 192.168.200.113:8080 cookie appl ch
```

Tomcat 官方 Session 复制代码（方法二）

```
<!--
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
-->
```

把以下内容写入：

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
channelSendOptions="8">
<Manager className="org.apache.catalina.ha.session.DeltaManager"
expireSessionsOnShutdown="false"
notifyListenersOnReplication="true"/>
<Channel className="org.apache.catalina.tribes.group.GroupChannel">
<Membership
className="org.apache.catalina.tribes.membership.McastService"
address="228.25.25.4"
port="45564"
frequency="500"
dropTime="3000"/>
<Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
address="192.168.200.109"          #在Tomcat2 上替换为 192.168.200.110
port="4000"
autoBind="100"
selectorTimeout="5000"
maxThreads="6"/>
<Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
<Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
```

```

</Sender>
<Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
<Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatch15Interceptor"/>
</Channel>
<Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
filter=""/>
<Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>
<Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
tempDir="/tmp/war-temp/"
deployDir="/tmp/war-deploy/"
watchDir="/tmp/war-listen/"
watchEnabled="false"/>
<ClusterListener
className="org.apache.catalina.ha.session.JvmRouteSessionIDBinderListener"/>
<ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>

```

常见实验报错，实验一启动而实验二启动不了，或者都启动了，但是session复制做不出来；出现这种情况，一般都是网络模式的原因；**解决办法**如下：

Session 添加组播地址设置

```

[root@node1 ~]# route add -net 224.0.0.0 netmask 240.0.0.0 dev ens32
[root@node2 ~]# route add -net 224.0.0.0 netmask 240.0.0.0 dev ens32

```