参考：<http://blog.csdn.net/res_cue/article/details/21756357>

**Tomcat简介**

 作者：杨晓(<http://blog.sina.com.cn/u/1237288325>)

**一、Tomcat背景**  
　　自从JSP发布之后，推出了各式各样的JSP引擎。Apache Group在完成GNUJSP1.0的开发以后，开始考虑在SUN的JSWDK基础上开发一个可以直接提供Web服务的JSP服务器，当然同时也支持Servlet,（me:servlet是servlet，jsp是jsp） 这样Tomcat就诞生了。Tomcat是jakarta项目中的一个重要的子项目，其被JavaWorld杂志的编辑选为2001年度最具创新的[**Java**](http://lib.csdn.net/base/java)产品，同时它又是sun公司官方推荐的servlet和jsp容器，因此其越来越多的受到软件公司和开发人员的喜爱。servlet和jsp的最新规范都可以在tomcat的新版本中得到实现。其次，Tomcat是完全免费的软件，任何人都可以从互联网上自由地下载。Tomcat与Apache的组合相当完美。

**二、Tomcat目录**

tomcat  
|---bin Tomcat：存放启动和关闭tomcat脚本；

|---confTomcat：存放不同的配置文件（server.xml和web.xml（全局web.xml配置））；  
|---doc：存放Tomcat文档；  
|---lib/japser/common：存放Tomcat运行需要的库文件（JARS）；  
|---logs：存放Tomcat执行时的LOG文件；  
|---src：存放Tomcat的源代码；  
|---webapps：Tomcat的主要Web发布目录（包括应用程序示例）；  
|---work：存放jsp编译后产生的class文件；

**三、Tomcat类加载**

Bootstrap($JAVA\_HOME/jre/lib/ext/\*.jar)  
System($CLASSPATH/\*.class和指定的jar)   
Common($CATALINA\_HOME/common 下的classes,lib,endores三个子目录)   
Catalina ($CATALINA\_HOME/server/下的classes和lib目录仅对Tomcat可见)  
&Shared($CATALINA\_HOME/shared/下的classes和lib目录以及$CATALINA\_HOME/lib目录)仅对Web应用程序可见,对Tomcat不可见WebApp($WEBAPP/Web-INF/\*仅对该WEB应用可见classes/\*.classlib/\*.jar)

加载类和资源的顺序为：  
1、/Web-INF/classes

2、/Web-INF/lib/\*.jar

3、Bootstrap

4、System

5、$CATALINA\_HOME/common/classes  
6、$CATALINA\_HOME/common/endores/\*.jar

7、$CATALINA\_HOME/common/lib/\*.jar

8、$CATALINA\_HOME/shared/classes  
9、$CATALINA\_HOME/shared/lib/\*.jar

**四、server.xml配置简介：**

下面讲述这个文件中的基本配置信息，更具体的配置信息请参考tomcat的文档：  
    **server:**

1、port 指定一个端口，这个端口负责监听关闭tomcat的请求   
          2、shutdown 指定向端口发送的命令字符串

输入：Telent locahost 8005

输入：SHUTDOWN //一定要与配置的命令的书写一样，包括大小写

**service:**

1、name 指定service的名字   
  **Connector**(表示客户端和service之间的连接)：  
          1、port 指定服务器端要创建的端口号，并在这个断口监听来自客户端的请求   
          2、minProcessors 服务器启动时创建的处理请求的线程数（类似线程池，总是处于等待或者活跃状态）   
          3、maxProcessors 最大可以创建的处理请求的线程数(me：其余的请求进入到等待队列)   
          4、enableLookups 如果为true，则可以通过调用request.getRemoteHost()进行DNS查询来得到远程客户端的实际主机名，若为false则不进行DNS查询，而是返回其ip地址

5、redirectPort 指定服务器正在处理http请求时收到了一个SSL传输请求后重定向的端口号   
          6、acceptCount 指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理   
          7、connectionTimeout 指定超时的时间数(以毫秒为单位)

8、protocol 指定请求的协议类型

**Engine**(表示指定service中的请求处理机，接收和处理来自Connector的请求)：  
          1、defaultHost 指定缺省的处理请求的主机名，它至少与其中的一个host元素的name属性值是一样的，当匹配不到时就会将请求转发到这个缺省的host   
  **Context**(表示一个web应用程序)：  
          1、docBase 应用程序的路径或者是WAR文件存放的路径   
          2、path 表示此web应用程序的url的前缀，这样请求的url为http://localhost:8080/path/\*\*\*\*   
          3、reloadable 这个属性非常重要，如果为true，则tomcat会自动检测应用程序的/WEB-INF/lib 和/WEB-INF/classes目录的变化，自动装载新的应用程序，我们可以在不重启tomcat的情况下改变应用程序   
**host** (表示一个虚拟主机)：  
          1、name 指定主机名   
          2、appBase 应用程序基本目录，即存放应用程序的目录   
          3、unpackWARs 如果为true，则tomcat会自动将WAR文件解压，否则不解压，直接从WAR文件中运行应用程序   
    **Logger**(表示日志，调试和错误信息)：  
          1、className 指定logger使用的类名，此类必须实现org.apache.catalina.Logger 接口   
          2、prefix 指定log文件的前缀   
          3、suffix 指定log文件的后缀   
          4、timestamp 如果为true，则log文件名中要加入时间，如下例:localhost\_log.2001-10-04.txt  
  **Realm** (表示存放用户名，密码及role的[**数据库**](http://lib.csdn.net/base/mysql))：  
          1、className 指定Realm使用的类名，此类必须实现org.apache.catalina.Realm接口   
  **Valve**(功能与Logger差不多，其prefix和suffix属性解释和Logger 中的一样)：  
          1、className 指定Valve使用的类名，如用org.apache.catalina.valves.AccessLogValve类可以记录应用程序的访问信息   
  **directory**（指定log文件存放的位置）：  
    1、pattern 有两个值，

common方式记录远程主机名或ip地址，用户名，日期，第一行请求的字符串，HTTP响应代码，发送的字节数。

combined方式比common方式记录的值更多

**五、web.xml配置简介：**

**1、默认(欢迎)文件的设置**

 在tomcat4\conf\web.xml中，<welcome-file-list>与IIS中的默认文件意思相同。

 <welcome-file-list>

 <welcome-file>index.html</welcome-file>

 <welcome-file>index.htm</welcome-file>

 <welcome-file>index.jsp</welcome-file>

 </welcome-file-list>

**2、报错文件的设置**

<error-page>

<error-code>404</error-code>

<location>/notFileFound.jsp</location>

</error-page>

<error-page>

<exception-type>java.lang.NullPointerException</exception-type>

<location>/null.jsp</location>

</error-page>

如果某文件资源没有找到，服务器要报404错误，按上述配置则会调用\webapps\ROOT\notFileFound.jsp。

如果执行的某个JSP文件产生NullPointException ，则会调用\webapps\ROOT\null.jsp

**3、会话超时的设置**

设置session 的过期时间，单位是分钟；

<session-config>

<session-timeout>30</session-timeout>

</session-config>

**4、过滤器的设置**

配置类似servlet的配置，先定义名字和类，然后对mapping

<filter>

<filter-name>FilterSource</filter-name>

<filter-class>project4. FilterSource</filter-class>

</filter>

<filter-mapping>

<filter-name>FilterSource</filter-name>

<url-pattern>/WwwServlet</url-pattern>

(<url-pattern>/haha/\*</url-pattern>)

</filter-mapping>

**过滤：**

1) 身份验证的过滤Authentication Filters

2) 日志和审核的过滤Logging and AuditingFilters

3) 图片转化的过滤Image conversionFilters

4) 数据压缩的过滤Data compressionFilters

5) 加密过滤Encryption Filters

6) Tokenizing Filters

7) 资源访问事件触发的过滤Filters that triggerresource access events XSL/T 过滤XSL/T filters

9) 内容类型的过滤Mime-type chain Filter 注意监听器的顺序，如：先安全过滤，然后资源，然后内容类型等，这个顺序可以自己定。

**六、管理  
    1、用户配置**  
      在进行具体Tomcat管理之前，先给tomcat添加一个用户，使这个用户有权限来进行管理。   
      打开conf目录下的tomcat-users.xml文件，在相应的位置添加下面一行：   
    <user name="user"password="user" roles="standard,manager"/>   
    然后重启tomcat，在浏览器中输入http://localhost:8080/manager/，会弹出对话框，输入上面的用户

名和密码即可。

Tomcat-9.0.0中需要这样配置：

<role rolename="manager-gui"/>

<user username="tomcat" password="s3cret" roles="manager-gui"/>

**2、应用程序列表**  
      在浏览器中输入http://localhost:8080/manager/list，浏览器将会显示如下的信息：   
    OK - Listed applications for virtualhost localhost   
    /ex:running:1   
    /examples:running:1   
    /webdav:running:0   
    /tomcat-docs:running:0   
    /manager:running:0   
    /:running:0   
     上面显示的信息分别为：应用程序的路径、当前状态、连接这个程序的session数

**3、重新装载应用程序**  
      在浏览器中输入http://localhost:8080/manager/reload?path=/examples，浏览器显示如下：   
    OK - Reloaded application at contextpath /examples

这表示example应用程序装载成功，如果我们将server.xml的Context元素的reloadable属性设为true，则没必要利用这种方式重新装载应用程序，因为tomcat会自动装载。

**4、显示session信息**  
    在浏览器中输入http://localhost:8080/manager/sessions?path=/examples，浏览器显示如下：   
    OK - Session information for application at contextpath /examples Default maximum session inactive

interval 30 minutes **5、启动和关闭应用程序**  
   在浏览器中输入[http://localhost:8080/manager/start?path=/examples和](http://localhost:8080/manager/start?path=/examples%E5%92%8C)

<http://localhost:8080/manager/stop?path=/examples>分别启动和关闭examples应用程序。

  注意：      
  1：经过我[**测试**](http://lib.csdn.net/base/softwaretest)，我设置Context的path="",reloadable=true，然后放一个WAR文件到webapps目录，结果tomcat不能检测出此文件(重启tomcat可以)，而把此文件解压，则tomcat会自动检测出这个新的应用程序。如果不能自动检测WAR文件，我们可以利用下面管理中讲的方法来部署应用程序。      
     
  2：默认的server.xml中，Realm元素只设置了一个className属性，但此文件中也包含几个通过JDBC连接到数据库进行验证的示例(被注释掉了)，通过Realm元素我们可以实现容器安全管理([**Container**](http://lib.csdn.net/base/docker)   Managed   Security)。      
     
  3：还有一些元素我们没有介绍，如Parameter，loader，你可以通过tomcat的文档获取这些元素的信息。     
   
此楼回复Re：

三)：管理         
  1：配置      
  在进行具体的管理之前，我们先给tomcat添加一个用户，使这个用户有权限来进行管理。     
  打开conf目录下的tomcat-users.xml文件，在相应的位置添加下面一行：    
  注意：这一行的最后部分一定是/>,tomcat的文档掉了/符号，如果没有/符号的话，tomcat重启时将无法访问应用程序。通过logs/catalina.out文件你可以看到这个错误的详细信息。        
  然后重启tomcat，在浏览器中输入http://localhost:8080/manager/，会弹出对话框，输入上面的用户名和密码即可。     
  2：应用程序列表      
  在浏览器中输入http://localhost:8080/manager/list，浏览器将会显示如下的信息：     
  OK   -   Listed   applications   for   virtual  host   localhost      
  /ex:running:1      
  /examples:running:1      
  /webdav:running:0      
  /tomcat-docs:running:0      
  /manager:running:0      
  /:running:0        
  上面的信息分别为应用程序的路径，当前状态(running   或者stopped)，与这个程序相连的session数。      
  3：重新装载应用程序      
  在浏览器中输入  http://localhost:8080/manager/reload?path=/examples，浏览器显示如下：    
  OK   -   Reloaded   application   at   context  path   /examples         
  表示example应用程序装载成功，如果我们将server.xml的Context元素的reloadable属性设为true(见上面表格)，则没必要利用这种方式重新装载应用程序，因为tomcat会自动装载。       
  4：显示session信息     
  在浏览器中输入http://localhost:8080/manager/sessions?path=/examples，浏览器显示如下：     
  OK   -   Session   information   for  application   at   context   path   /examples  Default   maximum   session   inactive   interval   30  minutes         
  5：启动和关闭应用程序      
  在浏览器中输入http://localhost:8080/manager/start?path=/examples和http://localhost:8080/manager/stop?path=/examples分别启动和关闭examples应用程序。     
  6：部署及撤销部署      
  WAR有两种组织方式，一种是按一定的目录结构组织文件，一种是一个后缀为WAR的压缩包，因此它的部署方式也有两种：      
  (1)：在浏览器中输入：http://localhost:8080/manager/install?path=/examples&war=file:/c:examples     
  就会将按目录结构组织的WAR部署     
     
  (2)：如果输入:http://localhost:8080/manager/install?path=/examples&war=jar:file:/c:examples.war!/     
  就会将按压缩包组织的WAR部署，注意此url后半部分一定要有!/号。      
  部署后就可以用   http://localhost:8080/examples访问了。     
  在浏览器中输入：http://localhost:8080/manager/remove?path=/examples  就会撤销刚才部署的应用程序。     
  (四)：与apache集成      
  虽然tomcat也可以作web服务器,但其处理静态html的速度比不上apache,且其作为web服务器的功能远不如apache,因此我们想把apache和tomcat集成起来。     
  我们以[**Linux**](http://lib.csdn.net/base/linux)系统为例介绍.     
  从apache网站下载apache1.3.22源代码版本，然后使用如下命令配置安装apache：    
  mkdir   /usr/local/apache      
  tar   zxvf   apache.1.32.tar.gz      
  cd   apache.1.32      
  ./configure   --prefix=/usr/local/apache   --enable-module=so     
  make      
  make   install

  注意configure命令指定目标安装目录，并且加入DSO(Dynamic   Shared   Object)支持，注意一定不要忘了这一个选项。      
     
  然后下载webapp模块，将解压后mod\_webapp.so文件放入apache的libexec目录，编辑apache的conf目录下的httpd.conf，在这个文件的最后加入下面三行：        
  LoadModule   webapp\_module   libexec/mod\_webapp.so     
  WebAppConnection   warpConnection   warp   localhost:8008     
  WebAppDeploy   examples   warpConnection   /examples/      
  第一行是加入webapp模块，如果编译apache时不增加DSO支持，则无法使用LoadModule指令，第二行指定tomcat与apache的连接，第三行指定部署那个应用，这两个指令使用格式如下：     
  WebAppConnection   [connection   name]   [provider]  [host:port]      
  WebAppDeploy   [application   name]   [connection  name]   [url   path]         
  其中connection   name指定连接名，provider只能是warp，port端口与你的tomcat的配置文件server.xml最后几行指定的要保持一致。文件如下：        
  port="8008"   minProcessors="5"  maxProcessors="75"      
  enableLookups="true"      
  acceptCount="10"   debug="0"/>      
  \*\*\*\*\*\*         
  application   name与你在tomcat中部署的应用名一致，url   path指定访问这个应用的url。例如上面的例子可以通过http://localhost/examples/来访问tomcat中的examples应用。

  (五)：中文问题     
  一般jsp的乱码问题可以通过在jsp中加入来解决，至于servlet的乱码在可以使用servlet2.3中提供的HttpServeletRequest.setCharacterEncoding函数。更详细的中文问题请见JSP/Servlet  中的汉字编码问题。         
  四：综述      
  tomcat作为一个servlet(jsp也被编译为servlet执行)容器，其应用前景是非常好的，如果与jboss结合起来，则可以实现sun的j2ee规范(用jboss作ejb服务器)。jboss的官方网站也提供集成了tomcat3.2\*的jboss以供下载。另外一个开源的应用服务器(enhydra)  也是基于tomcat的，其提供了更友好的管理界面，部署应用程序也更简单，功能也更强大。

Tomcat 原理解说：启动过程分析

2007-09-13 19:08

**1 - Tomcat Server的组成部分**

**1.1 - Server**

A Server element represents the entire Catalina servletcontainer. (Singleton)

Me:Server代表了tomcat的整个servlet容器

**1.2 - Service**

A Service element represents the combination of one ormore Connector components that share a single Engine  
Service是这样一个集合：它由一个或者多个Connector组成，以及一个Engine，负责处理所有Connector所获得的客户请求。

Me:在service中包含一个engine和多个connector,它是为了将一个engine与多个connector联系起来。

**1.3 - Connector**

一个Connector将在某个指定端口上侦听客户请求，并将获得的请求交给Engine来处理，从Engine处获得回应并返回客户  
TOMCAT有两个典型的Connector，一个直接侦听来自browser的http请求，一个侦听来自其它WebServer的请求  
Coyote Http/1.1 Connector 在端口8080处侦听来自客户browser的http请求  
Coyote JK2 Connector 在端口8009处侦听来自其它WebServer(Apache)的servlet/jsp代理请求

**1.4 - Engine**

The Engine element represents the entire requestprocessing machinery associated with a particular Service  
It receives and processes all requests from one or more Connectors  
and returns the completed response to the Connector for ultimate transmissionback to the client  
Engine下可以配置多个虚拟主机Virtual Host，每个虚拟主机都有一个域名  
当Engine获得一个请求时，它把该请求匹配到某个Host上，然后把该请求交给该Host来处理  
Engine有一个默认虚拟主机，当请求无法匹配到任何一个Host上的时候，将交给该默认Host来处理

Me：一个engine对应多个host,将connector接受到的请求转发到匹配的host，host就相当于是apache里面的虚拟主机，这里的host对应一个独立的app。

**1.5 - Host**

代表一个VirtualHost，虚拟主机，每个虚拟主机和某个网络域名Domain Name相匹配  
每个虚拟主机下都可以部署(deploy)一个或者多个WebApp，每个Web App对应于一个Context，有一个Context path  
当Host获得一个请求时，将把该请求匹配到某个Context上，然后把该请求交给该Context来处理  
匹配的方法是“最长匹配”，所以一个path==""的Context将成为该Host的默认Context  
所有无法和其它Context的路径名匹配的请求都将最终和该默认Context匹配。

Me：一个host对应一个或者多个app，每个app都有对应的context与之相匹配，host获得一个请求后将该请求交给context来处理。

**1.6 - Context**

一个Context对应于一个Web Application，一个WebApplication由一个或者多个Servlet组成  
Context在创建的时候将根据配置文件$CATALINA\_HOME/conf/web.xml和$WEBAPP\_HOME/WEB-INF/web.xml载入Servlet类  
当Context获得请求时，将在自己的映射表(mappingtable)中寻找相匹配的Servlet类  
如果找到，则执行该类，获得请求的回应，并返回

**2 - Tomcat Server的结构图**

**3 - 配置文件$CATALINA\_HOME/conf/server.xml的说明**

该文件描述了如何启动TomcatServer   
<!----------------------------------------------------------------------------------------------->  
<web-app>  
     <!-- 概述：  
          该文件是所有的WEBAPP共用的部署配置文件，  
          每当一个WEBAPP被DEPLOY，该文件都将先被处理，然后才是WEBAPP自己的/WEB-INF/web.xml  
         -->     <!--    +-------------------------+     -->  
     <!--    |       servlet类定义部分      |     -->  
     <!--    +-------------------------+    -->     <!-- DefaultServlet  
          当用户的HTTP请求无法匹配任何一个servlet的时候，该servlet被执行  
          URL PATTERN MAPPING : /  
         -->

<servlet>  
          <servlet-name>default</servlet-name>  
          <servlet-class>  
            org.apache.catalina.servlets.DefaultServlet  
          </servlet-class>

           <init-param>  
              <param-name>debug</param-name>  
              <param-value>0</param-value>  
          </init-param>  
           <init-param>  
              <param-name>listings</param-name>  
              <param-value>true</param-value>  
           </init-param>  
          <load-on-startup>1</load-on-startup>  
       </servlet>  
     <!-- InvokerServlet  
          处理一个WEBAPP中的匿名servlet  
          当一个servlet被编写并编译放入/WEB-INF/classes/中，却没有在/WEB-INF/web.xml中定义的时候  
          该servlet被调用，把匿名servlet映射成/servlet/ClassName的形式  
          URL PATTERN MAPPING :/servlet/\*  
         -->       <servlet>  
          <servlet-name>invoker</servlet-name>  
          <servlet-class>  
            org.apache.catalina.servlets.InvokerServlet  
          </servlet-class>  
           <init-param>  
              <param-name>debug</param-name>  
              <param-value>0</param-value>  
           </init-param>  
          <load-on-startup>2</load-on-startup>  
       </servlet>  
     <!-- JspServlet  
          当请求的是一个JSP页面的时候（\*.jsp）该servlet被调用  
          它是一个JSP编译器，将请求的JSP页面编译成为servlet再执行  
          URL PATTERN MAPPING :\*.jsp  
         -->       <servlet>  
          <servlet-name>jsp</servlet-name>  
          <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>  
           <init-param>  
              <param-name>logVerbosityLevel</param-name>  
              <param-value>WARNING</param-value>  
          </init-param>  
          <load-on-startup>3</load-on-startup>  
       </servlet>    <!--     +---------------------------+    -->  
     <!--    |       servlet映射定义部分      |     -->  
     <!--    +---------------------------+     -->     
       <servlet-mapping>  
          <servlet-name>default</servlet-name>  
          <url-pattern>/</url-pattern>  
      </servlet-mapping>      <servlet-mapping>  
           <servlet-name>invoker</servlet-name>  
          <url-pattern>/servlet/\*</url-pattern>  
      </servlet-mapping>      <servlet-mapping>  
          <servlet-name>jsp</servlet-name>  
          <url-pattern>\*.jsp</url-pattern>

//=========================20171028

       </servlet-mapping>  
     <!--    +------------------------+     -->  
     <!--    |       其它部分，略去先      |     -->  
     <!--    +------------------------+    -->       ... ... ......</web-app><!-- 启动Server  
        在端口8005处等待关闭命令  
        如果接受到"SHUTDOWN"字符串则关闭服务器  
        --><Serverport="8005" shutdown="SHUTDOWN" debug="0">  
     <!-- Listener ???  
          目前没有看到这里  
         -->     <ListenerclassName="org.apache.catalina.mbeans.ServerLifecycleListener"debug="0"/>  
     <ListenerclassName="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener"debug="0"/>  
     <!-- Global JNDI resources ???  
          目前没有看到这里，先略去  
         -->     <GlobalNamingResources>  
       ... ... ... ...  
     </GlobalNamingResources>  
     <!-- Tomcat的StandaloneService  
          Service是一组Connector的集合  
          它们共用一个Engine来处理所有Connector收到的请求  
         -->     <Service name="Tomcat-Standalone">  
       <!-- Coyote HTTP/1.1 Connector  
            className : 该Connector的实现类是org.apache.coyote.tomcat4.CoyoteConnector  
            port : 在端口号8080处侦听来自客户browser的HTTP1.1请求  
           minProcessors : 该Connector先创建5个线程等待客户请求，每个请求由一个线程负责  
           maxProcessors : 当现有的线程不够服务客户请求时，若线程总数不足75个，则创建新线程来处理请求  
            acceptCount: 当现有线程已经达到最大数75时，为客户请求排队  
                         当队列中请求数超过100时，后来的请求返回Connectionrefused错误  
            redirectport: 当客户请求是https时，把该请求转发到端口8443去  
            其它属性略  
           -->       <ConnectorclassName="org.apache.coyote.tomcat4.CoyoteConnector"  
                 port="8080"  
                 minProcessors="5" maxProcessors="75"acceptCount="100"  
                 enableLookups="true"  
                 redirectPort="8443"  
                 debug="0"  
                 connectionTimeout="20000"  
                 useURIValidationHack="false"  
                 disableUploadTimeout="true" />  
       <!-- Engine用来处理Connector收到的Http请求  
            它将匹配请求和自己的虚拟主机，并把请求转交给对应的Host来处理  
            默认虚拟主机是localhost  
           -->       <Enginename="Standalone" defaultHost="localhost"debug="0">  
            <!-- 日志类，目前没有看到，略去先-->         <LoggerclassName="org.apache.catalina.logger.FileLogger".../>         <!-- Realm，目前没有看到，略去先-->         <RealmclassName="org.apache.catalina.realm.UserDatabaseRealm" .../>  
         <!-- 虚拟主机localhost  
             appBase : 该虚拟主机的根目录是webapps/  
              它将匹配请求和自己的Context的路径，并把请求转交给对应的Context来处理  
             -->         <Host name="localhost"debug="0" appBase="webapps" unpackWARs="true"autoDeploy="true">  
               <!-- 日志类，目前没有看到，略去先-->           <LoggerclassName="org.apache.catalina.logger.FileLogger" .../>  
               <!-- Context，对应于一个Web App  
               path : 该Context的路径名是""，故该Context是该Host的默认Context  
               docBase : 该Context的根目录是webapps/mycontext/  
               -->           <Contextpath="" docBase="mycontext" debug="0"/>  
                 <!-- 另外一个Context，路径名是/wsota-->           <Contextpath="/wsota" docBase="wsotaProject"debug="0"/>  </Host>      </Engine></Service></Server>

**4 - Context的部署配置文件web.xml的说明**

一个Context对应于一个Web App，每个Web App是由一个或者多个servlet组成的  
当一个Web App被初始化的时候，它将用自己的ClassLoader对象载入“部署配置文件web.xml”中定义的每个servlet类  
它首先载入在$CATALINA\_HOME/conf/web.xml中部署的servlet类  
然后载入在自己的Web App根目录下的WEB-INF/web.xml中部署的servlet类  
web.xml文件有两部分：servlet类定义和servlet映射定义  
每个被载入的servlet类都有一个名字，且被填入该Context的映射表(mapping table)中，和某种URL PATTERN对应  
当该Context获得请求时，将查询mapping table，找到被请求的servlet，并执行以获得请求回应

分析一下所有的Context共享的web.xml文件，在其中定义的servlet被所有的Web App载入

**5 - Tomcat Server处理一个http请求的过程**

假设来自客户的请求为：  
http://localhost:8080/wsota/wsota\_index.jsp

1) 请求被发送到本机端口8080，被在那里侦听的Coyote HTTP/1.1 Connector获得  
2) Connector把该请求交给它所在的Service的Engine来处理，并等待来自Engine的回应  
3) Engine获得请求localhost/wsota/wsota\_index.jsp，匹配它所拥有的所有虚拟主机Host  
4) Engine匹配到名为localhost的Host（即使匹配不到也把请求交给该Host处理，因为该Host被定义为该Engine的默认主机）  
5) localhost Host获得请求/wsota/wsota\_index.jsp，匹配它所拥有的所有Context  
6) Host匹配到路径为/wsota的Context（如果匹配不到就把该请求交给路径名为""的Context去处理）  
7) path="/wsota"的Context获得请求/wsota\_index.jsp，在它的mapping table中寻找对应的servlet  
8) Context匹配到URL PATTERN为\*.jsp的servlet，对应于JspServlet类  
9) 构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用JspServlet的doGet或doPost方法  
10)Context把执行完了之后的HttpServletResponse对象返回给Host  
11)Host把HttpServletResponse对象返回给Engine  
12)Engine把HttpServletResponse对象返回给Connector  
13)Connector把HttpServletResponse对象返回给客户browser

### 

### 1 - Tomcat Server的组成部分

**1.1 - Server**

A Server element representsthe entire Catalina servlet container. (Singleton)

**1.2 - Service**

A Service element representsthe combination of o­ne or more Connector components that share a single Engine  
Service是这样一个集合：它由一个或者多个Connector组成，以及一个Engine，负责处理所有Connector所获得的客户请求

**1.3 - Connector**

一个Connector将在某个指定端口上侦听客户请求，并将获得的请求交给Engine来处理，从Engine处获得回应并返回客户  
TOMCAT有两个典型的Connector，一个直接侦听来自browser的http请求，一个侦听来自其它WebServer的请求  
Coyote Http/1.1 Connector 在端口8080处侦听来自客户browser的http请求  
Coyote JK2 Connector 在端口8009处侦听来自其它WebServer(Apache)的servlet/jsp代理请求

**1.4 - Engine**

The Engine element representsthe entire request processing machinery associated with a particular Service  
It receives and processes all requests from o­ne or more Connectors  
and returns the completed response to the Connector for ultimate transmissionback to the client  
Engine下可以配置多个虚拟主机VirtualHost，每个虚拟主机都有一个域名  
当Engine获得一个请求时，它把该请求匹配到某个Host上，然后把该请求交给该Host来处理  
Engine有一个默认虚拟主机，当请求无法匹配到任何一个Host上的时候，将交给该默认Host来处理

**1.5 - Host**

代表一个Virtual Host，虚拟主机，每个虚拟主机和某个网络域名Domain Name相匹配  
每个虚拟主机下都可以部署(deploy)一个或者多个Web App，每个Web App对应于一个Context，有一个Contextpath  
当Host获得一个请求时，将把该请求匹配到某个Context上，然后把该请求交给该Context来处理  
匹配的方法是“最长匹配”，所以一个path==""的Context将成为该Host的默认Context  
所有无法和其它Context的路径名匹配的请求都将最终和该默认Context匹配

**1.6 - Context**

一个Context对应于一个Web Application，一个Web Application由一个或者多个Servlet组成  
Context在创建的时候将根据配置文件$CATALINA\_HOME/conf/web.xml和$WEBAPP\_HOME/WEB-INF/web.xml载入Servlet类  
当Context获得请求时，将在自己的映射表(mappingtable)中寻找相匹配的Servlet类  
如果找到，则执行该类，获得请求的回应，并返回

### 2 - Tomcat Server的结构图

### 3 - 配置文件$CATALINA\_HOME/conf/server.xml的说明

该文件描述了如何启动Tomcat Server

### 4 - Context的部署配置文件web.xml的说明

一个Context对应于一个Web App，每个Web App是由一个或者多个servlet组成的  
当一个Web App被初始化的时候，它将用自己的ClassLoader对象载入“部署配置文件web.xml”中定义的每个servlet类  
它首先载入在$CATALINA\_HOME/conf/web.xml中部署的servlet类  
然后载入在自己的Web App根目录下的WEB-INF/web.xml中部署的servlet类  
web.xml文件有两部分：servlet类定义和servlet映射定义  
每个被载入的servlet类都有一个名字，且被填入该Context的映射表(mappingtable)中，和某种URLPATTERN对应  
当该Context获得请求时，将查询mappingtable，找到被请求的servlet，并执行以获得请求回应

分析一下所有的Context共享的web.xml文件，在其中定义的servlet被所有的Web App载入

### 5 - Tomcat Server处理一个http请求的过程

假设来自客户的请求为：  
http://localhost:8080/wsota/wsota\_index.jsp

1) 请求被发送到本机端口8080，被在那里侦听的CoyoteHTTP/1.1 Connector获得  
2) Connector把该请求交给它所在的Service的Engine来处理，并等待来自Engine的回应  
3) Engine获得请求localhost/wsota/wsota\_index.jsp，匹配它所拥有的所有虚拟主机Host  
4) Engine匹配到名为localhost的Host（即使匹配不到也把请求交给该Host处理，因为该Host被定义为该Engine的默认主机）  
5) localhost Host获得请求/wsota/wsota\_index.jsp，匹配它所拥有的所有Context  
6) Host匹配到路径为/wsota的Context（如果匹配不到就把该请求交给路径名为""的Context去处理）  
7) path="/wsota"的Context获得请求/wsota\_index.jsp，在它的mapping table中寻找对应的servlet  
8) Context匹配到URLPATTERN为\*.jsp的servlet，对应于JspServlet类  
9) 构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用JspServlet的doGet或doPost方法  
10)Context把执行完了之后的HttpServletResponse对象返回给Host  
11)Host把HttpServletResponse对象返回给Engine  
12)Engine把HttpServletResponse对象返回给Connector  
13)Connector把HttpServletResponse对象返回给客户browser

<!------------------------------------------------>

<web-app>

<!-- 概述：

该文件是所有的WEB APP共用的部署配置文件，

每当一个WEB APP被DEPLOY，该文件都将先被处理，然后才是WEB APP自己的/WEB-INF/web.xml-->

<!--  +-------------------------+  -->

<!--  |    servlet类定义部分    |  -->

<!--  +-------------------------+  -->

<!-- DefaultServlet 当用户的HTTP请求无法匹配任何一个servlet的时候，该servlet被执行

URL PATTERN MAPPING : /        -->

<servlet>

<servlet-name>default</servlet-name>

<servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>

<init-param>

<param-name>debug</param-name>

<param-value>0</param-value>

</init-param>

<init-param>

<param-name>listings</param-name>

<param-value>true</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- InvokerServlet 处理一个WEB APP中的匿名servlet

当一个servlet被编写并编译放入/WEB-INF/classes/中，却没有在/WEB-INF/web.xml中定义的时候

该servlet被调用，把匿名servlet映射成/servlet/ClassName的形式

URL PATTERN MAPPING : /servlet/\*        -->

<servlet>

<servlet-name>invoker</servlet-name>

<servlet-class>org.apache.catalina.servlets.InvokerServlet</servlet-class>

<init-param>

<param-name>debug</param-name>

<param-value>0</param-value>

</init-param>

<load-on-startup>2</load-on-startup>

</servlet>

<!-- JspServlet  当请求的是一个JSP页面的时候（\*.jsp）该servlet被调用

它是一个JSP编译器，将请求的JSP页面编译成为servlet再执行

URL PATTERN MAPPING : \*.jsp        -->

<servlet>

<servlet-name>jsp</servlet-name>

<servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>

<init-param>

<param-name>logVerbosityLevel</param-name>

<param-value>WARNING</param-value>

</init-param>

<load-on-startup>3</load-on-startup>

</servlet>

<!--  +---------------------------+  -->

<!--  |    servlet映射定义部分    |  -->

<!--  +---------------------------+  -->

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>invoker</servlet-name>

<url-pattern>/servlet/\*</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>jsp</servlet-name>

<url-pattern>\*.jsp</url-pattern>

</servlet-mapping>

<!--  +------------------------+  -->

<!--  |    其它部分，略去先    |  -->

<!--  +------------------------+  -->

... ... ... ...

</web-app>

<!----------------------------------->

<!------------------------------------------------------------------------->

<!-- 启动Server 在端口8005处等待关闭命令  如果接受到"SHUTDOWN"字符串则关闭服务器      -->

<Server port="8005" shutdown="SHUTDOWN" debug="0">

<!-- Listener ???        目前没有看到这里        -->

<Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" debug="0"/>

<Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" debug="0"/>

<!-- Global JNDI resources ???        目前没有看到这里，先略去        -->

<GlobalNamingResources>     ... ... ... ...   </GlobalNamingResources>

<!-- Tomcat的Standalone Service Service是一组Connector的集合

它们共用一个Engine来处理所有Connector收到的请求-->

<Service name="Tomcat-Standalone">

<!-- Coyote HTTP/1.1 Connector  className: 该Connector的实现类是

org.apache.coyote.tomcat4.CoyoteConnector

port : 在端口号8080处侦听来自客户browser的HTTP1.1请求

minProcessors : 该Connector先创建5个线程等待客户请求，每个请求由一个线程负责

maxProcessors : 当现有的线程不够服务客户请求时，若线程总数不足75个，则创建新线程来处理请求

acceptCount : 当现有线程已经达到最大数75时，为客户请求排队 当队列中请求数超过100时，后来的请求返回Connection refused错误

redirectport : 当客户请求是https时，把该请求转发到端口8443去 其它属性略          -->

<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"

port="8080"

minProcessors="5" maxProcessors="75" acceptCount="100"

enableLookups="true"                 redirectPort="8443"

debug="0"                 connectionTimeout="20000"

useURIValidationHack="false"                 disableUploadTimeout="true" />

<!-- Engine用来处理Connector收到的Http请求

它将匹配请求和自己的虚拟主机，并把请求转交给对应的Host来处理

默认虚拟主机是localhost          -->

<Engine name="Standalone" defaultHost="localhost" debug="0">

<!-- 日志类，目前没有看到，略去先 -->

<Logger className="org.apache.catalina.logger.FileLogger" .../>

<!-- Realm，目前没有看到，略去先 -->

<Realm className="org.apache.catalina.realm.UserDatabaseRealm" .../>

<!-- 虚拟主机localhost  appBase : 该虚拟主机的根目录是webapps/

它将匹配请求和自己的Context的路径，并把请求转交给对应的Context来处理            -->

<Host name="localhost" debug="0" appBase="webapps" unpackWARs="true" autoDeploy="true">

<!-- 日志类，目前没有看到，略去先 -->

<Logger className="org.apache.catalina.logger.FileLogger" .../>

<!-- Context，对应于一个Web App

path : 该Context的路径名是""，故该Context是该Host的默认Context

docBase : 该Context的根目录是webapps/mycontext/              -->

<Context path="" docBase="mycontext" debug="0"/>

<!-- 另外一个Context，路径名是/wsota -->

<Context path="/wsota" docBase="wsotaProject" debug="0"/>

</Host></Engine></Service></Server>

**tomcat启动脚本分析一**

2008-05-12 08:59

Tomcat是应用颇广的Servlet和JSP的容器，可以说几乎每个学习JSP的人都接触过它。在实际开发环境中，它还可以作为轻巧灵便的应用服务器用来调试和部署。在Tomcat上开发的Servlet和JSP程序，可以非常方便地部署到Weblogic及Websphere等生产服务器上去。有些情况下，它还可以和ApacheWeb Server配合使用，作为小型企业的Web Application Server。本文以Tomcat5.0.28为例，讨论一下Tomcat的启动过程及常见问题。大家都知道，Tomcat最普遍的启动方法就是在DOS命令行窗口中执行Tomcat的bin目录下的startup.bat文件（双击该文件即可启动)。   
**1. startup.bat脚本解析**下面我们首先来看看startup.bat脚本是如何启动Tomcat的。Startup脚本非常明了，它的主要任务就是先设置CATALINA\_HOME的环境变量，让该环境变量指向Tomcat的安装目录；然后在安装目录的bin子目录下找catalina.bat文件，如果找不到，就打印出错误信息并退出，如果在bin目录下找到catalina.bat文件，就准备执行该文件。不过，在执行该文件之前，startup脚本还做了一件工作，那就是寻找参数。startup脚本可以带多个参数，在Java主类中，第一个参数为args[0]，而在MS-DOS脚本中，第一个参数可以通过”%1”得到，如果您想要得到DOS脚本中得到第二个参数，调用”%2”。但是，在脚本参数数目未知的情况下，我们就只能通过goto循环调用shift命令来获得脚本的所有参数，下面这个简单的实验脚本有助于我们理解如何使用shift命令。 @echo off   
echo Print parameters by %%1 andshift loop   
:loopStart   
if""%1""=="""" goto loopEnd   
echo Paramter is %1   
shift   
goto loopStart   
:loopEnd   
如果把上面的脚本保存成test.bat并在DOS中执行该脚本，执行结果如下。   
下面就是Startup.bat脚本的详细注释，如果想要在DOS脚本中注释一行，前面加上rem即可，这相当于Java的双斜杠(//)注释，下面程序中，凡是以REM开头然后接着中文的行，都是笔者加上的注释，以帮助读者理解该脚本。@echo off   
if "%OS%" == "Windows\_NT" setlocal   
rem ---------------------------------------------------------------------------  
rem Start script for the CATALINA Server   
rem   
rem $Id: startup.bat,v 1.6 2004/05/27 18:25:11 yoavs Exp $   
rem ---------------------------------------------------------------------------rem 首先定义一个变量CURRENT\_DIR，然后把当前目录作为一个字符串赋给它   
rem Guess CATALINA\_HOME if not defined   
set CURRENT\_DIR=%cd% rem 如果CATALINA\_HOME变量已经赋值，程序转到gotHome行继续   
if not "%CATALINA\_HOME%" == "" goto gotHome rem 把当前目录的值作为字符串赋给CATALINA\_HOME变量   
set CATALINA\_HOME=%CURRENT\_DIR% rem 如果当前目录的bin子目录下发现catalina.bat文件，程序转到okHome行继续   
rem 否则执行下一行“cd..”。   
Rem 因为缺省情况下当前目录就是bin目录，所以缺省情况下程序不转入okHome行   
if exist "%CATALINA\_HOME%\bin\catalina.bat" goto okHome rem 回上层目录   
cd .. rem 把当前目录的上层目录作为字符串赋给再次CATALINA\_HOME变量，覆盖第一次的赋值   
set CATALINA\_HOME=%cd% rem 重新回到当前目录   
cd %CURRENT\_DIR% :gotHome rem 如果在当前目录的父目录的bin子文件夹下发现catalina.bat文件，程序转到okHome行继续   
rem 否则程序打印两行错误信息，然后退出   
if exist "%CATALINA\_HOME%\bin\catalina.bat" goto okHome   
echo The CATALINA\_HOME environment variable is not defined correctly   
echo This environment variable is needed to run this program   
goto end   
:okHome rem 设置一个变量，存储catalina.bat的完整路径   
set EXECUTABLE=%CATALINA\_HOME%\bin\catalina.bat rem 再次检查catalina.bat是否存在，如果不存在，打印两行错误信息，退出   
rem Check that target executable exists   
if exist "%EXECUTABLE%" goto okExec   
echo Cannot find %EXECUTABLE%   
echo This file is needed to run this program   
goto end   
:okExec rem 检查脚本是否带参数，如果发现参数，把第一个参数赋值给 CMD\_LINE\_ARGS变量   
rem 如果脚本带有多个参数，程序设置了一个循环setArgs，   
rem 不断把所带的参数加到CMD\_LINE\_ARGS变量中   
rem Get remaining unshifted command line arguments and save them in the   
set CMD\_LINE\_ARGS=   
:setArgs   
if ""%1""=="""" goto doneSetArgs   
set CMD\_LINE\_ARGS=%CMD\_LINE\_ARGS% %1   
shift   
goto setArgs   
:doneSetArgs rem 执行catalina.bat脚本，该脚本的第一个参数为start，后面的参数是从本脚本中传入的   
call "%EXECUTABLE%" start %CMD\_LINE\_ARGS% :end 那么读者也许要问，问什么要通过startup脚本来调用catalina脚本，直接一个脚本不更方便吗？细心的读者或许会发现，在缺省情况下，startup脚本只能放在两个位置，一是Tomcat的安装目录，一个就是Tomcat安装目录的bin子目录。这是什么意思呢？如果Tomcat安装目录是C:\carl\it\tomcat\_research\jakarta-tomcat-5.0.28，那么startup脚本只能放在C:\carl\it\tomcat\_research\jakarta-tomcat-5.0.28\目录或者C:\carl\it\tomcat\_research\jakarta-tomcat-5.0.28\bin目录下才能运行，如果拷贝startup文件到其它目录，我们就会看到下面的错误（以Tomcat运行在C:\carl目录下为例）： C:\carl>startup   
CATALINA\_HOME C:\   
The CATALINA\_HOME environment variable is not defined correctly   
This environment variable is needed to run this program   
那么，如果我们想让Tomcat在桌面上能成功启动，那么该怎么办呢？当然，我们首先想到的是为startup创建一个快捷方式，然后拷贝该快捷方式到桌面，这是一种办法。另一种办法就是修改startup.bat文件，在startup.bat脚本的第八行处添上一句脚本，让 CATALINA\_HOME变量指向用户Tomcat的安装路径，这样，startup.bat就可以拷贝到机器的任何目录执行。这个startup.bat脚本就是提供给使用者用来修改的，我们可以在其中设置JAVA\_HOME，CATALINA\_HOME等环境变量，但我们并不需要深入到较为复杂的catalina.bat脚本中，这正是startup.bat脚本的真正用意所在。我们知道，软件设计模式中有一个重要的原则就是开闭原则，即我们可以允许别人扩展我们的程序，但在程序发布后，我们拒绝任何修改，因为修改会产生新的Bug，使得我们已经Bug-free的程序又要重新测试。开闭原则是面向对象世界中的一个非常重要的原则，我们可以把这个原则从Java类扩展至源代码级别。startup脚本就是要求用户不要修改catalina.bat脚本，这是符合软件设计思想的。我们如果想要彻底贯彻这个重要的软件设计原则，可以写一个新脚本tomcat.bat，脚本内容大致如下： set JAVA\_HOME=C:\Program Files\Java\jdk1.5.0\_09   
set CATALINA\_HOME=C:\carl\it\tomcat\_research\jakarta-tomcat-5.0.28   
call %CATALINA\_HOME%\bin\startup.bat 这个tomcat.bat文件可以存放在任何目录并能执行，并且不需要修改tomcat自带的任何脚本及其它环境变量，这就彻底贯彻了开闭原则。

**有关Tomcat应用程序目录、端口、默认目录、应用程序默认打开文件、使用数据库连接池等方面的配置**

关键字

  有关Tomcat应用程序目录、端口、默认目录、应用程序默认打开文件、使用数据库连接池等方面的配置出处

修改Tomcat应用程序目录  
 <Host name="localhost" debug="0"appBase="webapps" unpackWARs="true"   
  autoDeploy="true" xmlValidation="false"xmlNamespaceAware="false">  
  修改成  
 <Host name="localhost" debug="0"appBase="mytest" unpackWARs="true"   
  autoDeploy="true" xmlValidation="false"xmlNamespaceAware="false">  
  mytest目录是和webapps在同一父目录下。  
  同时还要将原来webapps目录中的一些文件拷贝过来（如果不确定，请将安装后的  
webapps下的所以文件夹拷贝过来）  
  win2000 tomcat 5.0.27测试    
  网上有人这么写，我没有测试过。

 在Tomcat的conf目录下面的server.xml里面，修改    
 <Host   name="localhost"   debug="0"  appBase="E:\Project\www\_project\_com"     
   unpackWARs="true"  autoDeploy="true"     
   xmlValidation="false"  xmlNamespaceAware="false">         
  E:\Project\www\_project\_com这个硬盘目录下面还要建立一个ROOT（必须大写）目录，  
  里面放文件，还要有一个web-inf文件夹，这个你可以参考tomcat的webapps目录里面的  
  东西拷贝过去就可以

修改Tomcat端口  
  查找 <Connector port="8080" minProcessors="5"  maxProcessors="75"     
       enableLookups="true"  redirectPort="8443"     
       acceptCount="100"  debug="0"   connectionTimeout="20000"    
        useURIValidationHack="false"  disableUploadTimeout="true"   />  
  将port="8080"改成port="80"。

修改Tomcat默认目录  
  在conf文件夹中，打开server.xml,找到</host>,在它前面加上下面的代码：    
  <Context   path=""  docBase="f:\jsp"   debug="0"  reloadable="true"     
          crossContext="true">     
  </Context>

修改应用程序默认打开文件  
  在你的应用程序目录下修改WEB-INF\web.xml文件（注意：不是tomcat\conf\web.xml）  
  在<web-app...></web-app>中添加节点如下：  
  <welcome-file-list>     
                 <welcome-file>index.jsp</welcome-file>     
                 <welcome-file>index.htm</welcome-file>     
                 <welcome-file>index.html</welcome-file>     
                 <welcome-file>default.htm</welcome-file>     
                 <welcome-file>default.html</welcome-file>     
  </welcome-file-list>    
  这时，如果你在url中键入<http://localhost:8080/myweb> (myweb是你的应用程序目录)，  
  系统会自动寻找welcome-file-list中列出的文件

经过这样的配置，就可以在url中输入http:\\localhost（或http:\\IP）来访问自己的网页了，

在Tomcat中使用数据库连接池(sqlserver2000)  
    在conf文件夹中，打开server.xml,找到</host>,在它前面加上下面的代码：  
    这样就生成了一个虚拟目录webshop，并且创建了连接池（注意修改相应的连接参数）  
    注意：请首先将sqlser2000的jdbc的三个文件（msbase.jarms，sqlserver.jar，msutil.jar）复制到tomcat\common\lib目录下  
      <Context path="webshop"docBase="e:/webshop" debug="0" reloadable="true"crossContext="true">     
          <Resourcename="jdbc/mssql" auth="Container"type="javax.sql.DataSource"/>     
            <ResourceParams name="jdbc/mssql">   
         <parameter>    
             <name>factory</name>    
                <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>    
              </parameter>      
    <parameter>    
                   <name>driverClassName</name>    
               <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>    
        </parameter>       
       <parameter>   
            <name>url</name>    
             <value>jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=wjw\_test</value>      
    </parameter>    
         <parameter>       <name>username</name>              <value>sa</value>    </parameter>  
      <parameter>        <name>password</name>               <value>8021</value>      </parameter>      
    <parameter>          <name>maxActive</name>              <value>20</value>    </parameter>  
     <parameter>           <name>maxIdle</name>     <value>10</value>     </parameter>   
    <parameter>          <name>maxWait</name>      <value>-1</value>           </parameter>    
           </ResourceParams>   
         </Context>

在jsp中使用连接池concool.jsp

 <!--测试数据源-->   
 <%@ page contentType="text/html; charset=gb2312" %>  
 <%@ page import="javax.naming.Context" %>   
 <%@ page import="javax.sql.DataSource"%>   
 <%@ page import="javax.naming.InitialContext"%>   
 <%@ page import="java.sql.\*"%>

 <%   
   DataSource ds = null;   
   try{    Context initCtx = new InitialContext();   
     Context envCtx = (Context) initCtx.lookup("java:comp/env");  
    //从Context中lookup数据源。  
     ds =(DataSource)envCtx.lookup("jdbc/mssql");   
 if(ds!=null)   
 { out.println("已经获得DataSource!");   
 out.println("<br>");  
 Connection conn = ds.getConnection();  
 Statement stmt=conn.createStatement();  
 ResultSet rst=stmt.executeQuery("select \* from book");  
 out.println("以下是从数据库中读取出来的数据");  
 while(rst.next())  
 { out.println("bookName:"+rst.getString("bookName"));  
 out.println("<br>");  } }  
 else   
  out.println("连接失败!");  }  
 catch(Exception ne)  
 {  out.println(ne); }  %>