[反向代理介绍](http://blog.csdn.net/shaozengwei/article/details/40861227)

Apache可以被配置为**正向(forward)** 和**反向(reverse) 代理**。  
**正向代理**是一个位于客户端和原始服务器(originserver)之间的服务器,为了从原始服务器取得内容,客户端向代理发送一个请求并指定目标(原始服务器),然后代理向原始服务器转交请求并将获得的内容返回给客户端。客户端必须要进行一些特别的设置才能使用正向代理。  
正向代理的典型用途是为在防火墙内的局域网客户端提供访问Internet的途径。正向代理还可以使用缓冲特性(由**mod\_cache** 提供)减少网络使用率。  
使用ProxyRequests 指令即可激活正向代理 。  
因为正向代理允许客户端通过它访问任意网站并且隐藏客户端自身，因此你必须采取安全措施 以确保仅为经过授权的客户端提供服务。  
  
  
**反向代理**正好相反,对于客户端而言它就像是原始服务器,并且客户端不需要进行任何特别的设置.客户端向反向代理的命名空间(name-space)中的内容发送普通请求,接着反向代理将判断向何处(原始服务器)转交请求,并将获得的内容返回给客户端,就像这些内容原本就是它自己的一样。  
反向代理的典型用途是**将防火墙后面的服务器提供给Internet用户访问**.反向代理还可以为后端的多台服务器提供**负载平衡**,或为后端较慢的服务器提供缓冲服务。 另外，还可以启用高级URL策略和管理技术,从而使处于不同web服务器系统的web页面同时存在于同一个URL空间下。  
可以使用**ProxyPass**指令激活反向代理(在RewriteRule 指令中使用[P] 标记也可以)。配置反向代理并不需要 打开ProxyRequests 指令(**但有可能需要加载**)。

## Apache + Tomcat的session复制

步骤：

1.安装Apache基本模块

2.后台监控

3.负载均衡简单测试

4.配置Tomcat相关模块（AJP）

5.保持Session唯一，粘性会话

6.Tomcat集群，Session复制

1.安装Apache相关模块

负载均衡需要的主要是代理模块！

经过几次Apache配置尝试，在Ubuntu下配置Apache实在是太容易了。加载什么模块、取消什么模块两个命令搞定。

Shell代码

1.#启用模块

2.sudo a2enmod <model>

3.#禁用模块

4.sudo a2dismod <model>

这里，我们需要让Apache提供代理服务，其中又包含基于http、ftp、ajp等等协议的代理功能，同时还需要负载均衡模块。我们可以通过命令逐个加载：

Shell代码

1.#代理核心模块

2.sudo a2enmod proxy

3.#代理AJP模块

4.sudo a2enmod proxy\_ajp

5.#代理负载均衡模块

6.sudo a2enmod proxy\_balancer

7.#代理HTTP模块

8.sudo a2enmod proxy\_http

9.#代理FTP模块

10.sudo a2enmod proxy\_ftp

#LB request method

11.sudo a2enmod lbmethod\_byrequests

suod a2enmod lbmethod\_bytraffic

sudo a2enmod lbmethod\_bybusyness

sudo a2enmod lbmethod\_heartbeat

Apache module library path : /usr/lib/apache2/modules

完成上述操作后，系统会提示重启Apache！

先不着急重启，现学现卖，了解下Apache的目录结构。在Ubuntu下配置Apache主要是在/etc/apache2目录下：

分述：

apache2.conf核心配置文件，一般不需要修改！

conf.d目录，里面包含了一些字符集设置，文档等设置！

dav\_svn.authz和dav\_svn.passwd是前面做SVN时，相关权限、密码文件。

envvars定义了运行时的用户身份——www-data。

httpd.conf是Apache留给我们自己折腾的配置文件，默认为空。apache2.conf会加载这个文件。

ports.conf端口默认配置。apache2.conf会加载这个文件。

magic为mod\_mime\_magic模块服务。

mods-enabled和mods-available mods-enabled会被apache2.conf加载，里面包含\*.load和\*.conf文件。\*.load文件中是加载相应的模块（位于/usr/lib/apache2/modules/中），而\*.conf中是对应的基本配置。但这些文件其实都是链接到mods-available中相应的文件上。当我们通过a2enmod操作时，实际上正是操作了这些软链接。

sites-available和sites-enabled 与 mods-enabled和mods-available的关系类似，只是其中包含的是站点内容。

罗嗦了一堆，下面配置负载均衡部分。

执行修改：

Shell代码

sudo vi /etc/apache2/mods-available/proxy.conf

<IfModule mod\_proxy.c>

# If you want to use apache2 as a forward proxy, uncomment the

# 'ProxyRequests On' line and the <Proxy \*> block below.

# WARNING: Be careful to restrict access inside the <Proxy \*> block.

# Open proxy servers are dangerous both to your network and to the

# Internet at large.

#

# If you only want to use apache2 as a reverse proxy/gateway in

# front of some web application server, you DON'T need

# 'ProxyRequests On'.

#ProxyRequests On

ProxyRequests Off

# 我们只使用反向代理功能, ProxyRequests并不需要, 所以设置为Off

#<Proxy \*>

# AddDefaultCharset off

# Require all denied

# #Require local

#</Proxy>

# Enable/disable the handling of HTTP/1.1 "Via:" headers.

# ("Full" adds the server version; "Block" removes all outgoing Via: headers)

# Set to one of: Off | On | Full | Block

#ProxyVia Off

／\* \*

BalancerMember为群集的成员，即群集服务器Tomcat1, 2, ....，负载均衡服务器会根据均衡规则(loadfactor)来将请求转发给BalancerMember。Apache使用ajp协议与tomcat子服务器进行交互, 协议后的路径为每个tomcat的运行地址, 端口为tomcat的ajp服务运行端口, 端口在server.xml的connector标签中设置.route的值对应tomcat的server.xml中engine标签的jvmRoute属性.

\*\*/

<Proxy balancer://hello>

BalancerMember "ajp://192.168.1.104:8011/hello" route=node1 loadfactor=1

BalancerMember "ajp://192.168.1.104:8012/hello" route=node2 loadfactor=1

BalancerMember "ajp://192.168.1.105:8009/hello" route=tomcat1 loadfactor=1

</Proxy>

#ProxyPass为代理转发的Url,即将所有访问/\*的请求转发到群集balancer://hello

ProxyPass /hello balancer://hello

ProxyPassReverse /hello balancer://hello

#<Proxy balancer://hello>

#BalancerMember http://192.168.1.105:8080/hello

#BalancerMember http://192.168.1.104:8082/hello

#BalancerMember http://192.168.1.104:8081/hello

#</Proxy>

#ProxyPass /hello balancer://hello

#ProxyPassReverse /hello balancer://hello

</IfModule>

2.后台监控

我们如何知道某个节点负载多少，响应时间多久，服务是否正常呢？Apache提供了负载均衡监控平台：http://localhost/balancer-manager。但是，这个服务默认是不存在。

除了用于负载均衡配置、监控的balancer-manager还有http://localhost/server-status和http://localhost/server-info

我们需要添加proxy\_balancer模块：

Shell代码

同时，为了能够使用balancer-manager，我们需要配置：

sudo a2enmod proxy\_balancer

sudo vi /etc/apache2/mods-enable/proxy\_balancer.conf

add content like that:

<IfModule mod\_proxy\_balancer.c>

# Balancer manager enables dynamic update of balancer members

# (needs mod\_status). Uncomment to enable.

#

<IfModule mod\_status.c>

<Location /balancer-manager>

SetHandler balancer-manager

Require local

</Location>

</IfModule>

</IfModule>

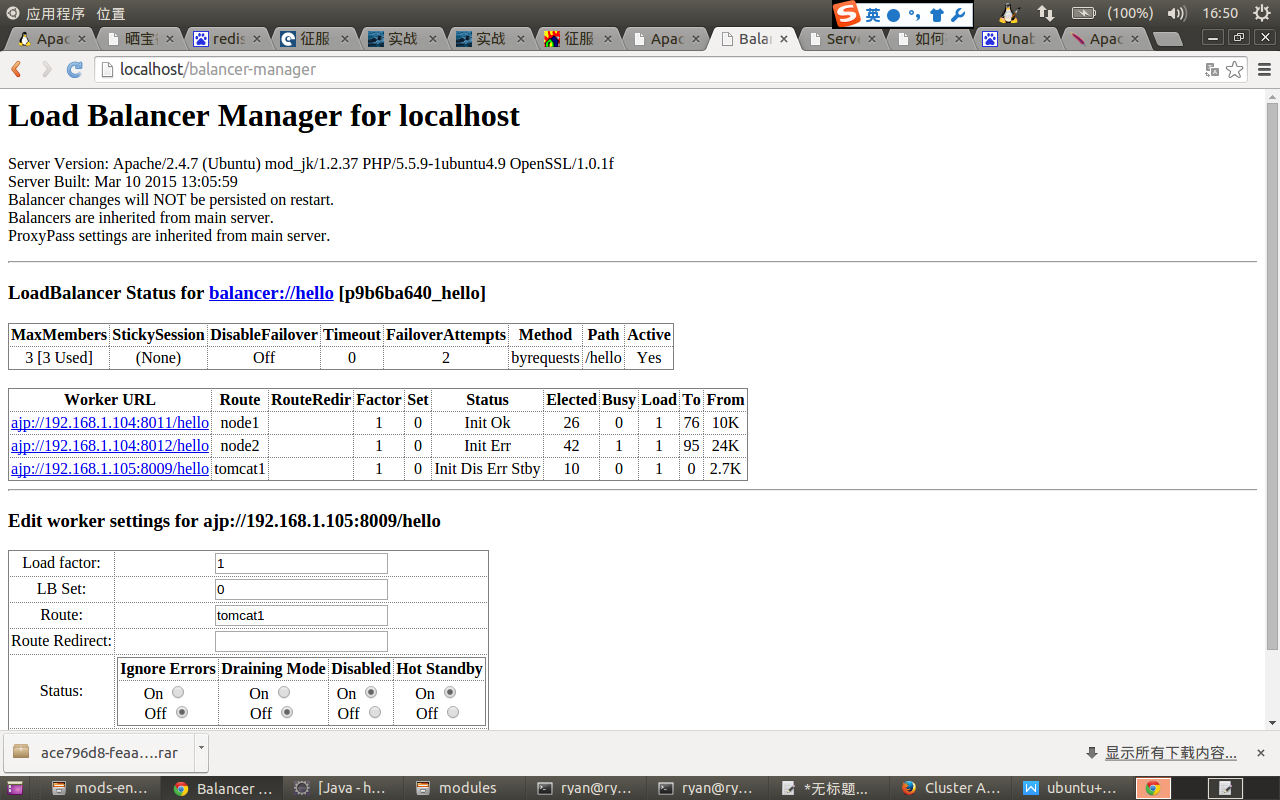
注意，这段代码放到了<IfModule mod\_info.c>和</IfModule>之间！

现在，我们重启Apache：

Shell代码 复制代码

sudo /etc/init.d/apache2 restart

来看看管理界面http://localhost/balancer-manager：



我们再来看看服务器基本信息 http://localhost/server-info,

服务器status :http://localhost/server-status

至此，我们完成了基本负载均衡的基本配置

/etc/apache2/mods-available/proxy.conf还有一些属性:

noFailOver是否打开失败转移，On|Off，默认为Off，添加在ProxyPass后面，如：

Conf代码 复制代码

ProxyPass /zlex balancer://zlex stickySession=JSESSIONID noFailOver=On

如果这样配置，当提供给你服务的服务器发生异常，那么你将一直看着它返回给你503，直到系统恢复正常！

loadfactor表示后台服务器负载到由Apache发送请求的权值,默认值为1添加在BalancerMember后面：

Conf代码 复制代码

<Proxy balancer://zlex>

BalancerMember ajp://localhost:8009/

BalancerMember ajp://192.168.49.1:8009/

</Proxy>

可以实现三种策略：

轮询均衡策略的配置

按权重分配均衡策略的配置

权重请求响应负载均衡策略的配置

5.Session唯一，粘性会话

Apache已经可以轻松将内容处理的工作分配给各个Tomcat了！

当然，这还不够，Session还是个问题！

WHY？

我们来做一系列修改，来检测Session到底出现了什么问题！

先来改造Tomcat，修改server.xml：

Shell代码 复制代码

sudo vi /etc/tomcat6/server.xml

修改<Engine />节点，增加jvmRoute属性：

Xml代码 复制代码

<Engine

name="Catalina"

defaultHost="localhost"

jvmRoute="tomcat1">

另一个Tomcat设置改为

Xml代码 复制代码

<Engine

name="Catalina"

defaultHost="localhost"

jvmRoute="tomcat2">

通过jvmRoute，指定了Tomcat唯一标识！

然后修改/etc/apache2/mods-available/proxy.conf

Shell代码 复制代码

sudo vi /etc/apache2/mods-available/proxy.conf

如下：

Java代码 复制代码

<Proxy balancer://zlex>

BalancerMember ajp://localhost:8009/zlex route=tomcat1

BalancerMember ajp://192.168.49.1:8009/zlex route=tomcat2

</Proxy>

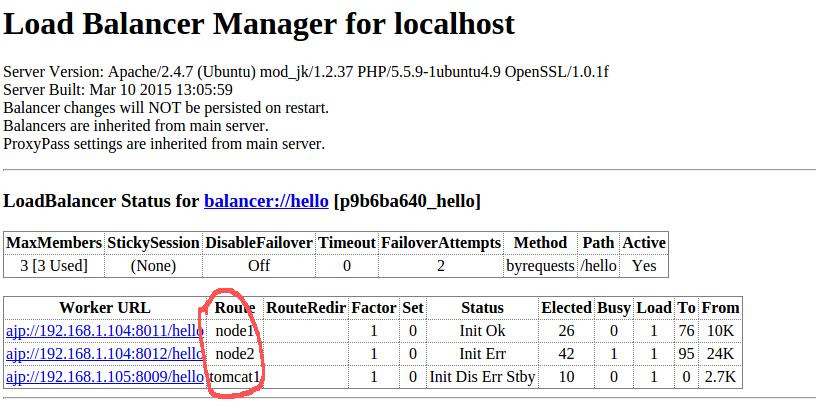
ProxyPass /zlex balancer://zlex

ProxyPassReverse /zlex balancer://zlex

这里需要通过修改route属性，将Apache与Tomcat关联起来！

注意，Tomcat中定义的jvmRoute需要与Apache定义的route相对应！

我们来看一下http://localhost/balancer-manager发生了什么变化：



我们注意到route字段有了新的标识，当然，我们也可以通过这个配置界面修改这些信息，但当前修改不会真的修改/etc/apache2/mods-available/proxy.conf文件，Apache重启后将丢失。

为了更细致的对比进过复杂均衡的结果，这里增加了应用！主要是监控Session的变化！

只看核心代码：

<%@ page contentType="text/html; charset=GBK" %>

<%@ page import="java.util.\*" %>

<html><head><title>Cluster App Test</title></head>

<body>

<b>Cookie信息：</b>

<br />

${header["cookie"]}

<br />

<b>Host信息：</b>

<br />

${header["host"]}

<br />

Server Info:

<%

out.println(request.getLocalAddr() + " : " + request.getLocalPort()+"<br>");%>

<%

out.println("<br> ID " + session.getId()+"<br>");

// 如果有新的 Session 属性设置

String dataName = request.getParameter("dataName");

if (dataName != null && dataName.length() > 0) {

String dataValue = request.getParameter("dataValue");

session.setAttribute(dataName, dataValue);

}

out.println("<b>Session 列表</b><br>");

System.out.println("============================");

Enumeration e = session.getAttributeNames();

while (e.hasMoreElements()) {

String name = (String)e.nextElement();

String value = session.getAttribute(name).toString();

out.println( name + " = " + value+"<br>");

System.out.println( name + " = " + value);

}

%>

<form action="session.jsp" method="POST">

名称:<input type=text size=20 name="dataName">

<br>

值:<input type=text size=20 name="dataValue">

<br>

<input type=submit>

</form>

</body>

</html>

分别部署到两个Tomcat上！

然后重启Apache：不断刷新http://localhost/hello，

仔细观察，每次请求都按照负载均衡配置的节点次序依次请求到不同的Tomcat上。尤其是当我们通过jvmRoute和route做了绑定之后，信息更加准确。但是，仔细观察，每次请求的SessionID都是不一样！对于纯Web应用，尤其是依靠SessionID区分唯一用户的应用，这将是一场噩梦——解决了服务器压力均衡问题，却带来了SessionID不唯一问题！这就需要SessionID绑定，或者说叫做“会话复制”。

如果这时候你在页面上提交表单，将键值对保持在session中，在页面刷新后，将无法获得该信息，因为Seesion丢失了！

接着修改/etc/apache2/mods-available/proxy.conf，让SeesionID保持唯一：

Shell代码 复制代码

sudo vi /etc/apache2/mods-available/proxy.conf

增加stickySession属性：

Conf代码 复制代码

ProxyPass /zlex balancer://hello stickySession=JSESSIONID

stickySession粘性会话，根据这一属性，浏览器将通过cookie绑定SeesionID。如果这个时候再次访问http://localhost/zlex，你会发现，页面不会来回跳转了！

sticky是什么？

引用

sticky模式

利用负载均衡器的sticky模式的方式把所有同一session的请求都发送到相同的Tomcat节点。这样不同用户的请求就被平均分配到集群中各个tomcat节点上，实现负载均衡的能力。这样做的缺点是没有灾难恢复的能力。一旦一个节点发生故障，这个节点上所有的session信息全部丢失；

同一用户同一session只和一个webServer交互，一旦这个webserver发生故障，本次session将丢失，用户不能继续使用 !

sticky模式的根本在于浏览器支持cookie，如果浏览器不支持cookie，则需要修改server.xml文件中的<Context />节点，将cookie置为false，关闭cookie功能，让jsessionid显式传递！

6.Tomcat集群，Session复制

集，是包含多个服务器实例的指定集合，这些服务器实例共享相同的应用程序、资源以及配置信息。您可以将不同计算机上的服务器实例分组到一个逻辑群集中并将其作为一个单元来管理。您可以使用 DAS 轻松控制多机群集的生命周期。

群集可以实现水平可伸缩性、负载平衡和故障转移保护。根据定义，群集中的所有实例都具有相同的资源和应用程序配置。当群集中的服务器实例或计算机出现故障时，负载平衡器检测到该故障，会将通信从出现故障的实例重定向至群集中的其他实例，并恢复用户会话状态。由于群集中所有实例上的应用程序和资源都相同，因此一个实例可以故障转移至群集中的任何其他实例。

引用

Session复制，主要是指集群环境下，多台应用服务器之间同步Session，确保Session保持一致，且Session中的内容保持一致，对外透明——看起来就像是一台应用服务器！

如果其中一台服务器发生故障，根据负载均衡的原理，Apache会遍历寻找可用节点，分发请求。与此同时，当前用户Session不能发生数据丢失，其余各节点服务器应保证用户Session数据同步。

Session复制核心内容主要是：

Session内容序列化（serialize），会消耗系统性能。

Session内容通过广播同步给成员，会造成网络流量瓶颈，即便是内网瓶颈。

因此，Session复制的这两个潜在问题，致使复杂均衡节点最多不会超过4个。因为，当节点数大于4时，整个集群的吞吐量将不再上升！

为了搭建Tomcat集群，我将两个Tomcat分别部署到两台虚拟机上，确保网段一致。

对于windows系统，不需要考虑网络问题，广播地址（这里用到224.0.0.0和240.0.0.0）默认开放，对于linux则需要通过命令开放地址。

Ubuntu上开放广播地址（eth0网卡）：

Shell代码 复制代码

sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0

然后通过-v参数查看当前开放的广播地址：

Shell代码 复制代码

route -v

然后，修改tomcat的server.xml文件：

<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat1">

<!--For clustering, please take a look at documentation at:

/docs/cluster-howto.html (simple how to)

/docs/config/cluster.html (reference documentation) -->

<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"

channelSendOptions="6">

<Manager className="org.apache.catalina.ha.session.BackupManager"

expireSessionsOnShutdown="false"

notifyListenersOnReplication="true"

mapSendOptions="6"/>

<!--

<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="224.0.0.4"

port="45564"

frequency="500"

dropTime="3000"/>

<Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"

address="192.168.1.105"

port="5000"

selectorTimeout="100"

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"/>

<Interceptor className="org.apache.catalina.tribes.group.interceptors.ThroughputInterceptor"/>

</Channel>

<Valve className="org.apache.catalina.ha.tcp.ReplicationValve"

filter=".\*\.gif|.\*\.js|.\*\.jpeg|.\*\.jpg|.\*\.png|.\*\.htm|.\*\.html|.\*\.css|.\*\.txt"/>

<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.ClusterSessionListener"/>

</Cluster>

这里需要注意<Membership />和Receiver：<Membership />节点的address属性是广播地址；Receiver节点的address属性是本地绑定地址。当然，默认为auto。由于我在启动Tomcat时，Tomcat频频将地址指向127.0.0.1，无奈只好使用固定IP。

Membership 里面是广播地址，加入ｃｌｕｓｔｅｒ　里的所有ｗｏｒｋｅｒ　保持一致

Receiver节点的address属性是本地绑定地址，如果同ＩＰ地址部署２个ｔｏｍｃａｔ，则用端口区别不同，端口范围从４０００－５０００，你可以取为ｎｏｄｅ１：４００１，　ｎｏｄｅ２：４００２

此外，为了降低Session复制的成本，Tomcat通过<Valve />节点，以过滤器的方式控制哪些请求可以忽略Session复制：

<Valve className="org.apache.catalina.ha.tcp.ReplicationValve"

filter=".\*\.gif;.\*\.js;.\*\.jpg;.\*\.png;.\*\.htm;.\*\.html;.\*\.css;.\*\.txt;"/>

另外，要修改应用中的web.xml文件，将<distributable />节点部署到<web-app />节点中，开启分布式服务：

如果仔细观察，当前SessionID在不断交替变化，这说明负载均衡在起作用！

我们再来看看2个Tomcat后台日志都做了什么！

两只猫都打印了相同的内容（a=1）不同的细节在于，sessionID带有服务器标识！

如果我们强行关闭Tomcat2：

首先，Tomcat1会很快侦测到Tomcat2离线，因为这是TCP通讯，成员之间很容易检测到其他成员是否离线！Tomcat1后台日志如下：

征服 <wbr>Apache <wbr>+ <wbr>Tomcat的session复制

其次，Apache会侦测到Tomcat2发生异常，将其余请求转交给其他节点，即交由Tomcat1处理！

继续刷新http://localhost/zlex当前页面，耐心等待几秒。你会发现，即便再次刷新页面，sessionID仍旧绑定在标识tomcat1服务器上。

然后，我们恢复Tomcat2服务，Tomcat1会马上侦测到Tomcat2已经恢复正常：

最后，我们再次刷新当前页，Apache已经将请求分发给Tomcat2了，从后台日志可以看到session信息会很快被同步了！

如果带有tomcatX标识的sessionID有很多不便之处，可以关闭粘性会话。简单的讲，就是取消Tomcat中[b]server.xml中<Engine/ >节点的jvmRoute属性！[/b]然后，重启tomcat、apache！

征服 <wbr>Apache <wbr>+ <wbr>Tomcat的session复制

页面提交一个b=3！

征服 <wbr>Apache <wbr>+ <wbr>Tomcat的session复制

左边为Tomcat1，右边为Tomcat2！SessionID一致！

除了上述几种方案外，还有Terracotta模式。一种第三方集群组件，2009年收购了缓存组件EhCache，可以结合Tomcat、JBoss等多种服务器，提供多种负载均衡、集群等功能实现，且当负载均衡节点超过8个时，仍然能够保持集群吞吐量的线性增长。

Eclipse插件地址：

http://download.terracotta.org/eclipse/update

下载地址：

http://www.terracotta.org/dl/oss-download-catalog