第一章　Tomcat 概述

一、Tomcat 简介

1、Tomcat

Tomcat 在严格意义上并不是一个真正的应用服务器，它只是一个可以支持运行 Serlvet/JSP 的 Web 容器，不过 Tomcat 也扩展了一些应用服务器的功能，如 JNDI，数据库连接池，用户事务处理等等。Tomcat 是 Apache 组织下 Jakarta 项目下的一个子项目，目前 Tomcat 被非常广泛的应用在中小规模的 Java Web 应用中。

Tomcat 是一种具有 JSP 环境的 Servlet 容器。Servlet 容器是代替用户管理和调用 Servlet 的运行时外壳。作为一个开放源代码的软件， Jakarta -Tomcat 有着自己独特的优势：

- 首先，它容易得到。事实上，任何人都可以从互联网上自由地下载这个软件。无论从 http://jakarta.Apache.org 还是从其他网站（Jakarta Tomcat 是 Apache 软件基金会开发的一个开放源码的应用服务器）。
- 其次，对于开发人员，特别是 Java 开发人员，Tomcat 提供了全部的源代码，包括 Servlet 引擎、JSP 引擎、HTTP 服务器。无论是对哪一方面感兴趣的程序员，都可以从这些由世界顶尖的程序员书写的代码中获得收益。
- 最后，由于源代码的开放及世界上许多程序员的卓有成效的工作， Tomcat 已经可以和大部分的主流服务器一起工作，而且是以相当高的效率一起工作。如：以模块的形式被载入 Apache，以 ISAPI 形式被载入 IIS 或 PWS，以 NSAPI 的形式被载入 Netscape Enterprise Server。
- 由于 Java 的跨平台特性，基于 Java 的 Tomcat 也具有跨平台性。

2、Tomcat5.0 包含三个主要的部分

包括：

* Catalina - 一个符合 Servlet API 规范 2.3 的 Servlet Container
* Jasper - 一个符合 JSP 规范 1.2 的 JSP 编译器和运行环境
* Webapps - Tomcat 中包含的一些例子和用于测试的 web 例程，以及相关文档。

3、应用服务器（如 WebLogic）与 Tomcat 有何区别。

应用服务器提供更多的 J2EE 特征，如 EJB，JMS，JAAS 等，同时也支持 Jsp 和 Servlet。而 Tomcat 则功能没有那么强大，它不提供 EJB 等支持。但如果与 JBoss（一个开源的应用服务器）集成到一块，则可以实现 J2EE 的全部功能。

4、Tomcat 目录的结构

（1）Tomcat 的安装

其实对于完全由 Java 写成的 Tomcat，Win32 版本和 Linux 版本没有多大区别，比如 Linux 版本，在 Solaris 下也没有问题。这里，主要以 Win32 版本作为示例。

注意：在安装使用 Tomcat 之前，先安装 JDK，最好是 Sun 的 JDK 1 .2 以上版。

（2）Tomcat 的目录结构

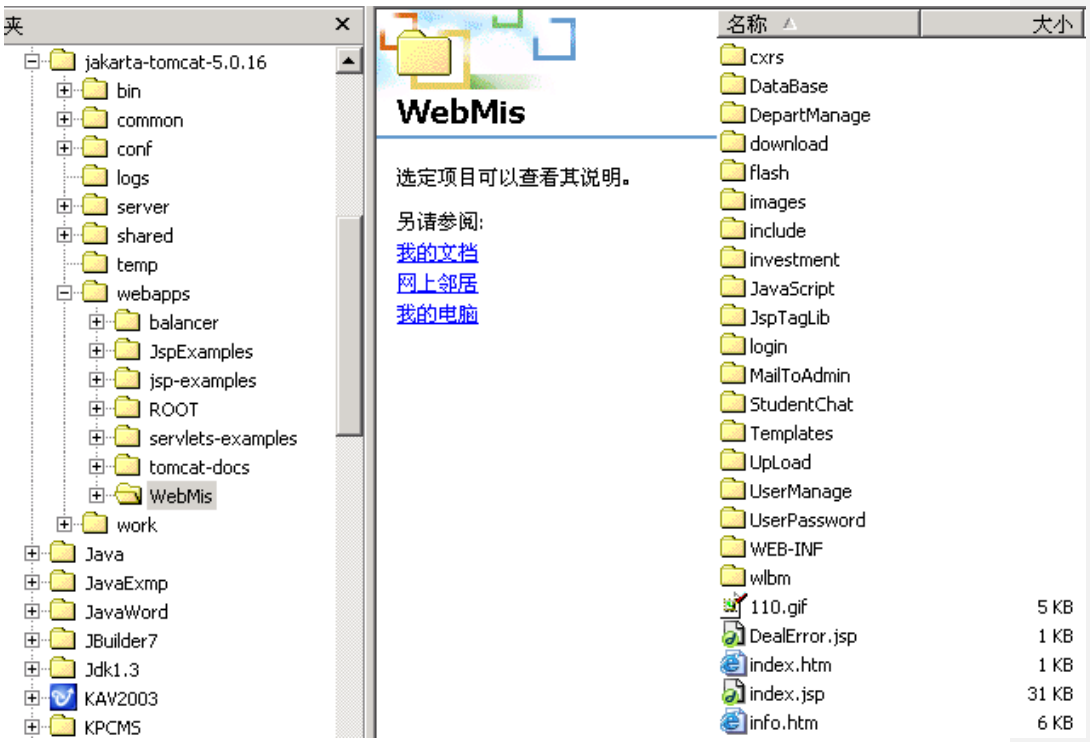
首先，下载 jakarta-tomcat.zip 包，解压缩到一个目录下，如:“c:\tomcat”。这时，会得到如下的 Tomcat 的目录结构：

```
- - - jakarta - tomcat
        | - - - bin              Tomcat执行脚本目录
        | - - - Common           放置一些通用类（如JDBC的驱动程序等）
        | - - - conf             Tomcat配置文件
        | - - - doc              Tomcat文档
        | - - - lib              Tomcat运行需要的库文件（JARS）
        | - - - logs             Tomcat执行时的LOG文件
        | - - - src              Tomcat的源代码
        | - - - webapps          Tomcat的主要Web发布目录（存放我们自己的JSP,SERVLET,
        类）
        | - - - work             Tomcat 的工作目录,Tomcat 将翻译 JSP 文件到的 Java 文件
                                和 class 文件放在这里。
```

目 录 名	该目录内的文件的一般功能描述
bin	包含有 Startup.bat（启动服务器）与 shutdown.bat（关闭服务器）文件
conf	包含设置部署在 Tomcat 上的 Web 应用的变量的初始值的设置文件，包括 *server.xml* (Tomcat 的全局配置文件) 和 *web.xml*（为不同的 Tomcat 配置的 web 应用设置缺省值的文件）
doc	包含关于 Tomcat 的各种各样的文档。
common	在其 lib 目录下，主要存放如 JDBC 的驱动程序等
lib	包含被 Tomcat 使用的各种各样的 jar 文件。在 UNIX 上，任何这个目录中的文件将被附加到 Tomcat 的 classpath 中。
logs	Tomcat 的 log 文件。
src	servlet API 的源文件。
webapps	包含 Web 应用的程序 （JSP、Servlet 和 JavaBean 等）
work	由 Tomcat 自动生成，这是 Tomcat 放置它运行期间的中间(intermediate)文件(诸如编译的 JSP 文件)地方。 如果当 Tomcat 运行时，你删除了这个目录那么将不能够执行包含 JSP 的页面。

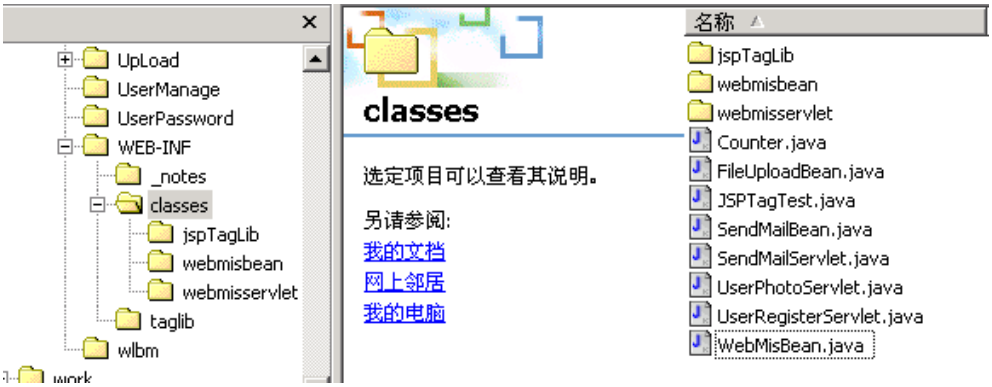
（3）、各个目录下所应该存放的文件：按照 Tomcat 的规范，Tomcat 的 Web 应用程序应该由如下目录组成

- 页面内容等文件的存放位置：*.html，*.jsp 等可以有许多目录层次，由用户的网站结构而定，实现的功能应该是网站的界面，也就是用户主要的可见部分。除了 HTML 文件、JSP 文件外，还有 js（JavaScript）文件和 css（样式表）文件以及其他多媒体文件等。

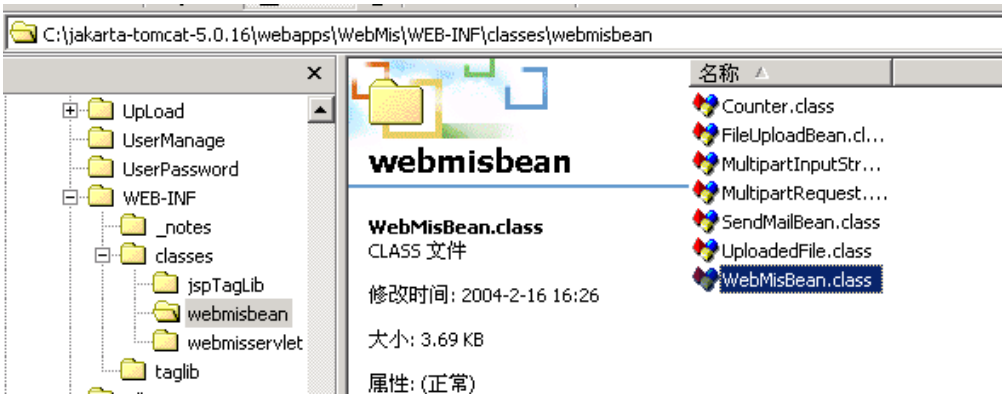


- Web-INF/web.xml 这是一个 Web 应用程序的描述文件。这个文件是一个 XML 文件，描述了 Servlet 和这个 Web 应用程序的其他组件信息，此外还包括一些初始化信息和安全约束等等。

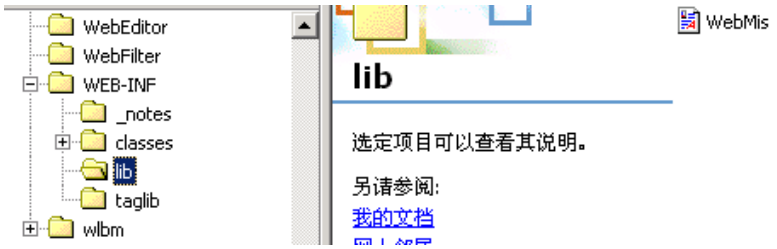
- Web-INF/classes/这个目录及其下的子目录应该包括这个 Web 应用程序的所有 JavaBean 及 Servlet 等编译好的 Java 类文件（*.class）文件，以及没有被压缩打入 JAR 包的其他 class 文件和相关资源。注意，在这个目录下的 Java 类应该按照其所属的包层次组织目录（即如果该*.class 文件具有包的定义，则该*.class 文件应该放在.\WEB-INF\classes\包名下）。




```
1  package webmisbean;
2  import java.sql.*;
3  public class WebMisBean
4  {
5      String sqlStatement;
6      Connection con=null;
7      PreparedStatement ps=null;
8      ResultSet rs=null;
9
10     public WebMisBean()
11        {
12 /*
13             try{
14
15                 Class.forName("com.microso
```



- 通常 Web-INF/classes/这个目录下的类文件也可以打包成 JAR 文件,并可以放到 WEB-INF 下的 lib 目录下。如将 classes 目录下的各个*.class 文件打包成 WebMis.jar 文件（jar cvf WebMis.jar *.*）



注意：

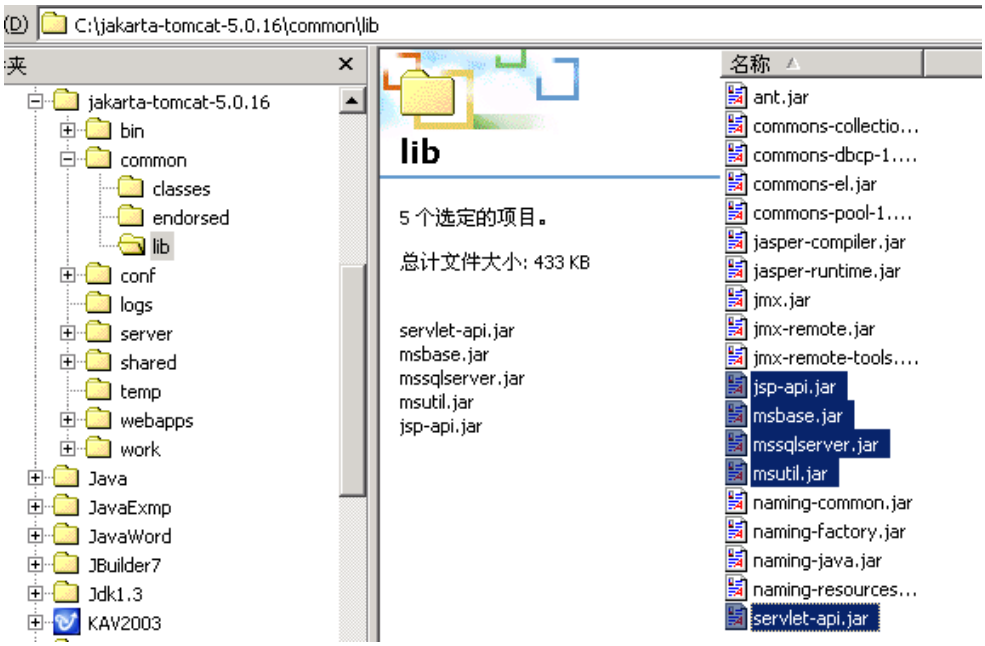
（1）WEB-INF 目录中包含应用软件所使用的资源，但是 WEB-INF 却不在公共文档根目录之中。在这个目录中所包含的文件都不能被客户机所访问。

（2）类目录中（在 WEB-INF 下）包含运行 Web 应用程序时所需的 Servlets，Beans 等类。

（3）lib 目录（在 WEB-INF 下）包含有 Java archive files (JARs)，例如标签库或者 Servlets，Beans 等类的*.jar 文件。

（4）如果一个类出现在 JAR 文件中同时也出现在类的目录中，类加载器会加载位于类目录中的那一个。

- common/lib/ 这个目录下包含了所有压缩到 JAR 文件中的类文件和相关文件。比如：第三方提供的 Java 库文件、JDBC 驱动程序等。
  - ✓ 其中 msbase.jar、mssqlserver.jar、msutil.jar 文件为 SqlServer2000 的 JDBC 驱动程序
  - ✓ 其中 servlet-api.jar 和 jsp-api.jar 为 Servlet 和 JSP 的 API 所在的包



二、Tomcat 的环境配置

1、启动 Tomcat

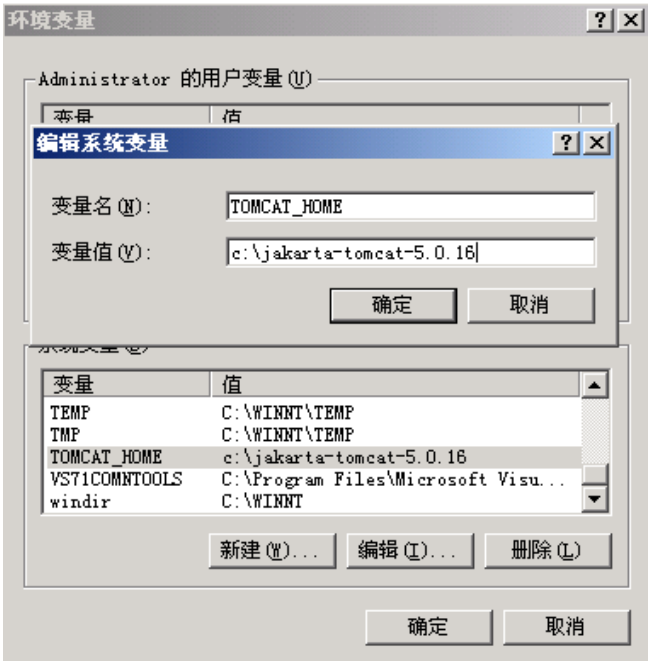
在 Bin 目录下,有一个名为 startup.bat 的脚本文件,执行这个脚本文件,就可以启动 Tomcat 服务器，不过，在启动服务器之前，还需要进行一些设置。

- **首先，设置系统的环境变量。**
  - ✓ **TOMCAT_HOME（或者：CATALINA_HOME）值：**

    d:\jakarta-tomcat-5.0.16（用 TOMCAT_HOME 指示 Tomcat 根目录，下面以 Tomcat 5.0.16 版为例)。
  - ✓ **JAVA_HOME 值：**

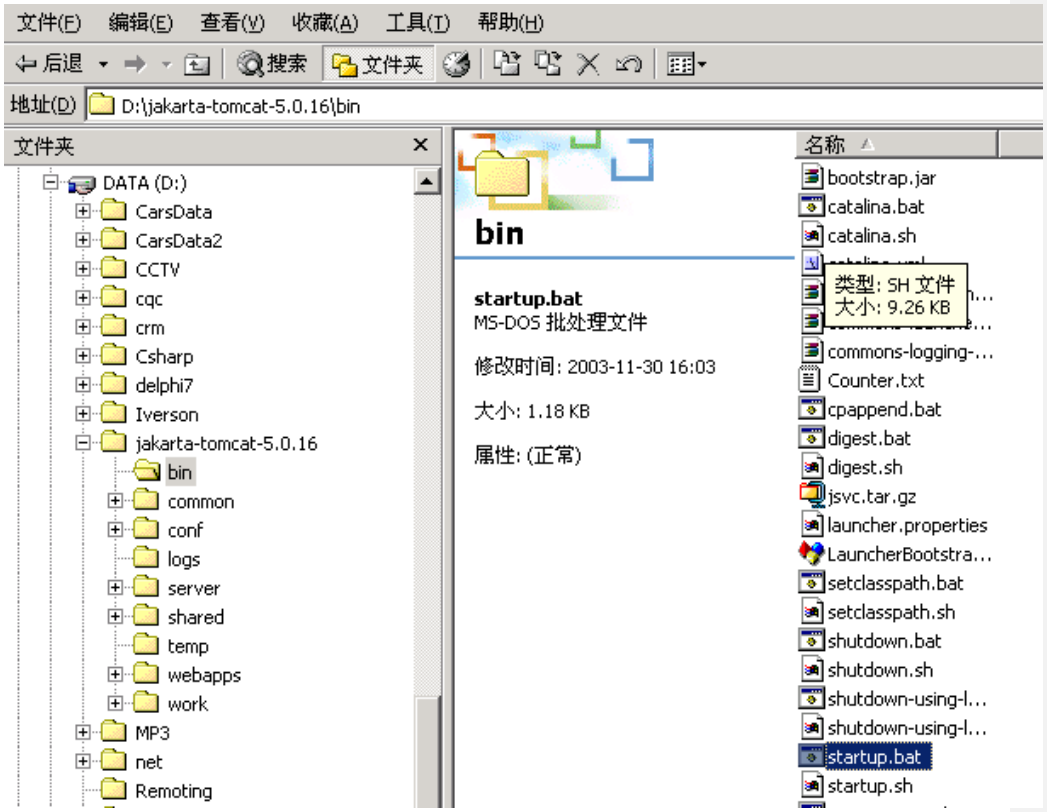
    c:\j2sdk1.4.0(用 JAVA_HOME 指示 jdk1.4 的安装目录)。

    注意：对于设置 Windows 的系统环境变量，可以打开控制面板中的“系统”程序；在“系统环境变量”中增加两个环境变量项目 JAVA_HOME（最好为大写）指向 JDK 的目录和 TOMCAT_HOME（最好为大写）指向所安装的 tomcat 的目录。

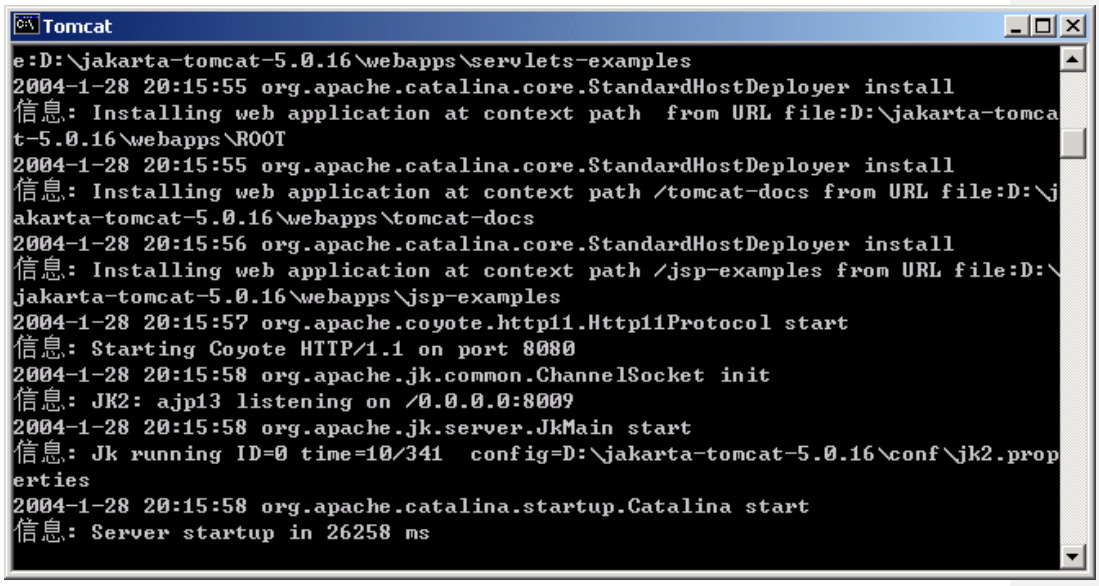


2、启动和关闭 Tomcat 服务器

（1)启动Tomcat服务器:执行在Bin目录下的名为startup.bat的脚本文件可以启动Tomcat服务器

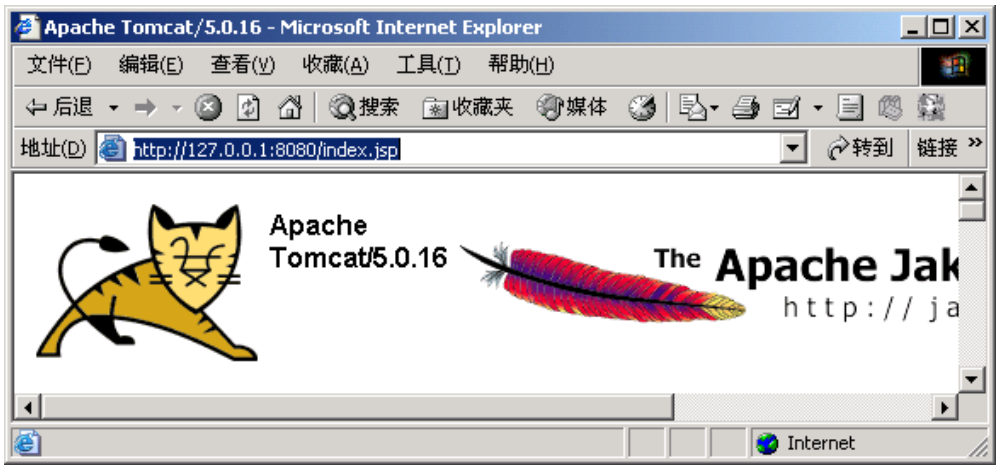


现在可以运行 TOMCAT 并作为一个独立的 Servlet 容器。



（2）测试Tomcat的服务器启动与否：

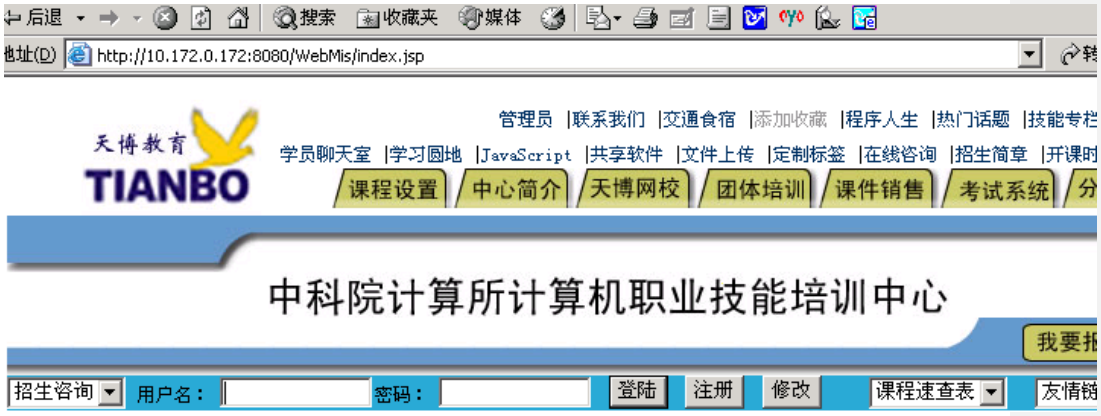
可以在浏览器中输入http://127.0.0.1:8080/index.jsp，是否出现如下内容。



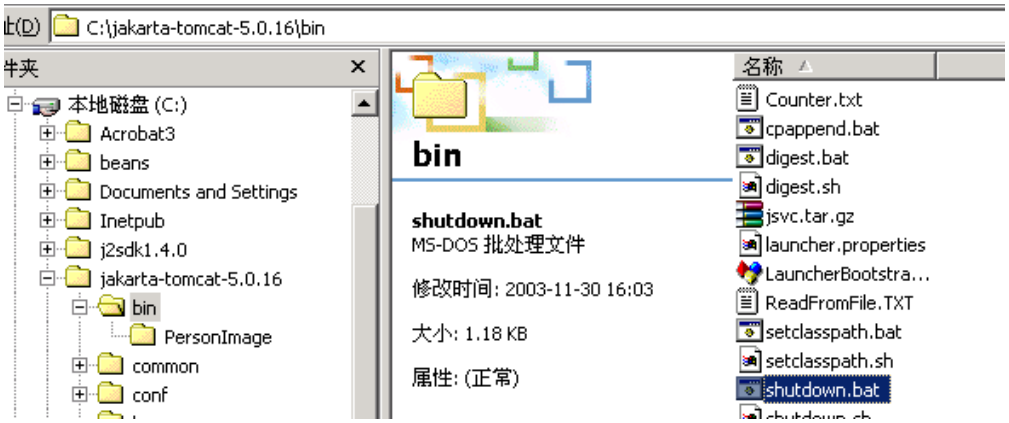
（3）启动本站点的JSP页面：在Tomcat中的JSP文件和JavaBean程序的存放位置

- ✓ JSP文件放在“Webapps\站点名称”的目录下
- ✓ 自定义的JavaBean程序*.java文件（可以不需要它）及*.class类文件存放在“Webapps\站点名称\ WEB-INF\classes\”目录下

因此，将*.jsp文件拷贝到“TOMCAT_HOME\Webapps\站点名称”目录下，然后输入其URL地址



（4）关闭 Tomcat 服务器：执行在 Bin 目录下的名为 shutdown.bat 的脚本文件可以终止 Tomcat 服务器。



三、配置 Tomcat 服务器

1、概述

Tomcat 为用户提供了一系列的配置文件来帮助用户配置自己的 Tomcat，Tomcat 的配置文件主要是基于 XML 的；如 server.xml、web.xml 等，下面将详细讨论 Tomcat 的主要配置文件以及如何利用这些配置文件解决常见问题。

2、server.xml 主配置文件

server.xml 是 Tomcat 的主配置文件，主要完成如下两个目标：

- ✓ 提供 Tomcat 组件的初始配置；
- ✓ 说明 Tomcat 的结构,含义,使得 Tomcat 通过实例化组件完成起动及构建自身。

观察 **server.xml**，可以发现其中有如下的一些元素。

（1）Server元素：

Server元素是**server.xml**文件的最高级别的元素， Server元素描述一个Tomcat服务器，一般来说用户不用关心这个元素。一个Server元素一般会包括Logger和ContextManager两个元素

- ✓ Logger：Logger元素定义了一个日志对象，一个日志对象包含有如下属性：
  1）name：表示这个日志对象的名称。
  2）path：表示这个日志对象包含的日志内容要输出到哪一个日志文件。
  3）verbosityLevel：表示这个日志文件记录的日志的级别。

一般来说，Logger对象是对Java Servlet、JSP和Tomcat运行期事件的记录

- ✓ ContextManager：ContextManager定义了一组ContextInterceptors（ContextManager的事件监听器） ，RequestInterceptors（的事件监听器）、Contexts（Web应用程序的上下文目录）和它们的Connectors（连接器）的结构和配置。ContextManager包含如下一些属性：
  1）debug：记录日志记录调试信息的等级。

2）home：webapps /、conf /、logs /和所有Context的根目录信息。这个属性的作用是从一个不同于TOMCAT _ HOME的目录启动Tomcat。

3）workDir：Tomcat工作目录。

ContextInterceptor 和RequestInterceptors两者都是监听ContextManager的特定事件的拦截器。ContextInterceptor监听Tomcat的启动和结束事件信息。而RequestInterceptors监听用户对服务器发出的请求信息。一般用户无需关心这些拦截器，对于开发人员需要了解这就是全局性的操作得以实现的方法

（2）Connector元素：

Connector（连接器）元素描述了一个到用户的连接，不管是直接由Tomcat到用户的浏览器还是通过一个Web服务器。Tomcat的工作进程和由不同的用户建立的连接传来的读/写信息和请求/答复信息都是由连接器对象管理的。对连接器对象的配置中应当包含管理类、TCP/IP端口等内容。

（3）Context元素：

每一个Context都描述了一个Tomcat的Web应用程序的目录。这个对象包含以下属性：

1)docBase。这是Context的目录。可以是绝对目录也可以是基于ContextManage的根目录的相对目录。

2)path。这是Context在Web服务时的虚拟目录位置和目录名。

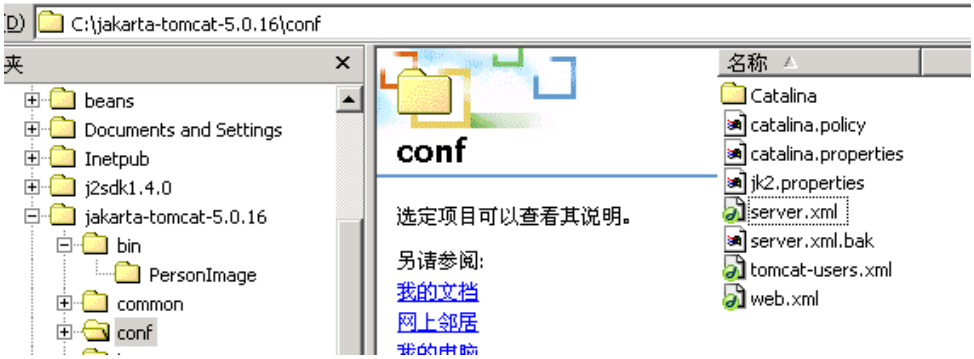
3)debug。日志记录的调试信息记录等级。

4)reloadable。这是为了方便 Servlet 的开发人员而设置的，当这个属性开关打开的时候，Tomcat 将检查 Servlet 是否被更新而决定是否自动重新载入它

3、**配置实例**：打开 Tomcat 下的 conf 文件夹下的 server.xml 文件

（1）改变 Tomcat 服务器的端口号

需要使用 Connector 元素，Connector 表示一个到用户的联接,不管是通过 web 服务器或直接到用户浏览器(在一个独立配置中)。Connector 负责管理 Tomcat 的工作线程和读/写连接到不同用户的端口的请求/响应。Connector 的配置包含如下信息：句柄类、句柄监听的 TCP/IP 端口、句柄服务器端口的 TCP/IP 的 backlog。修改后，必须重新启动 Tomcat 的服务器。



```
54      <!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
55      <Connector className="org.apache.catalina.connector.http.HttpConnector"
56                 port="8080" minProcessors="5" maxProcessors="75"
57                  enableLookups="true" redirectPort="8443"
58                  acceptCount="10" debug="0" connectionTimeout="60000"/>
59      <!-- Note : To disable connection timeouts, set connectionTimeout value
60       to -1 -->
```

注意：可以将端口号改变为 80，单要保证 80 端口没有被占用；另外，也可以同时分配两个端口号，只要产生两个 Connector 的配置信息。

```
    <!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
   <Connector    port="8080"              maxThreads="150"    minSpareThreads="25"
maxSpareThreads="75"
     enableLookups="false"   redirectPort="8443"   acceptCount="100"      debug="0"
connectionTimeout="20000"  disableUploadTimeout="true" />

    <!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8000 -->
    <Connector    port="8000"              maxThreads="150"    minSpareThreads="25"
maxSpareThreads="75"
               enableLookups="false"     redirectPort="8443"     acceptCount="100"
debug="0" connectionTimeout="20000"  disableUploadTimeout="true" />
```

（2）增加新的虚拟目录并指向物理目录

设立一个虚拟工作目录是比较简单的，只需要在 server.xml 文件中添加一个 Context 对象就可以了。如，要在 webapps\下增加一个 WebMis 文件夹以存放 jsp 页面文件，并且让用户可以使用 http://127.0.0.1:8080/WebMis 虚拟目录访问，则：需要使用 Context 元素，每个 Context 提供一个指向你放置你 Web 项目的 Tomcat 的下属目录。每个 Context 包含如下配置：

- Context 放置的路径，可以是与 ContextManager 主目录相关的路径；
- 纪录调试信息的调试级别；
- 可重载的标志，开发 Servlet 时，重载更改后的 Servlet。这是一个非常便利的特性,你可以调试或用 Tomcat 测试新代码而不用停止或重新启动 Tomcat。要打开重载,把 reloadable 设为 true 即可。

```
206         <!-- Tomcat Examples Context -->
207               <Context path="/WebMis" docBase="WebMis" debug="0" reloadable="true">
208               </Context>
209
```

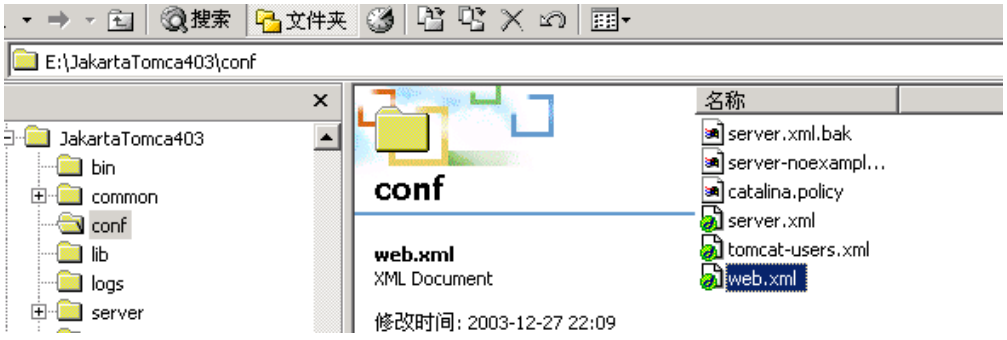
其中：path="/WebMis"说明其相对web URL的路径，是一个虚拟的路径，如：http://127.0.0.1:8080/WebMis，docBase="WebMis"说明其相对webapps的位置，是物理存在的目录，同时需要在webapps\下增加一个WebMis物理文件夹。


```
206         <!-- Tomcat Examples Context -->
207                 <Context path="/WebMis" docBase="WebMis" debug="0" reloadable="true">
208                 </Context>
209
```

（3） 加入自己的日志文件

添加 Logger 对象就可以加入自己的日志文件，添加工作相当简单，只需要将作为示例的 Logger 对象复制一份，然后修改一下前面介绍的几个属性就可以了。在设定了 Logger 以后，就可以在自己的 Servlet 中使用 ServletContext.log()方法来建立自己的日志文件。

4、**配置实例**：打开 conf 文件夹下的 web.xml 文件



（1）web.xml 文件：它包含了描述整个 Web 应用程序（Web 应用程序由一整套 Web 文件 jsp、servlet、html、jpg、gif、class 等组成）的信息。下面以一个 web.xml 文件为例，讲解里面的各个对象。

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
    <display-name>My Web Application</display-name>
    <description>在这里加入 Web 应用程序的描述信息</description>
    <!-
    下面定义了 Web 应用程序的初始化参数，在 JSP 或 Servlet 文件中使用下面的语句
    来得到初始化参数
        String value =
        getServletContext().getInitParameter("name");
        这里可以定义任意多的初始化参数
    -->
    <context-param>
        <param-name>webmaster</param-name>
        <param-value>myaddress@mycompany.com</param-value>
        <description>这里包含了初始化参数的描述</description>
    </context-param>
    <!-
    下面的定义描述了组成这个 Web 应用程序的 Servlet，还包含初始化参数。在 Tomcat 中，
    也可以将放在 Web-INF/classes 中的 Servlet 直接以 servlet/Servlet 名访问，但是一般来
    说，不推荐这样使用。而且这样的使用方法还会导致 Servlet 的相关资源组织的复杂性。所
    以一般来说推荐将所有的 Servlet 在这里定义出来。初始化参数可以在 Servlet 中使用如下
    语句来获得：
        String value =getServletConfig().getInitParameter("name");
    -->
    <servlet>
        <servlet-name>controller</servlet-name>
        <description>这里加入这个 Servlet 的描述</description>
        <servlet-class>com.mycompany.mypackage.ControllerServlet</servlet-class>
        <init-param>
            <param-name>listOrders</paramName>
            <param-value>com.mycompany.myactions.ListOrdersAction</param-value>
        </init-param>
        <init-param>
            <param-name>saveCustomer</paramName>
            <param-value>com.mycompany.myactions.SaveCustomerAction</param-value>
        </init-param>
    <!-
        服务器启动后这个 Servlet 加载的时间
    -->
        <load-on-startup>5</load-on-startup>
    </servlet>
    <servlet>
```

```
        <servlet-name>graph</servlet-name>
        <description>这个 Servlet 的描述</description>
    </servlet>
    <!-
        Servlet 映射对应了一个特殊的 URI 请求到一个特殊的 Servlet 的关系
    -->
    <servlet-mapping>
        <servlet-name>controller</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
        <servlet-mapping>
        <servlet-name>graph</servlet-name>
        <url-pattern>/graph</url-pattern>
    </servlet-mapping>
    <!-
        设定缺省的 Session 过期时间（单位为分）?单位为分？
    -->
        <session-config>
            <session-timeout>30</session-timeout>
        </session-config>
</web-app>
```

（2）配置实例：会话(session)超时修改，修改 conf\web.xml 中的如下数据值（单位为分）

```
295
296     <!-- ==================== Default Session Configuration ================ -->
297
298     <!-- You can set the default session timeout (in minutes) for all newly   -->
299     <!-- created sessions by modifying the value below.                       -->
300
301     <session-config>
302       <session-timeout>30</session-timeout>
303     </session-config>
304
```

5、在 Tomcat 中实现利用 JDBC 驱动程序访问 SQLServer2000 数据库

只需要将 SQLServer2000 的 JDBC 驱动程序的三个*.jar（msbase.jar、mssqlserver.jar 和 msutil.jar）文件放在\common\lib 目录下，然后在*.java 程序中访问它。

四、在 Tomcat5 中配置连接池和数据源

1、DataSource 接口介绍

（1）DataSource 概述

JDBC1.0 原来是用 DriverManager 类来产生一个对数据源的连接。JDBC2.0 用一种替代的方法，使用 DataSource 的实现，代码变的更小巧精致，也更容易控制。

一个 DataSource 对象代表了一个真正的数据源。根据 DataSource 的实现方法，数据源既可以是从关系数据库，也电子表格，还可以是一个表格形式的文件。当一个 DataSource 对象注册到名字服务中（JNDI），应用程序就可以通过名字服务获得 DataSource 对象，并用它来产生一个与 DataSource 代表的数据源之间的连接。

javax.sql 包中的 DataSource 接口，可以采用三种实现形式：简单的实现（只提供 Connection 对象）、连接池形式的实现和分布式事务形式的实现。

javax.sql 包中的 ConnectionPoolDataSource 提供对连接池实现的接口。

（2）使用 DataSource 的优点

- DataSource 与 DriverManager 的不同

关于数据源的信息和如何来定位数据源，例如数据库服务器的名字，在哪台机器上，端口号等等，都包含在 DataSource 对象的属性里面去了。这样，对应用程序的设计来说是更方便了，因为并不需要硬性的把驱动的名字写死到程序里面去。通常驱动名字中都包含了驱动提供商的名字，而在 DriverManager 类中通常是这么做的。

- 可移植性

如果数据源要移植到另一个数据库驱动中，代码也很容易做修改。所需要做的修改只是更改 DataSource 的相关的属性。而使用 DataSource 对象的代码不需要做任何改动。

（3）配置 DataSource

主要包括设定 DataSource 的属性，然后将它注册到 JNDI 名字服务中去。在注册 DataSource 对象的的过程中，系统管理员需要把 DataSource 对象和一个逻辑名字关联起来。名字可以是任意的，通常取成能代表数据源并且容易记住的名字。

在下面的例子中，名字起为：WebMisDB，按照惯例，逻辑名字通常都在 jdbc 的子上下文中。这样，逻辑名字的全名就是：jdbc/WebMisDB。

（4）产生一个与数据源的连接

一旦配置好了数据源对象，应用程序设计者就可以用它来产生一个与数据源的连接。下面的

代码片段示例了如何用 JNDI 上下文获得一个数据源对象，然后如何用数据源对象产生一个与数据源的连接。开始的两行用的是 JNDI API，第三行用的才是 JDBC 的 API：

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbc/WebMisDB");
Connection con = ds.getConnection("myPassword", "myUserName");
```

在一个基本的 DataSource 实现中，DataSource.getConnection 方法返回的 Connection 对象和用 DriverManager.getConnection 方法返回的 Connection 对象是一样的。因为 DataSource 提供的方便性，我们推荐使用 DataSource 对象来得到一个 Connection 对象。

（5）DataSource 的应用场合

对于普通的应用程序设计者，是否使用 DataSource 对象只是一个选择问题。但是，对于那些需要用的连接池或者分布式的事务的应用程序设计者来说，就必须使用 DataSource 对象来获得 Connection。需要注意的是对 Tomcat 而言，在 JNDI 的名称前面应该加上"java:comp/env/" ？其他的服务器不需要加？

（6）数据源（DataSource）的作用

它相当于客户端程序和连接池的中介，想要获得连接池中的连接对象，必须建立一个与该连接池相应的数据源，然后通过该数据源获得连接。

2、JNDI（JAVA NAMING AND DIRECTORY INTERFACE---Java 命名和目录接口）

（1）JNDI 简介

分布式计算环境通常使用命名和目录服务来获取共享的组件和资源。命名和目录服务将名称与位置、服务、信息和资源关联起来。它是一个为 JAVA 应用程序提供命名服务的应用程序编程接口（API）。

命名服务提供了一种为对象命名的机制，这样你就可以在无需知道对象位置的情况下获取和使用对象。只要该对象在命名服务器上注册过，且你必须知道命名服务器的地址和该对象在命名服务器上注册的 JNDI 名。就可以找到该对象，获得其引用，从而运用它提供的服务。

命名服务提供名称—对象的映射。目录服务提供有关对象的信息，并提供定位这些对象所需的搜索工具。

Java 命名和目录接口或 JNDI 提供了一个用于访问不同的命名和目录服务的公共接口（JAVA API）。运用一个命名服务来查找与一个特定名字相关的一个对象，JDBC 可以用 JNDI 来访问一个关系数据库。

（2）获得 JNDI 的初始环境

在 JNDI 中，**在目录结构中的每一个结点称为 Context 。**每一个 JNDI 名字都是相对于 Context 的。这里没有绝对名字的概念存在。对一个应用来说，它可以通过使用 InitialContext 类来得到其第一个 Context：

```
Context ctx = new InitialContext ();
```

应用可以通过这个初始化的 Context 经由这个目录树来定位它所需要的资源或对象。InitialContext 在网页应用程序初始化时被设置，用来支持网页应用程序组件。所有的入口和资源都放在 JNDI 命名空间里的 java:comp/env 段里。

（3）查找已绑定的对象

用 ctx..lookup(String name); 根据 name 找对象

例：

```
import javax.naming.*;
public class TestJNDI
{
    public static void main(String[] args)
```

```
    {
        try
        {
            Context ctx=new InitialContext();
            Object object=ctx.lookup("JNDIName");      //根据 JNDI 名查找绑定的对象
            String str=(String) object;                //强制转换
        }
        catch(NamingException e)
        {     e.printStackTrace();
        }
        catch(ClassCastException e)
        {     e.printStackTrace();
        }
    }
}
```

3、数据库连接池技术

（1）传统的 Web 数据库编程模式

- 在主程序（如 Servlet、Beans）中建立数据库连接。
- 进行 SQL 操作，取出数据。
- 断开数据库连接。

使用这种模式开发，存在很多问题。

- 首先，我们要为每一次 WEB 请求（例如察看某一篇文章的内容）建立一次数据库连接，对于一次或几次操作来讲，或许你觉察不到系统的开销，但是，对于 WEB 程序来讲，即使在某一较短的时间段内，其操作请求数也远远不是一两次，而是数十上百次（想想全世界的网友都有可能在您的网页上查找资料），在这种情况下，系统开销是相当大的。事实上，在一个基于数据库的 WEB 系统中，建立数据库连接的操作将是系统中代价最大的操作之一。很多时候，可能您的网站速度瓶颈就在于此。
- 其次，使用传统的模式，你必须去管理每一个连接，确保他们能被正确关闭，如果出现程序异常而导致某些连接未能关闭，将导致数据库系统中的内存泄露，最终我们将不得不重启数据库。
- 频繁的建立、关闭连接，会极大的减低系统的性能，因为对于连接的使用成了系统性能的瓶颈。

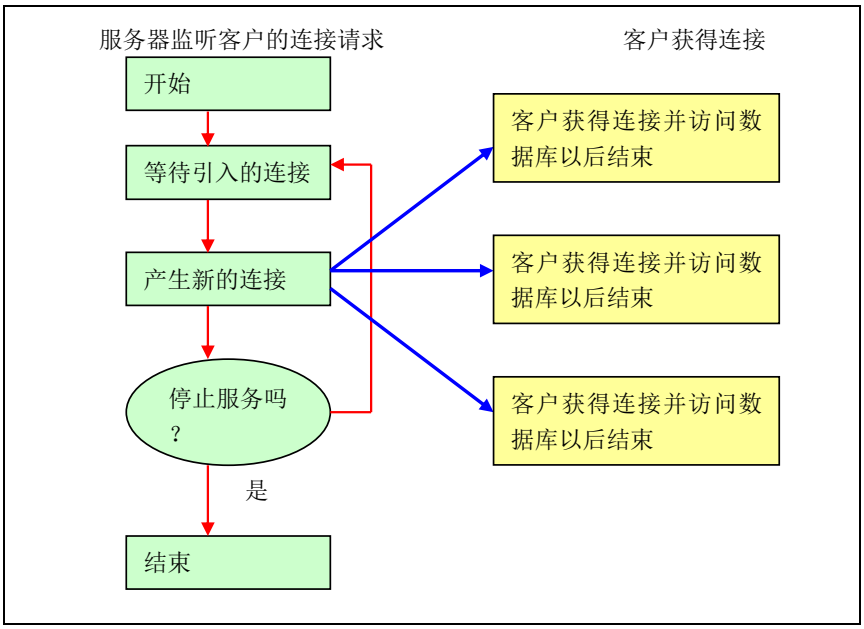
（2）数据库连接是一种关键的有限的昂贵的资源

这一点在多用户的网页应用程序中体现得尤为突出。对数据库连接的管理能显著影响到整个应用程序的伸缩性和健壮性，影响到程序的性能指标。数据库连接池正是针对这个问题提出来的。

连接池是这么一种机制，当应用程序关闭一个 Connection 的时候，这个连接被回收，而不是被 destroy，因为建立一个连接是一个很费资源的操作。如果能把回收的连接重新利用，会减少新创建连接的数目，显著的提高运行的性能。该策略的核心思想是：连接复用。

通过采用连接池的方法，服务器在启动时先打开一定数量的连接。 当应用需要连接时，就

可以从服务器请求一个连接。当应用结束该连接时，服务器就把它释放到连接池，以备其他客户机使用。

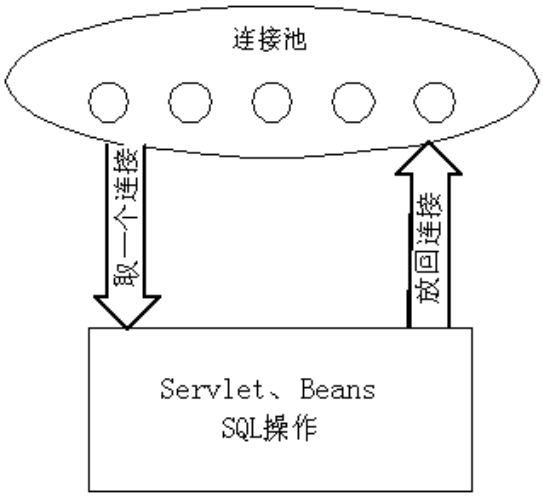


（3）连接池的主要作用

- 减少了建立和释放数据库连接的消耗
- 数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而再不是重新建立一个；
- 释放空闲时间超过最大空闲时间的数据库连接来避免因为没有释放数据库连接而引起的数据库连接遗漏。这项技术能明显提高对数据库操作的性能。
- 封装用户信息　使用连接池可以封装连接数据库系统所用的用户信息（帐号和密码），**这样客户端程序在建立连接时不用考虑安全信息。**

（4）数据库连接池的工作原理

当程序中需要建立数据库连接时，只须从内存中取一个来用而不用新建。同样，使用完毕后，只需放回内存即可。而连接的建立、断开都有连接池自身来管理。同时，我们还可以通过设置连接池的参数来控制连接池中的连接数、每个连接的最大使用次数等等



（5）数据库连接池的最小连接数和最大连接数

数据库连接池的最小连接数和最大连接数的设置要考虑到下列几个因素：

- 最小连接数是连接池一直保持的数据库连接，所以如果应用程序对数据库连接的使用量不大，将会有大量的数据库连接资源被浪费；
- 最大连接数是连接池能申请的最大连接数，如果数据库连接请求超过此数，后面的数据库连接请求将被加入到等待队列中，这会影响之后的数据库操作。

如果最小连接数与最大连接数相差太大，那么最先的连接请求将会获利，之后超过最小连接数量的连接请求等价于建立一个新的数据库连接。不过，这些大于最小连接数的数据库连接在使用完不会马上被释放，它将被放到连接池中等待重复使用或是空闲超时后被释放。

（6）使用连接池得到连接

假设应用程序需要建立到一个名字为 EmpolyeeDB 的 DataSource 的连接。使用连接池得到连接的代码如下：

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbc/EmployeeDB");
Connection con = ds.getConnection("myPassword", "myUserName");
```

或者：

```
Context ctx = new InitialContext();
ConnectionPoolDataSource
ds = (ConnectionPoolDataSource)ctx.lookup("jdbc/EmployeeDB");
PooledConnection con = ds.getConnection("myPassword", "myUserName");
```

是否使用连接池获得一个连接，在应用程序的代码上是看不出不同的。在使用这个 Connection 连接上也没有什么不一样的地方，唯一的不同是在 java 的 finally 语句块中来关闭一个连接。在 finally 中关闭连接是一个好的编程习惯。这样，即使方法抛出异常，Connection 也会被关闭并回收到连接池中去。代码应该如下所示：

```
try
{…
}
catch ()
```

```
{…
}
finally
{
    if (con!=null)
        con.close();
  }
```

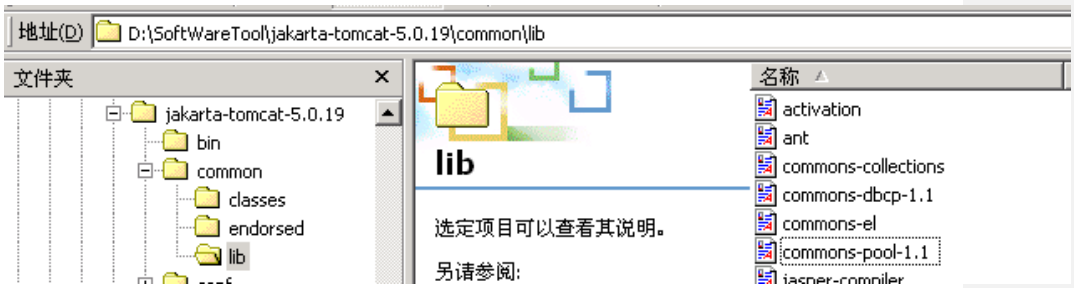
4、在 Tomcat 中配置数据库的连接池

（1）连接池配置(Database Connection Pool (DBCP) Configurations)

DBCP 使用的是 Jakarta-Commons Database Connection Pool 要使用连接池需要如下的组件即 jar 文件。

- Jakarta-Commons DBCP 1.1 对应 commons-dbcp-1.1.jar。
- Jakarta-Commons Collections 2.0 对应 commons-collections.jar。
- Jakarta-Commons Pool 1.1 对应 commons-pool-1.1.jar。

这三个 jar 文件要与你的 JDBC 驱动程序一起放到【TOMCAT_HOME】\common\lib 目录下以便让 tomcat 和你的 web 应用都能够找到。



注：

- 这三个 jar 文件是默认存在与【TOMCAT_HOME】\common\lib 下的。
- 需要注意的地方：第三方的驱动程序或者其他类只能以*.jar 的形式放到 Tomcat 的 common\lib 目录中,因为 Tomcat 只把*.jar 文件加到 CLASSPATH 中。
- 不要把上诉三个文件放到 WEB-INF/lib 或者其他地方因为这样会引起混淆。

（2）通过配置阻止连接池漏洞

数据库连接池创建和管理连接池中建立好的数据库连接，循环使用这些连接以得到更好的效率。这样比始终为一个用户保持一个连接和为用户的请求频繁的建立和销毁数据库连接要高效的多。

这样就有一个问题出现了，一个 Web 应用程序必须显示的释放 ResultSet，Statement 和 Connection。如果在关闭这些资源的过程中失败将导致这些资源永远不在可用，这就是所谓的连接池漏洞。这个漏洞最终会导致连接池中所有的连接不可用。

通过配置 Jakarta Common DBCP 可以跟踪和恢复那些被遗弃的数据库连接。

以下是一系列相关配置：

- 通过配置 DBCP 数据源中的参数 removeAbandoned 来保证删除被遗弃的连接使其可以被重新利用。

为 ResourceParams(见下文的数据源配置)标签添加参数 removeAbandoned

```
<parameter>
    <name>removeAbandoned</name>
    <value>true</value>
</parameter>
```

通过这样配置的以后当连接池中的有效连接接近用完时 DBCP 将试图恢复和重用被遗弃的连接。这个参数的值默认是 false。

- 通过设置 removeAbandonedTimeout 来设置被遗弃的连接的超时的时间，即当一个连接连接被遗弃的时间超过设置的时间时那么它会自动转换成可利用的连接。

```
<parameter>
     <name>removeAbandonedTimeout</name>
     <value>60</value>
 </parameter>
```

默认的超时时间是 300 秒。

- 设置 logAbandoned 参数，以将被遗弃的数据库连接的回收记入日志中

```
<parameter>
    <name>logAbandoned</name>
    <value>true</value>
</parameter>
```

这个参数默认为 false。

(3) 修改 server.xml 文件

```
<Context path="/WebMis" docBase="WebMis" debug="0" reloadable="true">

    <Resource name="jdbc/webmis" auth="Container" type="javax.sql.DataSource"/>
    <ResourceParams name="jdbc/webmis">
        <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>
```

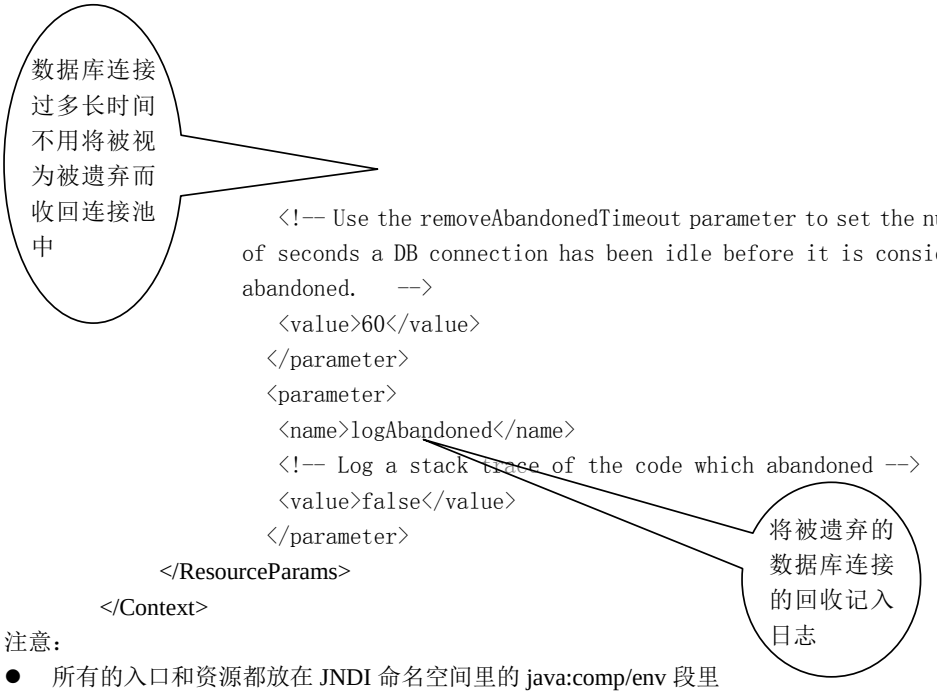
<value>**jdbc:microsoft:sqlserver://127.0.0.1:1433;DatabaseName=DataBase**
</value>
</parameter>
<parameter>
<name>
username
</name>
<value>
sa
</value>
</parameter>
<parameter>
<name>
password
</name>
<value>
</value>
</parameter>
<parameter>
<name>
maxActive
</name>
<value>
20
</value>
</parameter>
<parameter>
<name>
maxIdle
</name>
<value>10</value>
</parameter>
<parameter>
<name>maxWait</name>
<value>-1</value>
</parameter>
<parameter>
<name>removeAbandoned</name>
<!-- Abandoned DB connections are removed and recycled -->
<value>true</value>
</parameter>
<parameter>
<name>removeAbandonedTimeout</name>

maxActive 连接池的最大数据库连接数。设为 0 表示无限制。

maxIdle 数据库连接的最大空闲时间。超过此空闲时间，数据库连接将被标记为不可用，然后被释放。设为 0 表示无限制。

maxWait 最大建立连接等待时间。如果超过此时间将接到异常。设为－1 表示无限制。

回收被遗弃的（一般是忘了释放的）数据库连接到连接池中，设为－1 表示无限制。



注意：

- 所有的入口和资源都放在 JNDI 命名空间里的 java:comp/env 段里
- 设置 JNDI 资源要在$CATALINA_HOME/conf/server.xml 文件里使用下列标志符：
  1) <Resource>--设置应用程序可用的资源的名字和类型（同上面说的<resource-ref>等价）。
  2) <ResourceParams>--设置 Java 资源类工厂的名称或将用的 JavaBean 属性。
  上述这些标志符必须放在<Context>和</Context>之间

（2）、拷贝 SQLServer 的 JDBC 驱动程序到 Tomcat 的\common\lib 目录下

（3）、在程序中利用数据源来访问数据库

```
try
{
    Context initCtx = new InitialContext();
    Context envCtx = (Context) initCtx.lookup("java:comp/env");
    DataSource ds = (DataSource)envCtx.lookup("jdbc/webmis");
    Connection con=ds.getConnection();
}
catch (NamingException e)
{
    e.printStackTrace();
}
catch (SQLException e)
{
    e.printStackTrace();
}
```

5、在 server.xml 文件中与数据源的描述相关的标签含义

- maxActive 连接池的最大数据库连接数。设为 0 表示无限制。
- maxIdle 数据库连接的最大空闲时间。超过此空闲时间，数据库连接将被标记为不可用，然后被释放。设为 0 表示无限制。
- maxWait 最大建立连接等待时间。如果超过此时间将接到异常。设为－1 表示无限制。
- removeAbandoned 回收被遗弃的（一般是忘了释放的）数据库连接到连接池中。

- removeAbandonedTimeout 数据库连接过多长时间不用将被视为被遗弃而收回连接池中。
- logAbandoned 将被遗弃的数据库连接的回收记入日志。
- driverClassName JDBC 驱动程序。
- url 数据库 DSN 连接字符串

6、在 Web 应用的 web.xml 文件中引用该资源

将下面的标签放在放在<web-app>和</web-app>中间

```
<!-- Database Config start -->
<resource-ref>
<description>connectDB test</description>
<res-ref-name>jdbc/webmis</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
<!-- Database Config end -->
```

4，综合配置实例

首先在 C:根目录下建立文件夹 mywebapp，作为一个虚拟目录的位置。

建立一个 Sql Server 数据库 DataBonus

找到 C:\jakarta-tomcat-5.0.19\conf\server.xml，打开。

加入：的位置？

```
<Context path="/mywebapp" docBase="C:/mywebapp" debug="0" reloadable="true">

    <Resource name="jdbc/mybonusds" auth="Container" type="javax.sql.DataSource"/>
    <ResourceParams name="jdbc/mybonusds">
        <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://127.0.0.1:1433;DatabaseName=DataBonus
```

```
                </value>
            </parameter>
            <parameter>
              <name>username</name>
              <value>sa</value>
             </parameter>
             <parameter>
                <name>password</name>
                <value></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>
             <parameter>
                <name>removeAbandoned</name>
                <!-- Abandoned DB connections are removed and recycled -->
                <value>true</value>
             </parameter>
             <parameter>
                 <name>removeAbandonedTimeout</name>
   <!-- Use the removeAbandonedTimeout parameter to set the number of seconds a DB connection has been idle before it is considered abandoned.    -->
                 <value>60</value>
              </parameter>
              <parameter>
                 <name>logAbandoned</name>
               <!-- Log a stack trace of the code which abandoned -->
                 <value>false</value>
              </parameter>
        </ResourceParams>
</Context>
```

做一个 JSP 页面 index.jsp 放到 mywebapp 下面，代码：

```
<%--字符集设为"gb2312",使动态页面支持中文--%>
```

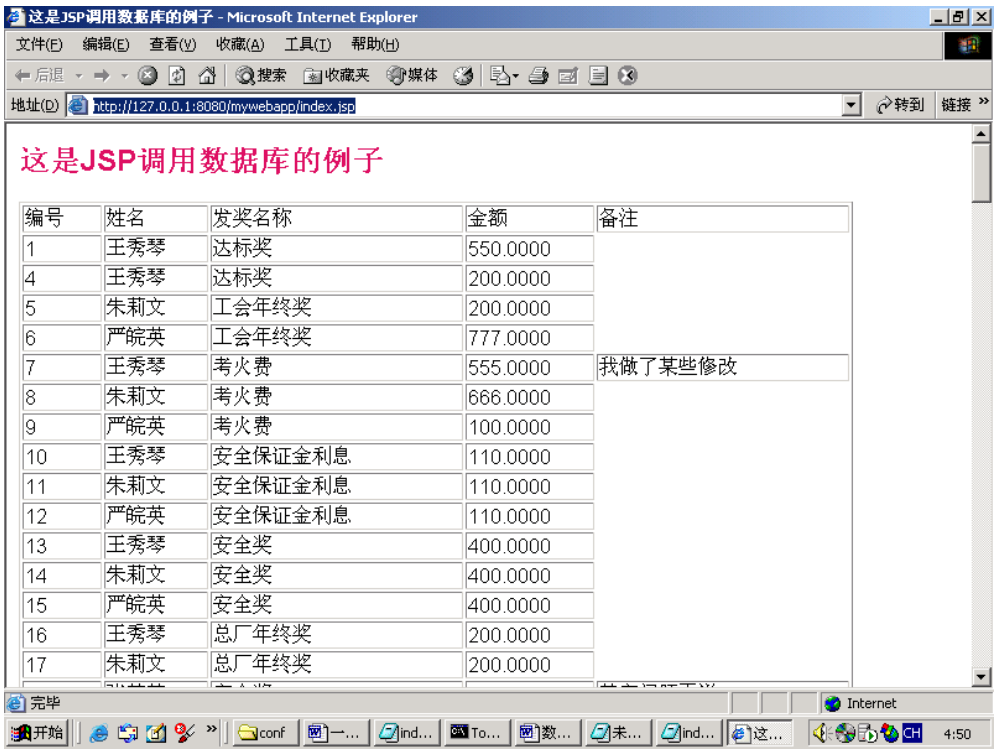
```
<%@ page contentType="text/html; charset=GB2312"%>

<!-- 这里使用一个字串变量 ("PAGETITLE") 保持题目和主标题的一致性。-->
<html>
<head>
<title>
<%= pagetitle %>
</title>
</head>

<body bgcolor=#FFFFFF>

<font face="Helvetica">

<h2>
<font color=#DB1260>
<%= pagetitle %>
</font>
</h2>

<!-- 导入必要的类和类库 -->

<%@ page import="
      javax.naming.*,
      java.sql.*,
      javax.sql.DataSource
"%>

<!-- 声明一个类方法 -->

<%!
//声明变量
//标题
   String pagetitle = "这是 JSP 调用数据库的例子";

%>

<!-- 下面这些代码将被插入到 servlet 中 -->

<%

     java.sql.Connection conn= null;
     java.sql.Statement stmt =null;
     java.sql.ResultSet rs=null;
```

```
try {
  // 通过 JNDI 获取主接口

    Context initCtx = new InitialContext();
    Context envCtx = (Context) initCtx.lookup("java:comp/env");
    DataSource ds = (DataSource)envCtx.lookup("jdbc/mybonusds");
    conn=ds.getConnection();

     stmt = conn.createStatement();


       //执行 SQL 语句
     stmt.execute("select * from  奖金");
  //取得结果集
     rs = stmt.getResultSet();

  %>

<table border="1">
 <tr>
     <td width="60" height="20"><% out.print("编号"); %></td>
     <td width="80" height="20"><% out.print("姓名"); %></td>
     <td width="200" height="20"><% out.print("发奖名称"); %></td>
     <td width="100" height="20"><% out.print("金额"); %></td>
     <td width="200" height="20"><% out.print("备注"); %></td>
 </tr>
<%     while (rs.next()) {

%>
   <tr>
     <td width="60" height="20"><% out.print(rs.getString("编号")); %></td>
     <td width="80" height="20"><% out.print(rs.getString("姓名")); %></td>
     <td width="200" height="20"><% out.print(rs.getString("发奖名称")); %></td>
     <td width="100" height="20"><% out.print(rs.getString("金额")); %></td>
     <td width="200" height="20"><% out.print(rs.getString("备注")); %></td>
 </tr>
    <%   } %>
</table>


<%
 // Catch exceptions
  }
```

```
  catch (Exception e) {
  }
  finally {

 if (rs != null)
       {
         try{rs.close();}catch(Exception ignore){};
         }

       if (stmt != null)
       {
         try{stmt.close();}catch(Exception ignore){};
         }
       if (conn != null)
       {
         try{conn.close();}catch(Exception ignore){};
         }


 %>


<%
   }
%>

</font>
</body>
</html>
```

启动 Tomcat。

浏览：http://127.0.0.1:8080/mywebapp/index.jsp



四、在 Tomcat 中实现系统和 Web 管理的配置

1、配置系统管理（Admin Web Application）

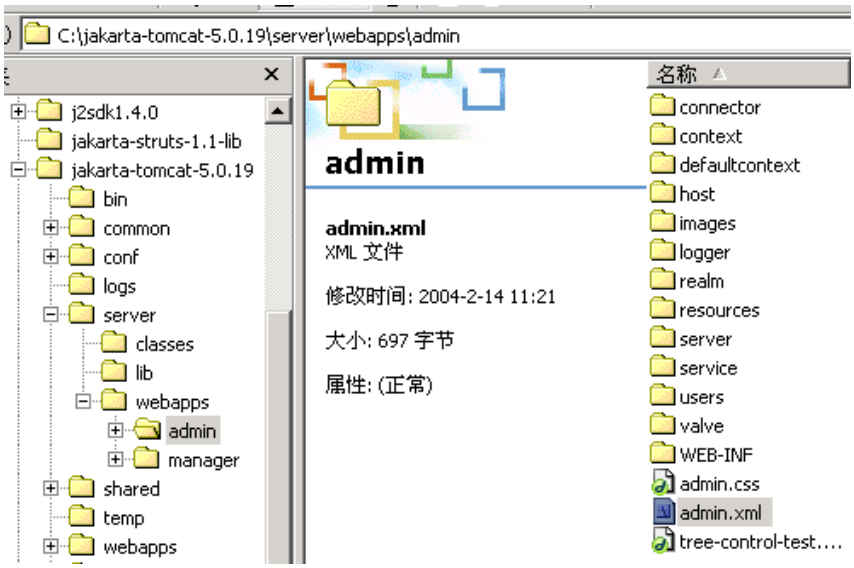
（1）概述

大多数商业化的 J2EE 服务器都提供一个功能强大的管理界面（如 Weblogic 的管理控制台），且大都采用易于理解的 Web 应用界面。Tomcat 按照自己的方式，同样提供一个成熟的管理工具，并且丝毫不逊于那些商业化的竞争对手。

Tomcat 的 Admin Web Application 最初在 4.1 版本时出现，当时的功能包括管理 context、data source、user 和 group 等。当然也可以管理像初始化参数，user、group、role 的多种数据库管理等。在后续的版本中，这些功能将得到很大的扩展，但现有的功能已经非常实用了。

（2）系统管理 Web 应用程序

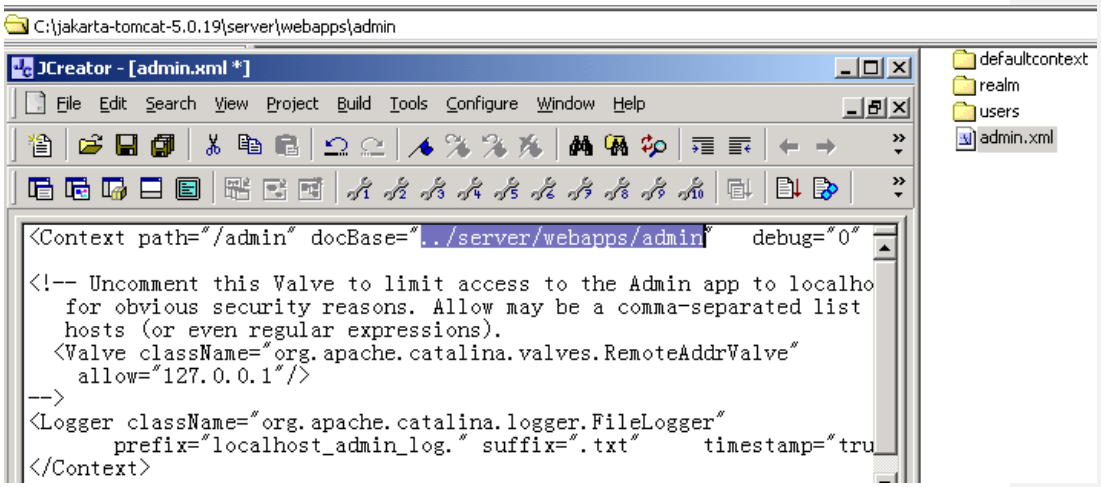
Tomcat 中的 Admin Web Application 被定义在自动部署文件：C:\jakarta-tomcat-5.0.19\server\webapps\admin\ admin.xml 中（请见下图所示）。



（3）编辑 admin.xml 文件

通过编辑 admin.xml 文件，以确定 Context 中的 docBase 参数设置为 Admin Web Application 所在的目录路径（应该是绝对路径）。作为另外一种选择，你也可以删除这个**自动部署文件，而在 C:\jakarta-tomcat-5.0.19\conf\server.xml 文件中建立一个 Admin Web Application 的 context，效果是一样的。**

你不能管理 Admin Web Application 这个应用，换而言之，除了删除 CATALINA_BASE/webapps/admin.xml ，你可能什么都做不了。

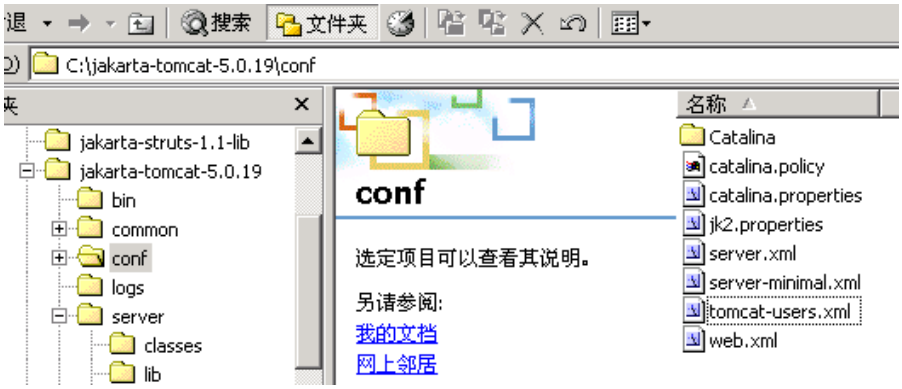


注意：如果将其中的被注释掉的<Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="127.0.0.1"/>打开，将能够限制访问 Admin Web Application 的程序主机为本机（服务器主机）；当然也可以设置为其它的主机 IP 地址（如设置为 Web 管理员所的工作主机）。

（4）在 C:\jakarta-tomcat-5.0.19\conf\ tomcat-users.xml 文件中添加系统管理员的角色和系统

管理员

Tomcat 中提供 UserDatabaseRealm（默认），这样我们可以根据管理的需要添加不同的用户角色和与该角色相配置的用户名称和密码



- 添加用户角色

  <role name="admin"/>

- 添加与该角色相配置的用户名称和密码

  <user name="admin" password="12345678" roles="admin"/>

当你完成这些步骤后，请重新启动 Tomcat，访问 http://localhost:8080/admin，你将看到一个登录界面。Admin Web Application 程序采用基于容器管理的安全机制，并采用了 Jakarta Struts 框架。下面是在原来的 tomcat-users.xml 文件中再添加了两个角色 admin 和 manager，同时也添加了与该两个角色相配置的用户 admin 和 manager。

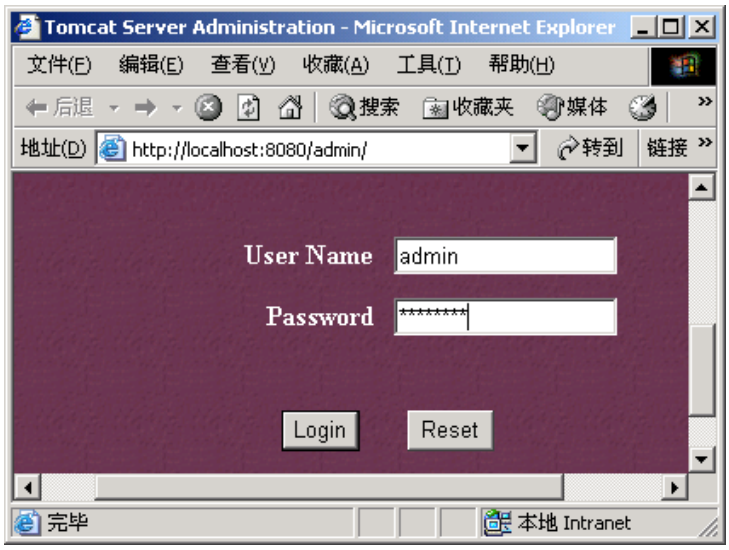
```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="role1"/>
  <role rolename="tomcat"/>
  <role rolename="admin"/>
  <role rolename="manager"/>
  <user username="admin" password="12345678" roles="admin"/>
  <user name="manager" password="12345678" roles="manager"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
</tomcat-users>
```

（5）登录 Admin Web Application 程序

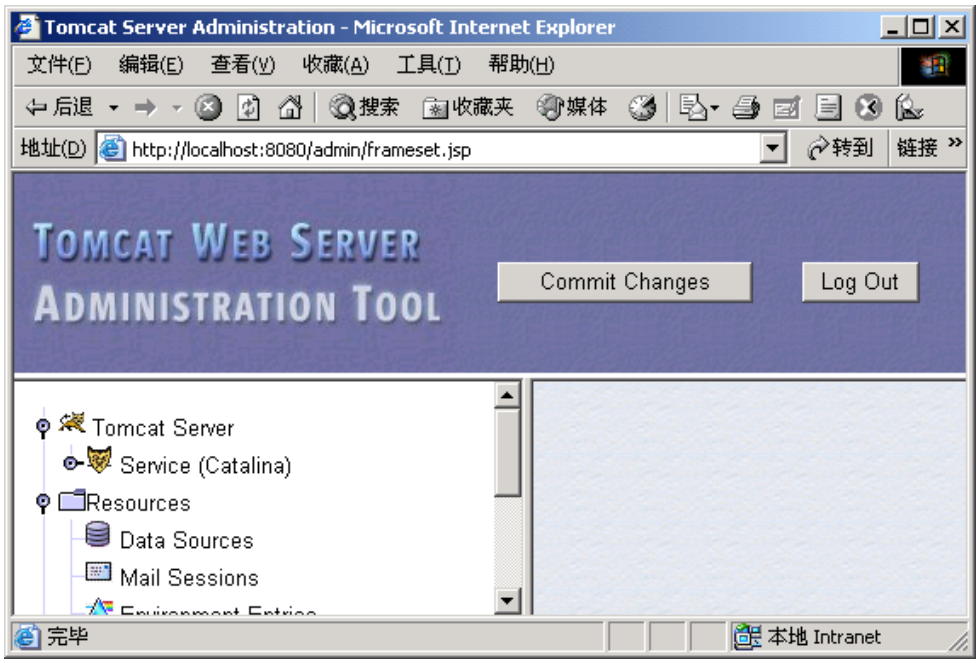
输入 http://localhost:8080/admin/进入系统管理员的登录页，然后在页中

输入用户名称：admin

密码： 12345678



将进入系统管理的界面，在该系统管理的程序中将可以配置各种资源如 Data Sources、Mail Sessions、Environment Entries，并且也可以管理 Users 和 Groups 以及 Roles 等功能。



2、配置应用管理（Manager Web Application）

（1）概述

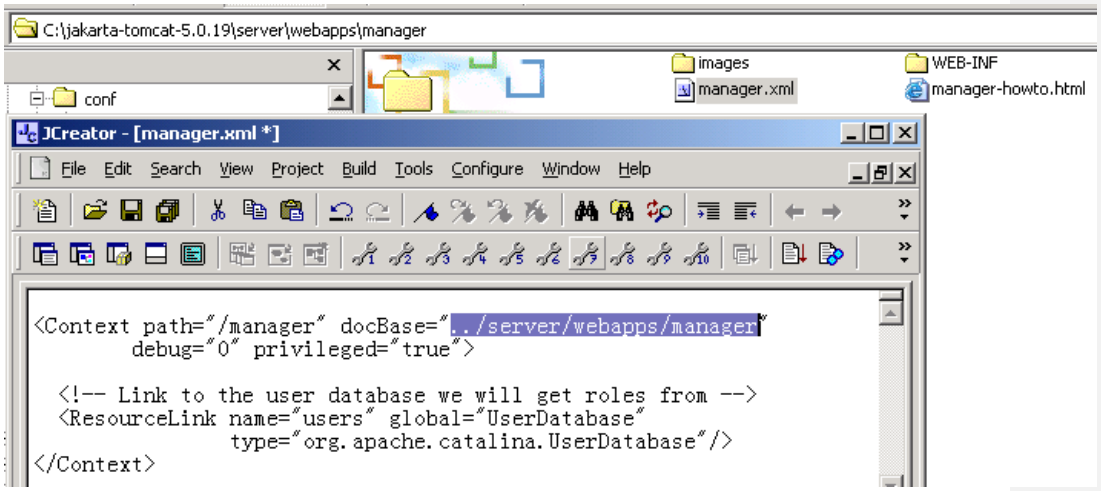
Tomcat 中所提供的 Manager Web Application 让你通过一个比 Admin Web Application 更为简单的用户界面，执行一些与 Web 应用任务相关的一些管理功能。

（2）Manager Web Application 程序

Manager Web Application 被被定义在一个自动部署文件中 C:\jakarta-tomcat-5.0.19\server\webapps\manager\manager.xml 。

（3）编辑 manager.xml 文件

通过编辑这个文件，以确保其中的 context 中的 docBase 属性参数是 C:\jakarta-tomcat-5.0.19\server\webapps\manager 的绝对路径。



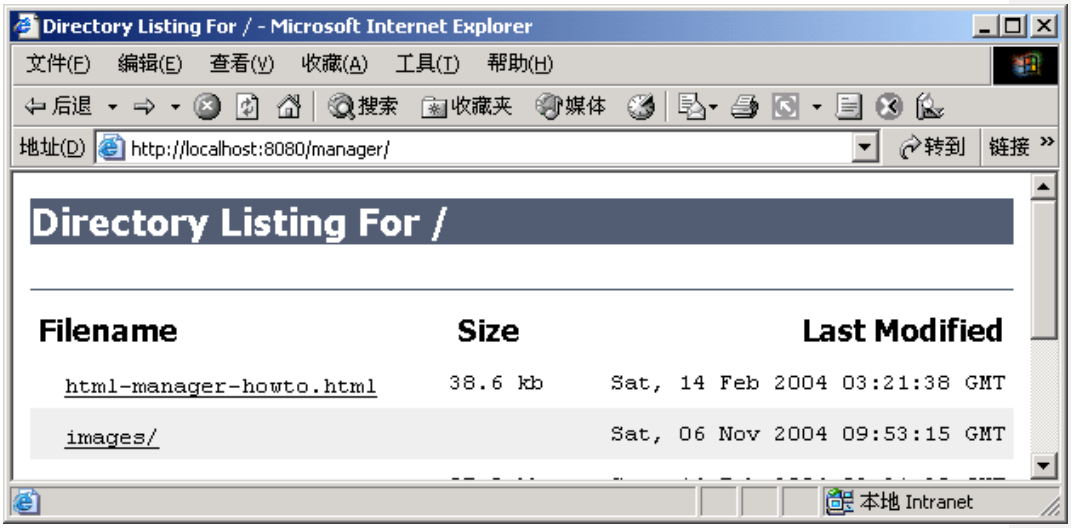
（4）在 C:\jakarta-tomcat-5.0.19\conf\ tomcat-users.xml 文件中添加 Web 管理员的角色和 Web 管理员

- 添加用户角色
  <role name=" manager "/>
- 添加与该角色相配置的用户名称和密码
  <user name="manager" password="12345678" roles="manager"/>

（5）登录 Web 管理员的页面

- 文本型管理界面
  然后重新启动 Tomcat，输入 http://localhost:8080/manager/，将进入看到一个很朴素的文

本型管理界面

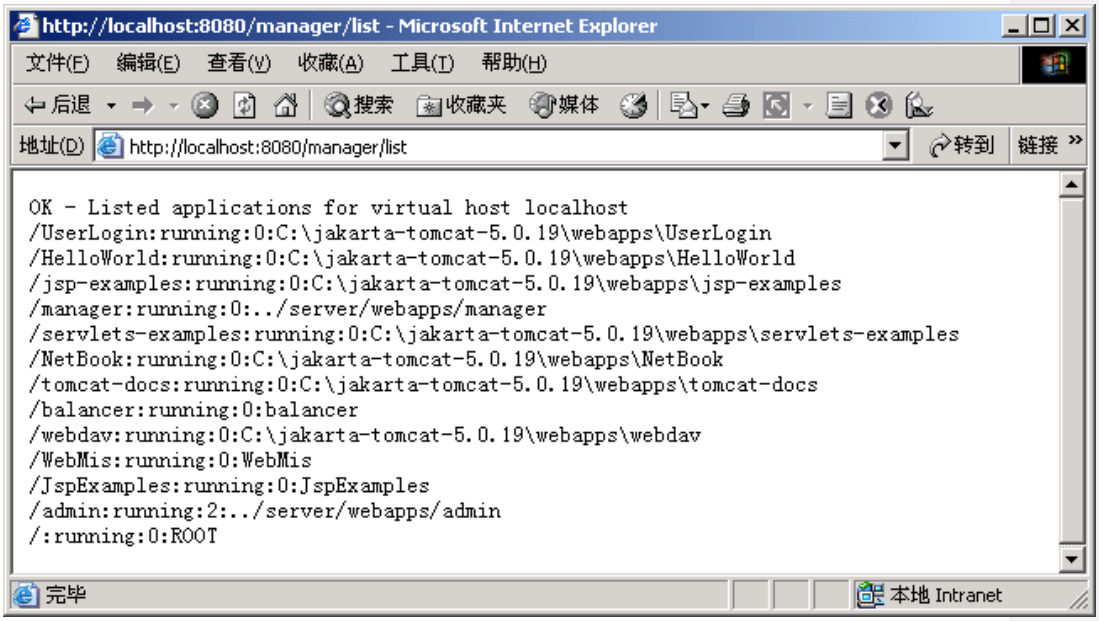


如果输入 http://localhost:8080/manager/list，将进入一个登录管理界面，然后

输入用户名称：manager（前面在 tomcat-users.xml 中设置的）

密码：12345678

将显示出

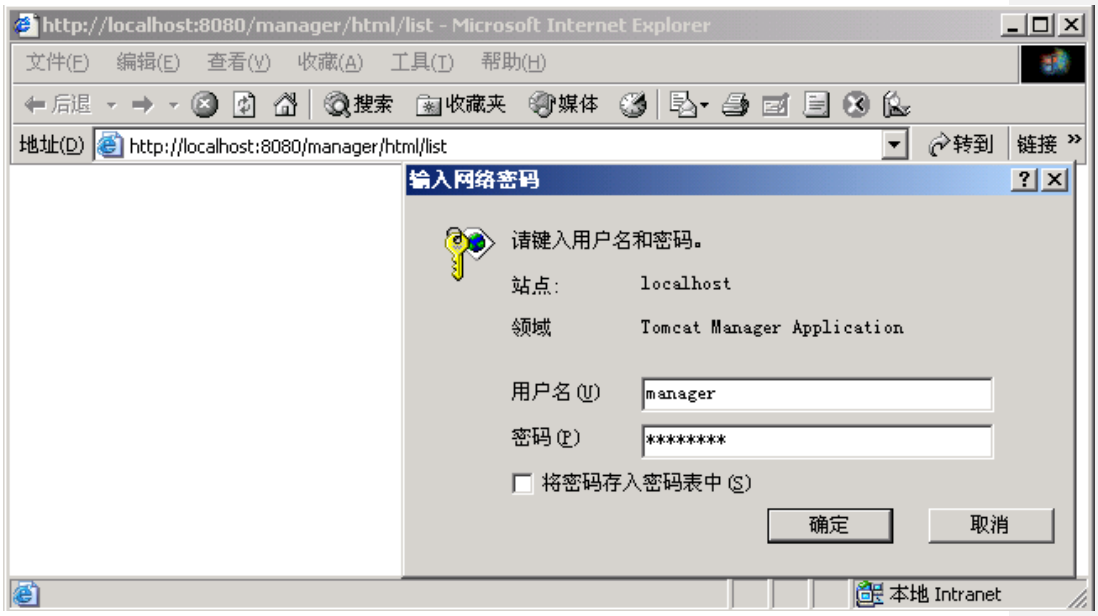


- HTML 型管理界面

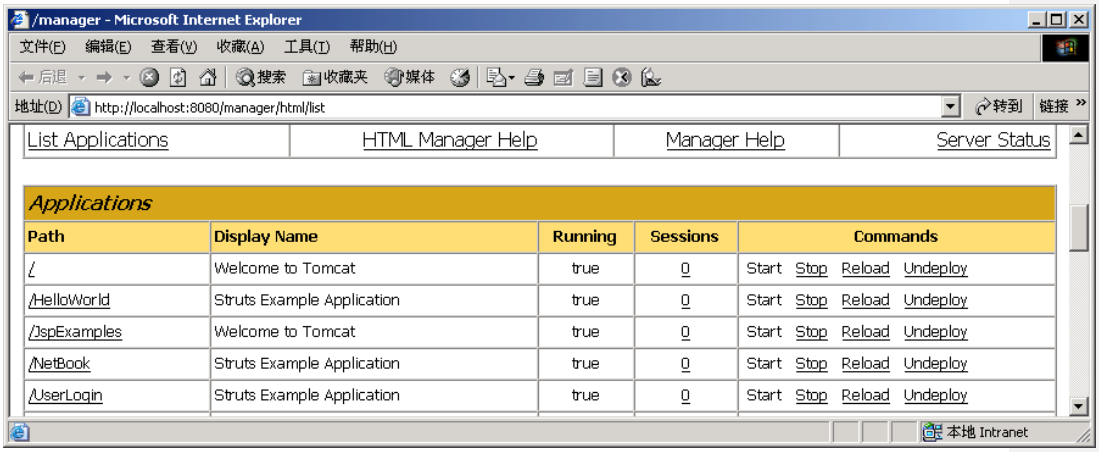
输入 http://localhost:8080/manager/html/list，将出现如下的页面，然后再

输入用户名称：manager

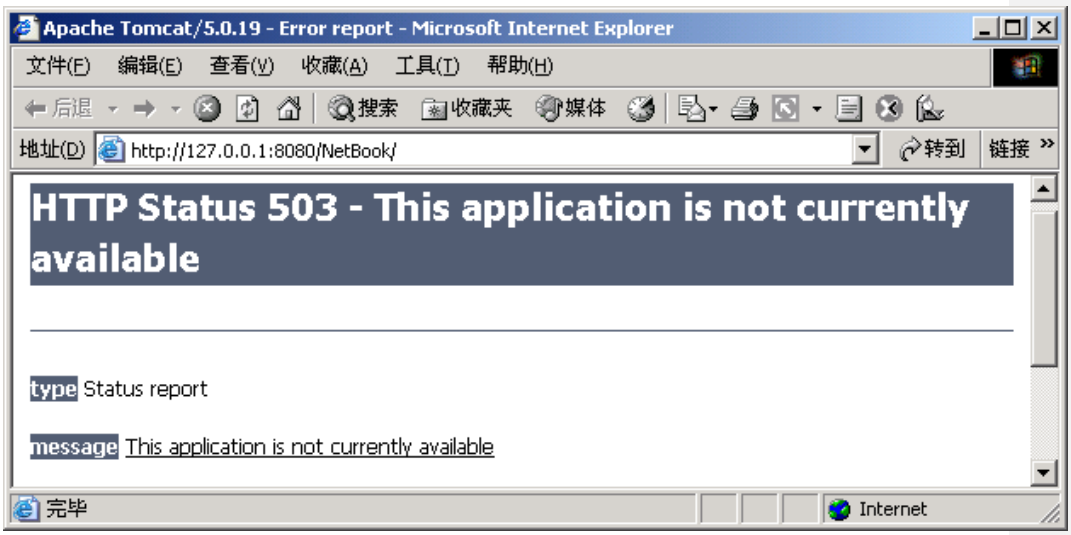
密码：12345678



将出现 Web 方式的管理页面

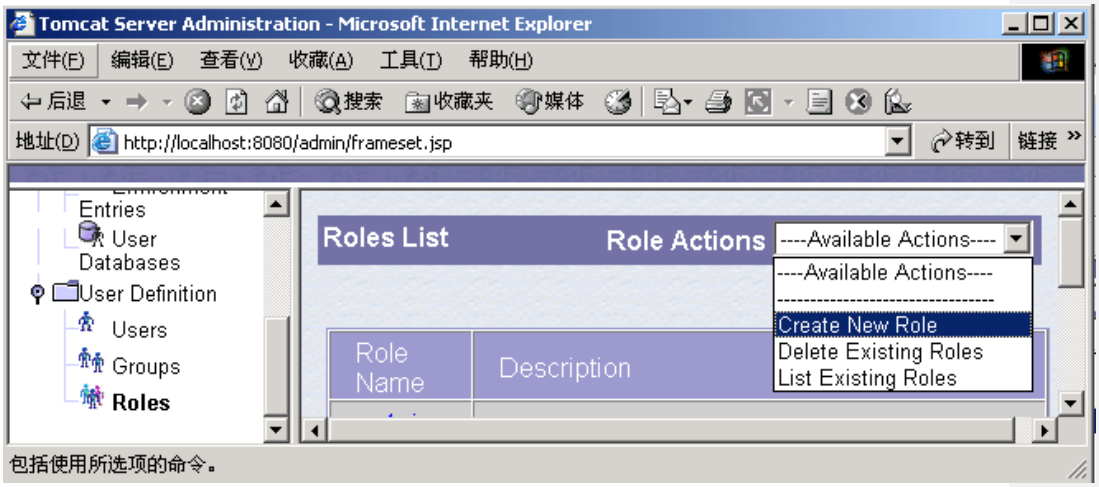


Manager application 可以让用户在没有系统管理特权的基础上，部署安装新的 Web 应用，以用于测试。同时也可以对所部署的 Web 应用程序的工作状态进行控制（Start 或者 Stop），以免重新启动服务器（这在对 web.xml 等配置的内容发生改变的情况下，特别有效）。当有用户尝试访问这个被停止的应用时，将看到一个 503 的错误——“503 - This application is not currently available”。

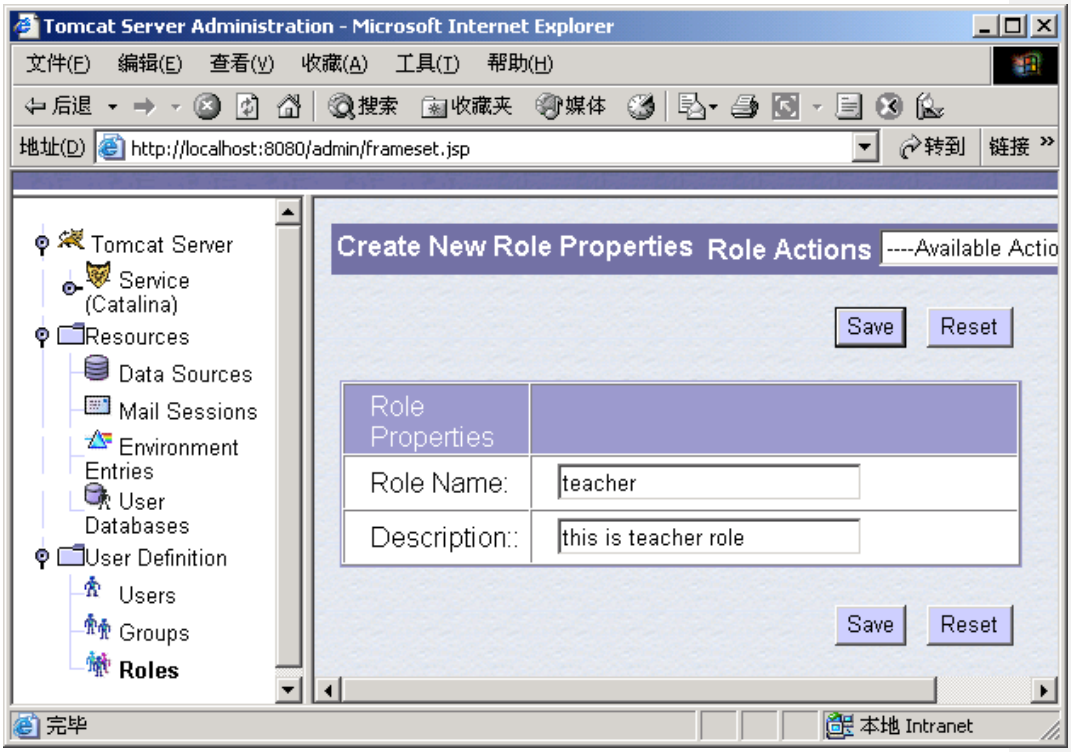


3、配置各种用户角色、用户组和用户

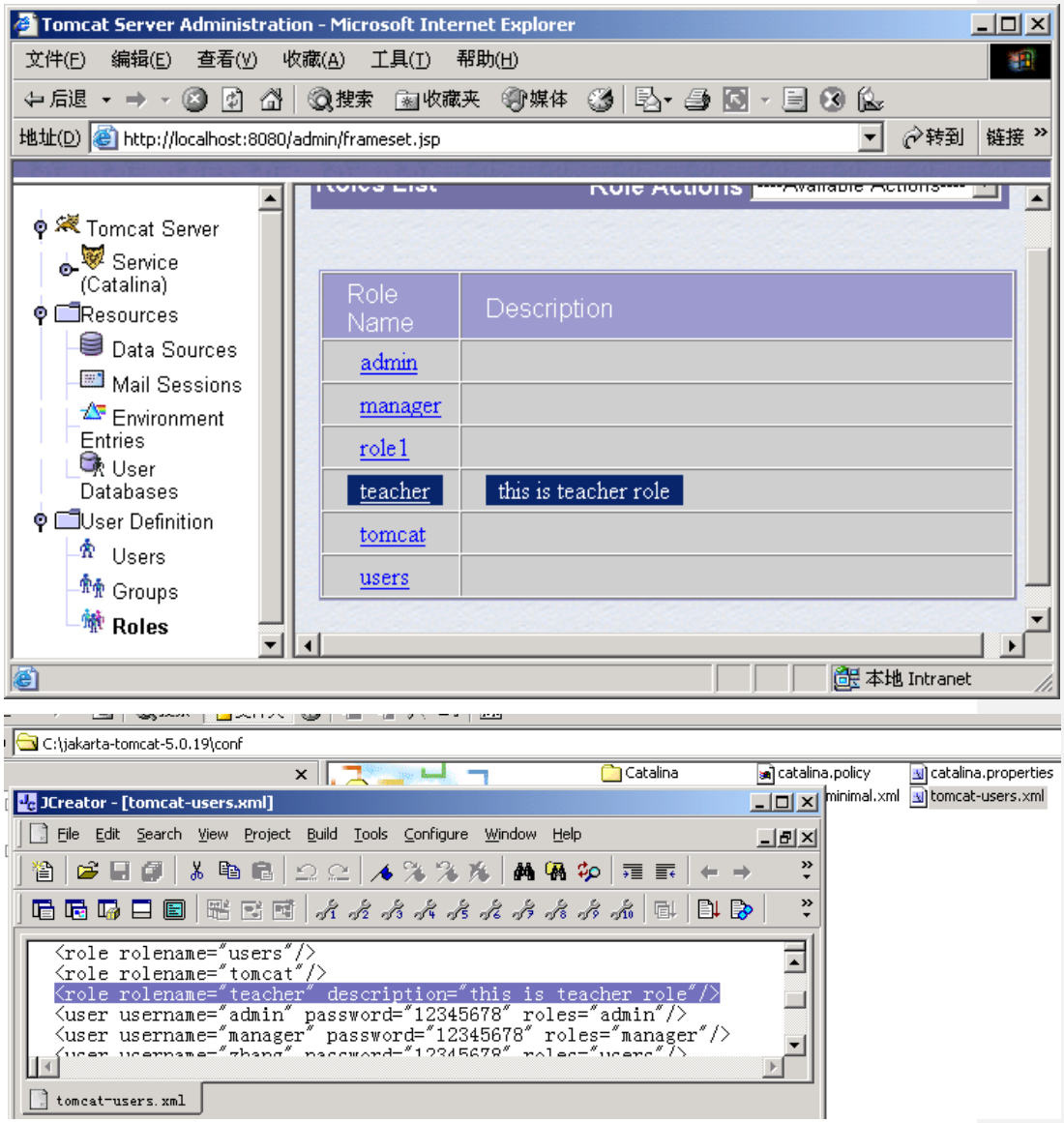
（1）添加用户角色：在 admin 的界面中点击左面的 Roles 节点，然后在右面的下拉列表框中选择 Create New Role 项目。



然后输入角色的名称和描述

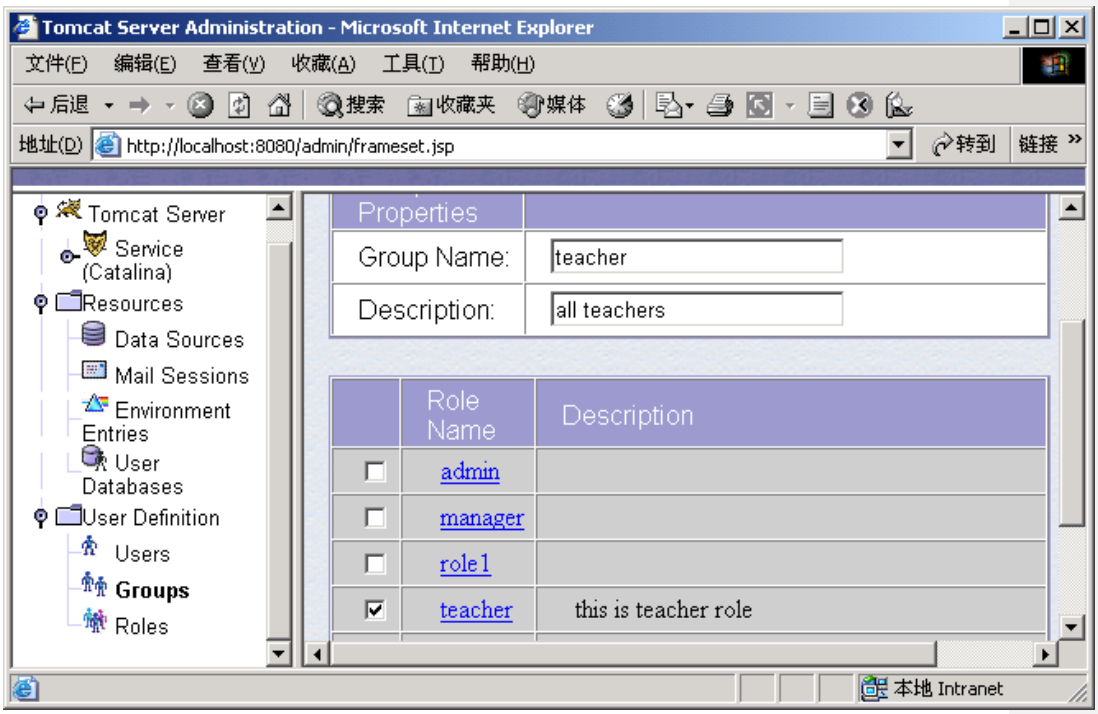


最后点击“保存”，将存储在 C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml 文件中并且在管理界面中显示出。

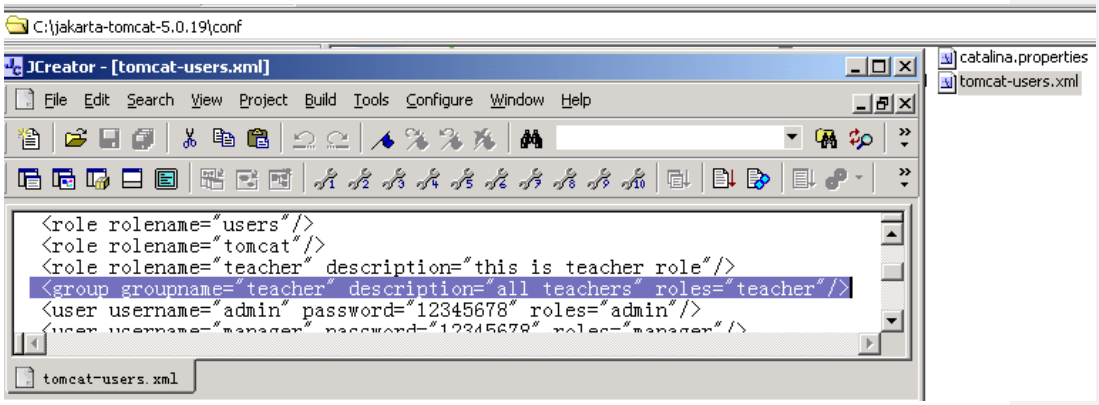


（2）添加用户组：在 admin 的界面中点击左面的 Groups 节点，然后在右面的下拉列表框中选择 Create New Group 项目。

然后输入组的名称和描述，并且设置该组的角色。所应该注意的是，给组分配角色，则意味着该组中的各个成员（用户）将具有该角色所分配的各种权限。

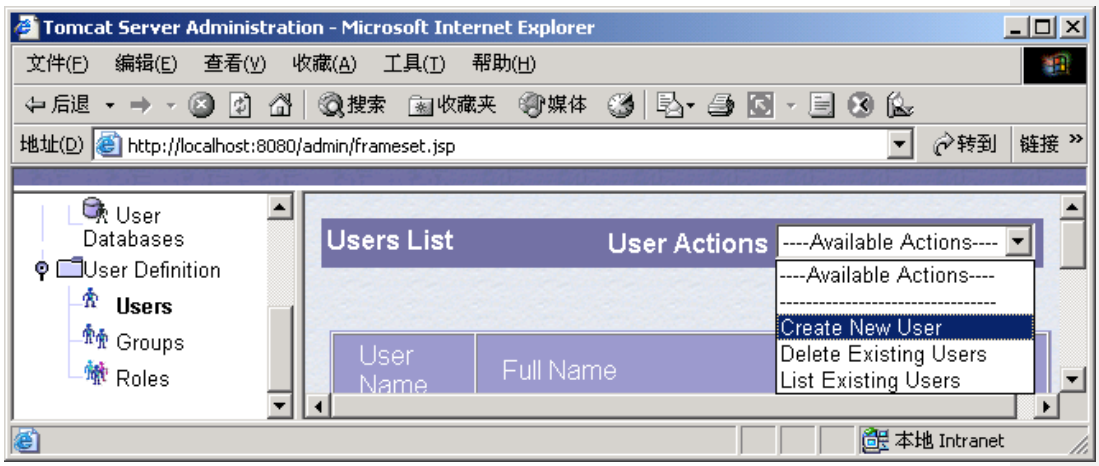


最后点击“Save”以保存它（仍然放在 C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml 文件中）



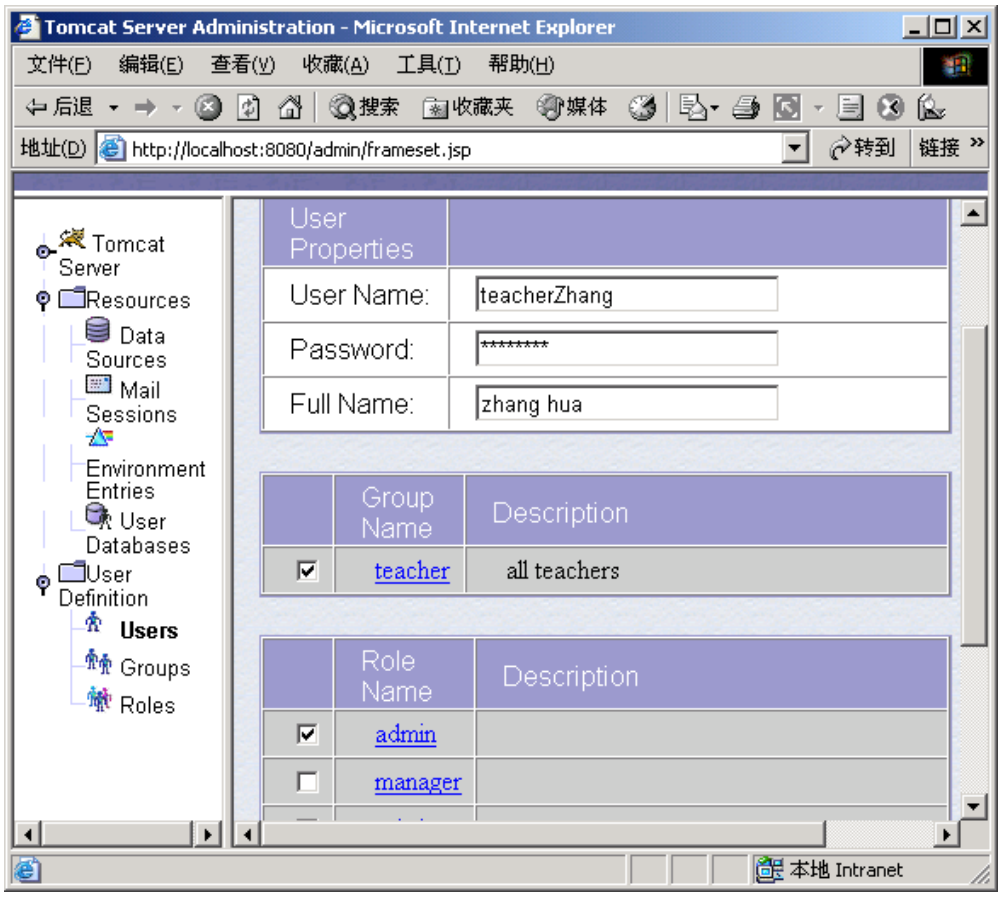
（3）添加属于某一用户组内的用户

在 admin 的界面中点击左面的 Users 节点，然后在右面的下拉列表框中选择 Create New User 项目。

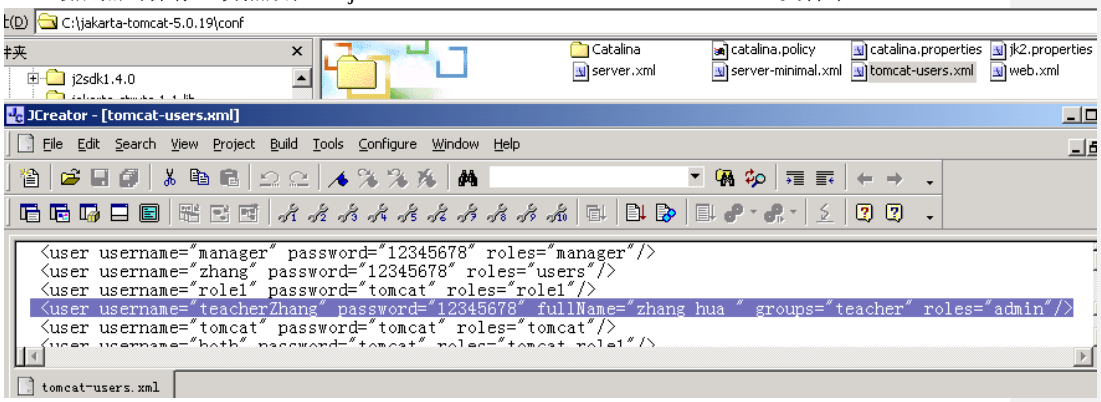


然后该用户的名称同时包括全名称、密谋，并且设置该用户所属的用户组；同时也可以为该用户再设置其它的角色以使该用户除了具有用户组的通用的权利以外，还具有其他方面的权利。

下面对“teacherZhang”这个用户进行设置，同时他也是系统管理员，因此将下面的 admin 的角色也选中。



最后点击保存（仍然放在 C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml 文件中）



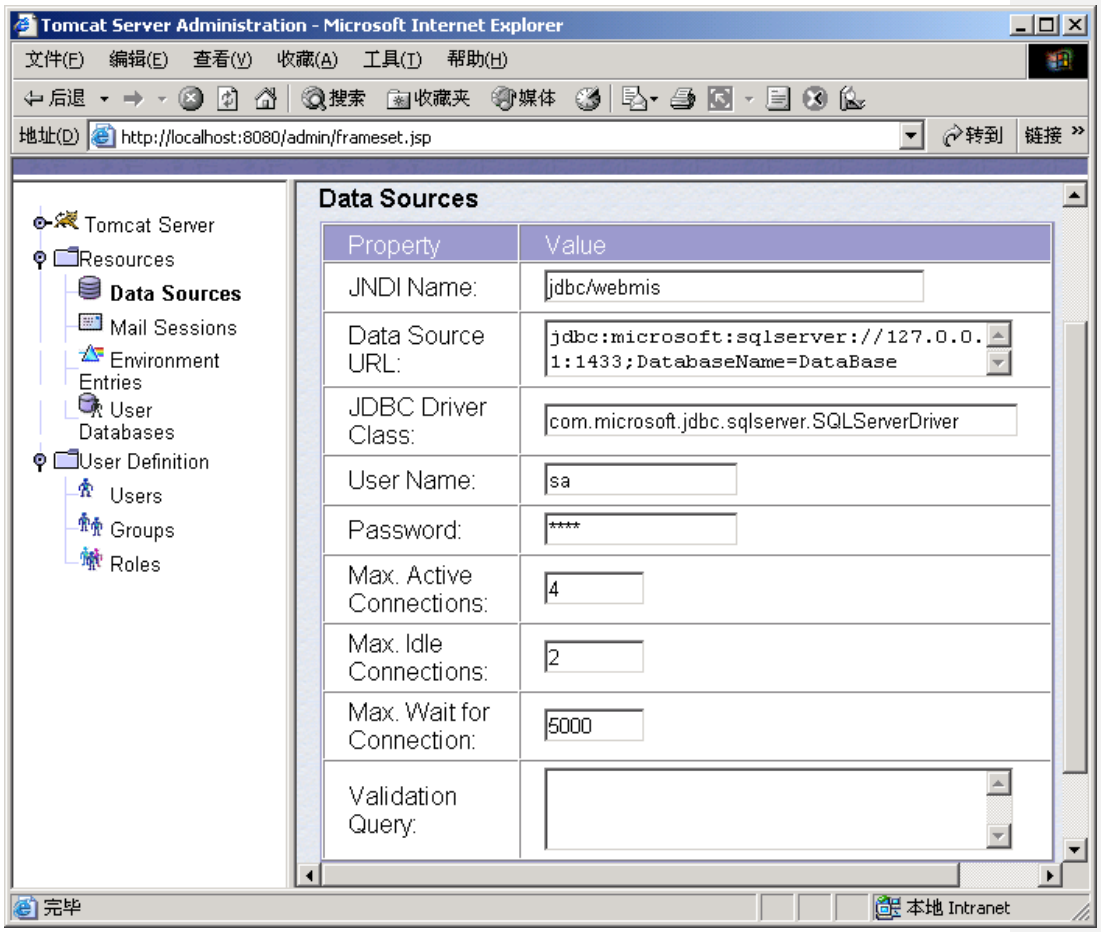
4、添加其它的系统资源

（1）DataSource

在 admin 的界面中点击左面的 DataSourcs 节点，然后在右面的下拉列表框中选择 Create New

DataSource 项目。

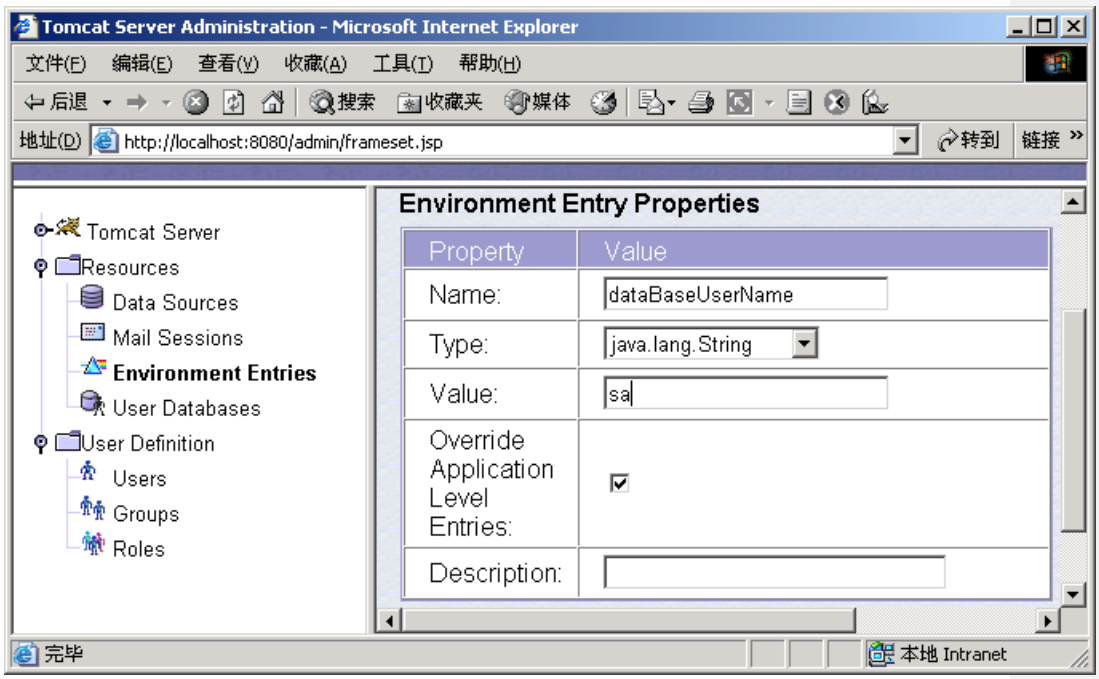
在各个输入的项目中根据数据库的特性进行输入。最后点击“Save”以保存。



（2）添加环境变量

在 admin 的界面中点击左面的 Environment Entries 节点，然后在右面的下拉列表框中选择 Create New Env Entry 项目。

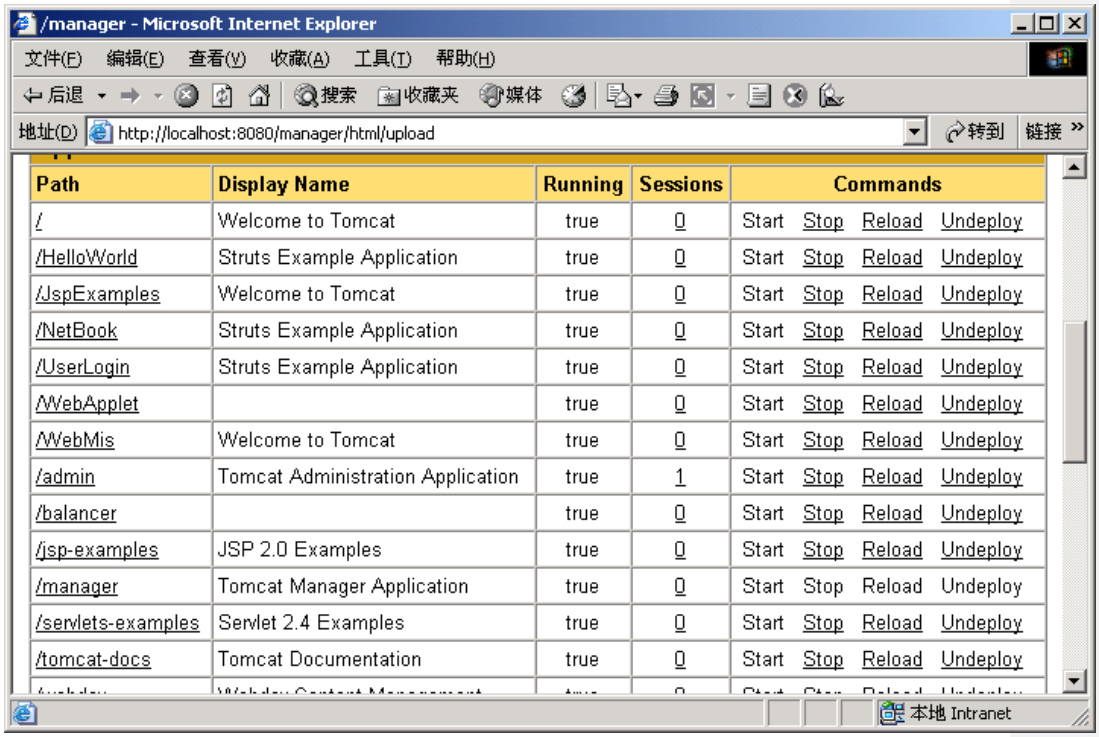
在各个输入的项目中根据数据库的特性进行输入。最后点击“Save”以保存。



5、对 Web 应用程序进行管理

（1）输入 http://localhost:8080/manager/html/list，将出现登录页并且进行登录，然后再进入 Tomcat Web Application Manager

（2）查看在 Web 服务中所发布的各个 Web 应用



Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/HelloWorld	Struts Example Application	true	0	Start Stop Reload Undeploy
/JspExamples	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/NetBook	Struts Example Application	true	0	Start Stop Reload Undeploy
/UserLogin	Struts Example Application	true	0	Start Stop Reload Undeploy
/WebApplet		true	0	Start Stop Reload Undeploy
/WebMis	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/admin	Tomcat Administration Application	true	1	Start Stop Reload Undeploy
/balancer		true	0	Start Stop Reload Undeploy
/jsp-examples	JSP 2.0 Examples	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/servlets-examples	Servlet 2.4 Examples	true	0	Start Stop Reload Undeploy
/tomcat-docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy

（3）启动或者终止、移除某一 Web 应用：

点击该 Web 应用右面的 Stop 链接，也可以点击 Start 再次启动它。Undeploy（移除）一个 Web 应用，只是指从 Tomcat 的运行拷贝中删除了该应用，如果你重新启动 Tomcat，被删除的应用将再次出现（也就是说，移除并不是指从硬盘上删除）。

（4）部署某一 Web 应用

有三种方式可以在 Tomcat 系统中部署 Web 应用。

- 直接拷贝你的 WAR 文件或者你的 Web 应用文件夹（包括该 Web 应用的所有内容）到 C:\jakarta-tomcat-5.0.19\webapps 目录下。

  该文件必须以“.war”作为扩展名。一旦 Tomcat 监听到这个文件，它将（缺省的）解开该文件包作为一个子目录，并以 WAR 文件的文件名作为子目录的名字。接下来，Tomcat 将在内存中建立一个 context，就好象你在 server.xml 文件里建立一样。当然，其他必需的内容，将从 server.xml 中的 DefaultContext 获得。

- 部署 web 应用的另一种方式是写一个 Context XML 片断文件，然后把该文件拷贝到 C:\jakarta-tomcat-5.0.19\webapps 目录下。

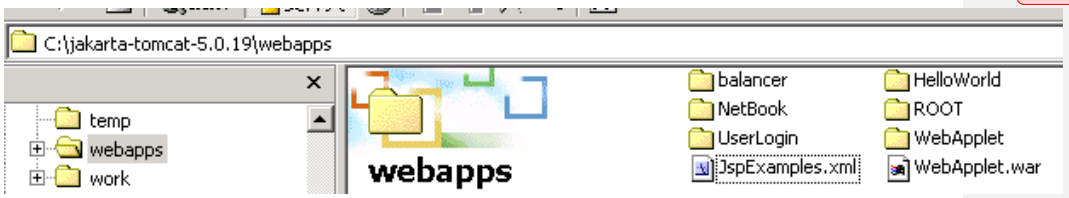
  一个 Context 片断并非一个完整的 XML 文件，而只是一个 Context 元素，以及对该应用的相应描述。这种片断文件就像是从 server.xml 中切取出来的 context 元素一样，所以这种片断被命名为“context 片断”。这个 web 应用本身可以存储在硬盘上的任何地方。

举个例子，如果我们想部署一个名叫 JspExamples 的 Web 应用，该应用使用 realm 作为访问控制方式，我们可以使用下面这个片断：

```
<!--
   Context fragment for deploying JspExamples
-->
<Context path="/JspExamples" docBase="JspExamples" debug="0" reloadable="true">
    <RealmclassName="org.apache.catalina.realm.UserDatabaseRealm"
resourceName="UserDatabase"/>
</Context>
```

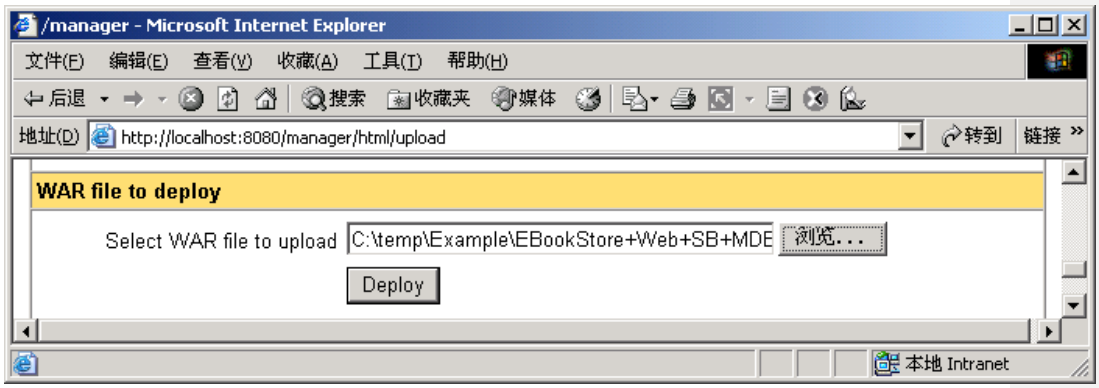
把该片断命名为“JspExamples.xml”，然后拷贝到 C:\jakarta-tomcat-5.0.19\webapps 目录下。这种 Context 片断提供了一种便利的方法来部署 web 应用，你不需要编辑 server.xml，除非你想改变缺省的部署特性，安装一个新的 Web 应用时不需要重启动 Tomcat。

批注 [linger1]: 难道不应该是放在 conf 文件夹中

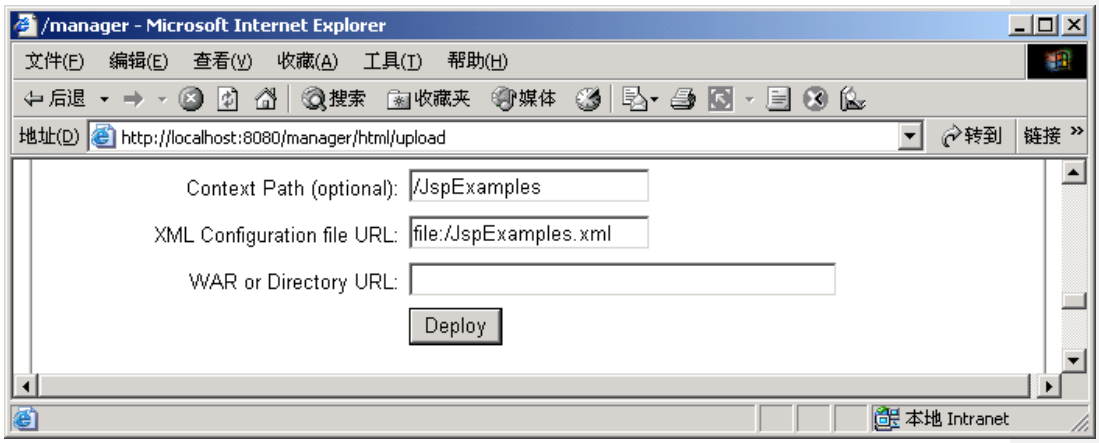


- **采用 GUI 管理界面进行发布**

如果提供了该 Web 应用的*.war 文件，直接浏览并发布它



如果 Web 应用是以目录形式存在的，则可以：



五、Tomcat 服务器的 Web 安全的解决方法

1、概述

在任何一种 WEB 应用开发中，不论大中小规模的，每个开发者都会遇到一些需要保护程序数据的问题，涉及到用户的 LOGIN ID 和 PASSWORD。那么如何执行验证方式更好呢？实际上，有很多方式来实现。

下面将讨论在 Tomcat 中实现基本的（BASIC）和基于表单的（FORM-BASED）验证方式。它通过 server.xml 和 web.xml 文件提供基本的和基于表单的验证。

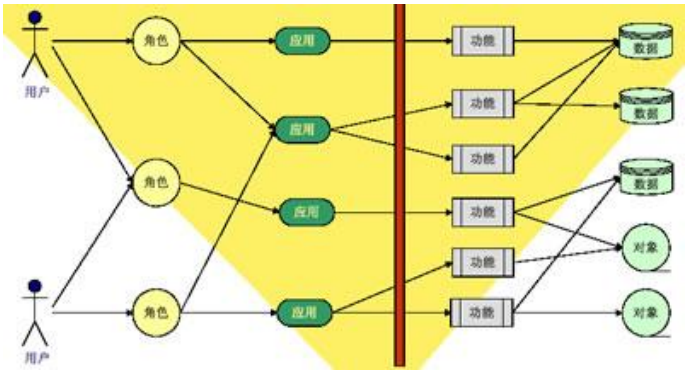
对于采用基于表单的（FORM-BASED）验证方式，只是要求在登录的 JSP 页面中的 j_security_check 表单(for FORM-based) 需要两个参数：j_username 和 j_password。

对于用户的登录的名称和密码在 Tomcat 中可以以两种形式来存放，一是采用 server.xml？；另一种也可以采用用户自己的数据库表来存储。

2、设计系统中的各种人员的角色

（1）设计思想

- 统一用户管理，实现基于角色、粗粒度（基于 URL）和细粒度（基于应用组件的方法调用）的访问策略管理体系，
- 基于分级角色的权限管理、统一证书管理和统一资源管理



	人事	财务	教务	房产			设备	信息	办公
学校主管领导									
主管部门	人事处	财务处	教务处 研究生院 成人教育				设备处		校办
单位主管									
业务人员	人事 干事	会计	教务员	宿舍	家属	公房	保管员	信息员	办公室任收员 公主发
普通公众									

（2）设计目标

一般采用数据库表（对于复杂的也可以采用 LDAP）记录每个系统用户的帐号信息、功能权限和数据权限信息，这样能够增加用户管理和权限设置的灵活性，同时也避免多个用户共用一个帐号的情况。

（3）优点

- 从用户角度来看，登录所有应用系统都使用唯一的用户名和口令（数字证书）同时在访问系统时，也只需要登录一次（单点登录全网漫游---SSO（Single Sign-On））。
- 从管理者角度来看，提供了统一、集中、有效的用户管理。

六、在 Tomcat 中实现基本的 HTTP 方式的验证

1、实现基本验证

（1）在 C:\jakarta-tomcat-5.0.19\conf 下的 tomcat-users.xml 文件中添加角色和用户（可以同时添加多个用户）

```
<role rolename="users"/>
<user name="yang"      password="12345678" roles="users"/>
<user name="zhang"     password="12345678" roles="users"/>
```

（2）在 Web 应用的 web.xml 文件中添加如下的项目

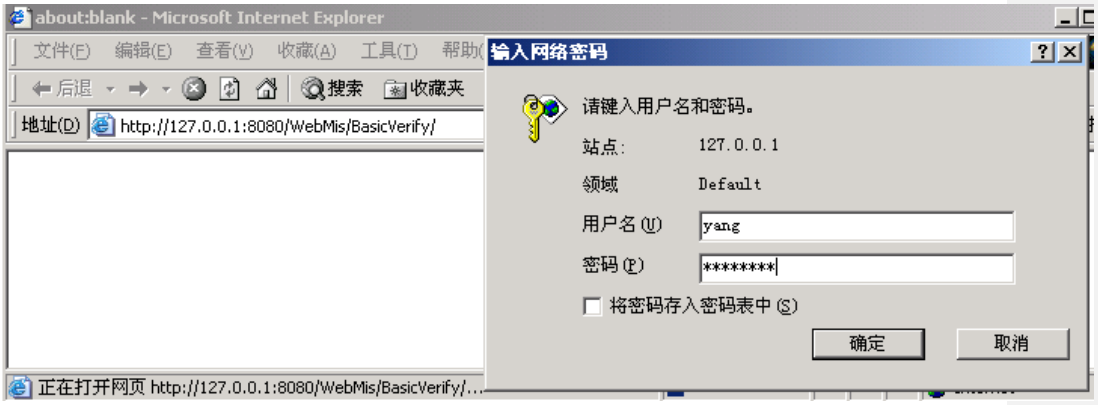
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
…..
      <security-constraint>
    <web-resource-collection>
            <web-resource-name>
              protected Resource
            </web-resource-name>
            <url-pattern>/BasicVerify/*</url-pattern>
    </web-resource-collection>
   <auth-constraint>
        <role-name>users</role-name>
   </auth-constraint>
   </security-constraint>
    <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Default</realm-name>
      </login-config>

<security-role>
          <description>this is a user</description>
          <role-name>users</role-name>
   </security-role>
…..
</web-app>
```

（3）重新启动 Tomcat 服务器

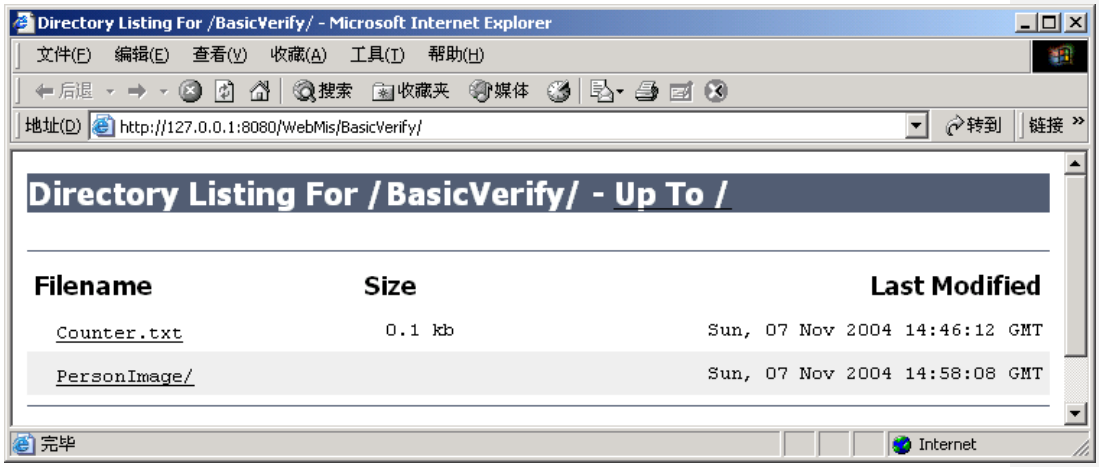
并在浏览器中直接输入所保护的目录 http://127.0.0.1:8080/WebMis/BasicVerify，将出现如下的登录页

输入用户名称：yang（请见前面的 tomcat-users.xml 文件的设置）

密码：12345678



将出现如下的



如果用户名称或者密码出现错误，将强制输入。

七、在 Tomcat 中采用基于表单的安全验证

1、概述

（1）基于表单的验证

基于 From 的安全认证可以通过 Tomcat Server 对 Form 表单中所提供的数据进行验证，基于表单的验证使系统开发者可以自定义用户的登陆页面和报错页面。这种验证方法与基本 HTTP 的验证方法的唯一区别就在于它可以根据用户的要求制定登陆和出错页面。

通过拦截并检查用户的请求，检查用户是否在应用系统中已经创建好 login session。如果没有，则将用户转向到认证服务的登录页面。但在 Tomcat 中的基于表单的验证凭证不被保护并以纯文本发送。

（2）在 Tomcat 中的实现

在 Tomcat 中，用户、用户组和角色都是在 XML 配置文件（C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml）中指定的，我们只需要提供一个登陆页面，包含一个名为 j_security_check 的 Form 表单，一个名为 j_username 的 TextBox 和一个名为 j_password 的 PasswordBox，然后在/WEB-INF/web.xml 中配置即可使用 Tomcat 默认的 JAAS 身份验证。

使用 JAAS 验证的好处是，验证逻辑从页面中分离，对页面的限制访问是通过/WEB-INF/web.xml 中的配置指定的，无需自定义过滤器。

（3）为了实现 Web 应用程序的安全，Tomcat Web 容器执行下面的步骤：

- 在受保护的 Web 资源被访问时，判断用户是否被认证。
- 如果用户没有得到认证，则通过重定向到部署描述符中定义的注册页面，要求用户提供安全信任状。
- 根据为该容器配置的安全领域，确认用户的信任状有效。
- 判断得到认证的用户是否被授权访问部署描述符（web.xml）中定义的 Web 资源。

2、设计步骤

（1）编写登录页面和错误处理页面：请见 FormSafeWebApp 程序中的页面

（2）登录的页面文件的内容如下

基于 FORM 的用户认证要求你返回一个包括用户名和密码的 HTML 表单，这个表单相对应与用户名和密码的元素必须是 j_username 和 j_password，并且表单的 action 描述必须为 j_security_check（其实是一个 Servlet）。该表单的具体操作以及 j_username 和 j_password 名字在 Servlet 中定义。当这个表单到达服务器的时候，由内部的 Tomcat Server 安全区对它进行确认。

包括这个表单的资源可以是一个 HTML 页面、一个 JSP 页面或者一个 Servlet。你可以在<form-login-page>元素中定义。基于表单的认证能够使开发人员定制认证的用户界面。在 web.xml 的 login-config 标签项目定义了认证机制的类型、登录的 URI 和错误页面。

下面为该页面的内容：

```
<%@ page contentType="text/html; charset=GBK" %>
<html><head><title>在 Tomcat 中采用 Form 验证方式实现的安全 Web 应用程序的登录页</title>
</head><body bgcolor="#ffffff">
<form method="post" name="Login" action="j_security_check">
```

注意：action 应该为 j_security_check

```
<table width="500" border="1" align="center"> <tr>
    <td colspan="2"> <div align="center"><strong>在 Tomcat 中采用</strong><strong>基于表单的
安全验证的登录表单 </strong> </div></td> </tr>
  <tr><td width="224"><div align="right">用户名称：</div></td>
    <td width="260"><input type="text" name="j_username"></td> </tr>
  <tr> <td><div align="right">密码：</div></td>
   <td><input type="password" name="j_password"></td>
  </tr>
  <tr><td><div align="right"><input type="submit" name="Submit" value="提交">
    </div></td> <td><input type="reset" name="Submit2" value="重置"></td>
  </tr></table></form></body></html>
```

注意：用户名称和密码的输入应该为 j_username 和 j_password

（3）修改 web.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
     </welcome-file-list>
 <!-- Security is active on entire directory -->
  <security-constraint>
    <display-name>Tomcat Server Form Security Constraint</display-name>
    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
      <description>A Page of Login Success</description>
       <url-pattern>/ProtectedDirOne/index.jsp</url-pattern>
      </web-resource-collection>
    <auth-constraint>
      <!-- Anyone with one of the listed roles may access this area -->
      <role-name>admin</role-name>
    </auth-constraint>
  </security-constraint>
  <!-- Login configuration uses form-based authentication -->
  <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>Tomcat Server Configuration Form-Based Authentication
Area</realm-name>
    <form-login-config>
        <form-login-page>/login.jsp </form-login-page>
        <form-error-page>/Error.htm </form-error-page>

    </form-login-config>
```

定义本 Web 应用的默认启始页面

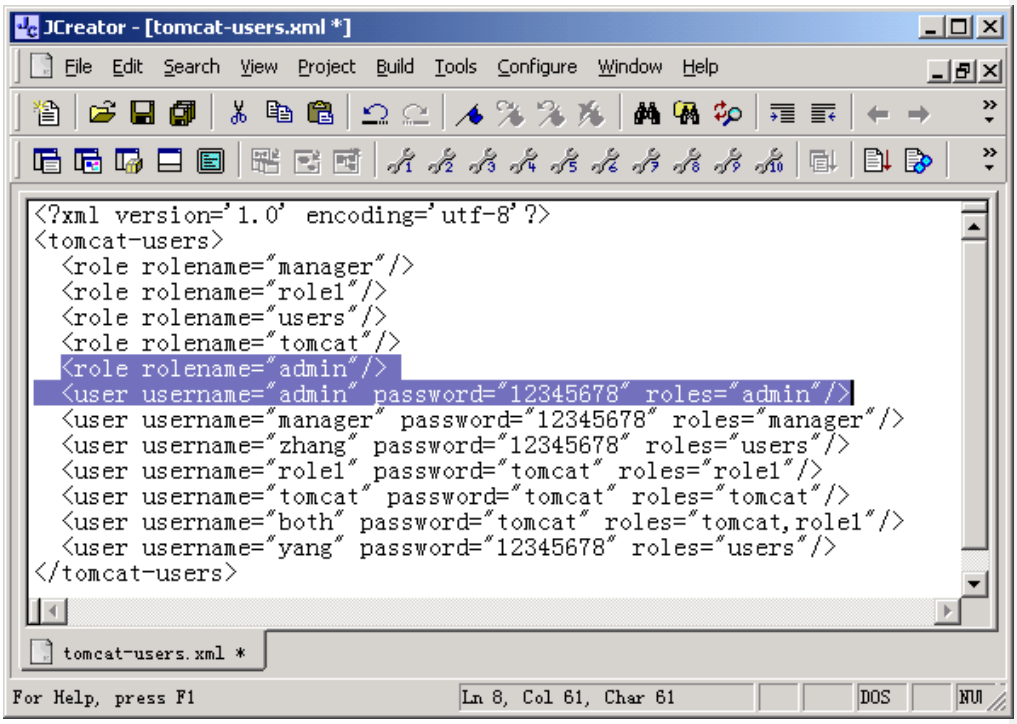
指定 Form 验证的用户的角色名称

指定验证的方式为 Form

```
    </login-config>
    <!-- Security roles referenced by this web application -->
    <security-role>
      <description>
        The role is Administration
      </description>
      <role-name>admin</role-name>
    </security-role>
</web-app>
```

关联 Tomcat 中的 admin 的角色

（4）在 C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml 文件中配置 admin 的角色以及与该 admin 角色相匹配的用户名称和密码

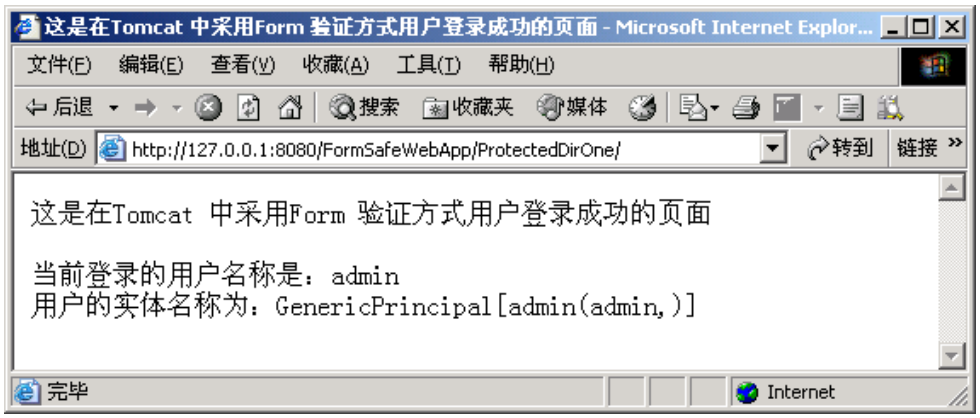


（5）执行该页面

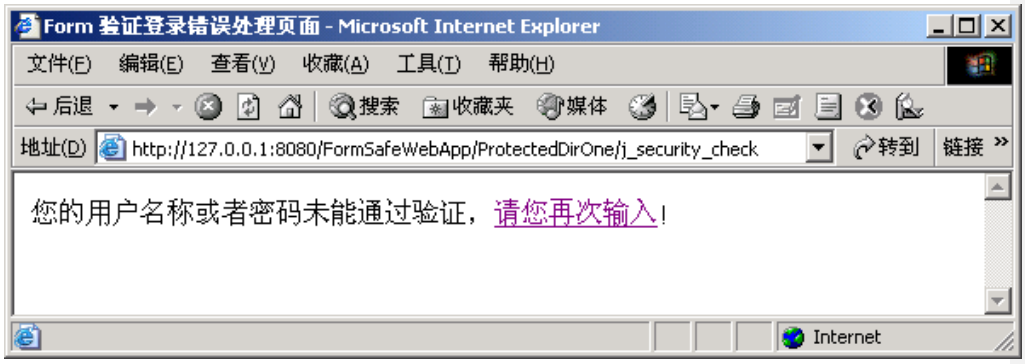
在浏览器中直接输入受保护的页面的 URL 地址：

http://127.0.0.1:8080/FormSafeWebApp/ProtectedDirOne/，将出现要求登录的页面。

在表单中输入用户名称为 admin（前面在 tomcat-users.xml 文件中所设置的某一用户名称），密码为 12345678。然后点击“提交”，将出现如下页面



如果用户名称或者密码输入不正确，将出现如下的页面也就是错误页面



（6）在页面中获得当前登录成功后的用户名称和实体名称

利用 request 对象中的 getRemoteUser()方法获得当前登录成功后的用户名称和利用 getUserPrincipal()方法获得当前登录成功后的实体名称。

八、在 Tomcat 中配置单点登录（Single Sign-On）

1、概述

一旦你设置了 realm 和验证的方法，你就需要进行实际的用户登录处理。一般说来，对用户而言登录系统是一件很麻烦的事情，你必须尽量减少用户登录验证的次数。作为缺省的情况，当用户第一次请求受保护的资源时，每一个 Web 应用都会要求用户登录。

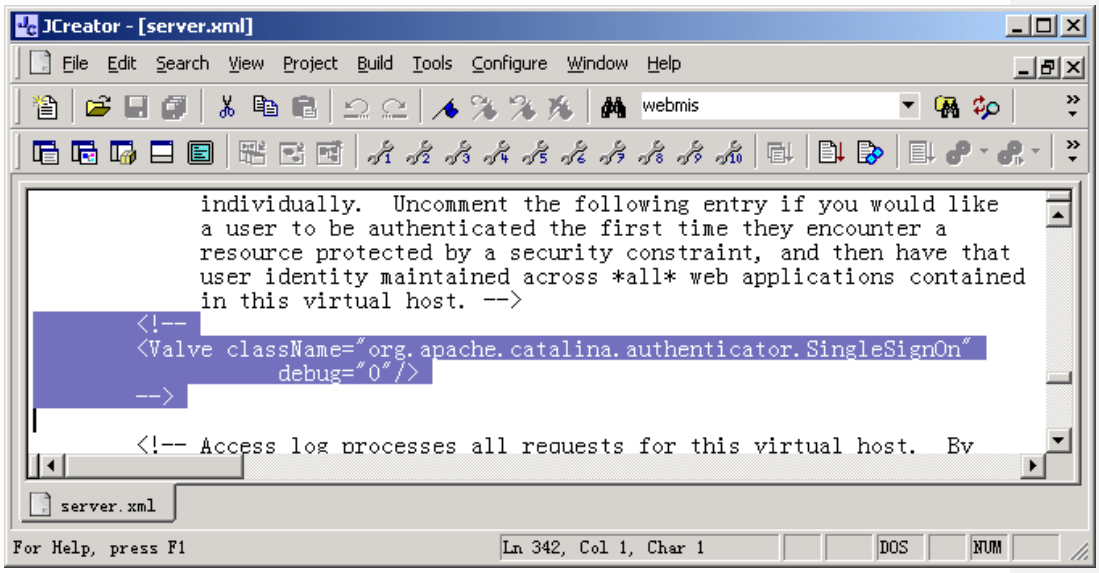
如果你运行了多个 Web 应用，并且每个应用都需要进行单独的用户验证，那这看起来就有点像你在与你的用户搏斗。用户们不知道怎样才能把多个分离的应用整合成一个单独的系统，所有他们也就不知道他们需要访问多少个不同的应用，只是很迷惑，为什么总要不停的登录。

2、Tomcat 中的“Single Sign-On”特性及配置

其主要的特性是能够允许用户在访问同一虚拟主机下所有 Web 应用时，只需登录一次。为了使用这个功能，你只需要在 C:\jakarta-tomcat-5.0.19\conf \server.xml 文件中的 Host 标签上添加一个 SingleSignOn Valve 元素即可，如下所示：

<Valve className="org.apache.catalina.authenticator.SingleSignOn" debug="0"/>

在 Tomcat 初始安装后，server.xml 的注释里面包括 SingleSignOn Valve 配置的例子，你只需要去掉注释（**在 339 行左右**），即可使用。那么，任何用户只要登录过一个应用，则对于同一虚拟主机下的所有应用同样有效。

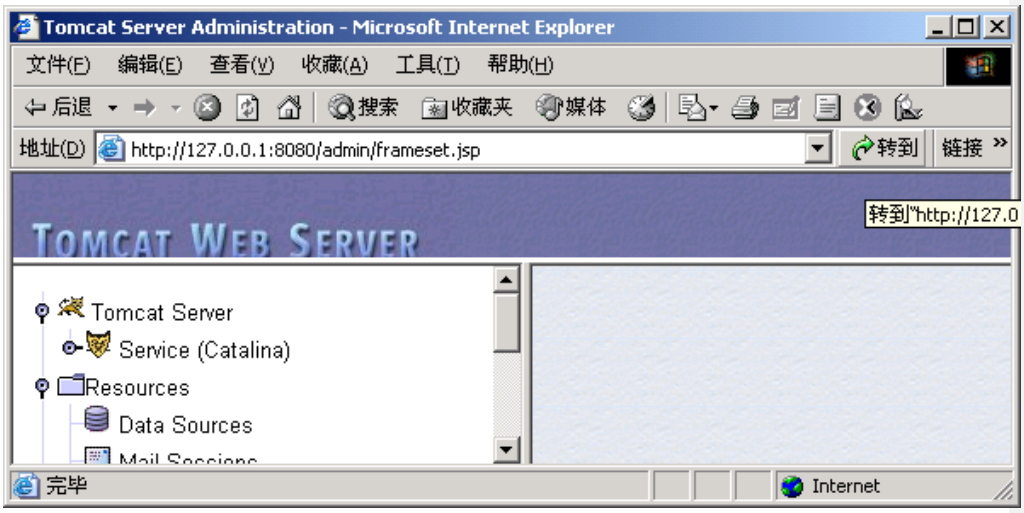


3、测试单点登录

（1） 直接进入前面的 Form 验证所产生的 Web 应用
（http://127.0.0.1:8080/FormSafeWebApp/ProtectedDirOne/）将出现要求登录的页面

在表单中输入用户名称为 admin（前面在 tomcat-users.xml 文件中所设置的某一用户名称），密码为 12345678。然后点击“提交”，将以用户名 admin 进行成功登录该 Web 应用。

（2）再在该浏览器窗口内（不能在新窗口，否则会成为另一用户）直接输入 http://127.0.0.1:8080/admin/frameset.jsp，此时将以 admin 的用户浏览另一 Web 应用。观察能否直接进入 Tomcat 的系统管理的页面，此时应该可以并且出现下面的页面。



如果新开一浏览器窗口并直接输入 http://127.0.0.1:8080/admin/frameset.jsp，看能否直接进入 Tomcat 的系统管理的页面，此时将会出现要求登录的页面。

4、使用 single sign-on valve 所应该注意的问题

- value 必须被配置和嵌套在相同的 Host 元素里，**并且所有需要进行单点验证的 web 应用（必须通过 context 元素定义）都位于该 Host 下**。
- 包括共享用户信息的 realm 必须被设置在同一级 Host 中或者嵌套之外。
- 不能被 context 中的 realm 覆盖。
- 使用单点登录的 web 应用最好使用一个 Tomcat 的内置的验证方式（Basic 或者 Form）（被定义在 web.xml 中的<auth-method>中），这比自定义的验证方式强。
- 单点登录需要使用 cookies。

九、对 Struts 中的 Action 进行授权利

1、应用的意义

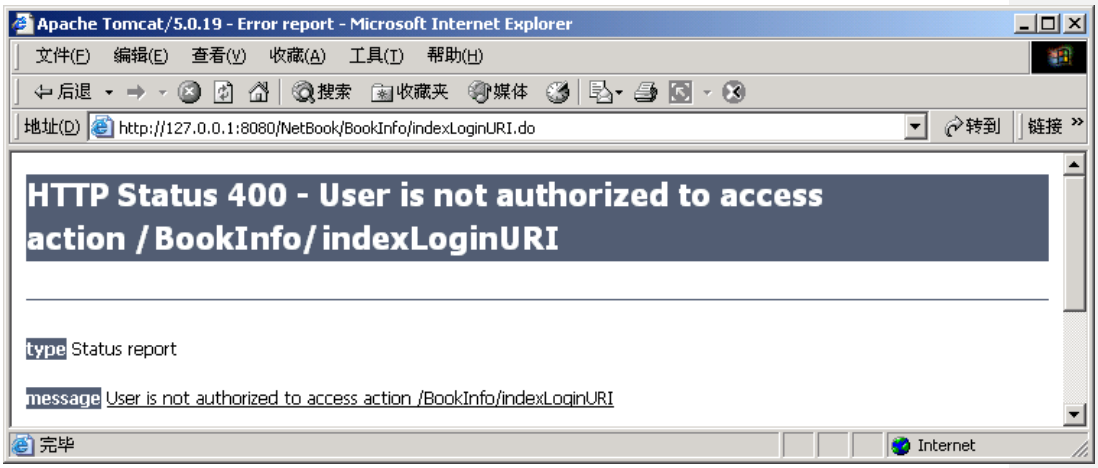
在某些应用下，如果 Action 类执行的功能比较重要，可以对该 Action 类进行授权利以实现只有特定角色的用户能够访问，此时可以在 struts-config.xml 文件中进行配置

2、在 struts-config.xml 文件中进行配置

```
<action-mappings>
        <action    path="/BookInfo/indexLoginURI"              type="netbook.IndexLoginAction"
name="IndexLoginForm"
            scope="request"      validate="true"      input="/index.jsp" roles="admin">
            <forward name="loginSuccess"   path="/BookInfo/ShowBookInfo.jsp" />
            <forward name="loginFailure" path="/BookInfo/gotoIndex.htm"/>
        </action>
  </action-mappings>
```

3、 问该 Action

如果访问者不是 admin 角色的用户，此时将出现如下的错误



九、在 Tomcat 上配置虚拟主机

1、Tomcat 服务器的 server.xml 文件

（1）Tomcat 组件

Tomcat 服务器是由一系列可配置的组件构成，其中核心组件是 Catalina Servlet 容器，它是所有其他 Tomcat 组件的顶层容器。Tomcat 的组件可以在<CATALINA_HOME>/conf/server.xml 文件中进行配置，每个 Tomcat 组件在 server.xml 文件中对应一种配置元素。

（2）Tomcat 组件之间的关系

以下代码以 XML 的形式展示了各种 Tomcat 组件之间的关系：

```
<Server>
    <Service>
        <Connector />
        <Engine>
            <Host>
                <Context>
                </Context>
            </Host>
        </Engine>
    </Service>
</Server>
```

（3）各个 Tomcat 组件的说明

在以上 XML 代码中，每个元素都代表一种 Tomcat 组件。这些元素可分 4 类：

- 顶层类元素：主要包括 <Server> 元素和 <Service> 元素，他们位于整个配置文件的顶层。
- 连接器类元素：代表了介于客户与服务之间的通信接口，负责将客户的请求发送给服务器，并将服务器的响应结果传递给客户。
- 容器类元素：代表处理客户请求并生成响应结果的组件，有 3 种容器类元素，它们是 Engine、Host 和 Context。Engine 组件为特定的 Service 组件处理所有的客户请求，Host 组件为特定的虚拟主机处理所有客户请求，Context 组件为特定的 Web 应用处理所有客户请求。
- 嵌套类元素：嵌套类元素代表了可以加入到容器中的组件，如 <Logger> 元素、<Valve> 元素和 <Realm> 元素。

（4）<Server> 元素

<Server> 元素代表整个 Catalina Servlet 容器，它是 Tomcat 实例的顶层元素，由 org.apache.catalina.Server 接口来定义。

<Server> 元素中可以包含一个或多个 <Service> 元素，但 <Server> 元素不能做为任何其他元素的子元素。

（5）<Service> 元素

<Service> 元素由 org.apache.catalina.Service 接口来定义，它包含一个 <Engine> 元素，以及一个或多个 <Connector> 元素，这些 <Connector> 元素共享同一个 <Engine> 元素。

（6）<Connector> 元素

<Connector> 元素由 org.apache.catalina.Connector 接口来定义。<Connector> 元素代表和客户程序实际交互的组件，它负责接受客户请求，以及向客户返回响应结果。

（7）<Engine> 元素

<Engine> 元素由 org.apache.catalina.Engine 接口来定义。每个 <Service> 元素只能包含一个 <Engine> 元素。<Engine> 元素处理在同一个 <Service> 元素中所有 <Connector> 元素接受到的客户请求。

<Engine> 元素可包括如下子元素：

<Loggor>
<Realm>
<Valve>
<Host>

（8）<Host> 元素

<Host> 元素由 org.apache.catalina.Host 接口来定义。一个 <Engine> 元素中可以包含多个 <Host> 元素。每个 <Host> 元素定义了一个虚拟主机，他可以包含一个或多个 Web 应用。

<Host> 元素可包括如下子元素：

<Loggor>
<Realm>
<Valve>
<Context>

（9）<Context> 元素

<Context> 元素由 org.apache.catalina.Context 接口来定义。<Context> 元素是使用最频繁的元素。每个 <Context> 元素代表运行在虚拟主机上的单个 Web 应用。一个 <Host> 元素中可以包含多个 <Context> 元素。

<Context> 元素可包括如下子元素：

<Loggor>

<Realm>

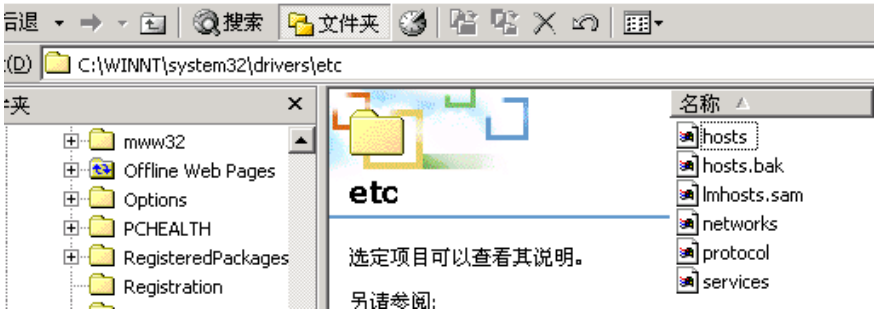
<Valve>

<Resource>

<ResourceParams>

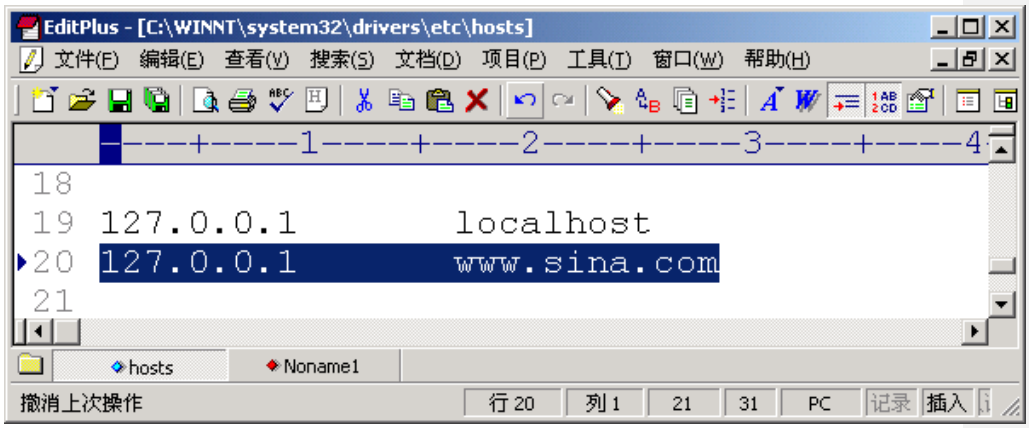
2、为主机配置域名

（1）编辑 C:\WINNT\system32\drivers\etc 下的 hosts 文件，在其中增加对本机 IP 地址的映射的域名



（2）本例为

127.0.0.1 www.sina.com



（3）保存该文件

3、修改 Tomcat 下的 C:\jakarta-tomcat-5.0.19\conf\server.xml 文件以增加一个主机 Host 的设置

Host 标记是用来配置虚拟主机的，就是可以多个域名指向一个 tomcat，<context>是 Host 标记的子元素吧，表示一个虚拟目录，它主要有两个属性，path 就相当于虚拟目录名字，而 docbase 则是具体的文件位置。

<Host name="www.sina.com" debug="0" appBase="webapps" unpackWARs="true" autoDeploy="true" xmlValidation="false" xmlNamespaceAware="false">

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

</Host>

注意：

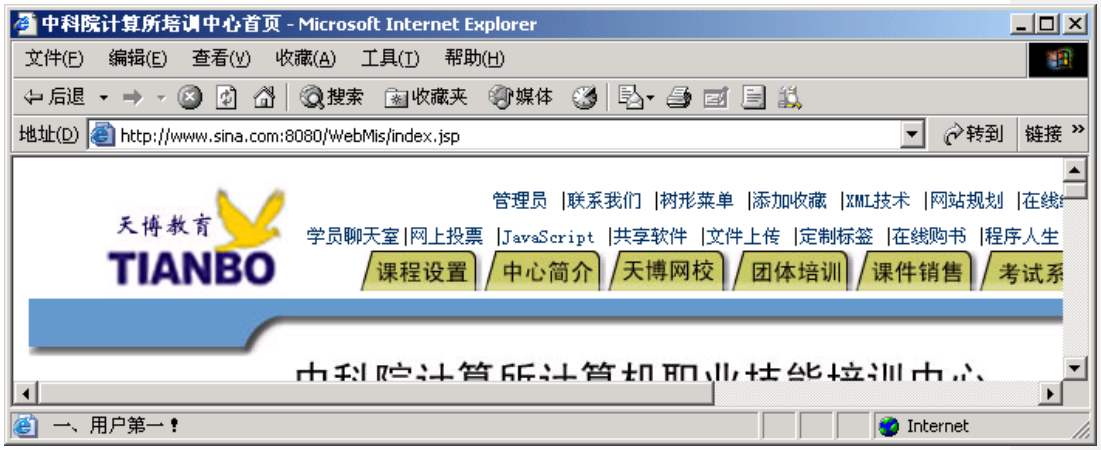
（1）可以将 Tomcat 自己带的 localhost 主机的 Host 的整个设置全部拷贝，然后将“localhost”改名为 www.sina.com 即可以。

（2）必须保证在<Host></Host>之间至少有一个<Context path="" docBase="ROOT" debug="0" />的根 Web 应用程序的设置项目存在。

（3）可以根据应用的需要，在<Host></Host>之间设置其它的基于该主机名称下的其它 Web 应用程序的**<Context>**设置。

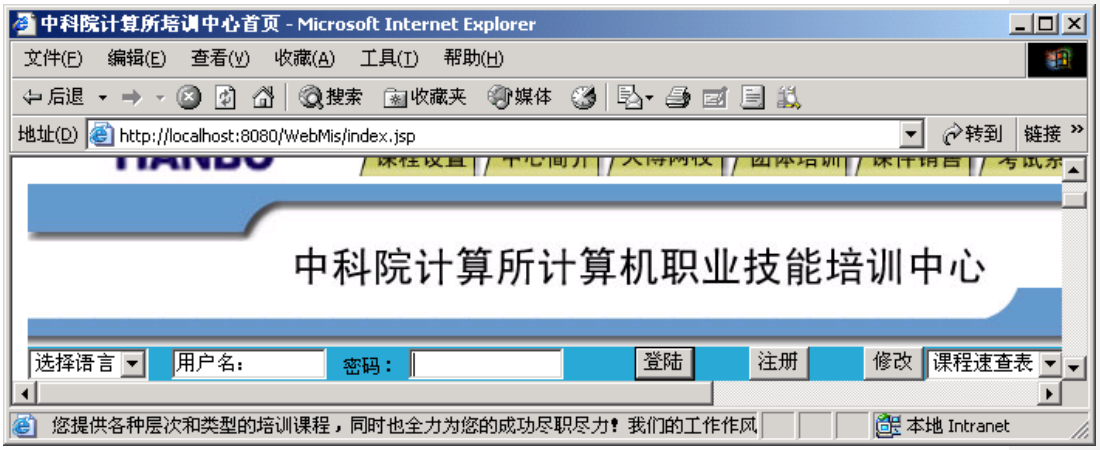
4、启动 Tomcat 后再浏览本 Web 应用

输入 http://www.sina.com:8080/WebMis/index.jsp



5、本例也可以以 localhost 缺省的主机名称来访问

http://localhost:8080/WebMis/index.jsp



四、在 Tomcat 中实现系统和 Web 管理的配置

1、配置系统管理（Admin Web Application）

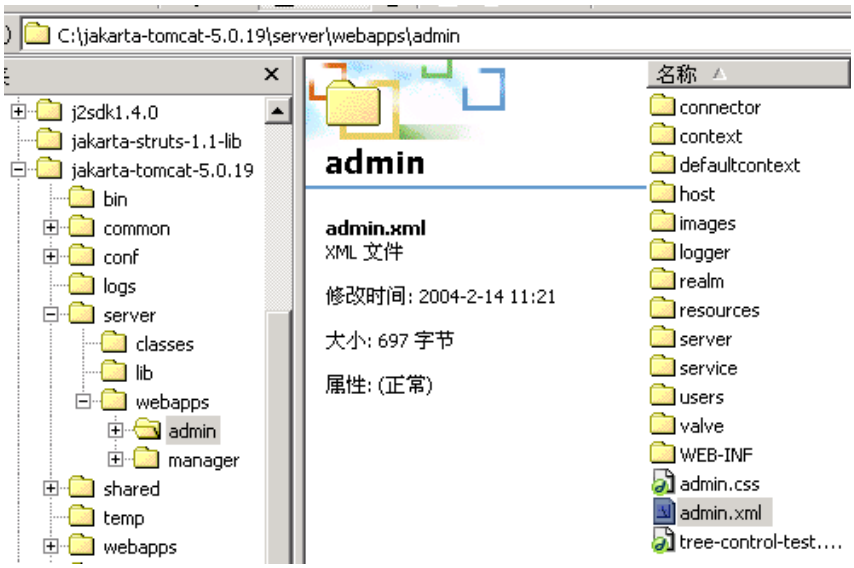
（1）概述

大多数商业化的 J2EE 服务器都提供一个功能强大的管理界面（如 Weblogic 的管理控制台），且大都采用易于理解的 Web 应用界面。Tomcat 按照自己的方式，同样提供一个成熟的管理工具，并且丝毫不逊于那些商业化的竞争对手。

Tomcat 的 Admin Web Application 最初在 4.1 版本时出现，当时的功能包括管理 context、data source、user 和 group 等。当然也可以管理像初始化参数，user、group、role 的多种数据库管理等。在后续的版本中，这些功能将得到很大的扩展，但现有的功能已经非常实用了。

（2）系统管理 Web 应用程序

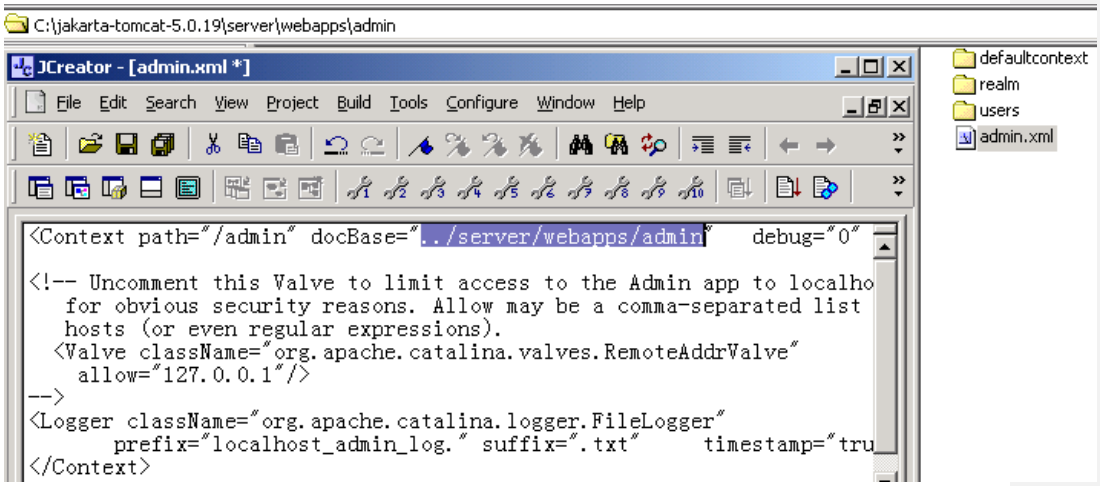
Tomcat 中的 Admin Web Application 被定义在自动部署文件：C:\jakarta-tomcat-5.0.19\server\webapps\admin\ admin.xml 中（请见下图所示）。



（3）编辑 admin.xml 文件

通过编辑 admin.xml 文件，以确定 Context 中的 docBase 参数设置为 Admin Web Application 所在的目录路径（应该是绝对路径）。作为另外一种选择，你也可以删除这个自动部署文件，而在 C:\jakarta-tomcat-5.0.19\conf\server.xml 文件中建立一个 Admin Web Application 的 context，效果是一样的。

你不能管理 Admin Web Application 这个应用，换而言之，除了删除 CATALINA_BASE/webapps/admin.xml ，你可能什么都做不了。

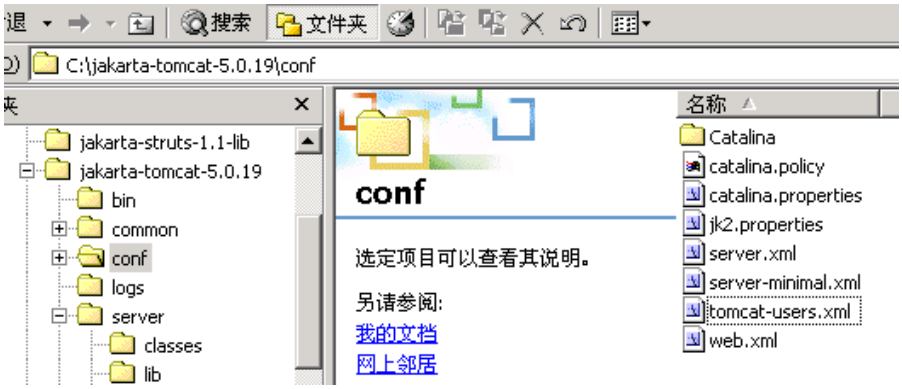


注意：如果将其中的被注释掉的<Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="127.0.0.1"/>打开，将能够限制访问 Admin Web Application 的程序主机为本机（服务器主机）；当然也可以设置为其它的主机 IP 地址（如设置为 Web 管理员所的工作主机）。

（4）在 C:\jakarta-tomcat-5.0.19\conf\ tomcat-users.xml 文件中添加系统管理员的角色和系统

管理员

Tomcat 中提供 UserDatabaseRealm（默认），这样我们可以根据管理的需要添加不同的用户角色和与该角色相配置的用户名称和密码



- 添加用户角色

  <role name="admin"/>

- 添加与该角色相配置的用户名称和密码

  <user name="admin" password="12345678" roles="admin"/>

当你完成这些步骤后，请重新启动 Tomcat，访问 http://localhost:8080/admin，你将看到一个登录界面。Admin Web Application 程序采用基于容器管理的安全机制，并采用了 Jakarta Struts 框架。下面是在原来的 tomcat-users.xml 文件中再添加了两个角色 admin 和 manager，同时也添加了与该两个角色相配置的用户 admin 和 manager。

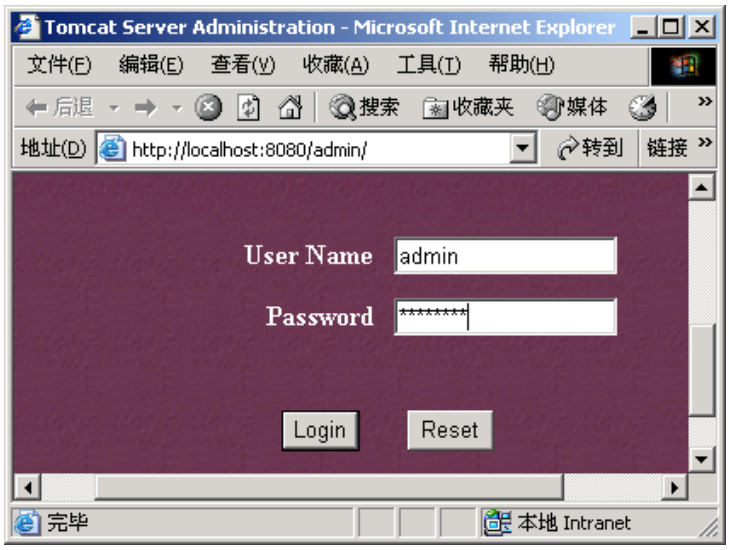
```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="role1"/>
  <role rolename="tomcat"/>
  <role rolename="admin"/>
  <role rolename="manager"/>
  <user username="admin" password="12345678" roles="admin"/>
  <user name="manager" password="12345678" roles="manager"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
</tomcat-users>
```

（5）登录 Admin Web Application 程序

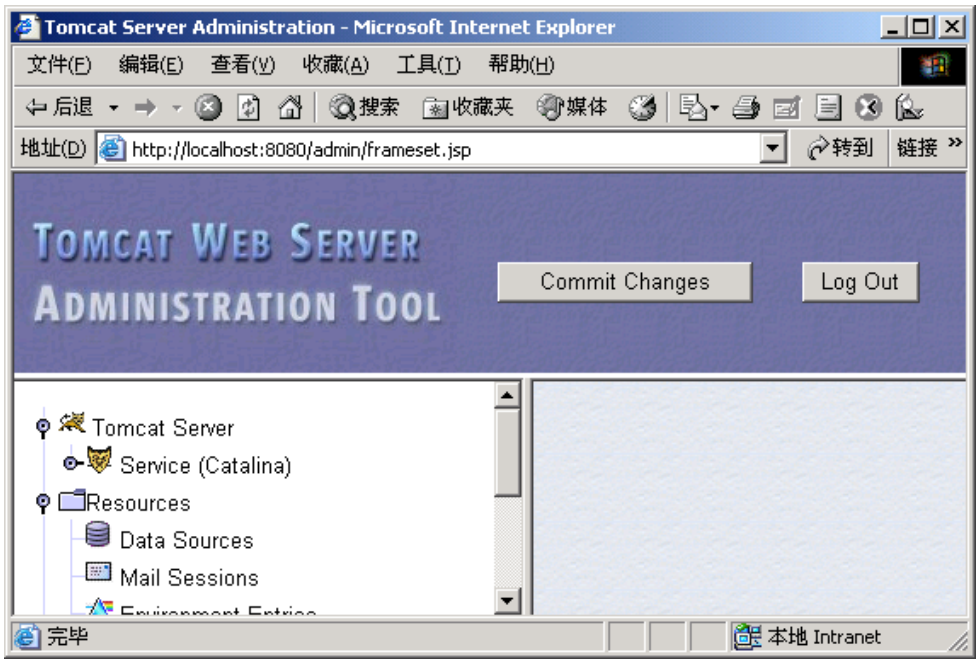
输入 http://localhost:8080/admin/进入系统管理员的登录页，然后在页中

输入用户名称：admin

密码： 12345678



将进入系统管理的界面，在该系统管理的程序中将可以配置各种资源如 Data Sources、Mail Sessions、Environment Entries，并且也可以管理 Users 和 Groups 以及 Roles 等功能。



2、配置应用管理（Manager Web Application）

（1）概述

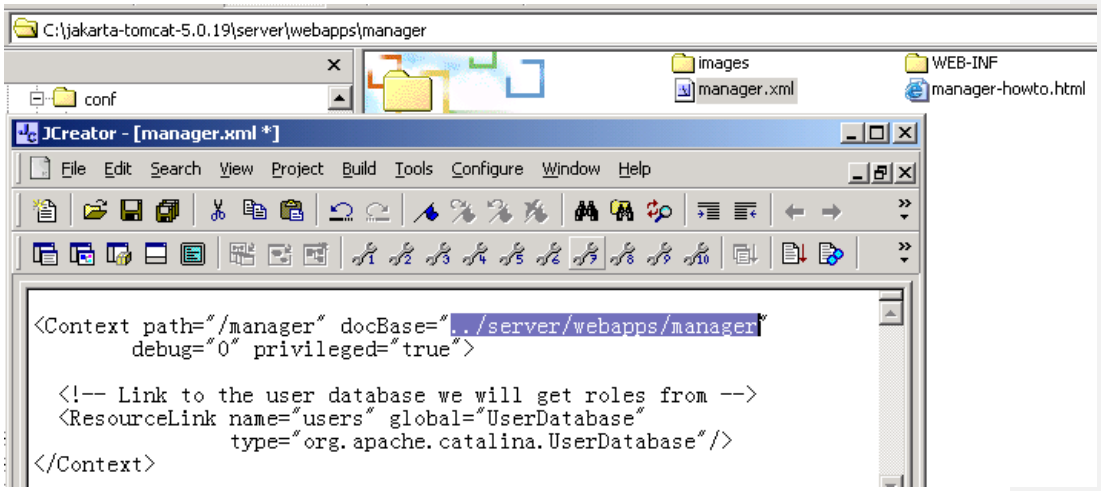
Tomcat 中所提供的 Manager Web Application 让你通过一个比 Admin Web Application 更为简单的用户界面，执行一些与 Web 应用任务相关的一些管理功能。

（2）Manager Web Application 程序

Manager Web Application 被被定义在一个自动部署文件中 C:\jakarta-tomcat-5.0.19\server\webapps\manager\manager.xml 。

（3）编辑 manager.xml 文件

通过编辑这个文件，以确保其中的 context 中的 docBase 属性参数是 C:\jakarta-tomcat-5.0.19\server\webapps\manager 的绝对路径。



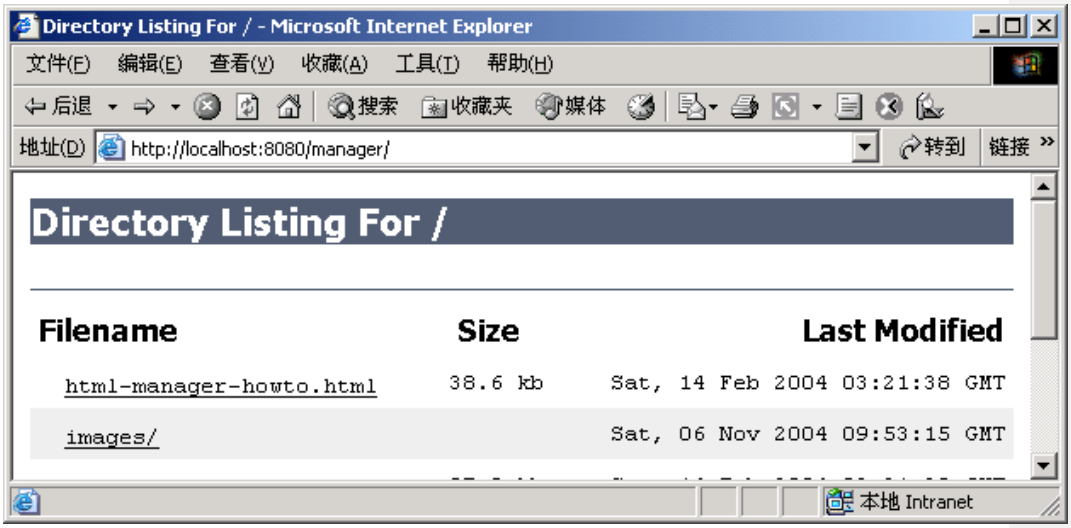
（4）在 C:\jakarta-tomcat-5.0.19\conf\ tomcat-users.xml 文件中添加 Web 管理员的角色和 Web 管理员

- 添加用户角色
  <role name=" manager "/>
- 添加与该角色相配置的用户名称和密码
  <user name="manager" password="12345678" roles="manager"/>

（5）登录 Web 管理员的页面

- 文本型管理界面
  然后重新启动 Tomcat，输入 http://localhost:8080/manager/，将进入看到一个很朴素的文

本型管理界面

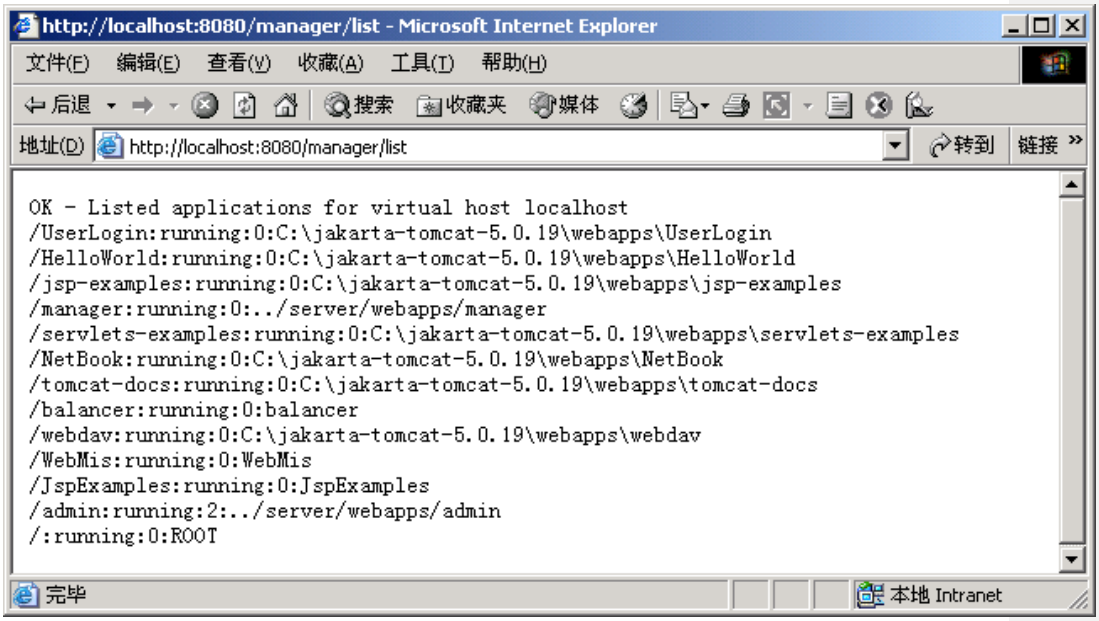


如果输入 http://localhost:8080/manager/list，将进入一个登录管理界面，然后

输入用户名称：manager（前面在 tomcat-users.xml 中设置的）

密码：12345678

将显示出

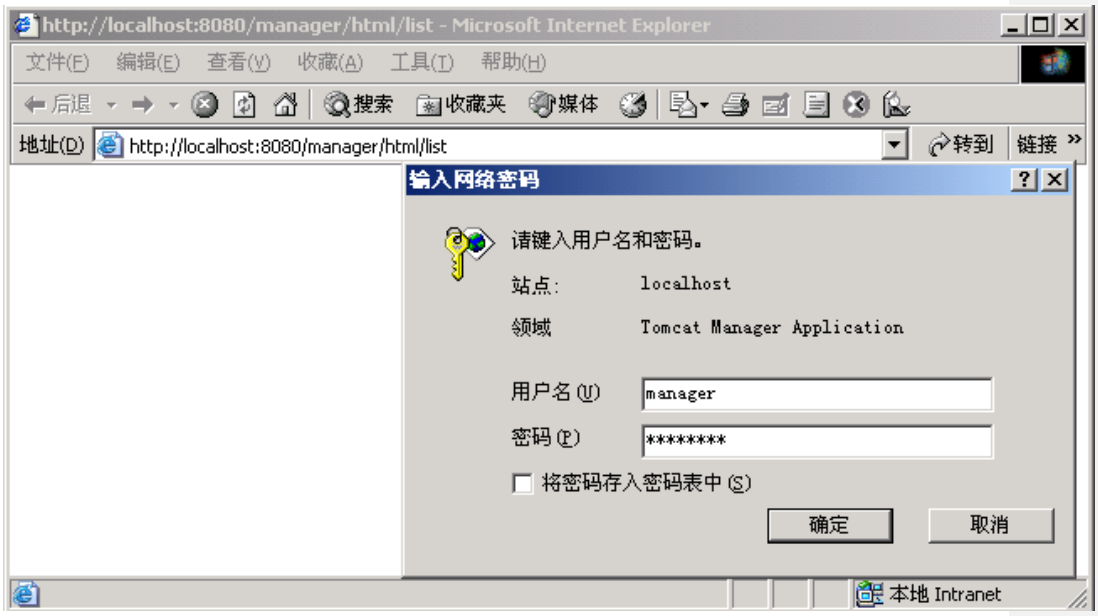


- HTML 型管理界面

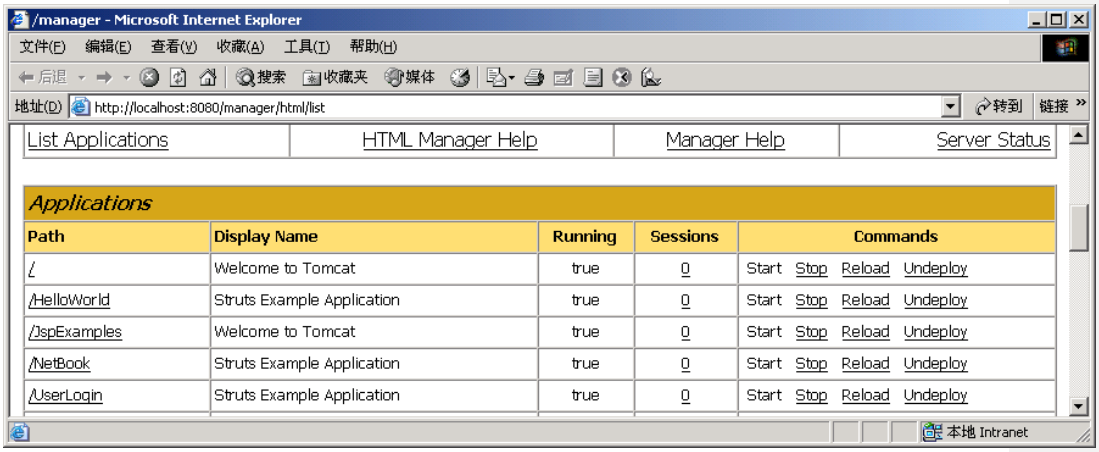
输入 http://localhost:8080/manager/html/list，将出现如下的页面，然后再

输入用户名称：manager

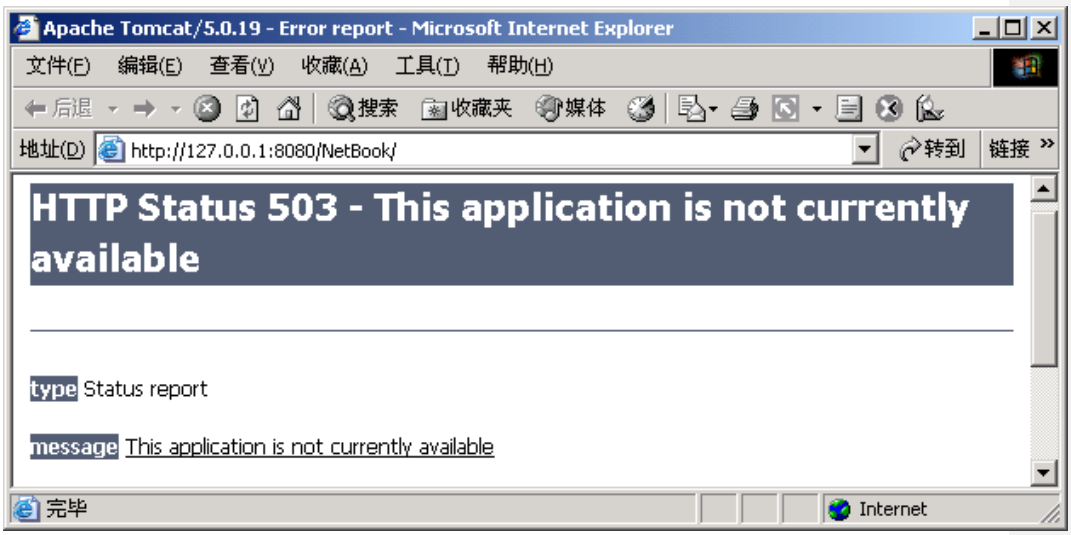
密码：12345678



将出现 Web 方式的管理页面

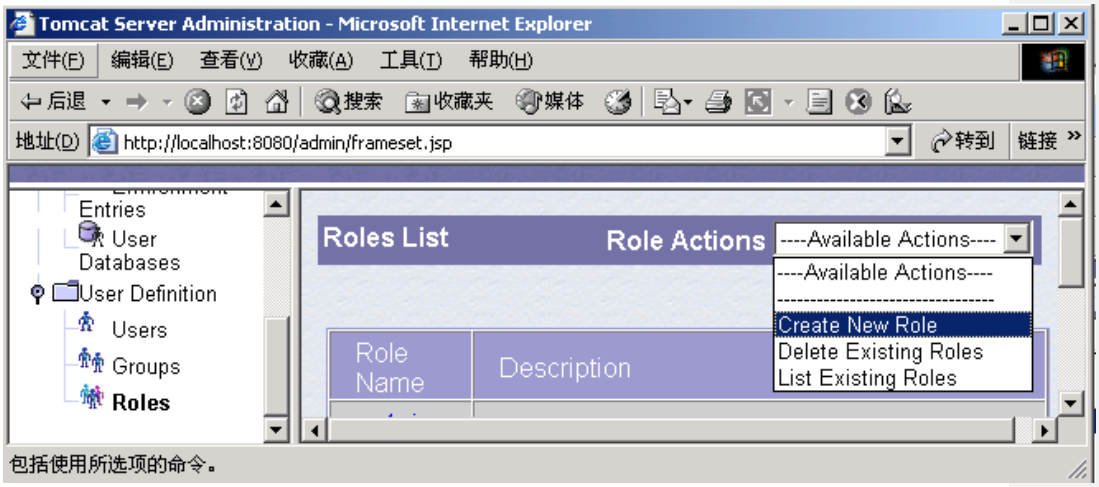


Manager application 可以让用户在没有系统管理特权的基础上，部署安装新的 Web 应用，以用于测试。同时也可以对所部署的 Web 应用程序的工作状态进行控制（Start 或者 Stop），以免重新启动服务器（这在对 web.xml 等配置的内容发生改变的情况下，特别有效）。当有用户尝试访问这个被停止的应用时，将看到一个 503 的错误——“503 - This application is not currently available”。

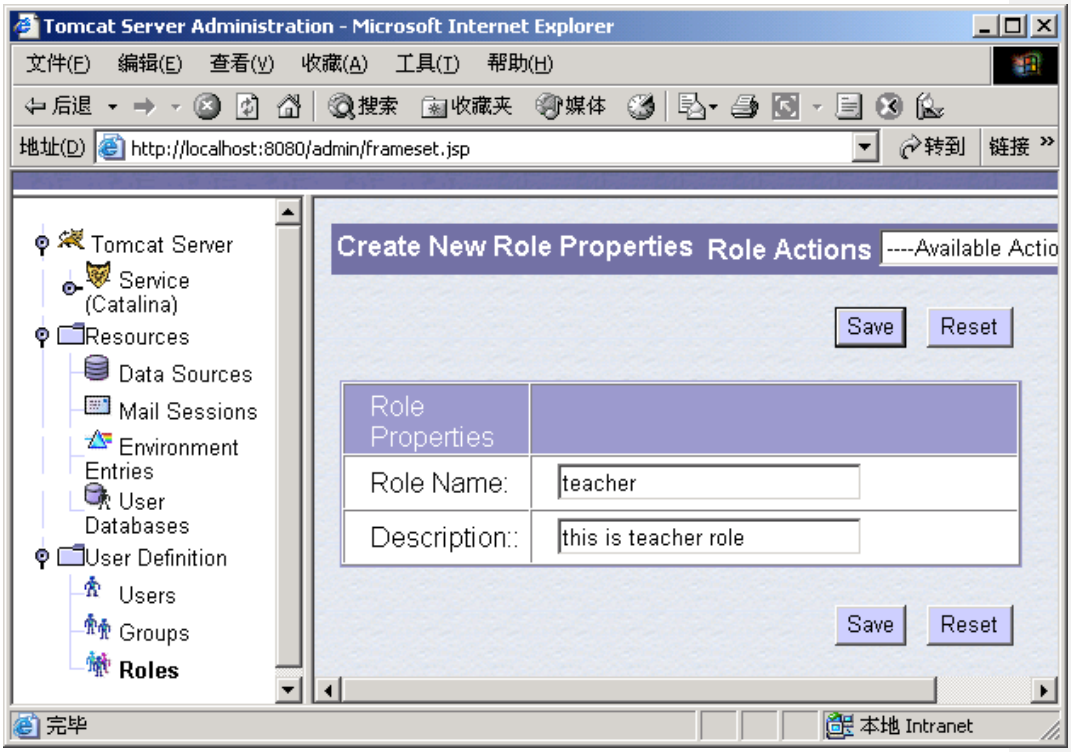


3、配置各种用户角色、用户组和用户

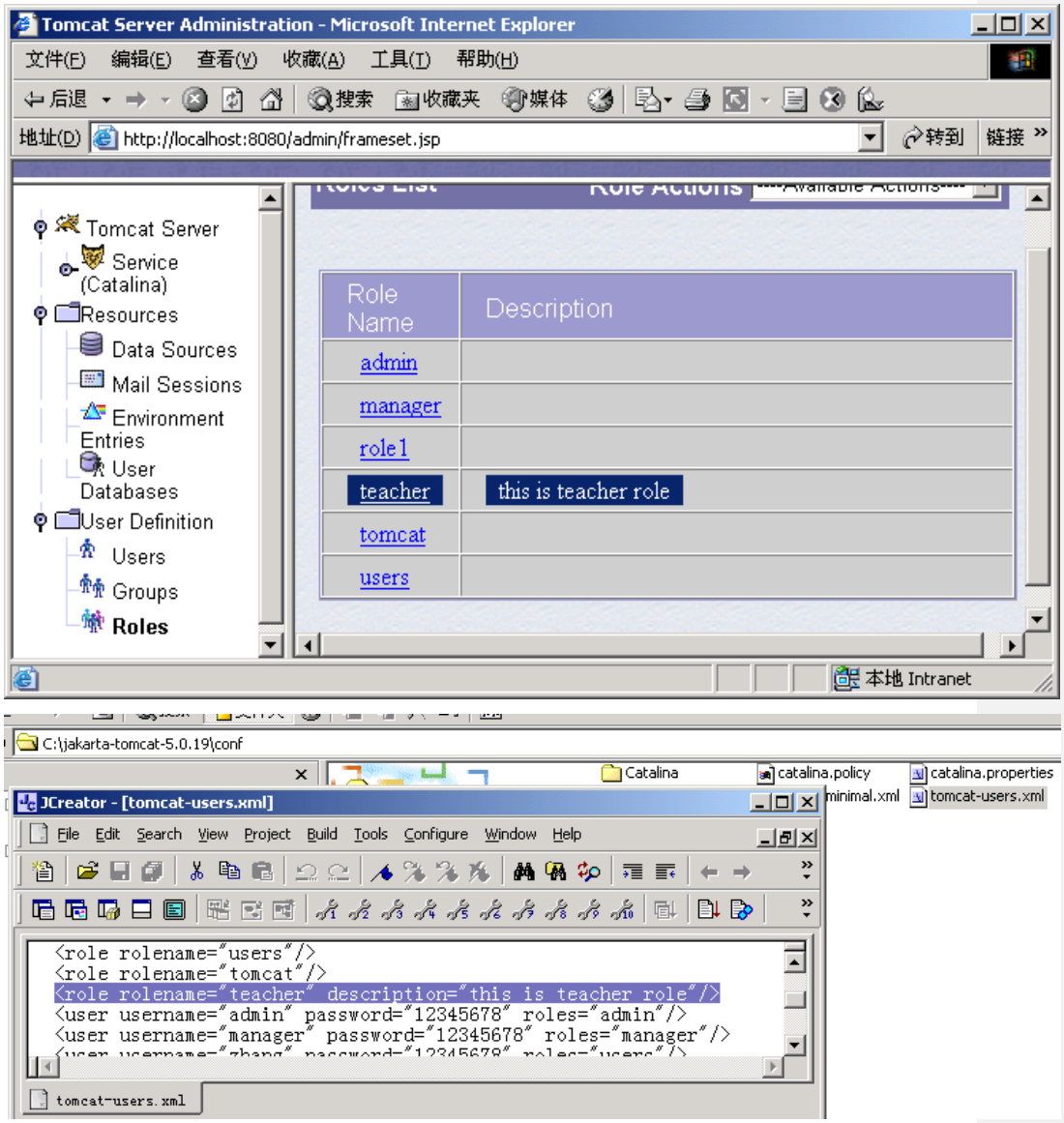
（1）添加用户角色：在 admin 的界面中点击左面的 Roles 节点，然后在右面的下拉列表框中选择 Create New Role 项目。



然后输入角色的名称和描述

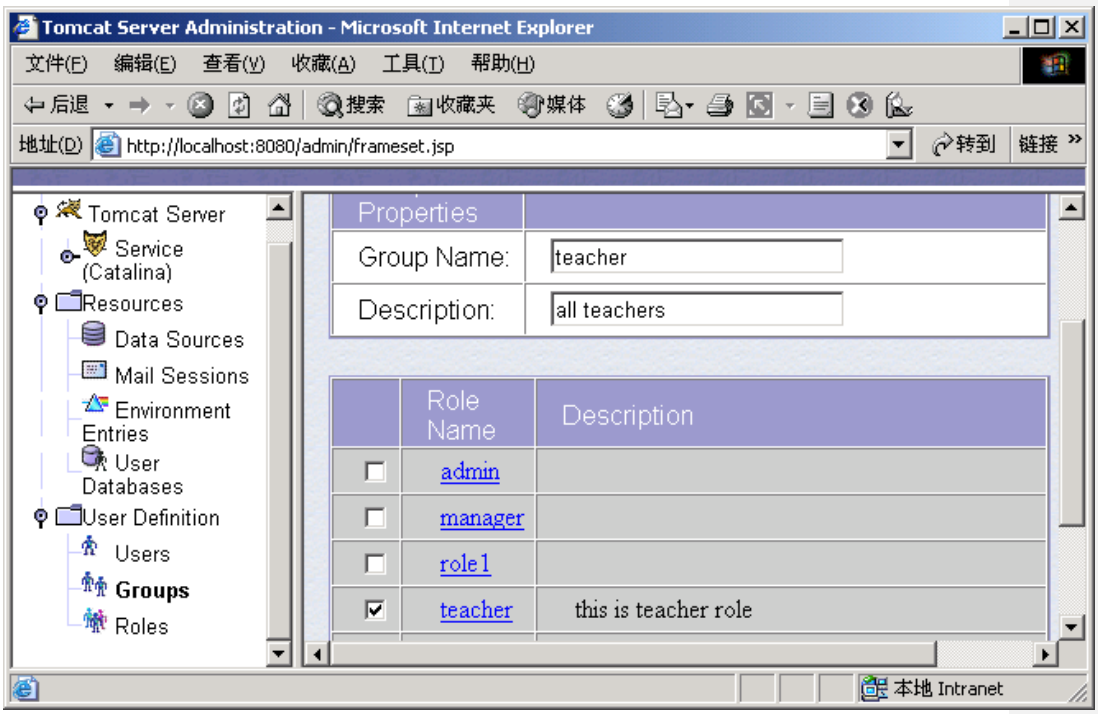


最后点击“保存”，将存储在 C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml 文件中并且在管理界面中显示出。

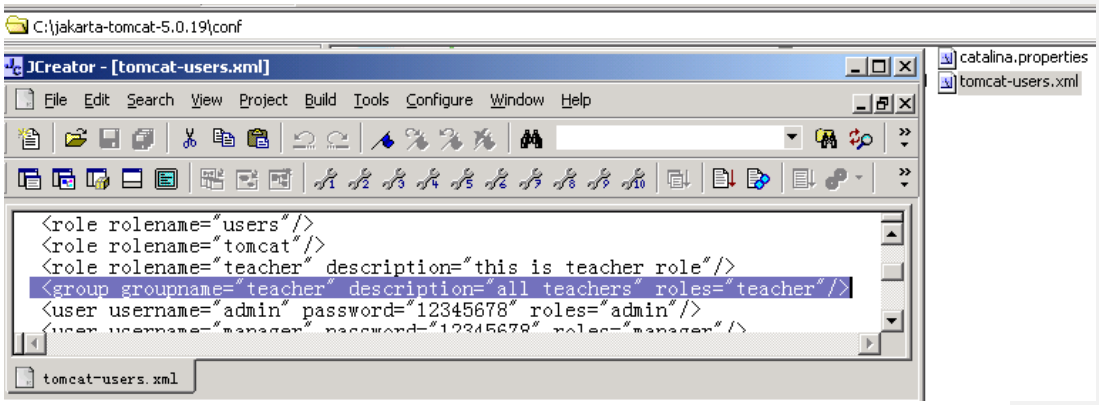


（2）添加用户组：在 admin 的界面中点击左面的 Groups 节点，然后在右面的下拉列表框中选择 Create New Group 项目。

然后输入组的名称和描述，并且设置该组的角色。所应该注意的是，给组分配角色，则意味着该组中的各个成员（用户）将具有该角色所分配的各种权限。

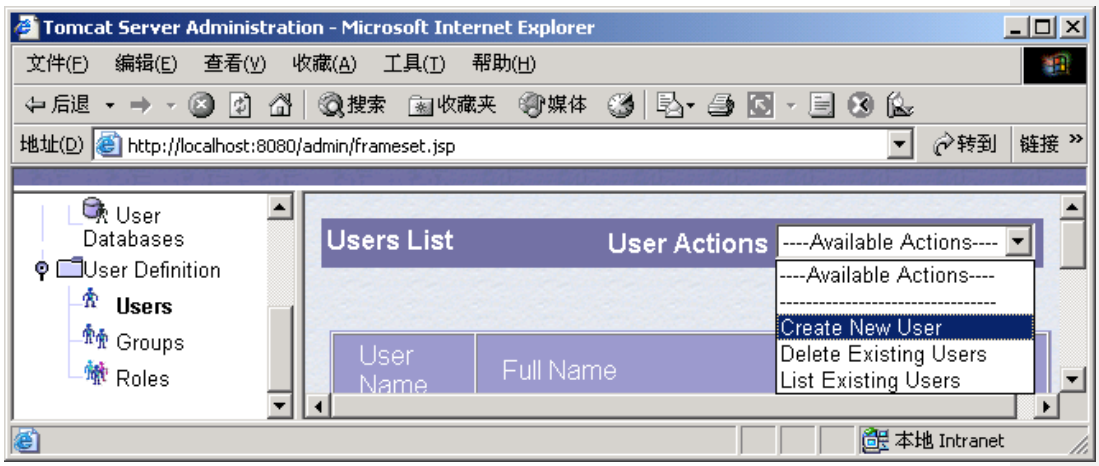


最后点击“Save”以保存它（仍然放在 C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml 文件中）



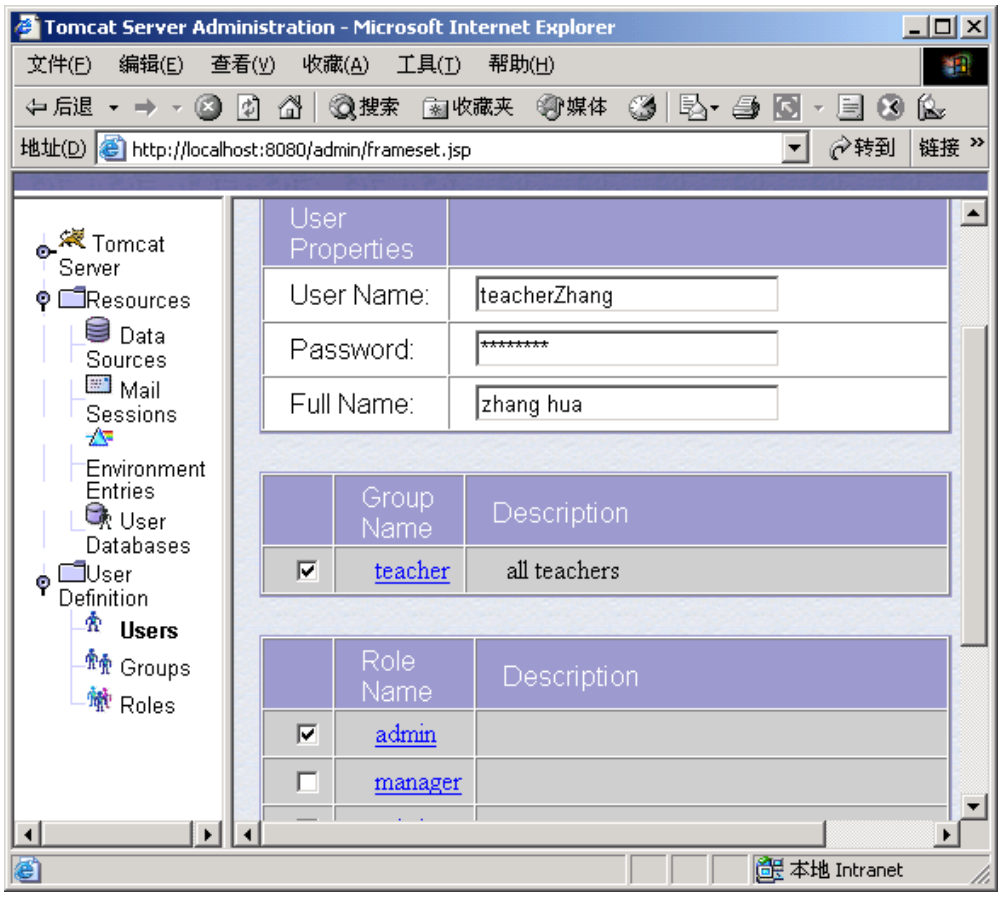
（3）添加属于某一用户组内的用户

在 admin 的界面中点击左面的 Users 节点，然后在右面的下拉列表框中选择 Create New User 项目。

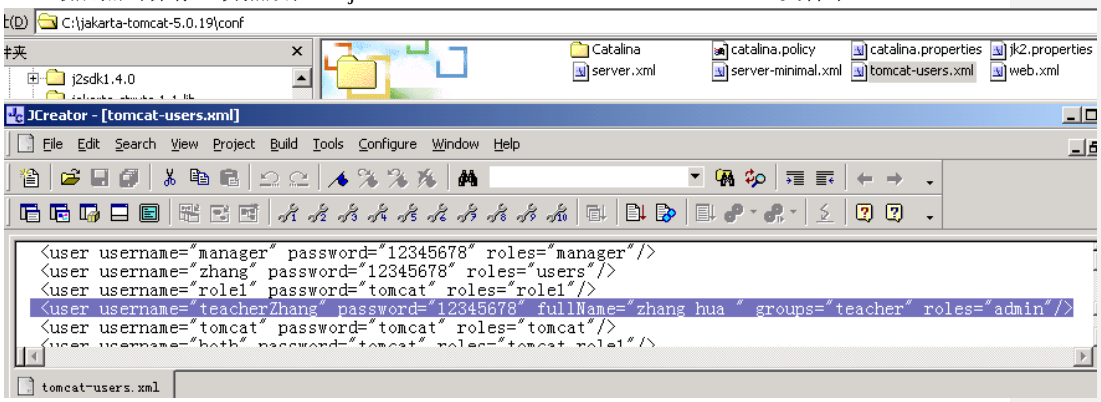


然后该用户的名称同时包括全名称、密谋，并且设置该用户所属的用户组；同时也可以为该用户再设置其它的角色以使该用户除了具有用户组的通用的权利以外，还具有其他方面的权利。

下面对“teacherZhang”这个用户进行设置，同时他也是系统管理员，因此将下面的 admin 的角色也选中。



最后点击保存（仍然放在 C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml 文件中）



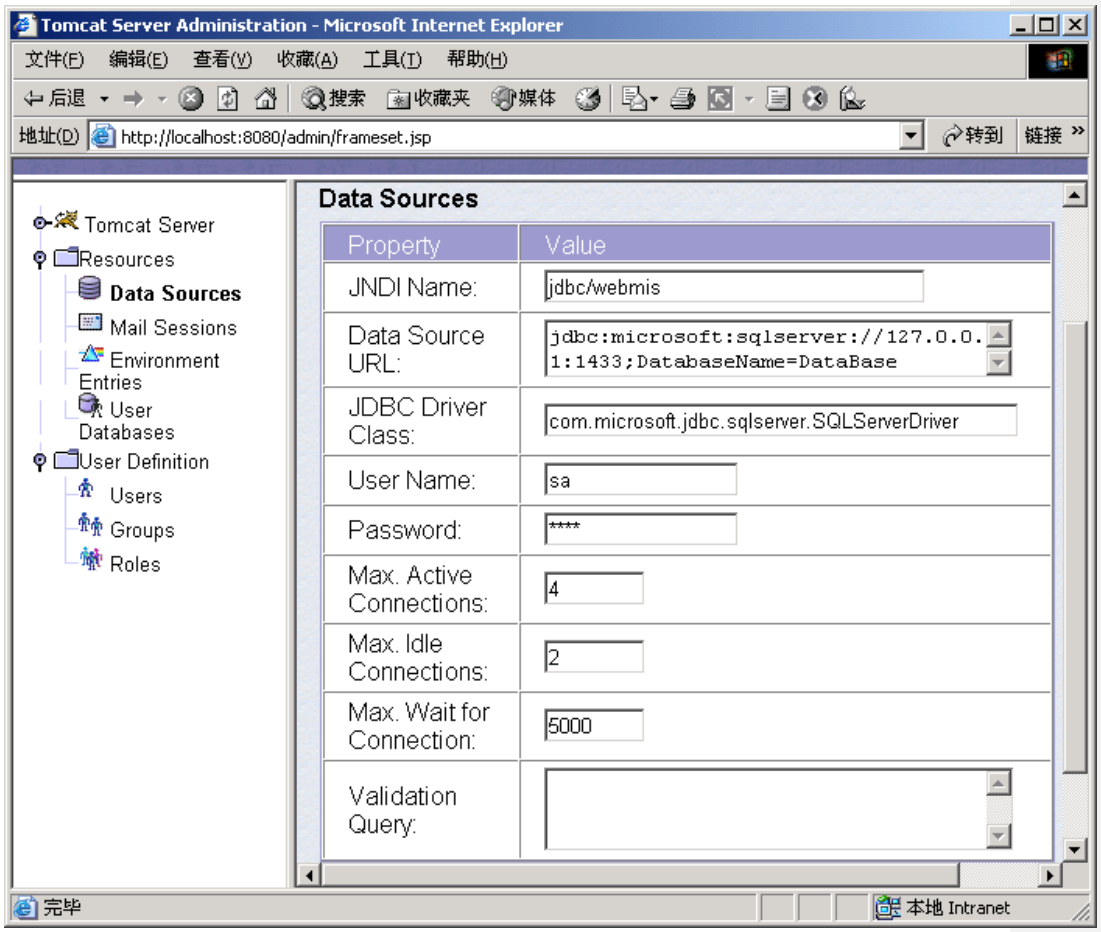
4、添加其它的系统资源

（1）DataSource

在 admin 的界面中点击左面的 DataSourcs 节点，然后在右面的下拉列表框中选择 Create New

DataSource 项目。

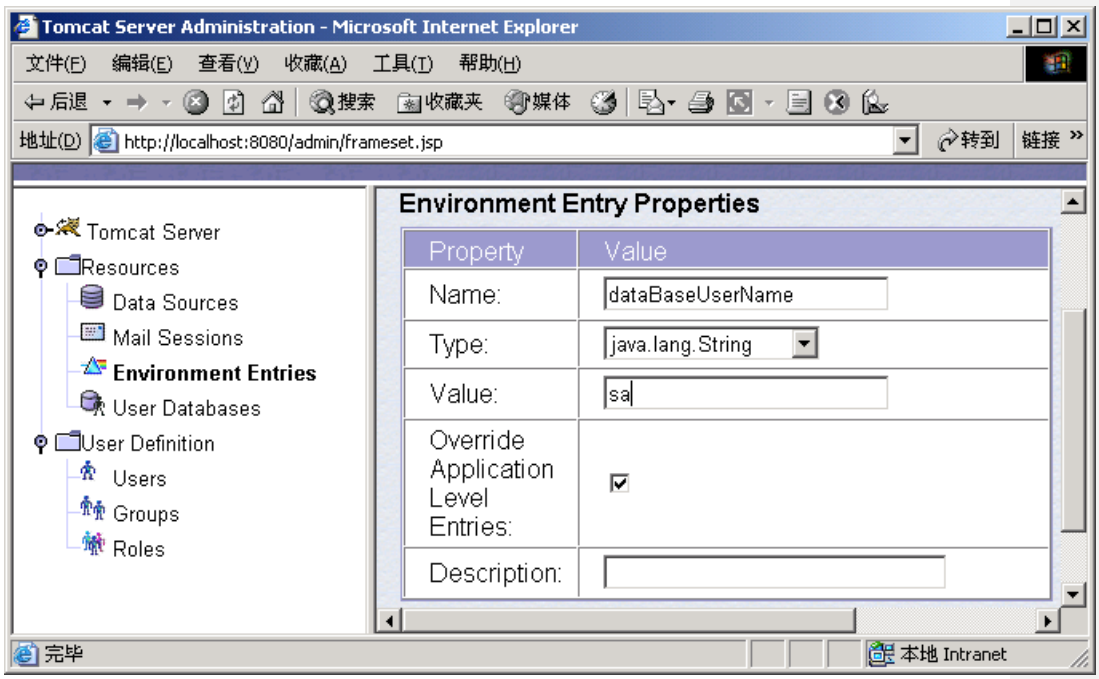
在各个输入的项目中根据数据库的特性进行输入。最后点击“Save”以保存。



（2）添加环境变量

在 admin 的界面中点击左面的 Environment Entries 节点，然后在右面的下拉列表框中选择 Create New Env Entry 项目。

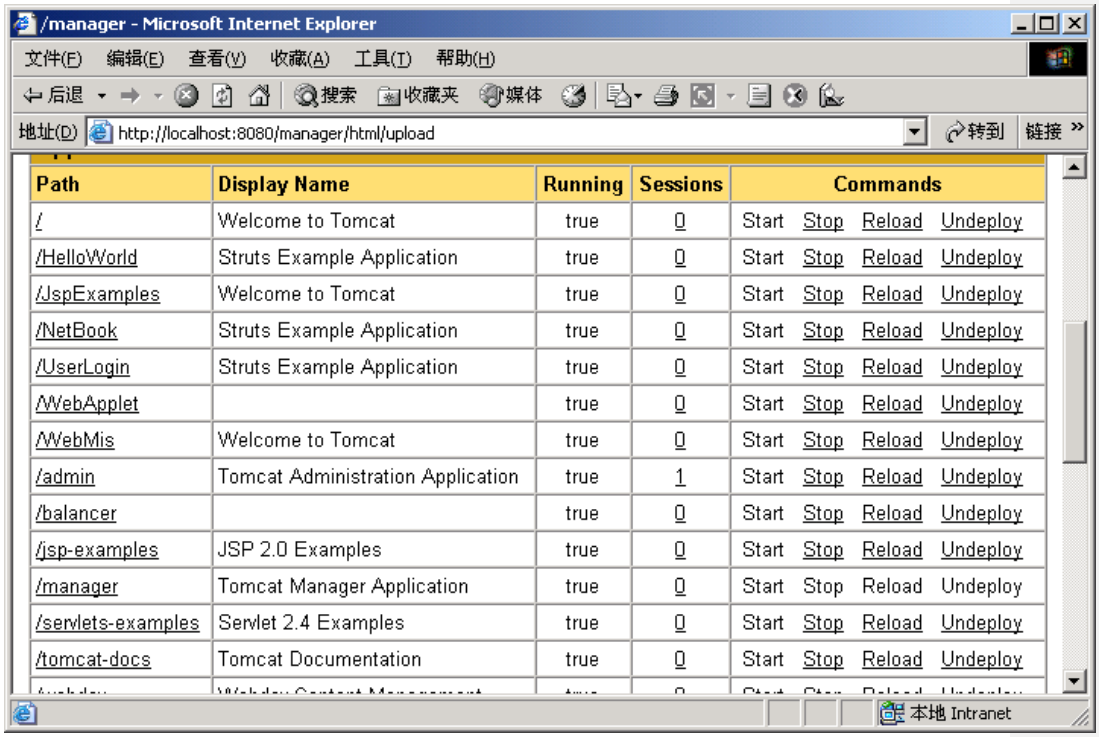
在各个输入的项目中根据数据库的特性进行输入。最后点击“Save”以保存。



5、对 Web 应用程序进行管理

（1）输入 http://localhost:8080/manager/html/list，将出现登录页并且进行登录，然后再进入 Tomcat Web Application Manager

（2）查看在 Web 服务中所发布的各个 Web 应用



Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/HelloWorld	Struts Example Application	true	0	Start Stop Reload Undeploy
/JspExamples	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/NetBook	Struts Example Application	true	0	Start Stop Reload Undeploy
/UserLogin	Struts Example Application	true	0	Start Stop Reload Undeploy
/WebApplet		true	0	Start Stop Reload Undeploy
/WebMis	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/admin	Tomcat Administration Application	true	1	Start Stop Reload Undeploy
/balancer		true	0	Start Stop Reload Undeploy
/jsp-examples	JSP 2.0 Examples	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/servlets-examples	Servlet 2.4 Examples	true	0	Start Stop Reload Undeploy
/tomcat-docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy

（3）启动或者终止、移除某一 Web 应用：

点击该 Web 应用右面的 Stop 链接，也可以点击 Start 再次启动它。Undeploy（移除）一个 Web 应用，只是指从 Tomcat 的运行拷贝中删除了该应用，如果你重新启动 Tomcat，被删除的应用将再次出现（也就是说，移除并不是指从硬盘上删除）。

（4）部署某一 Web 应用

有三种方式可以在 Tomcat 系统中部署 Web 应用。

- 直接拷贝你的 WAR 文件或者你的 Web 应用文件夹（包括该 Web 应用的所有内容）到 C:\jakarta-tomcat-5.0.19\webapps 目录下。

  该文件必须以“.war”作为扩展名。一旦 Tomcat 监听到这个文件，它将（缺省的）解开该文件包作为一个子目录，并以 WAR 文件的文件名作为子目录的名字。接下来，Tomcat 将在内存中建立一个 context，就好象你在 server.xml 文件里建立一样。当然，其他必需的内容，将从 server.xml 中的 DefaultContext 获得。

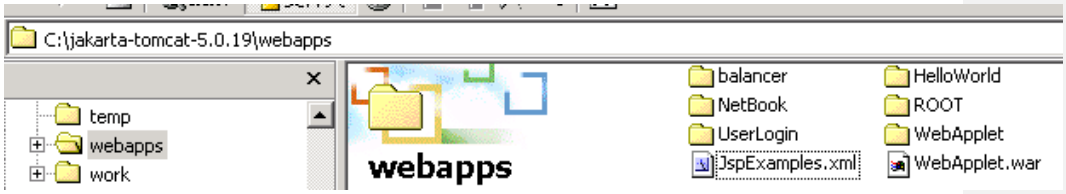
- 部署 web 应用的另一种方式是写一个 Context XML 片断文件，然后把该文件拷贝到 C:\jakarta-tomcat-5.0.19\webapps 目录下。

  一个 Context 片断并非一个完整的 XML 文件，而只是一个 Context 元素，以及对该应用的相应描述。这种片断文件就像是从 server.xml 中切取出来的 context 元素一样，所以这种片断被命名为“context 片断”。这个 web 应用本身可以存储在硬盘上的任何地方。

举个例子，如果我们想部署一个名叫 JspExamples 的 Web 应用，该应用使用 realm 作为访问控制方式，我们可以使用下面这个片断：

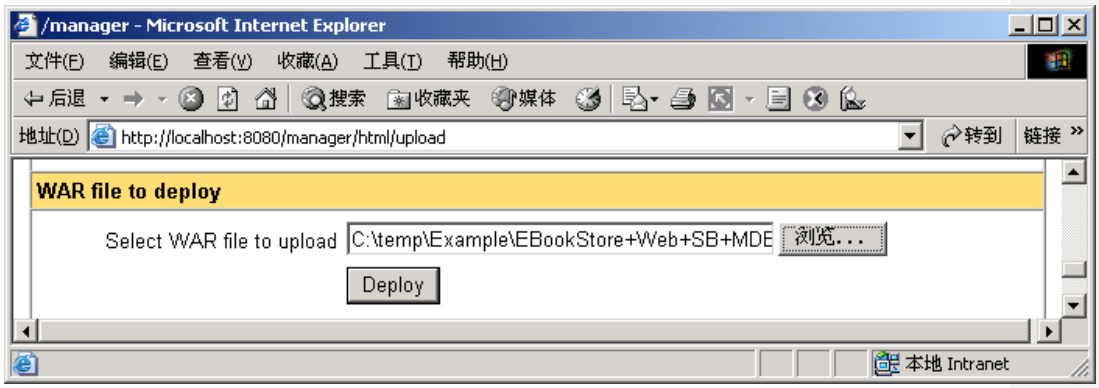
```
<!--
   Context fragment for deploying JspExamples
-->
<Context path="/JspExamples" docBase="JspExamples" debug="0" reloadable="true">
    <RealmclassName="org.apache.catalina.realm.UserDatabaseRealm"
resourceName="UserDatabase"/>
</Context>
```

把该片断命名为“JspExamples.xml”，然后拷贝到 C:\jakarta-tomcat-5.0.19\webapps 目录下。这种 Context 片断提供了一种便利的方法来部署 web 应用，你不需要编辑 server.xml，除非你想改变缺省的部署特性，安装一个新的 Web 应用时不需要重启动 Tomcat。

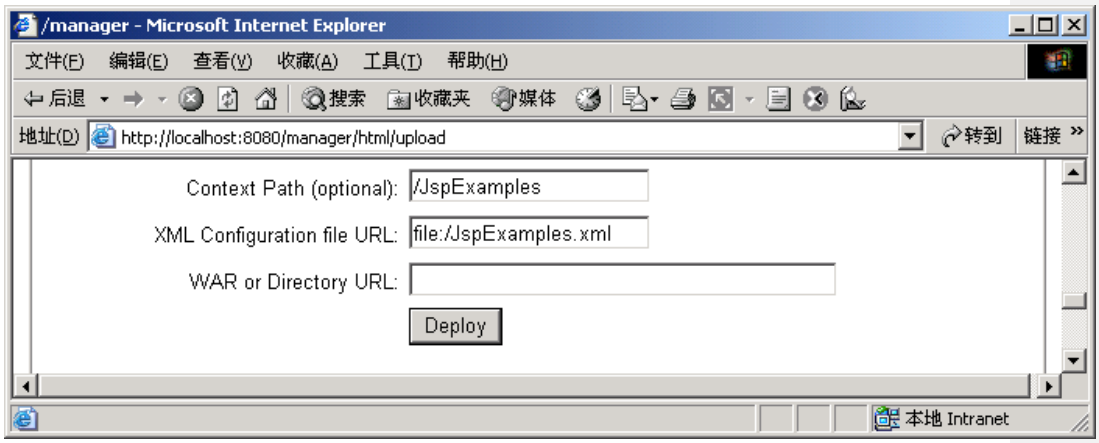


- 采用 GUI 管理界面进行发布

如果提供了该 Web 应用的*.war 文件，直接浏览并发布它



如果 Web 应用是以目录形式存在的，则可以：



五、Tomcat 服务器的 Web 安全的解决方法

1、概述

在任何一种 WEB 应用开发中，不论大中小规模的，每个开发者都会遇到一些需要保护程序数据的问题，涉及到用户的 LOGIN ID 和 PASSWORD。那么如何执行验证方式更好呢？实际上，有很多方式来实现。

下面将讨论在 Tomcat 中实现基本的（BASIC）和基于表单的（FORM-BASED）验证方式。它通过 server.xml 和 web.xml 文件提供基本的和基于表单的验证。

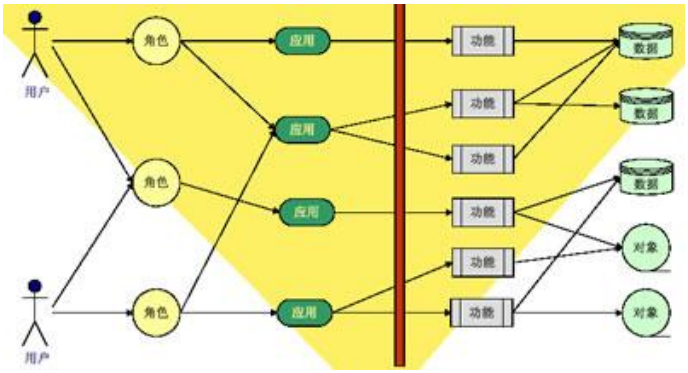
对于采用基于表单的（FORM-BASED）验证方式，只是要求在登录的 JSP 页面中的 j_security_check 表单(for FORM-based) 需要两个参数：j_username 和 j_password。

对于用户的登录的名称和密码在 Tomcat 中可以以两种形式来存放，一是采用 server.xml；另一种也可以采用用户自己的数据库表来存储。

2、设计系统中的各种人员的角色

（1）设计思想

- 统一用户管理，实现基于角色、粗粒度（基于 URL）和细粒度（基于应用组件的方法调用）的访问策略管理体系，
- 基于分级角色的权限管理、统一证书管理和统一资源管理



	人事	财务	教务	房产			设备	信息	办公
学校主管领导									
主管部门	人事处	财务处	教务处 研究生院 成人教育				设备处		校办
单位主管									
业务人员	人事 干事	会计	教务员	宿舍	家属	公房	保管员	信息员	办公室任收员 公主发
普通公众									

（2）设计目标

一般采用数据库表（对于复杂的也可以采用 LDAP）记录每个系统用户的帐号信息、功能权限和数据权限信息，这样能够增加用户管理和权限设置的灵活性，同时也避免多个用户共用一个帐号的情况。

（3）优点

- 从用户角度来看，登录所有应用系统都使用唯一的用户名和口令（数字证书）同时在访问系统时，也只需要登录一次（单点登录全网漫游---SSO（Single Sign-On））。
- 从管理者角度来看，提供了统一、集中、有效的用户管理。

六、在 Tomcat 中实现基本的 HTTP 方式的验证

1、实现基本验证

（1）在 C:\jakarta-tomcat-5.0.19\conf 下的 tomcat-users.xml 文件中添加角色和用户（可以同时添加多个用户）

```
<role rolename="users"/>
<user name="yang"      password="12345678" roles="users"/>
<user name="zhang"     password="12345678" roles="users"/>
```

（2）在 Web 应用的 web.xml 文件中添加如下的项目

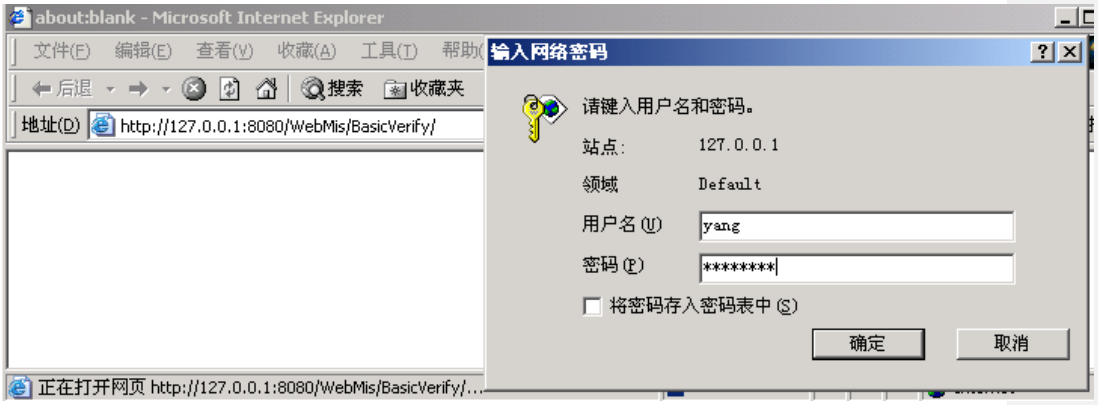
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
…..
      <security-constraint>
    <web-resource-collection>
            <web-resource-name>
              protected Resource
            </web-resource-name>
            <url-pattern>/BasicVerify/*</url-pattern>
     </web-resource-collection>
   <auth-constraint>
        <role-name>users</role-name>
   </auth-constraint>
   </security-constraint>
    <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Default</realm-name>
      </login-config>

<security-role>
          <description>this is a user</description>
          <role-name>users</role-name>
   </security-role>
…..
</web-app>
```

（3）重新启动 Tomcat 服务器

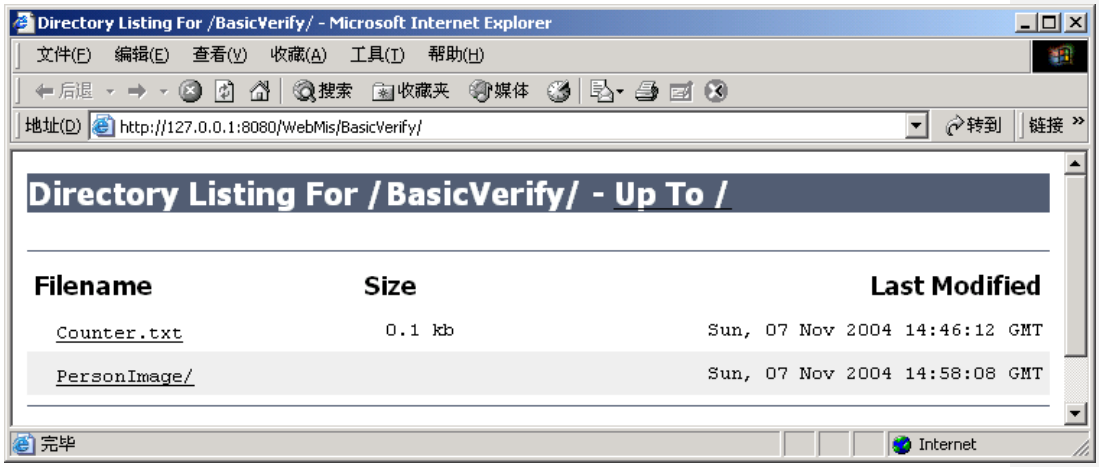
并在浏览器中直接输入所保护的目录 http://127.0.0.1:8080/WebMis/BasicVerify，将出现如下的登录页

输入用户名称：yang（请见前面的 tomcat-users.xml 文件的设置）

密码：12345678



将出现如下的



如果用户名称或者密码出现错误，将强制输入。

七、在 Tomcat 中采用基于表单的安全验证

1、概述

（1）基于表单的验证

基于 From 的安全认证可以通过 Tomcat Server 对 Form 表单中所提供的数据进行验证，基于表单的验证使系统开发者可以自定义用户的登陆页面和报错页面。这种验证方法与基本 HTTP 的验证方法的唯一区别就在于它可以根据用户的要求制定登陆和出错页面。

通过拦截并检查用户的请求，检查用户是否在应用系统中已经创建好 login session。如果没有，则将用户转向到认证服务的登录页面。但在 Tomcat 中的基于表单的验证凭证不被保护并以纯文本发送。

（2）在 Tomcat 中的实现

在 Tomcat 中，用户、用户组和角色都是在 XML 配置文件（C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml）中指定的，我们只需要提供一个登陆页面，包含一个名为 j_security_check 的 Form 表单，一个名为 j_username 的 TextBox 和一个名为 j_password 的 PasswordBox，然后在/WEB-INF/web.xml 中配置即可使用 Tomcat 默认的 JAAS 身份验证。

使用 JAAS 验证的好处是，验证逻辑从页面中分离，对页面的限制访问是通过/WEB-INF/web.xml 中的配置指定的，无需自定义过滤器。

（3）为了实现 Web 应用程序的安全，Tomcat Web 容器执行下面的步骤：

- 在受保护的 Web 资源被访问时，判断用户是否被认证。
- 如果用户没有得到认证，则通过重定向到部署描述符中定义的注册页面，要求用户提供安全信任状。
- 根据为该容器配置的安全领域，确认用户的信任状有效。
- 判断得到认证的用户是否被授权访问部署描述符（web.xml）中定义的 Web 资源。

2、设计步骤

（1）编写登录页面和错误处理页面：请见 FormSafeWebApp 程序中的页面

（2）登录的页面文件的内容如下

基于 FORM 的用户认证要求你返回一个包括用户名和密码的 HTML 表单，这个表单相对应与用户名和密码的元素必须是 j_username 和 j_password，并且表单的 action 描述必须为 j_security_check（其实是一个 Servlet）。该表单的具体操作以及 j_username 和 j_password 名字在 Servlet 中定义。当这个表单到达服务器的时候，由内部的 Tomcat Server 安全区对它进行确认。

包括这个表单的资源可以是一个 HTML 页面、一个 JSP 页面或者一个 Servlet。你可以在<form-login-page>元素中定义。基于表单的认证能够使开发人员定制认证的用户界面。在 web.xml 的 login-config 标签项目定义了认证机制的类型、登录的 URI 和错误页面。

下面为该页面的内容：

```
<%@ page contentType="text/html; charset=GBK" %>
<html><head><title>在 Tomcat 中采用 Form 验证方式实现的安全 Web 应用程序的登录页</title>
</head><body bgcolor="#ffffff">
<form method="post" name="Login" action="j_security_check">
```

注意：action 应该为 j_security_check

```
<table width="500" border="1" align="center"> <tr>
    <td colspan="2"> <div align="center"><strong>在 Tomcat 中采用</strong><strong>基于表单的
安全验证的登录表单 </strong> </div></td> </tr>
  <tr><td width="224"><div align="right">用户名称：</div></td>
    <td width="260"><input type="text" name="j_username"></td> </tr>
  <tr> <td><div align="right">密码：</div></td>
   <td><input type="password" name="j_password"></td>
  </tr>
  <tr><td><div align="right"><input type="submit" name="Submit" value="提交">
    </div></td> <td><input type="reset" name="Submit2" value="重置"></td>
  </tr></table></form></body></html>
```

注意：用户名称和密码的输入应该为 j_username 和 j_password

（3）修改 web.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
     </welcome-file-list>
 <!-- Security is active on entire directory -->
  <security-constraint>
    <display-name>Tomcat Server Form Security Constraint</display-name>
    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
      <description>A Page of Login Success</description>
       <url-pattern>/ProtectedDirOne/index.jsp</url-pattern>
      </web-resource-collection>
    <auth-constraint>
      <!-- Anyone with one of the listed roles may access this area -->
      <role-name>admin</role-name>
    </auth-constraint>
  </security-constraint>
  <!-- Login configuration uses form-based authentication -->
  <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>Tomcat Server Configuration Form-Based Authentication
Area</realm-name>
    <form-login-config>
        <form-login-page>/login.jsp </form-login-page>
        <form-error-page>/Error.htm </form-error-page>

    </form-login-config>
```

定义本 Web 应用的默认启始页面

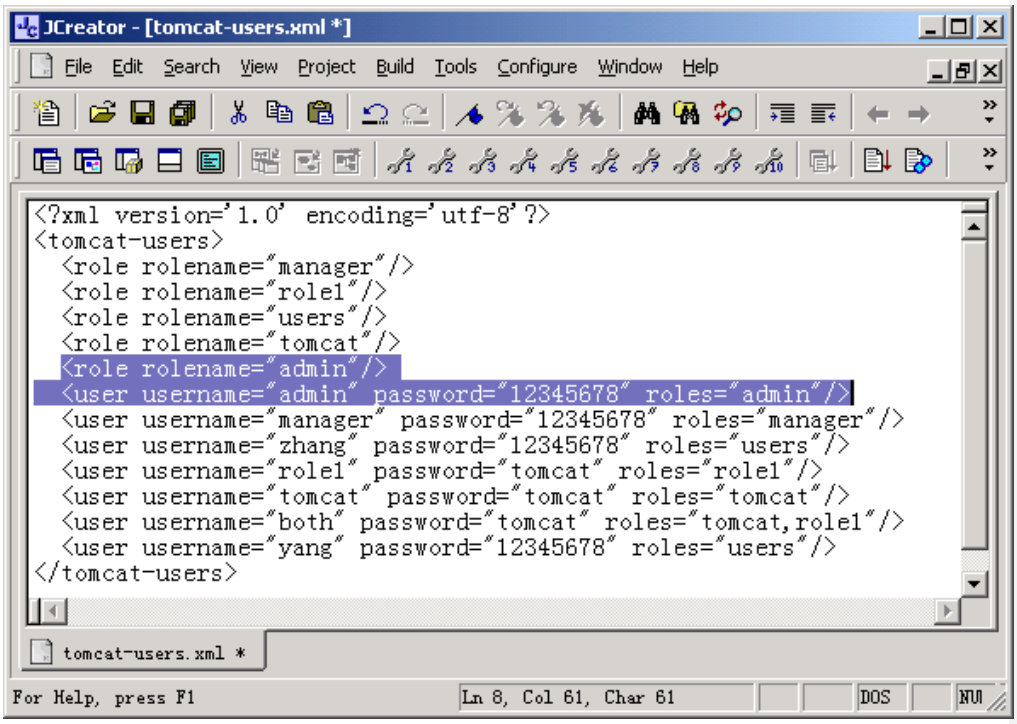
指定 Form 验证的用户的角色名称

指定验证的方式为 Form

```
    </login-config>
    <!-- Security roles referenced by this web application -->
    <security-role>
      <description>
        The role is Administration
      </description>
      <role-name>admin</role-name>
    </security-role>
</web-app>
```

关联 Tomcat 中的 admin 的角色

（4）在 C:\jakarta-tomcat-5.0.19\conf\tomcat-users.xml 文件中配置 admin 的角色以及与该 admin 角色相匹配的用户名称和密码

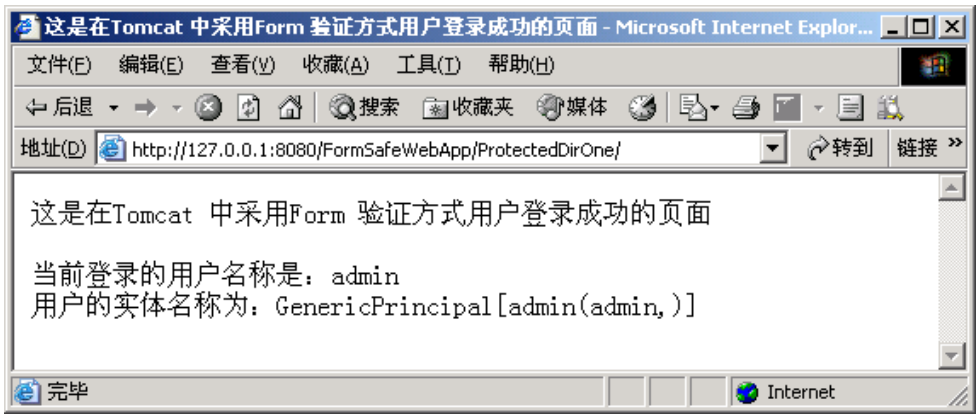


（5）执行该页面

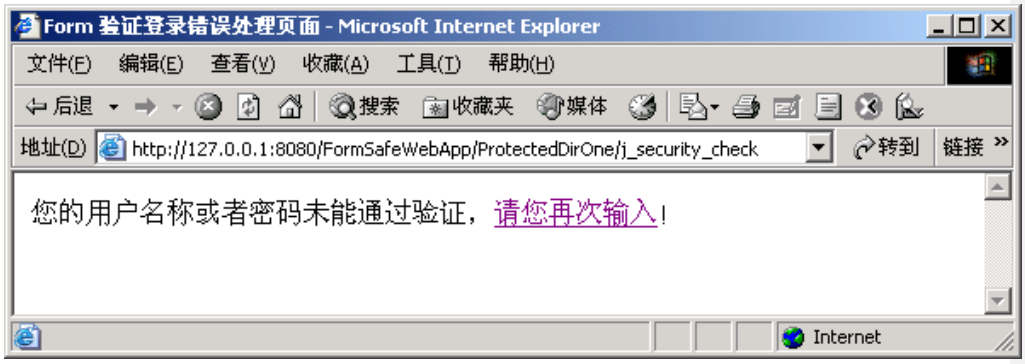
在浏览器中直接输入受保护的页面的 URL 地址：

http://127.0.0.1:8080/FormSafeWebApp/ProtectedDirOne/，将出现要求登录的页面。

在表单中输入用户名称为 admin（前面在 tomcat-users.xml 文件中所设置的某一用户名称），密码为 12345678。然后点击“提交”，将出现如下页面



如果用户名称或者密码输入不正确，将出现如下的页面也就是错误页面



（6）在页面中获得当前登录成功后的用户名称和实体名称

利用 request 对象中的 getRemoteUser()方法获得当前登录成功后的用户名称和利用 getUserPrincipal()方法获得当前登录成功后的实体名称。

八、在 Tomcat 中配置单点登录（Single Sign-On）

1、概述

一旦你设置了 realm 和验证的方法，你就需要进行实际的用户登录处理。一般说来，对用户而言登录系统是一件很麻烦的事情，你必须尽量减少用户登录验证的次数。作为缺省的情况，当用户第一次请求受保护的资源时，每一个 Web 应用都会要求用户登录。

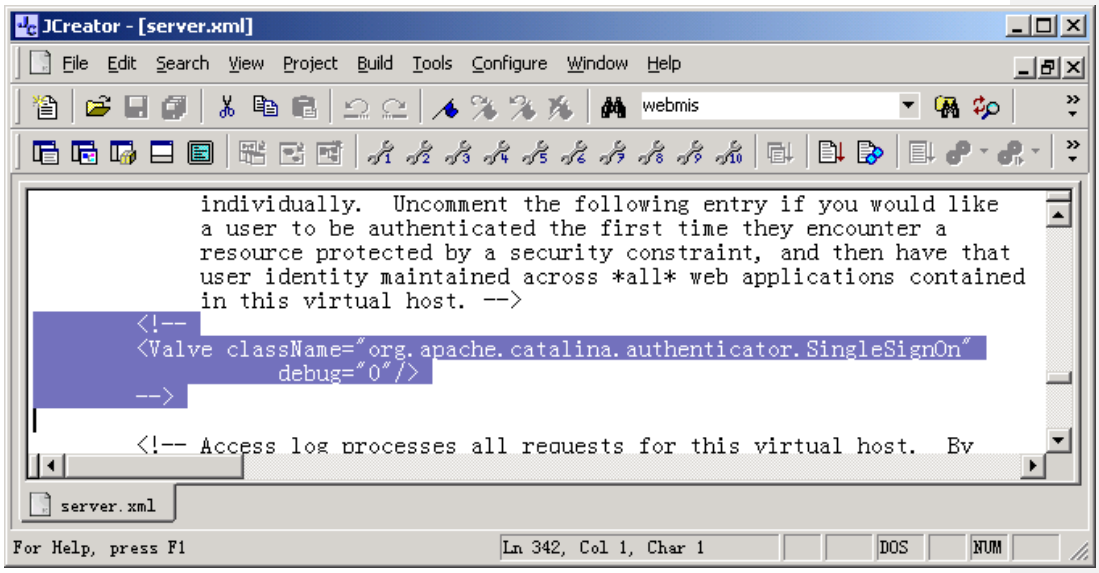
如果你运行了多个 Web 应用，并且每个应用都需要进行单独的用户验证，那这看起来就有点像你在与你的用户搏斗。用户们不知道怎样才能把多个分离的应用整合成一个单独的系统，所有他们也就不知道他们需要访问多少个不同的应用，只是很迷惑，为什么总要不停的登录。

2、Tomcat 中的“Single Sign-On”特性及配置

其主要的特性是能够允许用户在访问同一虚拟主机下所有 Web 应用时，只需登录一次。为了使用这个功能，你只需要在 C:\jakarta-tomcat-5.0.19\conf \server.xml 文件中的 Host 标签上添加一个 SingleSignOn Valve 元素即可，如下所示：

<Valve className="org.apache.catalina.authenticator.SingleSignOn" debug="0"/>

在 Tomcat 初始安装后，server.xml 的注释里面包括 SingleSignOn Valve 配置的例子，你只需要去掉注释（**在 339 行左右**），即可使用。那么，任何用户只要登录过一个应用，则对于同一虚拟主机下的所有应用同样有效。



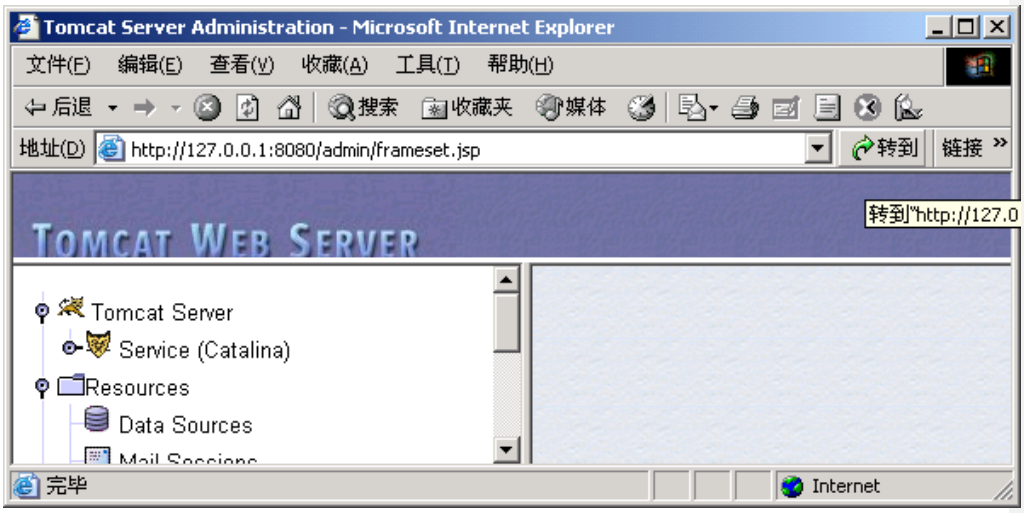
3、测试单点登录

（2） 直接进入前面的 Form 验证所产生的 Web 应用

（http://127.0.0.1:8080/FormSafeWebApp/ProtectedDirOne/）将出现要求登录的页面

在表单中输入用户名称为 admin（前面在 tomcat-users.xml 文件中所设置的某一用户名称），密码为 12345678。然后点击“提交”，将以用户名 admin 进行成功登录该 Web 应用。

（2）再在该浏览器窗口内（不能在新窗口，否则会成为另一用户）直接输入 http://127.0.0.1:8080/admin/frameset.jsp，此时将以 admin 的用户浏览另一 Web 应用。观察能否直接进入 Tomcat 的系统管理的页面，此时应该可以并且出现下面的页面。



如果新开一浏览器窗口并直接输入 http://127.0.0.1:8080/admin/frameset.jsp，看能否直接进入 Tomcat 的系统管理的页面，此时将会出现要求登录的页面。

4、使用 single sign-on valve 所应该注意的问题

- value 必须被配置和嵌套在相同的 Host 元素里，并且所有需要进行单点验证的 web 应用（必须通过 context 元素定义）都位于该 Host 下。
- 包括共享用户信息的 realm 必须被设置在同一级 Host 中或者嵌套之外。
- 不能被 context 中的 realm 覆盖。
- 使用单点登录的 web 应用最好使用一个 Tomcat 的内置的验证方式（Basic 或者 Form）（被定义在 web.xml 中的<auth-method>中），这比自定义的验证方式强。
- 单点登录需要使用 cookies。

九、对 Struts 中的 Action 进行授权利

1、应用的意义

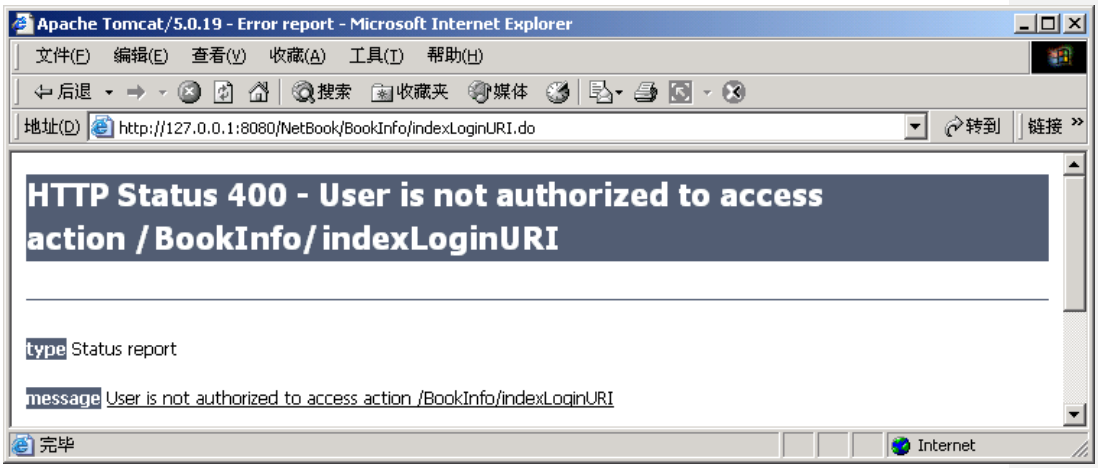
在某些应用下，如果 Action 类执行的功能比较重要，可以对该 Action 类进行授权利以实现只有特定角色的用户能够访问，此时可以在 struts-config.xml 文件中进行配置

2、在 struts-config.xml 文件中进行配置

```
<action-mappings>
        <action    path="/BookInfo/indexLoginURI"              type="netbook.IndexLoginAction"
name="IndexLoginForm"
            scope="request"      validate="true"      input="/index.jsp" roles="admin">
            <forward name="loginSuccess"   path="/BookInfo/ShowBookInfo.jsp" />
            <forward name="loginFailure" path="/BookInfo/gotoIndex.htm"/>
        </action>
  </action-mappings>
```

4、 问该 Action

如果访问者不是 admin 角色的用户，此时将出现如下的错误



九、在 Tomcat 上配置虚拟主机

1、Tomcat 服务器的 server.xml 文件

（1）Tomcat 组件

Tomcat 服务器是由一系列可配置的组件构成，其中核心组件是 Catalina Servlet 容器，它是所有其他 Tomcat 组件的顶层容器。Tomcat 的组件可以在<CATALINA_HOME>/conf/server.xml 文件中进行配置，每个 Tomcat 组件在 server.xml 文件中对应一种配置元素。

（2）Tomcat 组件之间的关系

以下代码以 XML 的形式展示了各种 Tomcat 组件之间的关系：

```
<Server>
    <Service>
        <Connector />
        <Engine>
            <Host>
                <Context>
                </Context>
            </Host>
        </Engine>
    </Service>
</Server>
```

（3）各个 Tomcat 组件的说明

在以上 XML 代码中，每个元素都代表一种 Tomcat 组件。这些元素可分 4 类：

- 顶层类元素：主要包括 <Server> 元素和 <Service> 元素，他们位于整个配置文件的顶层。
- 连接器类元素：代表了介于客户与服务之间的通信接口，负责将客户的请求发送给服务器，并将服务器的响应结果传递给客户。
- 容器类元素：代表处理客户请求并生成响应结果的组件，有 3 种容器类元素，它们是 Engine、Host 和 Context。Engine 组件为特定的 Service 组件处理所有的客户请求，Host 组件为特定的虚拟主机处理所有客户请求，Context 组件为特定的 Web 应用处理所有客户请求。
- 嵌套类元素：嵌套类元素代表了可以加入到容器中的组件，如 <Logger> 元素、<Valve> 元素和 <Realm> 元素。

（4）<Server> 元素

<Server> 元素代表整个 Catalina Servlet 容器，它是 Tomcat 实例的顶层元素，由 org.apache.catalina.Server 接口来定义。

<Server> 元素中可以包含一个或多个 <Service> 元素，但 <Server> 元素不能做为任何其他元素的子元素。

（5）<Service> 元素

<Service> 元素由 org.apache.catalina.Service 接口来定义，它包含一个 <Engine> 元素，以及一个或多个 <Connector> 元素，这些 <Connector> 元素共享同一个 <Engine> 元素。

（6）<Connector> 元素

<Connector> 元素由 org.apache.catalina.Connector 接口来定义。<Connector> 元素代表和客户程序实际交互的组件，它负责接受客户请求，以及向客户返回响应结果。

（7）<Engine> 元素

<Engine> 元素由 org.apache.catalina.Engine 接口来定义。每个 <Service> 元素只能包含一个 <Engine> 元素。<Engine> 元素处理在同一个 <Service> 元素中所有 <Connector> 元素接受到的客户请求。

<Engine> 元素可包括如下子元素：

<Loggor>
<Realm>
<Valve>
<Host>

（8）<Host> 元素

<Host> 元素由 org.apache.catalina.Host 接口来定义。一个 <Engine> 元素中可以包含多个 <Host> 元素。每个 <Host> 元素定义了一个虚拟主机，他可以包含一个或多个 Web 应用。

<Host> 元素可包括如下子元素：

<Loggor>
<Realm>
<Valve>
<Context>

（9）<Context> 元素

<Context> 元素由 org.apache.catalina.Context 接口来定义。<Context> 元素是使用最频繁的元素。每个 <Context> 元素代表运行在虚拟主机上的单个 Web 应用。一个 <Host> 元素中可以包含多个 <Context> 元素。

<Context> 元素可包括如下子元素：

<Loggor>

<Realm>

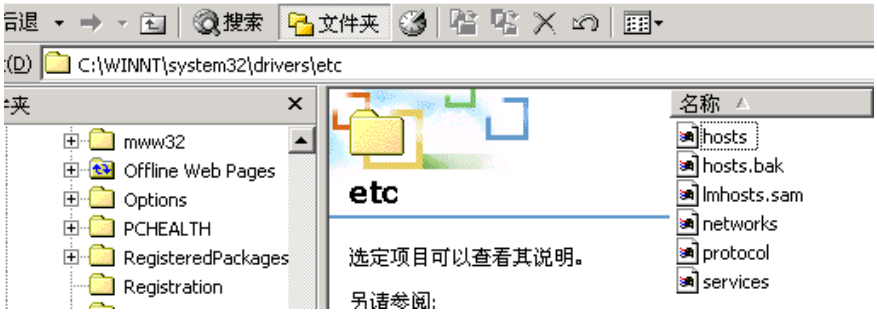
<Valve>

<Resource>

<ResourceParams>

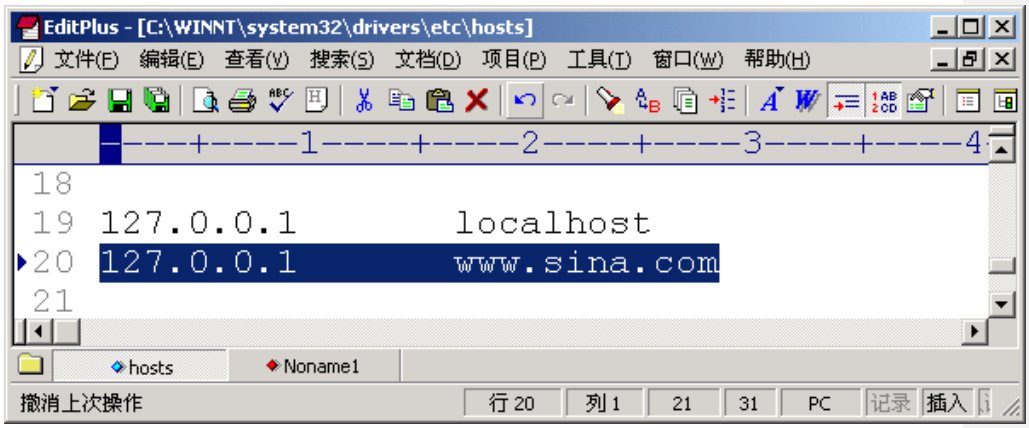
2、为主机配置域名

（1）编辑 C:\WINNT\system32\drivers\etc 下的 hosts 文件，在其中增加对本机 IP 地址的映射的域名



（2）本例为

127.0.0.1 www.sina.com



（3）保存该文件

3、修改 Tomcat 下的 C:\jakarta-tomcat-5.0.19\conf\server.xml 文件以增加一个主机 Host 的设置

Host 标记是用来配置虚拟主机的，就是可以多个域名指向一个 tomcat，<context>是 Host 标记的子元素吧，表示一个虚拟目录，它主要有两个属性，path 就相当于虚拟目录名字，而 docbase 则是具体的文件位置。

<Host name="www.sina.com" debug="0" appBase="webapps" unpackWARs="true" autoDeploy="true" xmlValidation="false" xmlNamespaceAware="false">

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

</Host>

注意：

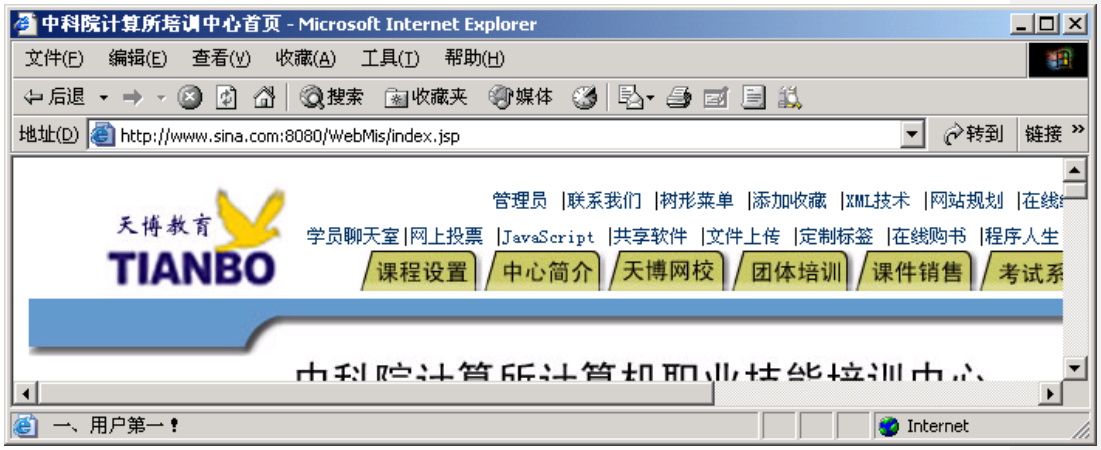
（1）可以将 Tomcat 自己带的 localhost 主机的 Host 的整个设置全部拷贝，然后将“localhost”改名为 www.sina.com 即可以。

（2）必须保证在<Host></Host>之间至少有一个<Context path="" docBase="ROOT" debug="0" />的根 Web 应用程序的设置项目存在。

（3）可以根据应用的需要，在<Host></Host>之间设置其它的基于该主机名称下的其它 Web 应用程序的**<Context>**设置。

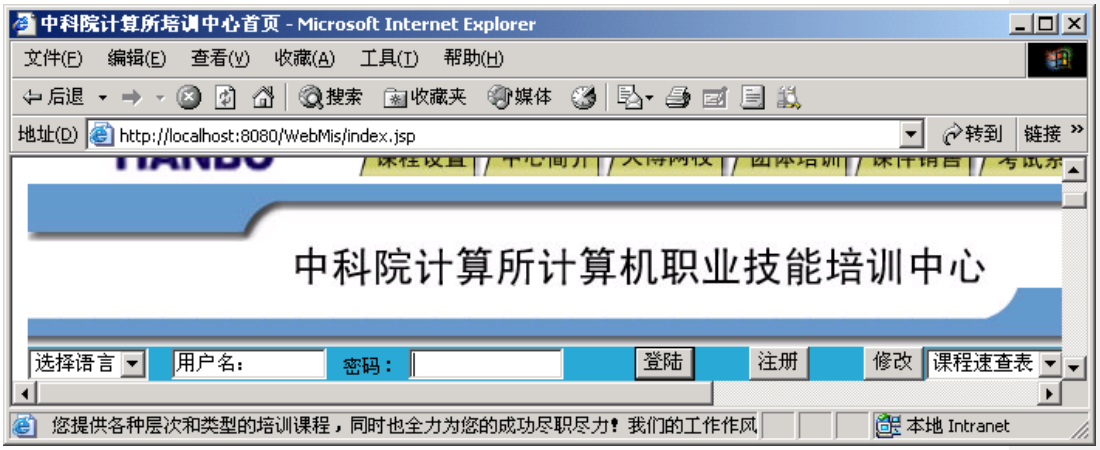
4、启动 Tomcat 后再浏览本 Web 应用

输入 http://www.sina.com:8080/WebMis/index.jsp



5、本例也可以以 localhost 缺省的主机名称来访问

http://localhost:8080/WebMis/index.jsp


中科院计算所培训中心首页 - Microsoft Internet Explorer
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 收藏夹 媒体
地址(D) http://localhost:8080/WebMis/index.jsp
转到 链接
中科院计算所计算机职业技能培训中心
选择语言
用户名:
密码:
登陆
注册
修改
课程速查表
您提供各种层次和类型的培训课程，同时也全力为您的成功尽职尽力！我们的工作作风
本地 Intranet