**注意：本篇博文涉及的知识内容如下，实验中所用的系统环境为RHEL6.4。**

**1.Java基础知识讲解**

**2.**Tomcat的安装和实现****

****3.通过apache的mod\_proxy模块代理并实现Tomcat负载均衡****

****4.基于Tomcat集群的网上商城的实现****

------------------------------------------------------------------------------------------

**1.Java基础知识讲解**

1.1.Java相关的体系结构

1.Java程序设计语言

2.Java class 文件格式

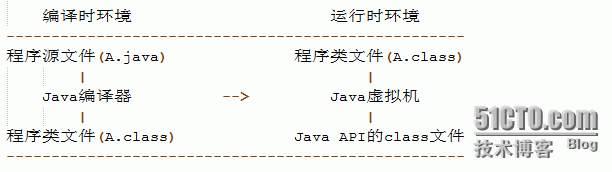
3.Java 应用编程接口

4.Java 虚拟机

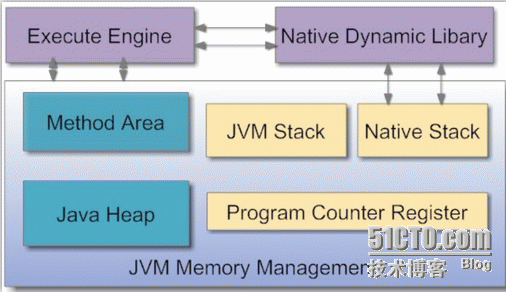
1.2.Java运行环境

用Java语言编译源代码，把它编译成Java Class文件，然后在Java VM中运行class文件；当编写程序时，通过调用类(Java API)中的方法来访问系统资源，而当程序运行时，它通过调用class文件中实现了Java API的方法也满足程序的Java API调用。Java VM和Java API一起组成了一个“平台”，所有Java程序都在其上编译和运行，因此，它们有时也被称作Java运行时环境。

相关实现模型如下：

[](http://img1.51cto.com/attachment/201310/094949885.png)

1.3.JVM（虚拟机）运行数据区域

[](http://img1.51cto.com/attachment/201310/095304567.png)

1.4.相关术语概念

1.JVM : Java虚拟机 主要包含类加载器和相关的执行引擎

2.JDK : Java开发工具 主要包括Java运行环境，和javac编辑器及相关的API

3.Java SE : 包含JDK和一些核心的JAVA API

4.Java EE : 由Java SE 和一些Java企业级的API组成

1.5.web容器（Tomcat）

当JSP应用程序第一次调用之后，JSP会被编译成一个servlet类，后续的操作直接使用此类，从而避免了每次调用的都要重新分析和编译。因此，类似servlet，JSP的执行需要在container中完成JSP的container跟servlet的container基本相同。但在JSP执行之前，需要一些额外的步骤如与servlet代码建立会话等。Tomcat包含了一个叫做Catalina的Servlet container（执行servlet和编译过的JSP）和一个JSP编译器(Jasper)。

事实上，一个包含了JSP编译器和Servlet容器的应用程序组合通过被称作Web容器。

**2.Tomcat的安装和实现**

2.1.JDK的安装

目前开源的JDK有ApacheHarmony, OpenJDK, SunJDK等。下载链接：

[http://www.oracle.com/technetwork/java/javase/downloads/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html" \t "/home/caojx/文档\\x/_blank)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | [root@TomcatA ~]# rpm -ivh jdk-7u9-linux-x64.rpm  [root@TomcatA ~]# ll /usr/java/  total 4  lrwxrwxrwx  1 root root   16 Oct 3 19:52 default -> /usr/java/latest  drwxr-xr-x 10 root root 4096 Oct 3 19:52 jdk1.7.0\_09  lrwxrwxrwx  1 root root   21 Oct 3 19:52 latest ->/usr/java/jdk1.7.0\_09  [root@TomcatA ~]# vim /etc/profile.d/java.sh  export JAVA\_HOME=/usr/java/latest  export PATH=$JAVA\_HOME/bin:$PATH  [root@TomcatA ~]# . /etc/profile.d/java.sh  [root@TomcatA ~]# java -version  java version "1.7.0\_09"  Java(TM) SE Runtime Environment (build 1.7.0\_09-b05)  Java HotSpot(TM) 64-Bit Server VM (build 23.5-b02, mixed mode)  [root@TomcatA ~]# |

2.2.Tomcat安装（JAVA程序） --> 解压部署

官方网站: [http://tomcat.apache.org/](http://tomcat.apache.org/" \t "/home/caojx/文档\\x/_blank)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | [root@TomcatA ~]# tar xf apache-tomcat-7.0.42.tar.gz -C /usr/local/  [root@TomcatA ~]# cd /usr/local/  [root@TomcatA ~]# ln -sv apache-tomcat-7.0.42 tomcat  `tomcat' -> `apache-tomcat-7.0.42  [root@TomcatA ~]# vim /etc/profile.d/tomcat.sh  export CATALINA\_HOME=/usr/local/tomcat  exportPATH=$CATALINA\_HOME/bin:$PATH  [root@TomcatA ~]# catalina.sh --help     #显示相关的命令选项  [root@TomcatA ~]# jps                    #显示当前系统运行的JVM程序 |

2.3.编辑启动服务脚本：

[root@TomcatA ~]# vim/etc/rc.d/init.d/tomcat

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **#!/bin/sh**  # Tomcat init script for Linux.  #  # chkconfig: 2345 96 14  # description: The Apache Tomcat servlet/JSP container.  JAVA\_OPTS='-Xms64m -Xmx256m'         #设置JAVA运行时的内容大小  JAVA\_HOME=/usr/java/latest  CATALINA\_HOME=/usr/local/tomcat  export JAVA\_HOME CATALINA\_HOME  exec $CATALINA\_HOME/bin/catalina.sh $\* |

2.4.配置文件介绍：

Tomcat的配置文件默认存放在$CATALINA\_HOME/conf目录中，主要有以下几个：

1.server.xml: Tomcat的主配置文件，Service, Connector, Engine, Realm, Valve,Hosts主组件的相关配置信息；

2.web.xml：遵循Servlet规范标准，用于配置servlet，并为所有的Web应用程序提供包括MIME映射等默认配置信息；

3.tomcat-user.xml：Realm认证时用到的相关角色、用户和密码等信息；Tomcat自带的manager默认情况下会用到此文件；在Tomcat中添加/删除用户，为用户指定角色等将通过编辑此文件实现；

4.catalina.policy：Java相关的安全策略配置文件，在系统资源级别上提供访问控制的能力；

5.catalina.properties：Tomcat内部package的定义及访问相关的控制，也包括对通过类装载器装载的内容的控制；Tomcat6在启动时会事先读取此文件的相关设置；

6.ogging.properties:Tomcat6通过自己内部实现的JAVA日志记录器来记录操作相关的日志，此文件即为日志记录器相关的配置信息，可以用来定义日志记录的组件级别以及日志文件的存在位置等；

7.context.xml：所有host的默认配置信息；

2.5.主配置文件 /usr/local/tomcat/conf/server.xml中常用的组件介绍：

Server组件：使tomcat服务器启动一个实例

Service组件：关联一个引擎和此引擎相关的连接器

Connector组件：为引擎设置相关的连接器以接受客户端的访问请求

Engine组件：是servlet处理器的一个实例，即servlet引擎

Host组件：位于engine容器中用于接收请求并进行相应处理的主机或虚拟主机

Context组件：在某些意义上类似于apache中的路径别名，一个Context定义用于标识tomcat实例中的一个Web应用程序

其余组建和详细参数信息请参考官方文档，在这里我就不再叙述。

2.6.应用程序目录的结构：

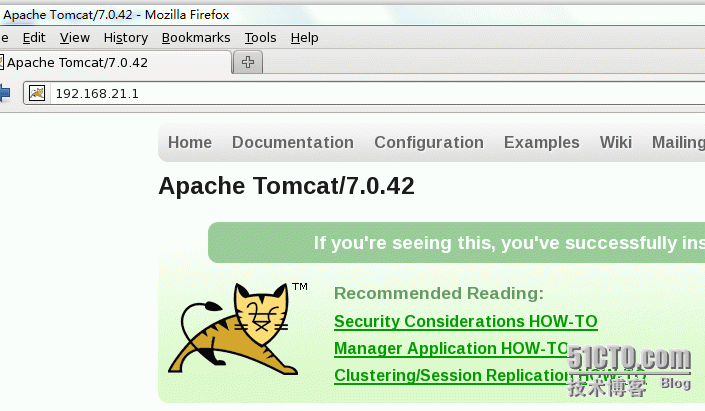
/WEB-INF/web.xml：包含当前webapp的deploy描述符，如所有的servlets和JSP等动态文件的详细信息，会话超时时间和数据源等；因此，其也通常用于定义当前webapp特有的资源；

/WEB-INF/classes: 包含所有服务器端类及当前应用程序相关的其它第三方类等；

/WEB-INF/lib: 包含JSP所用到的JAR文件；

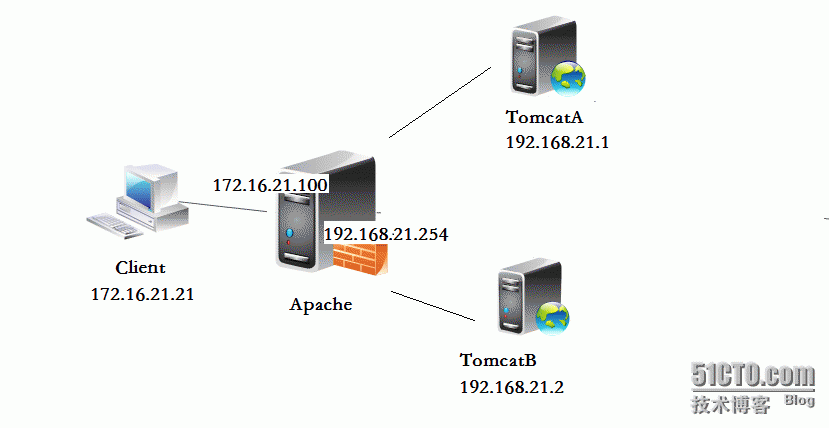
2.7.简单测试：

我们将上述connector的端口由默认的8080修改为80，重启tomcat服务器并进行简单的访问测试

[](http://img1.51cto.com/attachment/201310/102057629.png)

**3.通过apache的mod\_proxy模块代理并实现Tomcat负载均衡**

3.1.实验拓扑环境

[](http://img1.51cto.com/attachment/201310/102239657.png)

3.2.简单应用程序的部署

在这里我们使用默认的/usr/local/tomcat/webapps/目录直接部署虚拟目录实现，而不再使用单独的虚拟主机定义。配置简单实现如下：

**TomcatA：**

|  |  |
| --- | --- |
| 1  2  3 | [root@TomcatA ~]# cd /usr/local/tomcat/webapps/  [root@TomcatA webapps]# mkdir -pv test/WEB-INF/{classes,lib}  [root@TomcatA webapps]# vim test/index.jsp |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <%@ page language="java" %>  <html>   <head><title>TomcatA</title></head>    <body>  <h1><font color="red">TomcatA</font></h1>  <table align="centre" border="1">    <tr>  <td>Session ID</td>  <% session.setAttribute("abc","abc"); %>  <td><%= session.getId() %></td>    </tr>    <tr>  <td>Created on</td>  <td><%= session.getCreationTime() %></td>   </tr>  </table>    </body>  </html> |

**TomcatB：**

主要配置同TomcatA相同，只需要设置不同的主页文件：

|  |  |
| --- | --- |
| 1 | [root@TomcatA webapps]# vim test/index.jsp |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <%@ page language="java" %>  <html>   <head><title>TomcatB</title></head>    <body>  <h1><font color="blue">TomcatB</font></h1>  <table align="centre" border="1">    <tr>  <td>Session ID</td>  <% session.setAttribute("abc","abc"); %>  <td><%= session.getId() %></td>    </tr>    <tr>  <td>Created on</td>  <td><%= session.getCreationTime() %></td>   </tr>  </table>    </body>  </html> |

#配置完成之后，利用之前提供的脚本重启两台tomcat服务器。

**Apache：**利用mod\_proxy模块实现负载均衡。

1.启动RHEL6.4上默认的httpd服务。

2.新添加配置文件如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | [root@apache ~]# vim /etc/httpd/conf.d/mod\_proxy.conf  ProxyVia on  ProxyRequests Off                        #关闭正向代理  ProxyPreserveHost Off  <Proxy balancer://xkun>                  #定义代理的后端real server          BalancerMemberajp://192.168.21.1:8009 loadfactor=1          BalancerMemberajp://192.168.21.2:8009 loadfactor=1          ProxySetlbmethod=byrequests  </Proxy>  <Location /xkun>                         #状态检测的实现          SetHandler balancer-manager      #调用处理模块          Proxypass !                      #不向后方代理          Order allow,deny                 #定义访问权限          Allow from all                   #建议在实际生产中做好访问控制  </Location>  <Proxy \*>                                #相关反向代理的权限控制            Order allow,deny            Allow from all  </Proxy>  ProxyPass  /  balancer://xkun/           #设置代理路径的映射管理  ProxyPassReverse  /  balancer://xkun/  <Location  / >                           #定义相关的访问控制策略            Order allow,deny            Allow from all  </Location> |

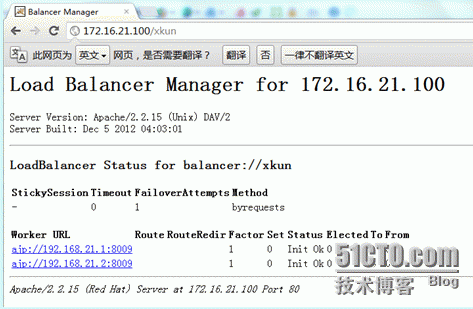
3.重启所有服务器进行访问测试

#注意：重启之前，最好编辑/etc/sysctl.conf文件打开ip\_forward转发

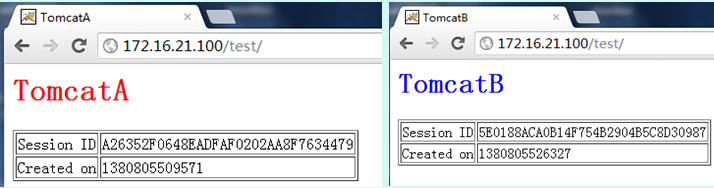
# service httpd start

3.3.简单测试如下：

1.查看状态信息

[](http://img1.51cto.com/attachment/201310/103953249.png)

2.查看部署好的jsp程序

[](http://img1.51cto.com/attachment/201310/104235125.png)

3.4.实验总结：

1.通过上述实验我们实现了如何利用apache的mod\_proxy模块实现了tomcat的负载均衡。

2.虽然本次实现了tomcat负载均衡但我们发现session无法得到保持。我将在一下叙述中进行实现

3.5.基于内存复制实现tomcat集群中的session共享

1.分别在TomcatA和TomcatB服务器的主配置文件上添加如下信息（只需修改各自的receiver）：

# vim server.xml #添加在引擎容器中以实现所有虚拟主机的共享

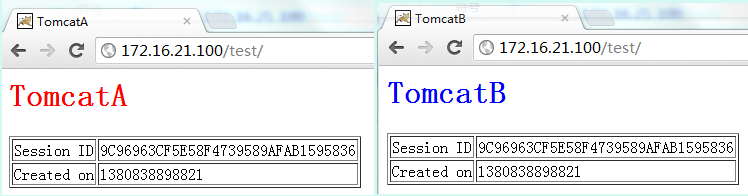
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | <**Cluster** className="org.apache.catalina.ha.tcp.SimpleTcpCluster"                   channelSendOptions="8">            <**Manager** className="org.apache.catalina.ha.session.DeltaManager"                     expireSessionsOnShutdown="false"                     notifyListenersOnReplication="true"/>            <**Channel** className="org.apache.catalina.tribes.group.GroupChannel">              <**Membership** className="org.apache.catalina.tribes.membership.McastService"                          address="228.0.0.4"                          port="45564"                          frequency="500"                          dropTime="3000"/>              <**Receiver** className="org.apache.catalina.tribes.transport.nio.NioReceiver"                        address="auto" <!--注意修改地址为自己的网卡IP -->                        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"/>  </**Cluster**> |

2.完成之后，还需要在应用程序部署目录中添加下配置文件并修改，实现两台tomcat的内存共享：

|  |  |
| --- | --- |
| 1  2  3  4 | [root@TomcatA ~]# cd /usr/local/tomcat/conf/  [root@TomcatA conf]# cp -a web.xml ../webapps/test/WEB-INF/  # 修改web.xml文件在 web-app 容器中添加如下行：  <distributable /> |

3.同样在TomcatB上进行同样的配置，添加相同的配置文件。只是需要将地址修改为自己网卡地址，在上述代码中已给有提示。关于server.xml中添加集群相关的配置文件详解请参考官方文档。

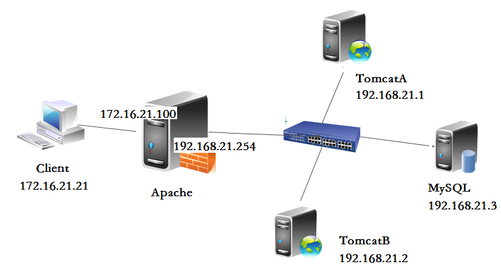
4.同启重启后端的两台tomcat服务器，并进行访问测试结果如下：

[](http://img1.51cto.com/attachment/201310/110236858.png)

我们发现无论如何刷新，session id的信息都不会再改变。至此，我们完成了如何利用apache的mod\_proxy模块实现tomcat服务器的负载均衡，同时我们也完成了tomcat内存共享集群的实现。下面我将通过一个网上购物商城实例来给大家做进一步的讲解和总结。

**4.基于Tomcat集群的网上商城的实现**

4.1.实验拓扑环境：

[](http://img1.51cto.com/attachment/201310/110733978.png)

4.2.shop应用程序的部署

注意：同样在这里我们使用默认的/usr/local/tomcat/webapps/目录直接部署虚拟目录实现。

下载开源网上购物商城程序包如下：shopxx-a5-Beta.zip

下载路径：

简单配置如下：

1.TomcatA：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [root@TomcatA ~]# unzip shopxx-a5-Beta.zip  [root@TomcatA ~]# cd shopxx-v3.0-Beta/  [root@TomcatA shopxx-v3.0-Beta]# ls shopxx-3.0Beta/  admin          favicon.ico  install      META-INF   robots.txt  upload  changelog.txt  index.jsp    license.html  resources shopxx.txt  WEB-INF  [root@TomcatA shopxx-v3.0-Beta]# mv shopxx-3.0Beta /usr/local/tomcat/webapps/shop |

2.MySQL： (安装方法我就不再叙述，可以参考之前的博客)

|  |  |
| --- | --- |
| 1  2  3 | mysql> create database shopxx;  mysql> create user 'showuser'@'%' identified by'showpass';  mysql> grant all on shopxx.\* to 'showuser'@'%'; |

3.Apache： (代理配置文件和上一个实现相同，这里就不再给出)

先注释BalancerMemberajp://192.168.21.2:8009 loadfactor=1

4.然后访问：[http://172.16.21.100/shop](http://172.16.21.100/shop" \t "/home/caojx/文档\\x/_blank) 根据提示进行安装：直至出现如下界面：

[](http://img1.51cto.com/attachment/201310/113539507.png)

5.TomcatB： 将TomcatA的关于shop/目录的配置复制到TomcatB中。

|  |  |
| --- | --- |
| 1 | # scp -rp shop 192.168.21.2:/usr/local/tomcat/webapps/root@192.168.21.2 |

问题：在购物商城的实现过程中，我也遇见了session无法保持的情况。即同一用户再次登录时购物车的商品有可能为空。

解决方案：利用tomcat集群实现内存共享，这在上面的案例中也有介绍。

同样修改配置文件server.xml添加上述内容（在上面我已经做过详细的说明）。

同时修改配置文件#vim shop/WEB-INF/web.xml添加如下一行

<distributable />

同时启动tomcat服务器。

当端口启动时，再次访问站点主页进行测试[http://172.16.21.100/shop](http://172.16.21.100/shop" \t "/home/caojx/文档\\x/_blank)

[](http://img1.51cto.com/attachment/201310/114139322.png)

注册用户并将商品加入购物车。多次执行退出登录操作。我们发现购物车的商品一直存在：

[](http://img1.51cto.com/attachment/201310/114421164.png)

无论我如何刷新，登录用户的购物车信息都会存在。实现了后端集群的信息共享。至此基于tomcat的集群的相关实现我已经写完了。如果大家有什么疑问欢迎和我交流。

QQ:572807025 邮箱：xk1129@163.com

注意：本次实验所用的系统环境为RHEL6.4,整个实验过程全部由笔者亲自实践。笔者将会在后续的博客中陆续更新：如何利用HAProxy代替apache实现反向代理并实现动静分离，并利用keepalived实现HAProxy的高可用。同时也会将HAProxy替换成Varnish实现相应的反向代理和缓存。通过对多种web架构的实现并进行相应的压力测试，比较各种架构的性能好坏。

本文出自 “[Haris](http://harisxiong.blog.51cto.com/)” 博客，请务必保留此出处<http://harisxiong.blog.51cto.com/7513022/1304746>