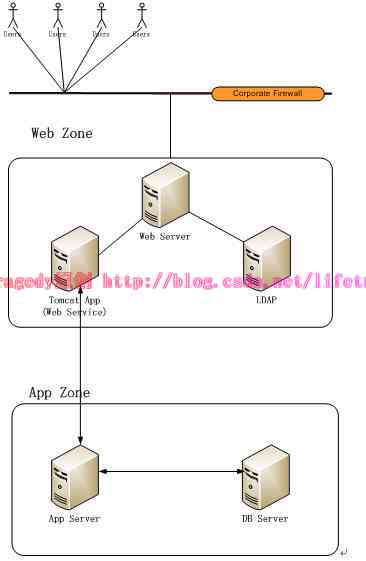
**一、为何要集群**

单台App Server再强劲，也有其瓶劲，先来看一下下面这个真实的场景。



当时这个工程是这样的，tomcat这一段被称为web zone，里面用spring+ws，还装了一个jboss的规则引擎Guvnor5.x，全部是ws没有service layer也没有dao layer。

然后App Zone这边是weblogic，传输用的是spring rmi，然后App Zone这块全部是service layer, dao layer和数据库打交道。

用户这边用的是.net，以ws和web zone连的。

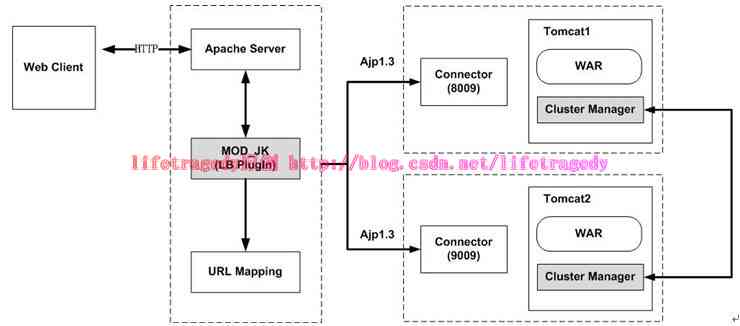
时间一长，数据一多，就出问题了。

拿Loader Runner跑下来，发觉是Web Zone这块，App Server已经被用到极限了。因为客户钱不多，所以当时的Web Zone是2台服务器，且都是32位的，内存不少，有8GB，测试下来后发觉cpu loader又不高，但是web server这边的吞吐量始终上不去，且和.net客户端那边响应越来越慢。

分析了一下原因：单台tomcat能够承受的最大负载已经到头了，单台tomcat的吞吐量就这么点，还要负担Guvnor的运行，Guvnor内有数百条业务规则要执行。

再看了一下其它方面的代码、SQL调优都已经到了极限了，所以最后没办法，客户又不肯拿钱投在内存和新机器上或者是再买台Weblogic，只能取舍一下，搞Tomcat集群了。

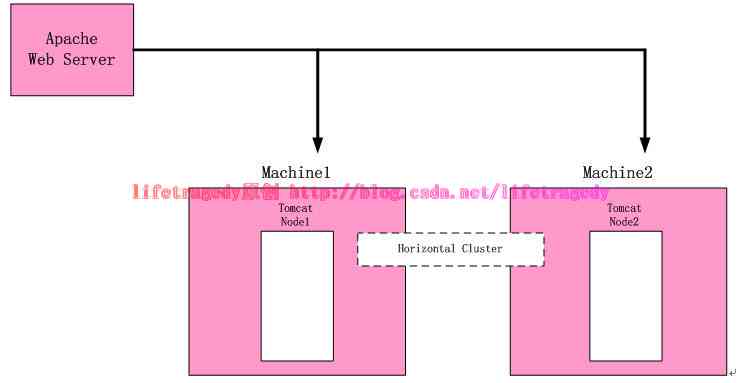
**二、集群分类**



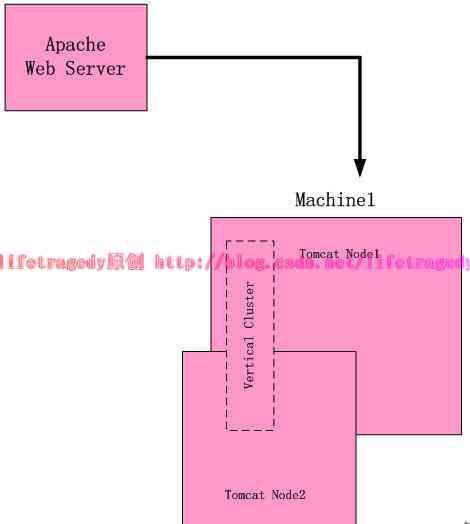
Tomcat作集群的逻辑架构是上面这样的一张图，关键是我们的production环境还需要规划好我们的物理架构。

**2.1 横向集群**

比如说，有两台Tomcat，分别运行在2台物理机上，好处是最大的即CPU扩展，内存也扩展了，处理能力也扩展了。



**2.2 纵向集群**

即，两个Tomcat的实例运行在一台物理器上，充分利用原有内存，CPU未得到扩展。  
  


**2.3 横向还是纵向**

一般来说，广为人们接受的是横向扩展的集群，可做大规模集群布署。但是我们这个case受制于客户即：

ü   不会再投入新机器了

ü   不会增加内存了

但是呢，通过压力测试报告我们可知：

ü   原有TomcatServer的CPU Loader不高，在23%左右

ü   原有TomcatServer上有8GB内存，而且是32位的，单台Tomcat只使用了1800MB左右的内存

ü   网络流量不高，单块千兆以太网卡完全可以处理掉

因此，我们只能做熊掌与鱼不能兼得的事，即采用了：纵向集群。

**2.4 Load Balance与High Available**

ü   Load Balance

简称LB即负载均衡，相当于1000根线程每个集群节点：Node负责处理500个，这样的效率是最高的。

ü   High Available

简称HA即高可用性，相当于1000根线程还是交给一台机器去慢慢处理，如果这台机器崩了，另一台机器顶上。

**三、集群架构中需要解决的问题**

集群规划好了怎么分，这不等于就可以开始实现集群了，一旦你的系统实现了集群，随之而来的问题就会出现了。

我们原有系统中有这样几个问题，在集群环境中是需要解决的，来看：

**3.1 解决上传文件同步的问题**

集群后就是两个Tomcat了，即和两个线程读同一个resource的问题是一样的，还好，我们原有上传文件是专门有一台文件伺服器的，这个问题不大，两个tomcat都往一台file server里上传，文件伺服器已经帮我们解决了**同名文件冲突的这个问题了**，如果原先的做法是把文件上传到Tomcat的目录中，那问题就大了，来看：

集群环境中，对于用户来说一切操作都是透明的，他也不知道我有几个Tomcat的实例运行在那边。

用户一点上传，可能上传到了Tomcat2中，但是下次要显示这个文件时，可能用到的是Tomcat1内的jsp或者是class，对不对？

于是，因为你把图片存在了Tomcat的目录中，因此导致了Tomcat1在显示图片时，取不到Tomcat2目录中存放的图片。

因此我们在工程一开始就强调存图片时要用一台专门的文件服务器或者是FTP服务器来存，就是为了避免将来出现这样的问题。

**3.2 解决Quartz在集群环境中的同步问题**

我们的系统用到一个Quartz（一个定时服务组件）来定时触发一些自动机制，现在有了两个Tomcat，粗想想每个Tomcat里运行自己的Quartz不就行了？

但是问题来了，如果两个Quartz在同一时间都触发了处理同一条定单，即该条定单会被处理两边。。。这不是影响效率和增加出错机率了吗？

因为本身Quartz所承受的压力几乎可以忽略不计的，它只是定时会触发脚本去运行，关键在于这个定时脚本的同步性，一致性的问题上。

**我们曾想过的解决方法：**

我们可以让一个Tomcat布署Quartz，另一个Tomcat里不布署Quartz

但这样做的结果就是如果布署Quartz的这个Tomcat崩溃掉了，这个Quartz是不是也崩啦？

**最后解决的办法：**

所 以我们还是必须在两台Tomcat里布署Quartz，然后使用HA的原则，即一个Quartz在运行时，另一台Quartz在监视着，并且不断的和另一 个Quartz之间保持勾通，一旦运行着的Quartz崩掉了，另一个Quartz在指定的秒数内起来接替原有的Quartz继续运行，对于 Quartz，我们同样也是面临着一个熊掌与鱼不能皆得的问题了，Quartz本身是支持集群的，而它支持的集群方式正是HA，和我们想的是一致的。

具体Quartz是如何在集群环境下作布署的，请见我的另一篇文章：[quartz在集群环境下的最终解决方案](http://blog.sina.com.cn/s/blog_9538eca90101gap9.html" \t "_blank)

解决了上述的问题后基本我们可以开始布署Tomcat这个集群了。

**四、布署Tomcat集群**

准备两个版本一致的Tomcat，分别起名为tomcat1，tomcat2。

**4.1 Apache中的配置**

²  worker.properties文件内容的修改

打开Apache HttpServer中的apache安装目录/conf/work.properties文件，大家还记得这个文件吗？

这是原有文件内容：

|  |
| --- |
| workers.tomcat\_home=d:/tomcat2  workers.java\_home=C:/jdk1.6.32  ps=/  worker.list=ajp13  worker.ajp13.port=8009  worker.ajp13.host=localhost  worker.ajp13.type=ajp13 |

现在开始改动成下面这样的内容（**把原有的worker.properties中的内容前面都加上#注释掉**）：

|  |
| --- |
| #workers.tomcat\_home=d:/tomcat2  #workers.java\_home=C:/jdk1.6.32  #ps=/  #worker.list=ajp13  #worker.ajp13.port=8009  #worker.ajp13.host=localhost  #worker.ajp13.type=ajp13  worker.list = controller  #tomcat1  worker.tomcat1.port=8009  worker.tomcat1.host=localhost  worker.tomcat1.type=ajp13  worker.tomcat1.lbfactor=1  #tomcat2  worker.tomcat2.port=9009  worker.tomcat2.host=localhost  worker.tomcat2.type=ajp13  worker.tomcat2.lbfactor=1  #========controller========  worker.controller.type=lb  worker.controller.balance\_workers=tomcat1,tomcat2  worker.lbcontroller.sticky\_session=0  worker.controller.sticky\_session\_force=true  worker.connection\_pool\_size=3000  worker.connection\_pool\_minsize=50  worker.connection\_pool\_timeout=50000 |

上面的这些设置的意思用中文来表达就是：

ü   两个tomcat，都位于localhost

ü   两个tomcat,tomcat1用8009，tomcat2用9009与apache保持jk\_mod的通讯

ü   不采用sticky\_session的机制

sticky\_session即：假设现在用户正连着tomcat1，而tomcat1崩了，那么此时它的session应该被复制到tomcat2上，由tomcat2继续负责该用户的操作，这就是load balance，此时这个用户因该可以继续操作。

如果你的sticky\_session设成了1，那么当你连的这台tomcat崩了后，你的操作因为是sticky（粘）住被指定的集群节点的，因此你的session是不会被复制和同步到另一个还存活着的tomcat节点上的。

ü   两台tomcat被分派到的任务的权重(lbfactor)为一致

你也可以设tomcat1 的worker.tomcat2.lbfactor=10，而tomcat2的worker.tomcat2.lbfactor=2，这个值越高，该tomcat节点被分派到的任务数就越多

²  httpd.conf文件内容的修改

找到下面这一行：

|  |
| --- |
| Include conf/extra/httpd-ssl.conf |

我们将它注释掉，因为我们在集群环境中不打算采用https，如果采用是https也一样，只是为了减省开销（**很多人都是用自己的开发电脑在做实验哦**）。

|  |
| --- |
| #Include conf/extra/httpd-ssl.conf |

找到原来的“”段

改成如下形式：

|  |
| --- |
| DocumentRoot d:/www      AllowOverride None      Order allow,deny            Allow from all           Order deny,allow           Deny from all  ServerAdmin localhost  DocumentRoot d:/www/  ServerName shnlap93:80  DirectoryIndex index.html index.htm index.jsp index.action  ErrorLog logs/shsc-error\_log.txt  CustomLog logs/shsc-access\_log.txt common    JkMount /\*WEB-INF controller  JkMount /\*j\_spring\_security\_check controller  JkMount /\*.action controller  JkMount /servlet/\* controller  JkMount /\*.jsp controller  JkMount /\*.do controller  JkMount /\*.action controller    JkMount /\*fckeditor/editor/filemanager/connectors/\*.\* controller  JkMount /fckeditor/editor/filemanager/connectors/\* controller |

注意：

原来的JKMount \*\*\* 后的 ajp13变成了什么了？

**controller**

**4.2 tomcat中的配置**

可以拿原有的tomcat复制成另一个tomcat，分别为d:\tomcat, d:\tomcat2。

打开tomcat中的conf目录中的server.xml，找到下面这行

1）

|  |
| --- |
|  |

**记得：**

***一定要把tomcat2中的这边的”SHUTDOWN”的port改成另一个端口号，两个tomcat如果是在集群环境中，此处的端口号绝不能一样。***

2）找到

|  |
| --- |
|  |

确保tomcat2中此处的端口不能为8080，我们就使用9090这个端口吧

3）把两个tomcat中原有的https的配置，整段去除

4）找到

|  |
| --- |
| URIEncoding="UTF-8"  minSpareThreads="25" maxSpareThreads="75"                 enableLookups="false" disableUploadTimeout="true" connectionTimeout="20000"                     acceptCount="300"  maxThreads="300" maxProcessors="1000" minProcessors="5"                     useURIValidationHack="false"                               compression="on" compressionMinSize="2048"                               compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"                 redirectPort="8443" /> |

确保tomcat2中这边的redirectPort为9443

5）找到

|  |
| --- |
|  |

改为：

|  |
| --- |
| URIEncoding="UTF-8"  minSpareThreads="25" maxSpareThreads="75"                              enableLookups="false" disableUploadTimeout="true" connectionTimeout="20000"                              acceptCount="300"  maxThreads="300" maxProcessors="1000" minProcessors="5"                              useURIValidationHack="false"                                        compression="on" compressionMinSize="2048"  compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"    /> |

**确保tomcat2的server.xml中此处的8009被改成了9009**且其它内容与上述内容一致（**redirectPort不要忘了改成9443**）

6）找到

|  |
| --- |
|  |

改成

|  |
| --- |
| tomcat1"> |

同时把tomcat2中此处内容改成

|  |
| --- |
| tomcat2"> |

7）

在刚才的

|  |
| --- |
| tomcat1"> |

的下面与在

|  |
| --- |
|  |

之上，在这之间加入如下一大陀的东西：

|  |
| --- |
| channelSendOptions="6">                        expireSessionsOnShutdown="false"                      notifyListenersOnReplication="true"                      mapSendOptions="6"/>                               bind="127.0.0.1"                           address="228.0.0.4"                           port="45564"                           frequency="500"                           dropTime="3000"/>                           address="auto"  **port="4001"**                         selectorTimeout="100"                         maxThreads="6"/>                                    filter=".\*\.gif;.\*\.js;.\*\.jpg;.\*\.png;.\*\.htm;.\*\.html;.\*\.css;.\*\.txt;"/> |

此处有一个Receiver port=”xxxx”，两个tomcat中此处的端口号必须唯一，即tomcat中我们使用的是port=4001，那么我们在tomcat2中将使用port=4002

8）把系统环境变更中的CATALINA\_HOME与TOMCAT\_HOME这两个变量去除掉

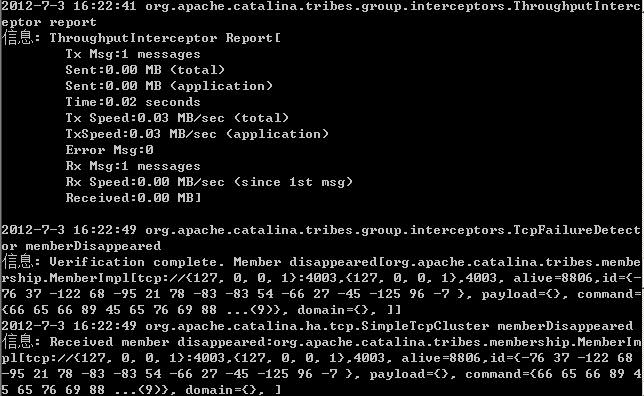
9）在每个tomcat的webapps目录下布署同样的一个工程，在布署工程前先确保你把工程中的WEB-INF\we b.xml文件做了如下的修改，在web.xml文件的最未尾即“”这一行前加入如下的一行：

|  |
| --- |
|  |

使该工程中的session可以被tomcat的集群节点进行轮循复制。

**4.3 启动集群**

好了，现在启动tomcat1, 启动tomcat2（其实无所谓顺序的），来看效果：



分别访问<http://localhost:8080/cbbs>与<http://localhost:9090/cbbs>

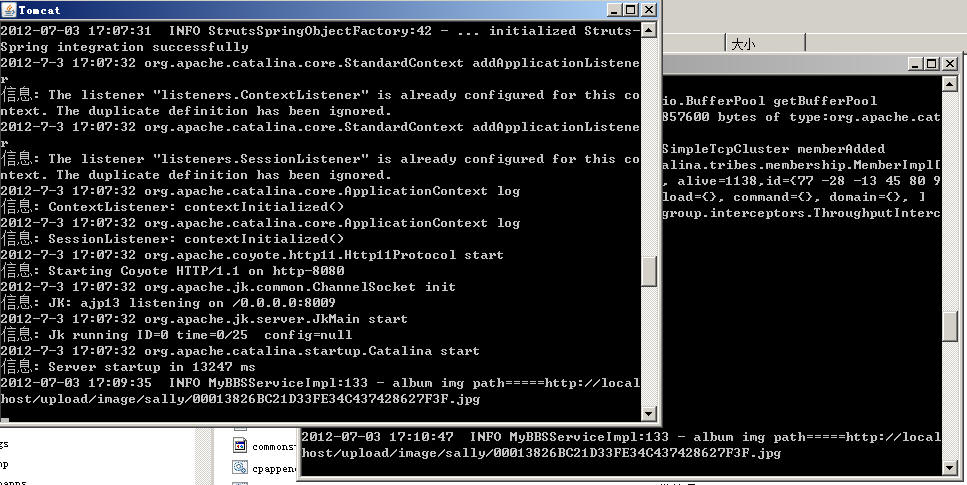
确保两个tomcat节点都起来了，然后此时，我们启动Apache

然后访问直接用<http://localhost/cbbs>不加端口的形式访问：



用sally/abcdefg登录，瞧，应用起来了。

然后我们拿另一台物理客户端，登录这个web应用，我们可以看到：



第一个tomcat正在负责处理我们第一次登录的请求。

当有第二个HTTP请求时，另一个tomcat自动开始肩负起我们第二个HTTP请求了，这就是Load Balance。