**如何在eclipse的maven环境中创建Servlet3.0并部署到Tomcat？？**

**工具：maven3.0+；eclipse EE；jdk 9；Tomcat8.5**

环境的搭建：

<http://www.open-open.com/lib/view/open1389227144328.html>

删除web.xml后会出现pom.xml文件报错。

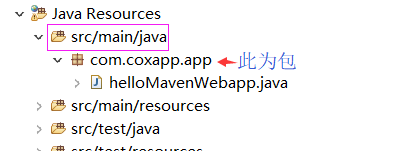


解决如右：

创建Servlet：

如上述操作创建好maven工程后

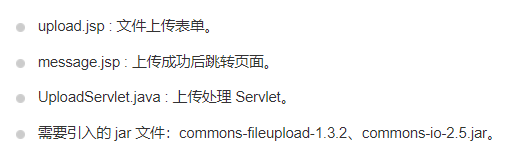
方法1：

* 在src/main/java下创建包，右击鼠标new—>package.-->填写包名。
* 在包下创建类class.—>填写类名。然后再编写Servlet。

方法2：

* 右击项目名称后new—>servlet🡪填写包名类名。

栗子：Servlet文件上传。



1）编写upload.jsp文件。（右击项目new🡪jsp file）

代码如下：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>文件上传实例 - 菜鸟教程</title>

</head>

<body>

<h1>文件上传实例 - 菜鸟教程</h1>

<form method="post" action=" /HelloMaven/helloMavenWebapp " enctype="multipart/form-data">

    选择一个文件:

    <input type="file" name="uploadFile" />

    <br/><br/>

    <input type="submit" value="上传" />

</form>

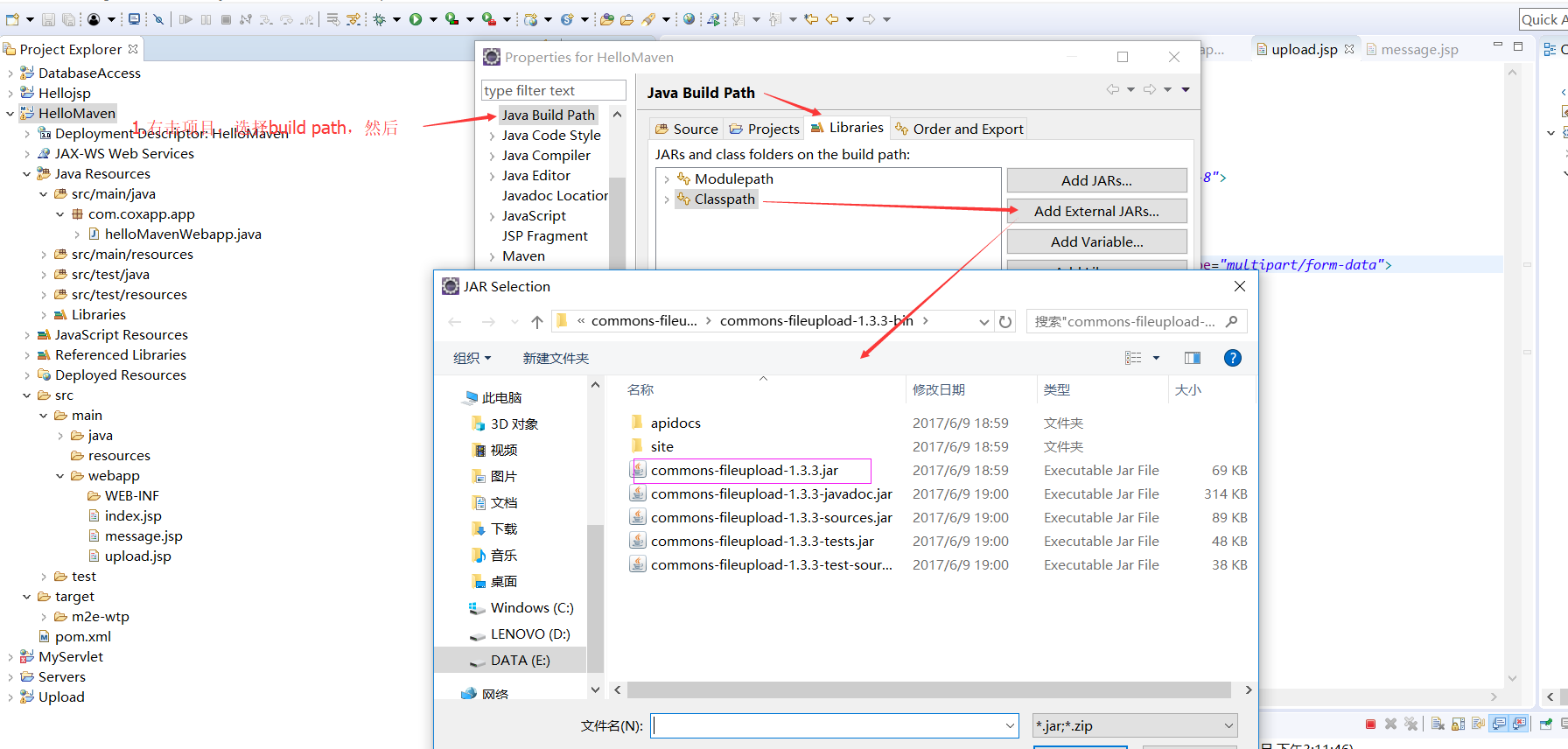
</body>

</html>

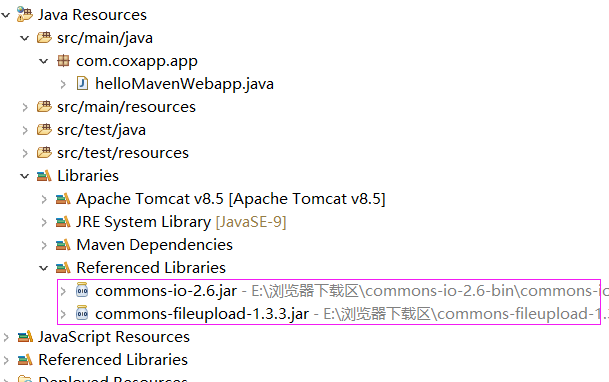
## 2）编写后台 Servlet

以下是 UploadServlet 的源代码，同于处理文件上传，在这之前我们先确保依赖包已经引入到项目的 WEB-INF/lib 目录下：

* 下面的实例依赖于 FileUpload，所以一定要确保在您的 classpath 中有最新版本的 **commons-fileupload.x.x.jar** 文件。可以从 <http://commons.apache.org/proper/commons-fileupload/> 下载。
* FileUpload 依赖于 Commons IO，所以一定要确保在您的 classpath 中有最新版本的 **commons-io-x.x.jar** 文件。可以从 <http://commons.apache.org/proper/commons-io/> 下载。
* 添加方法如下图。**commons-io-x.x.jar文件也是如此。**



然后可以看到



之后，需要把这两个文件放到Tomcat安装目录下的lib目录中。不然会出现启动Tomcat失败。

接着编写

helloMavenWebapp源代码 如下所示：

package com.coxapp.app;

import java.io.File;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItem;

import org.apache.commons.fileupload.disk.DiskFileItemFactory;

import org.apache.commons.fileupload.servlet.ServletFileUpload;

/\*\*

\* Servlet implementation class UploadServlet

\*/

@WebServlet("/helloMavenWebapp ")

public class helloMavenWebapp extends HttpServlet {

private static final long serialVersionUID = 1L;

// 上传文件存储目录

private static final String UPLOAD\_DIRECTORY = "upload";

// 上传配置

private static final int MEMORY\_THRESHOLD = 1024 \* 1024 \* 3; // 3MB

private static final int MAX\_FILE\_SIZE = 1024 \* 1024 \* 40; // 40MB

private static final int MAX\_REQUEST\_SIZE = 1024 \* 1024 \* 50; // 50MB

/\*\*

\* 上传数据及保存文件

\*/

protected void doPost(HttpServletRequest request,

        HttpServletResponse response) throws ServletException, IOException {

        // 检测是否为多媒体上传

        if (!ServletFileUpload.isMultipartContent(request)) {

         // 如果不是则停止

         PrintWriter writer = response.getWriter();

         writer.println("Error: 表单必须包含 enctype=multipart/form-data");

         writer.flush();

         return;

        }

// 配置上传参数

DiskFileItemFactory factory = new DiskFileItemFactory();

// 设置内存临界值 - 超过后将产生临时文件并存储于临时目录中

factory.setSizeThreshold(MEMORY\_THRESHOLD);

// 设置临时存储目录

factory.setRepository(new File(System.getProperty("java.io.tmpdir")));

ServletFileUpload upload = new ServletFileUpload(factory);

// 设置最大文件上传值

upload.setFileSizeMax(MAX\_FILE\_SIZE);

// 设置最大请求值 (包含文件和表单数据)

upload.setSizeMax(MAX\_REQUEST\_SIZE);

// 中文处理

upload.setHeaderEncoding("UTF-8");

// 构造临时路径来存储上传的文件

// 这个路径相对当前应用的目录

String uploadPath = request.getServletContext().getRealPath("./") + File.separator + UPLOAD\_DIRECTORY;

// 如果目录不存在则创建

File uploadDir = new File(uploadPath);

if (!uploadDir.exists()) {

uploadDir.mkdir();

}

try {

// 解析请求的内容提取文件数据

//@SuppressWarnings("unchecked")

List<FileItem> formItems = upload.parseRequest(request);

if (formItems != null && formItems.size() > 0) {

// 迭代表单数据

for (FileItem item : formItems) {

// 处理不在表单中的字段

if (!item.isFormField()) {

String fileName = new File(item.getName()).getName();

String filePath = uploadPath + File.separator + fileName;

File storeFile = new File(filePath);

// 在控制台输出文件的上传路径

System.out.println(filePath);

// 保存文件到硬盘

item.write(storeFile);

request.setAttribute("message",

"文件上传成功!");

}

}

}

} catch (Exception ex) {

request.setAttribute("message",

"错误信息: " + ex.getMessage());

}

// 跳转到 message.jsp

request.getServletContext().getRequestDispatcher("/message.jsp").forward(

request, response);

}

}

message.jsp 文件代码如下：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>文件上传结果</title>

</head>

<body>

<center>

<h2>${message}</h2>

</center>

</body>

</html>

**Pom.xml文件如下**

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.coxapp.app</groupId>

<artifactId>HelloMaven</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>HelloMaven Maven Webapp</name>

<url>http://maven.apache.org</url>

<properties>

<failOnMissingWebXml>false</failOnMissingWebXml>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<finalName>HelloMaven</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.0</version>

<configuration>

<source>9</source>

<target>9</target>

</configuration>

</plugin>

</plugins>

</build>

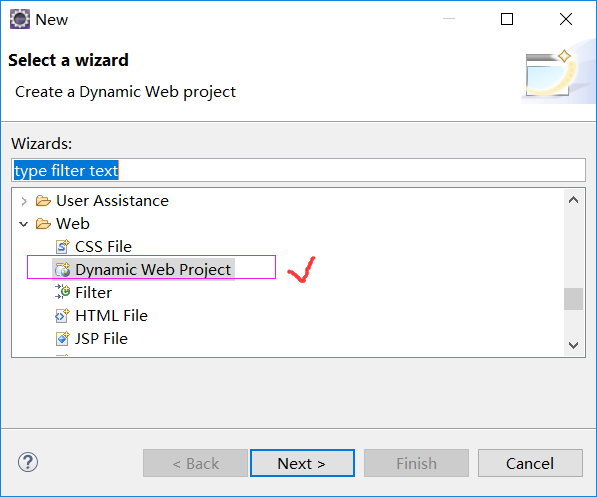
</project>

**。。**当您在浏览器中访问：http://localhost:8080/HelloMaven/upload.jsp 后选择文件上传将跳转到<http://localhost:8080/HelloMaven/helloMavenWebapp>

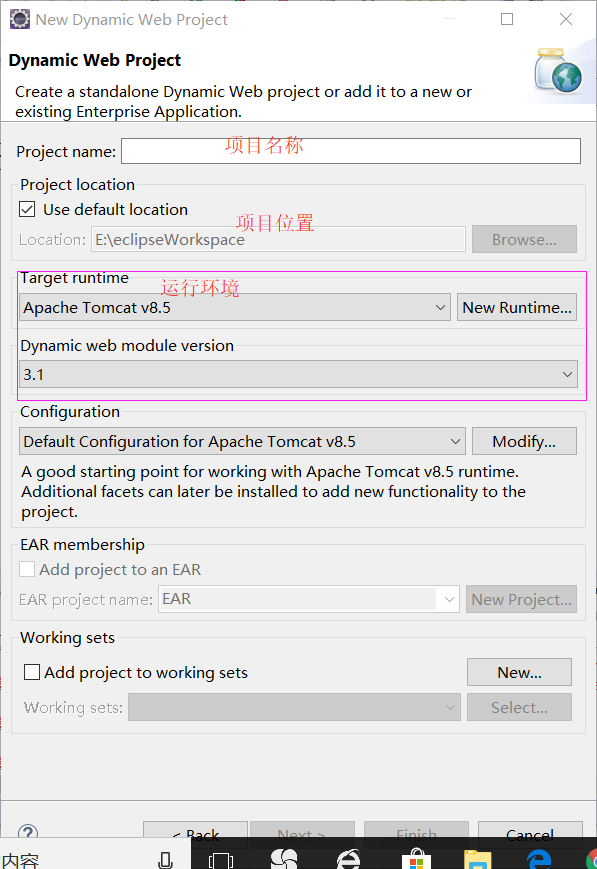
**Eclipse与Tomcat+MySQL的简单应用**

1. 在eclipse中创建Servlet项目并在Tomcat中显示。

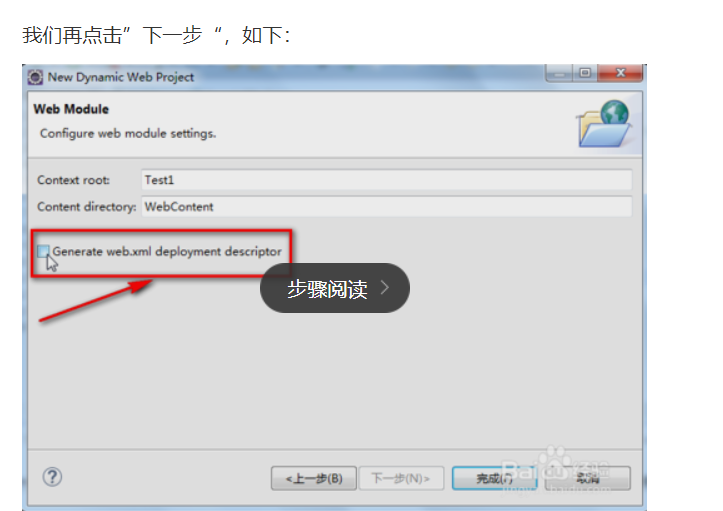
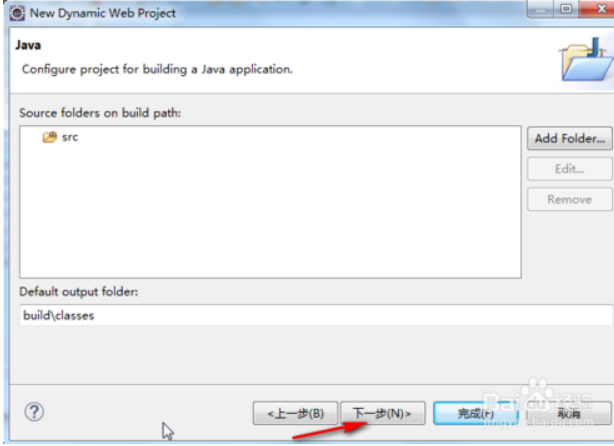
File->new->other->



R然后填写项目名称，选择运行环境



点击下一步

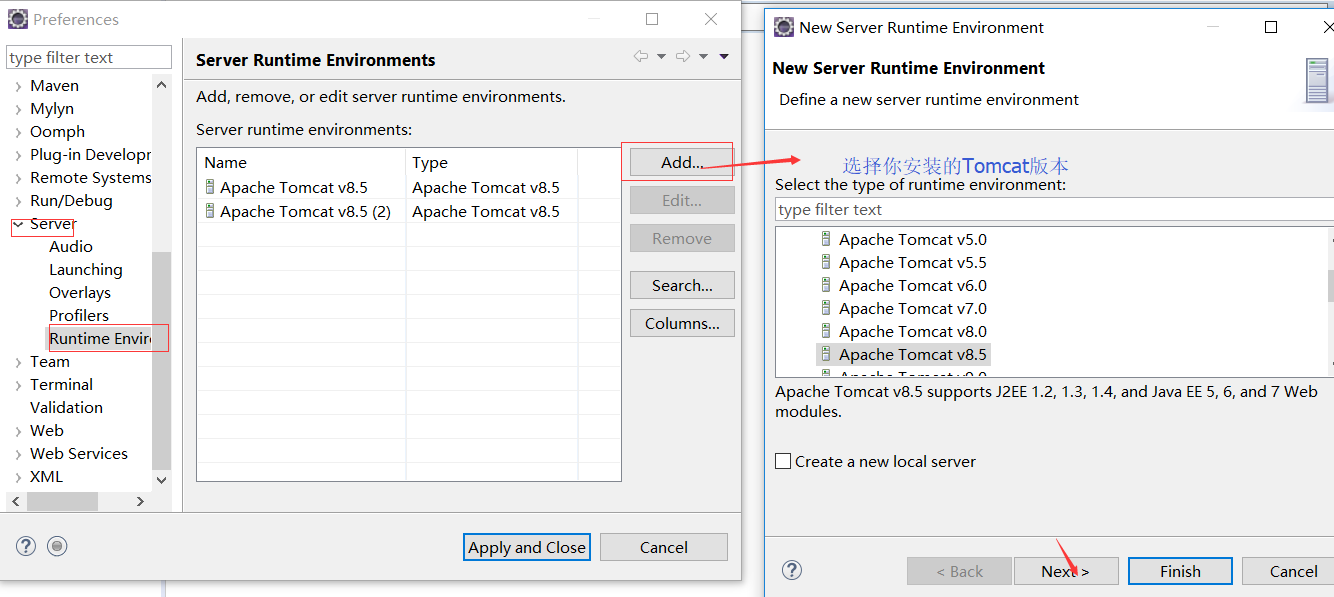


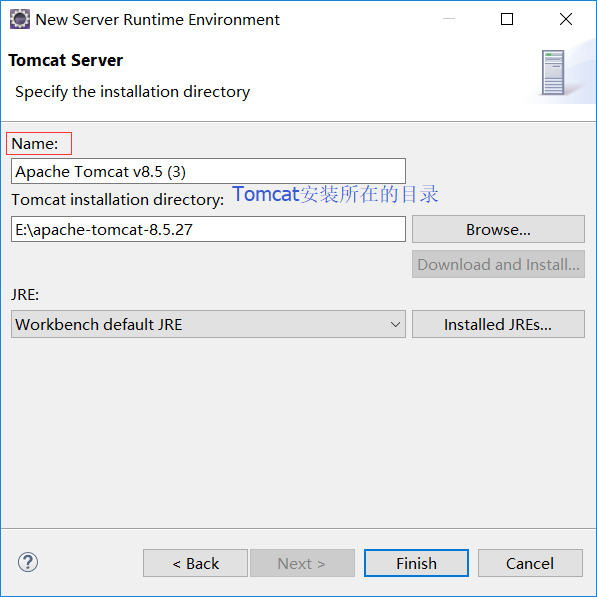
然后完成。



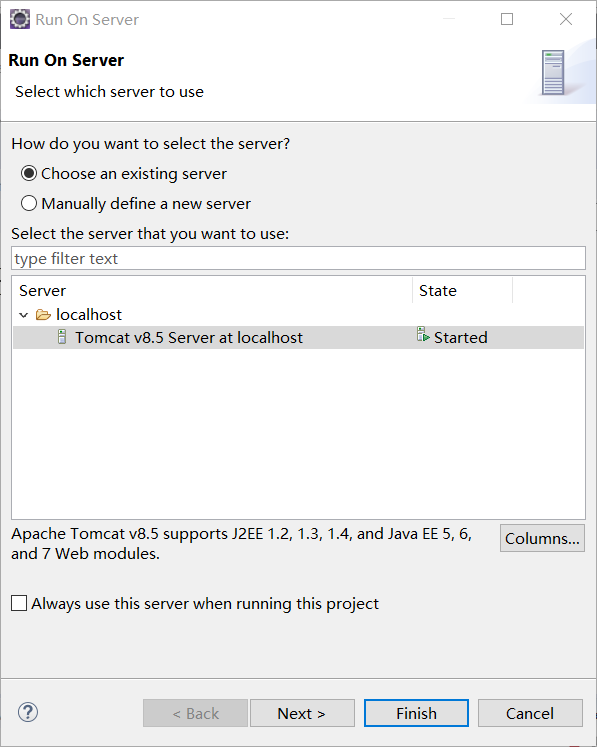
若需创建一个静态的HTML。点击左侧栏WebContent->new->HTML file.

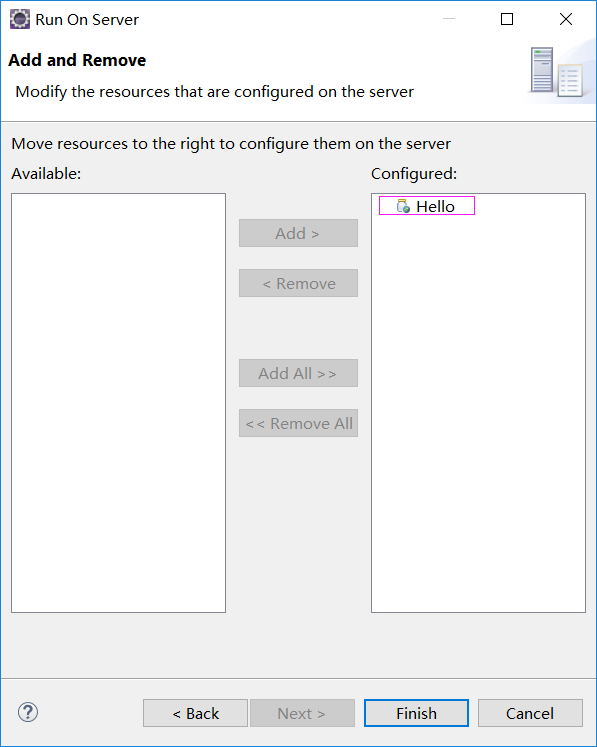
创建成功，填写需要的内容，然后在Window->Preferences





点击项目总名称，run on Servlet，就可以了，步骤如下。

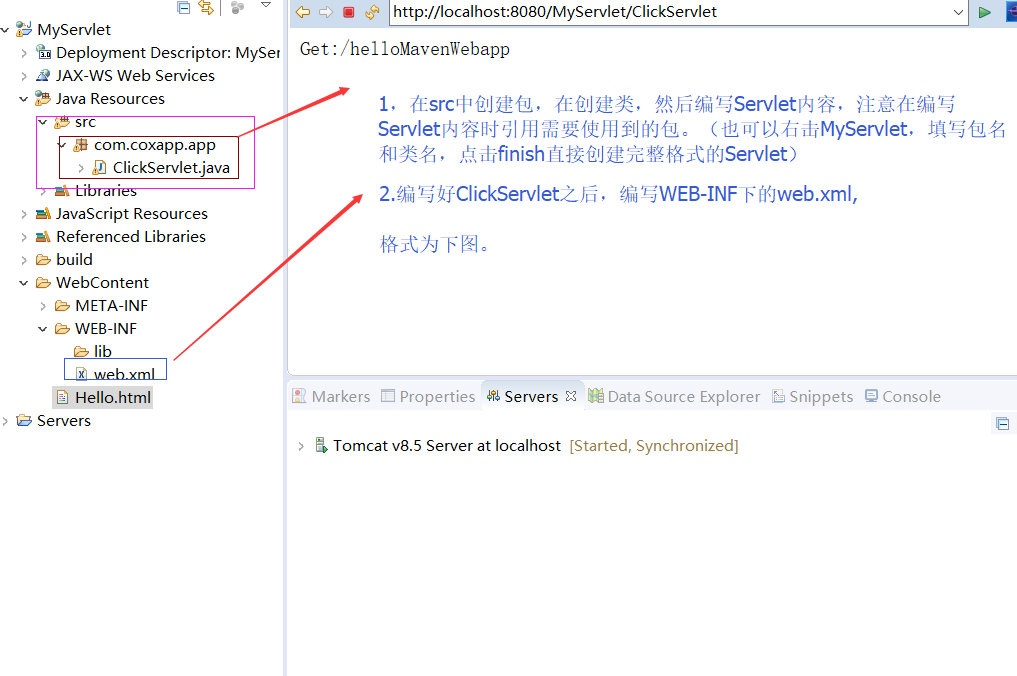


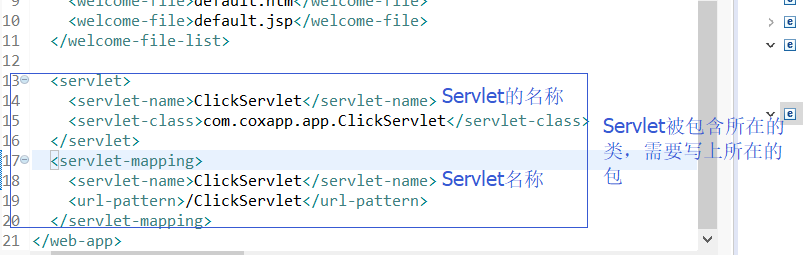


注意：若在这之前打开了Tomcat，则会报错（因为在eclipse中运行时会再次打开Tomcat）。此时应该先关闭Tomcat。

1. 编写第一个Servlet

首先创建一个动态网页项目。步骤如上，在上面的侧栏中的





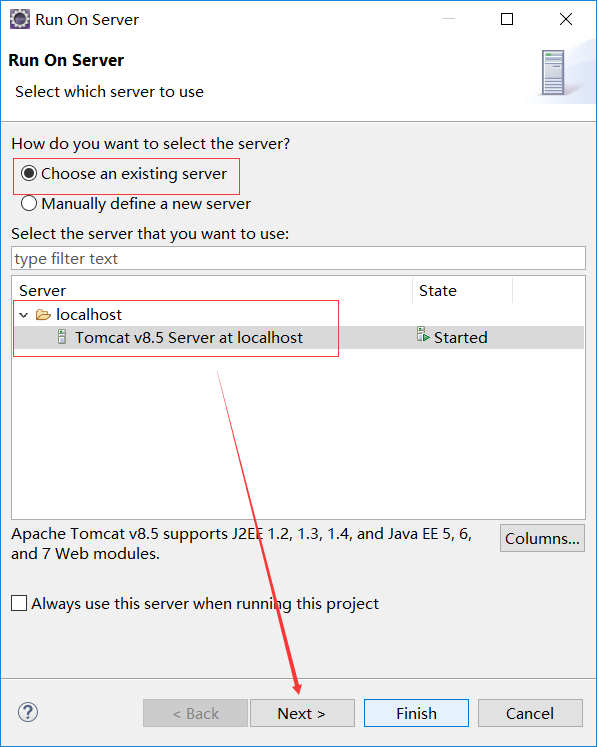
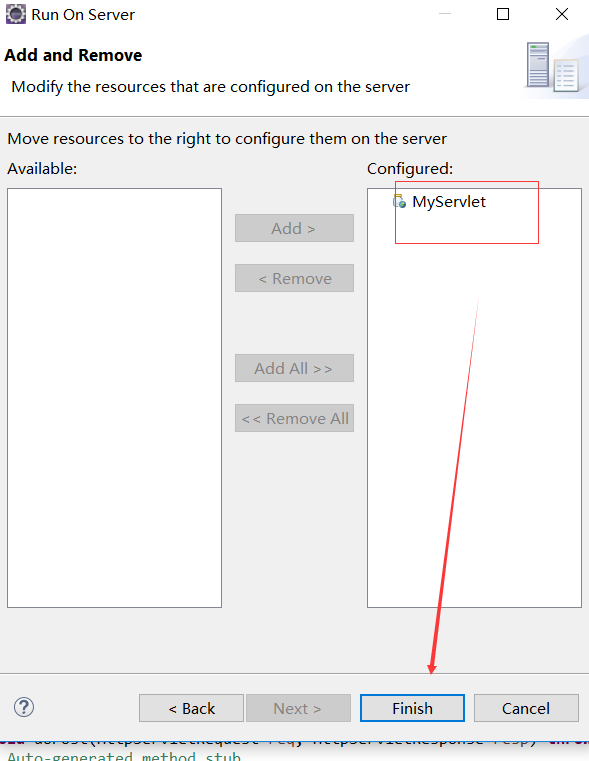
上图中<url-pattern>/ClickServlet</url-pattern>（表映射，当某Servlet被触发时，根据此找到与此对应的Servlet，也就是<servlet>中的内容,将此内容渲染在浏览器）；

其中的/ClickServlet与

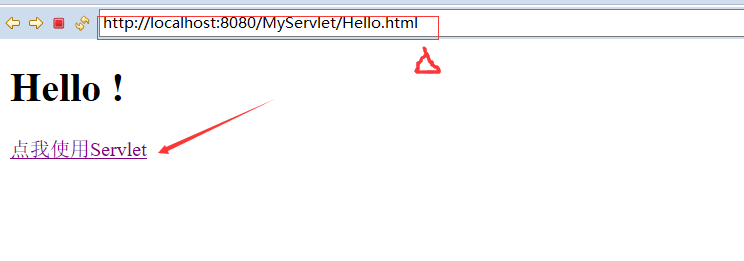


中的href=”ClickServlet”相对应。（当点击这个链接时，会执行对应的Servlet）

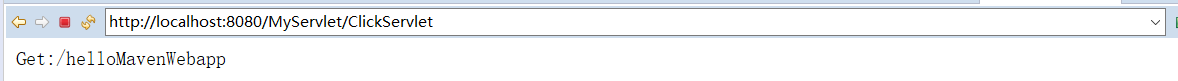
保存然后点击MyServlet->Run as->Run on server

然后输入要访问的页面，单击后会跳转



跳转后显示



也有可以会出现启动Tomcat失败的错误提示，原因及解决方法如下



附录：

Servlet代码如下，展开查看

package com.coxapp.app;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class ClickServlet

\*/

//@WebServlet("/ClickServlet")

public class ClickServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* Default constructor.

\*/

public ClickServlet() {

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest req, HttpServletResponse resp)

\*/

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

// TODO Auto-generated method stub

resp.getWriter().println("Get:/helloMavenWebapp");

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest req, HttpServletResponse resp)

\*/

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

// TODO Auto-generated method stub

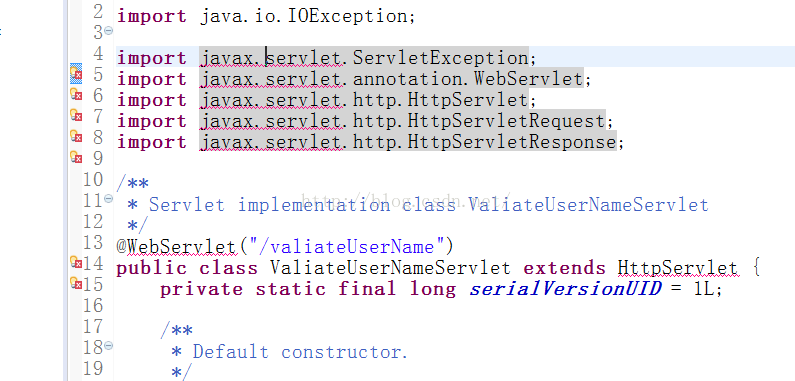
resp.getWriter().println("Post:/helloMavenWebapp");

}

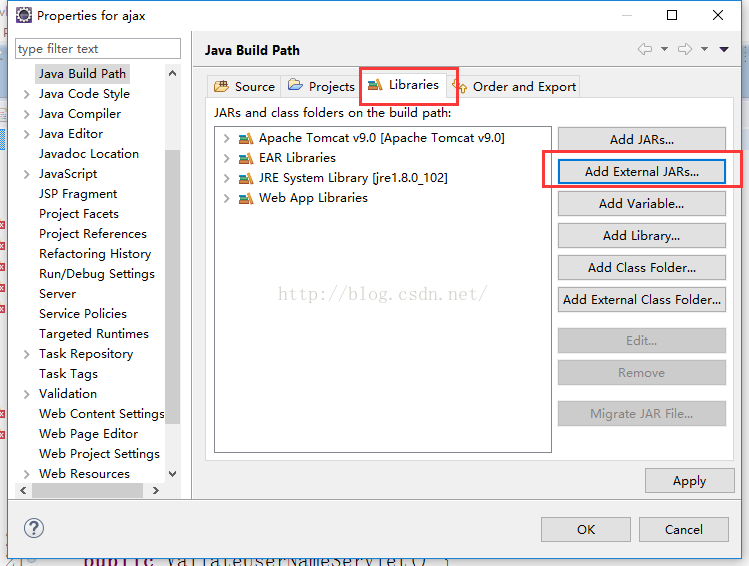
}

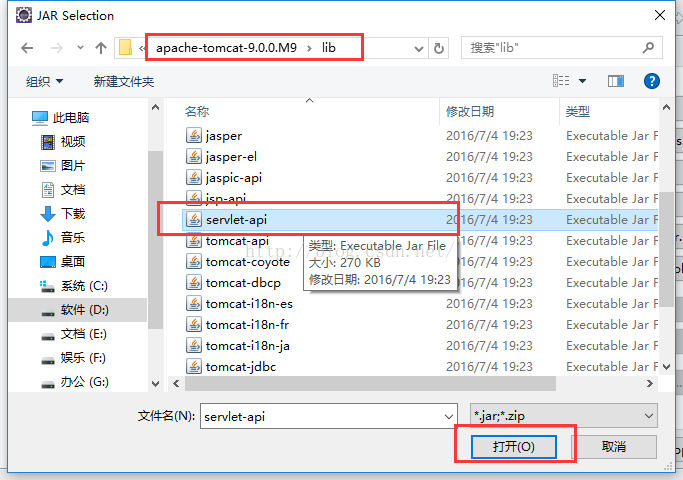
若出现出现----The import javax.servlet cannot be resolved 的解决方法

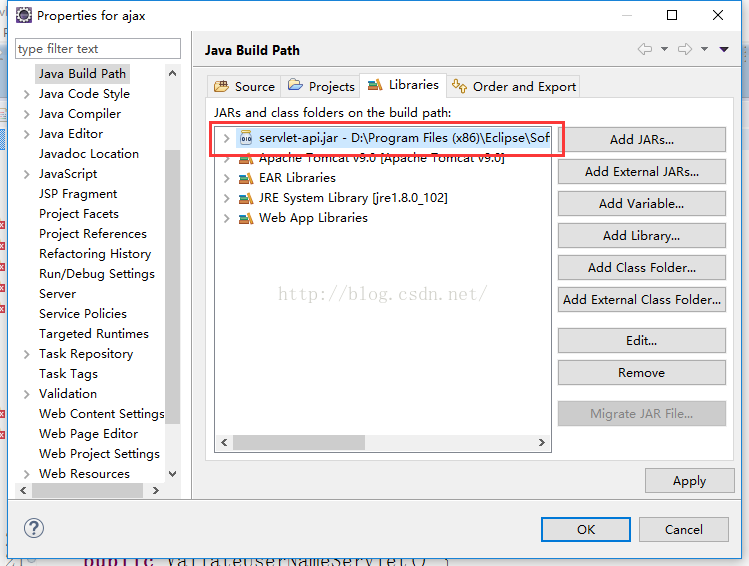
在Eclipse 中编写servlet时出现：The import javax.servlet cannot be resolved 错误，如下图

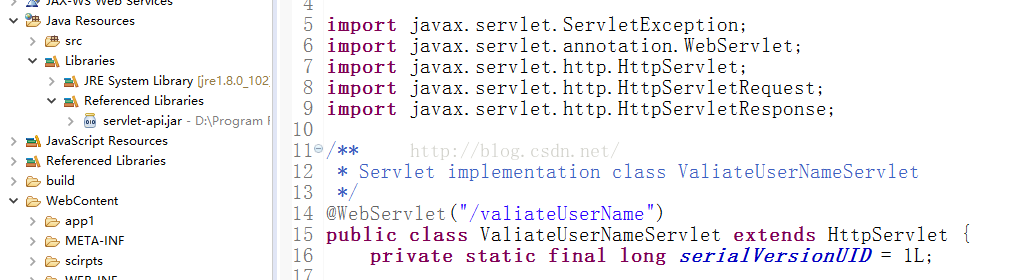


解决方法就是：在Eclipse中，右击项目，选择Build Path->configure build path->Libraries->Add External JARs，找到你计算机中tomcat的解压路径，在lib文件夹下选中"servlet-api.jar"，添加点击“确定”  ，如下图：









心得：在学习使用一个新的编译器时，应该先确定使用的版本号，查找对应的视频教程，除了认真听讲的同时，注意自己编译器出现的错误，查看控制台信息，百度或者进行翻译查看错误的原因及解决方法。

1. Servlet基础

**Servlet生命周期**

**Servlet生命周期**可被定义为从创建直到毁灭的整个过程。以下是 Servlet 遵循的过程：

Servlet 通过调用 init () 方法进行初始化—》调用 service() 方法来处理客户端的请求-》Servlet 通过调用 **destroy()** 方法终止（结束）--》最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的。



Init（）方法、service() 方法、doGet() 方法、doPost() 方法、destroy() 方法的使用查看菜鸟教程<http://www.runoob.com/servlet/servlet-life-cycle.html>

1. POST 请求来自于一个特别指定了 METHOD 为 POST 的 HTML 表单，它由 doPost() 方法处理。

**Servlet 表单数据**

将信息从浏览器传到web服务器，浏览器使用两种方法：get或post。

GET方法：（默认）会产生一个很长的字符串，显示在浏览器地址栏，如<http://www.test.com/hello?key1=value1&key2=value2>

大小限制：请求字符串中最多只能有 1024 个字符。（要向服务器传递的是密码或其他的敏感信息，请不要使用 GET 方法）

* POST方法：POST 方法打包信息的方式与 GET 方法基本相同，但是 POST 方法不是把信息作为 URL 中 ? 字符后的文本字符串进行发送，而是把这些信息作为一个单独的消息。消息以标准输出的形式传到后台程序，您可以解析和使用这些标准输出。Servlet 使用 doPost() 方法处理这种类型的请求。

（比较可靠：但不是绝对的安全）

*注意：如果表单提交的数据中有中文数据则需要转码：*

String name =new String(request.getParameter("name").getBytes("ISO8859-1"),"UTF-8");

**读取所有的表单参数**

以下是通用的实例，使用 HttpServletRequest 的 **getParameterNames()** 方法读取所有可用的表单参数。该方法返回一个枚举，其中包含未指定顺序的参数名。

一旦我们有一个枚举，我们可以以标准方式**循环枚举**，使用 *hasMoreElements()* 方法来确定何时停止，使用 *nextElement()* 方法来获取每个参数的名称。

**读取 HTTP 头的方法**

下面的方法可用在 Servlet 程序中读取 HTTP 头。这些方法通过 *HttpServletRequest* 对象可用。

|  |  |
| --- | --- |
| **序号** | **方法 & 描述** |
| 1 | **Cookie[] getCookies()** 返回一个数组，包含客户端发送该请求的所有的 Cookie 对象。 |
| 2 | **Enumeration getAttributeNames()** 返回一个枚举，包含提供给该请求可用的属性名称。 |
| 3 | **Enumeration getHeaderNames()** 返回一个枚举，包含在该请求中包含的所有的头名。 |
| 4 | **Enumeration getParameterNames()** 返回一个 String 对象的枚举，包含在该请求中包含的参数的名称。 |
| 5 | **HttpSession getSession()** 返回与该请求关联的当前 session 会话，或者如果请求没有 session 会话，则创建一个。 |
| 6 | **HttpSession getSession(boolean create)** 返回与该请求关联的当前 HttpSession，或者如果没有当前会话，且创建是真的，则返回一个新的 session 会话。 |
| 7 | **Locale getLocale()** 基于 Accept-Language 头，返回客户端接受内容的首选的区域设置。 |
| 8 | **Object getAttribute(String name)** 以对象形式返回已命名属性的值，如果没有给定名称的属性存在，则返回 null。 |
| 9 | **ServletInputStream getInputStream()** 使用 ServletInputStream，以二进制数据形式检索请求的主体。 |
| 10 | **String getAuthType()** 返回用于保护 Servlet 的身份验证方案的名称，例如，"BASIC" 或 "SSL"，如果JSP没有受到保护则返回 null。 |
| 11 | **String getCharacterEncoding()** 返回请求主体中使用的字符编码的名称。 |
| 12 | **String getContentType()** 返回请求主体的 MIME 类型，如果不知道类型则返回 null。 |
| 13 | **String getContextPath()** 返回指示请求上下文的请求 URI 部分。 |
| 14 | **String getHeader(String name)** 以字符串形式返回指定的请求头的值。 |
| 15 | **String getMethod()** 返回请求的 HTTP 方法的名称，例如，GET、POST 或 PUT。 |
| 16 | **String getParameter(String name)** 以字符串形式返回请求参数的值，或者如果参数不存在则返回 null。 |
| 17 | **String getPathInfo()** 当请求发出时，返回与客户端发送的 URL 相关的任何额外的路径信息。 |
| 18 | **String getProtocol()** 返回请求协议的名称和版本。 |
| 19 | **String getQueryString()** 返回包含在路径后的请求 URL 中的查询字符串。 |
| 20 | **String getRemoteAddr()** 返回发送请求的客户端的互联网协议（IP）地址。 |
| 21 | **String getRemoteHost()** 返回发送请求的客户端的完全限定名称。 |
| 22 | **String getRemoteUser()** 如果用户已通过身份验证，则返回发出请求的登录用户，或者如果用户未通过身份验证，则返回 null。 |
| 23 | **String getRequestURI()** 从协议名称直到 HTTP 请求的第一行的查询字符串中，返回该请求的 URL 的一部分。 |
| 24 | **String getRequestedSessionId()** 返回由客户端指定的 session 会话 ID。 |
| 25 | **String getServletPath()** 返回调用 JSP 的请求的 URL 的一部分。 |
| 26 | **String[] getParameterValues(String name)** 返回一个字符串对象的数组，包含所有给定的请求参数的值，如果参数不存在则返回 null。 |
| 27 | **boolean isSecure()** 返回一个布尔值，指示请求是否使用安全通道，如 HTTPS。 |
| 28 | **int getContentLength()** 以字节为单位返回请求主体的长度，并提供输入流，或者如果长度未知则返回 -1。 |
| 29 | **int getIntHeader(String name)** 返回指定的请求头的值为一个 int 值。 |
| 30 | **int getServerPort()** 返回接收到这个请求的端口号。 |
| 31 | **int getParameterMap()** 将参数封装成 Map 类型。 |

# Servlet 编写过滤器

过滤器通过 Web 部署描述符（web.xml）中的 XML 标签来声明，然后映射到您的应用程序的部署描述符中的 Servlet 名称或 URL 模式。

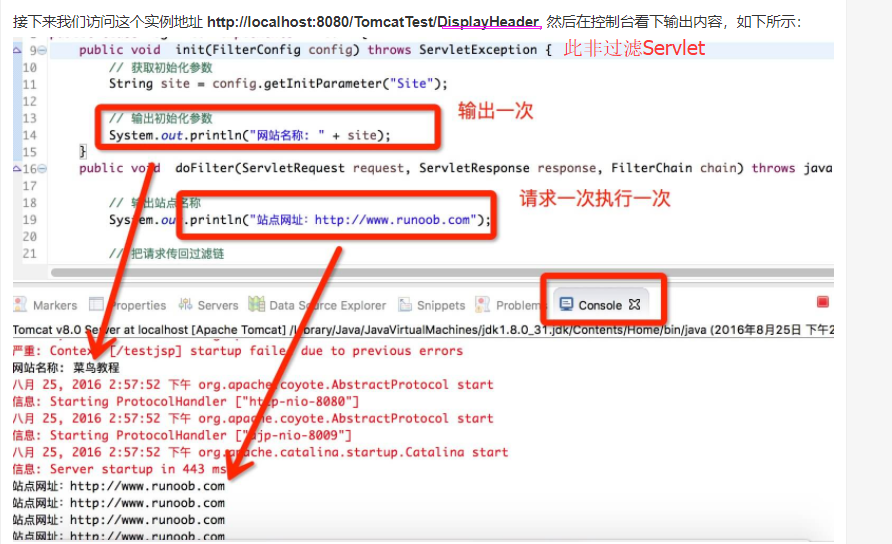
当 Web 容器启动 Web 应用程序时，它会为您在部署描述符中声明的每一个过滤器创建一个实例。

Filter的执行顺序与在web.xml配置文件中的配置顺序一致，一般把Filter配置在所有的Servlet之前。

假设有一个名为LogFilter的过滤器。那么在部署描述符文件web.xml中为filter标签创建下面的条目



上述过滤器适用于所有的 Servlet，因为我们在配置中指定 **/\*** 。如果您只想在少数的 Servlet 上应用过滤器，您可以指定一个特定的 Servlet 路径。



也可以使用多个过滤器，映射的编写如下



## 过滤器的应用顺序

web.xml 中的 filter-mapping 元素的顺序决定了 Web 容器应用过滤器到 Servlet 的顺序。若要反转过滤器的顺序，您只需要在 web.xml 文件中反转 filter-mapping 元素即可。

例如，上面的实例将先应用 LogFilter，然后再应用 AuthenFilter，但是下面的实例将颠倒这个顺序：

<filter-mapping>

<filter-name>AuthenFilter</filter-name>

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

</filter-mapping>

<filter-mapping>

<filter-name>LogFilter</filter-name>

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

</filter-mapping>

**web.xml配置各节点说明**

* <filter>指定一个过滤器。
  + <filter-name>用于为过滤器指定一个名字，该元素的内容不能为空。
  + <filter-class>元素用于指定过滤器的完整的限定类名。
  + <init-param>元素用于为过滤器指定初始化参数，它的子元素<param-name>指定参数的名字，<param-value>指定参数的值。
  + 在过滤器中，可以使用FilterConfig接口对象来访问初始化参数。
* <filter-mapping>元素用于设置一个 Filter 所负责拦截的资源。一个Filter拦截的资源可通过两种方式来指定：Servlet 名称和资源访问的请求路径
  + <filter-name>子元素用于设置filter的注册名称。该值必须是在<filter>元素中声明过的过滤器的名字
  + <url-pattern>设置 filter 所拦截的请求路径(过滤器关联的URL样式)
* <servlet-name>指定过滤器所拦截的Servlet名称。
* <dispatcher>指定过滤器所拦截的资源被 Servlet 容器调用的方式，可以是REQUEST,INCLUDE,FORWARD和ERROR之一，默认REQUEST。用户可以设置多个<dispatcher>子元素用来指定 Filter 对资源的多种调用方式进行拦截。
* <dispatcher>子元素可以设置的值及其意义
  + REQUEST：当用户直接访问页面时，Web容器将会调用过滤器。如果目标资源是通过RequestDispatcher的include()或forward()方法访问时，那么该过滤器就不会被调用。
  + INCLUDE：如果目标资源是通过RequestDispatcher的include()方法访问时，那么该过滤器将被调用。除此之外，该过滤器不会被调用。
  + FORWARD：如果目标资源是通过RequestDispatcher的forward()方法访问时，那么该过滤器将被调用，除此之外，该过滤器不会被调用。
  + ERROR：如果目标资源是通过声明式异常处理机制调用时，那么该过滤器将被调用。除此之外，过滤器不会被调用。

## Servlet 错误处理程序实例

以下是 Servlet 实例，将应对任何您所定义的错误或异常发生时的错误处理程序。

本实例让您对 Servlet 中的异常处理有基本的了解，您可以使用相同的概念编写更复杂的异常处理应用程序：

//导入必需的 java 库

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.\*;

//扩展 HttpServlet 类

public class ErrorHandler extends HttpServlet {

    // 处理 GET 方法请求的方法

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

    {

        Throwable throwable = (Throwable)

        request.getAttribute("javax.servlet.error.exception");

        Integer statusCode = (Integer)

        request.getAttribute("javax.servlet.error.status\_code");

        String servletName = (String)

        request.getAttribute("javax.servlet.error.servlet\_name");

        if (servletName == null){

            servletName = "Unknown";

        }

        String requestUri = (String)

        request.getAttribute("javax.servlet.error.request\_uri");

        if (requestUri == null){

            requestUri = "Unknown";

        }

        // 设置响应内容类型

        response.setContentType("text/html;charset=UTF-8");

        PrintWriter out = response.getWriter();

        String title = "菜鸟教程 Error/Exception 信息";

        String docType = "<!DOCTYPE html>\n";

        out.println(docType +

         "<html>\n" +

         "<head><title>" + title + "</title></head>\n" +

         "<body bgcolor=\"#f0f0f0\">\n");

         out.println("<h1>菜鸟教程异常信息实例演示</h1>");

         if (throwable == null && statusCode == null){

         out.println("<h2>错误信息丢失</h2>");

         out.println("请返回 <a href=\"" +

     response.encodeURL("http://localhost:8080/") +

         "\">主页</a>。");

         }else if (statusCode != null) {

         out.println("错误代码 : " + statusCode);

        }else{

             out.println("<h2>错误信息</h2>");

         out.println("Servlet Name : " + servletName +

     "</br></br>");

         out.println("异常类型 : " +

     throwable.getClass( ).getName( ) +

     "</br></br>");

         out.println("请求 URI: " + requestUri +

     "<br><br>");

         out.println("异常信息: " +

     throwable.getMessage( ));

         }

         out.println("</body>");

         out.println("</html>");

    }

    // 处理 POST 方法请求的方法

    public void doPost(HttpServletRequest request,

     HttpServletResponse response)

     throws ServletException, IOException {

        doGet(request, response);

    }

}

以通常的方式编译 **ErrorHandler.java**，把您的类文件放入<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中。（注意：POM和Eclipse项目忽略这一步）

让我们在 web.xml 文件中添加如下配置来处理异常：

<?xml version="1.0" encoding="UTF-8"?>

<web-app>

<servlet>

<servlet-name>ErrorHandler</servlet-name>

<servlet-class>com.runoob.test.ErrorHandler</servlet-class>

</servlet>

<!-- servlet mappings -->

<servlet-mapping>

<servlet-name>ErrorHandler</servlet-name>

<url-pattern>/TomcatTest/ErrorHandler</url-pattern>

</servlet-mapping>

<error-page>

<error-code>404</error-code>

<location>/TomcatTest/ErrorHandler</location>

</error-page>

<error-page>

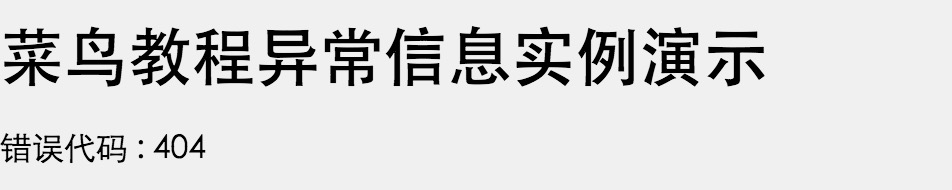
<exception-type>java.lang.Throwable</exception-type >

<location>/ErrorHandler</location>

</error-page>

</web-app>

现在，尝试使用一个会产生异常的 Servlet，或者输入一个错误的 URL，这将触发 Web 容器调用 **ErrorHandler** 的 Servlet，并显示适当的消息。例如，如果您输入了一个错误的 URL（如：http://localhost:8080/TomcatTest/UnKonwPage），那么它将显示下面的结果：（注意：有些浏览器可能无法显示，有些则可以）



# Servlet Cookie 处理

## 通过 Servlet 设置 Cookie

通过 Servlet 设置 Cookie 包括三个步骤：

**(1) 创建一个 Cookie 对象：**您可以调用带有 cookie 名称和 cookie 值的 Cookie 构造函数，cookie 名称和 cookie 值都是字符串。

Cookie cookie = new Cookie("key","value");

请记住，无论是名字还是值，都不应该包含空格或以下任何字符：

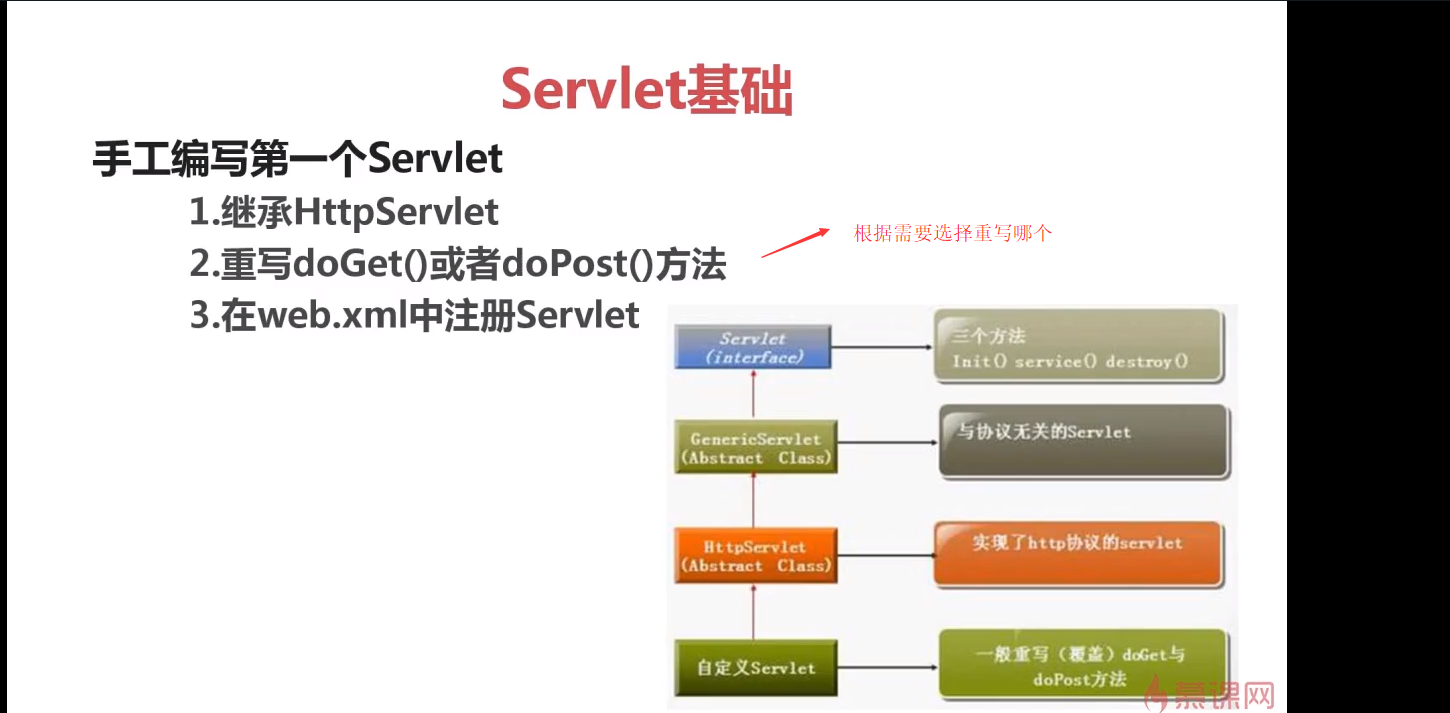
[ ] ( ) = , " / ? @ : ;

**(2) 设置最大生存周期：**您可以使用 setMaxAge 方法来指定 cookie 能够保持有效的时间（以秒为单位）。下面将设置一个最长有效期为 24 小时的 cookie。

cookie.setMaxAge(60\*60\*24);

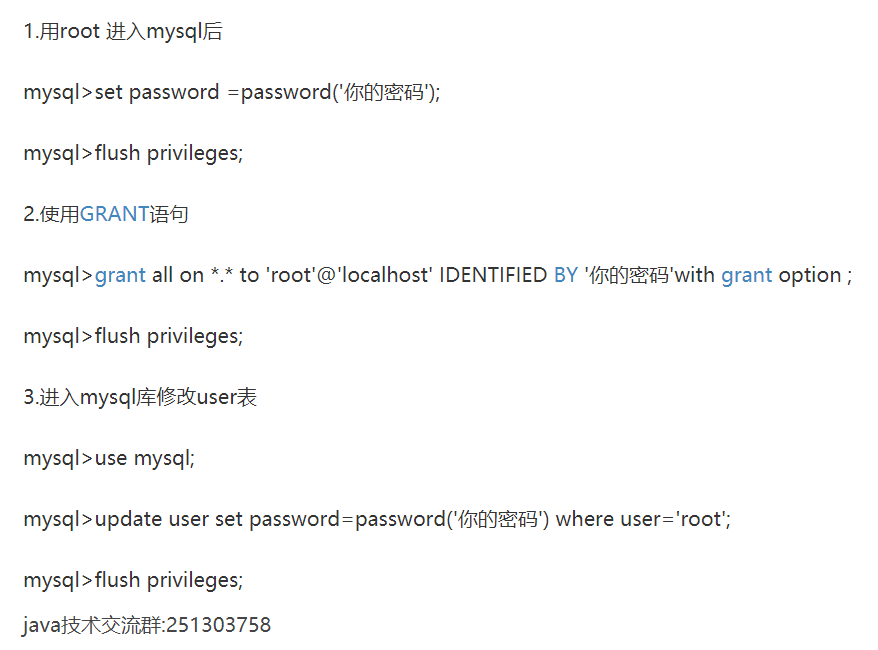
**(3) 发送 Cookie 到 HTTP 响应头：**您可以使用 **response.addCookie** 来添加 HTTP 响应头中的 Cookie，如下所示：

response.addCookie(cookie);

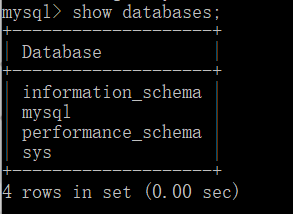
1. 
2. 二被告
3. MySQL的下载与安装。
4. 基本安装—

mysql-5.7.16-winx64解压版安装超详细图文教程<http://blog.csdn.net/li_magic/article/details/70186609>

1. 设置密码 <http://blog.csdn.net/li_magic/article/details/70187609>



1. 使用mysql -hlocalhost -uroot -p或者是mysql -u root -p；命令进入MySQL
2. 数据库的创建；
3. 进入MySQL后，输入show databases 命令，展示数据库列表。



使用时，首先创建数据库，



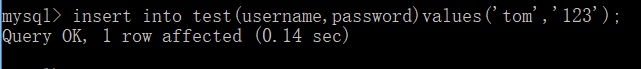
切换到需要使用的数据库

创建表；

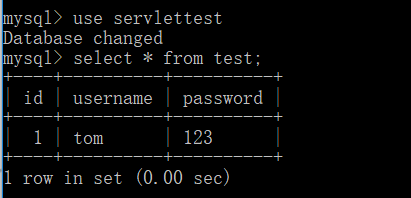


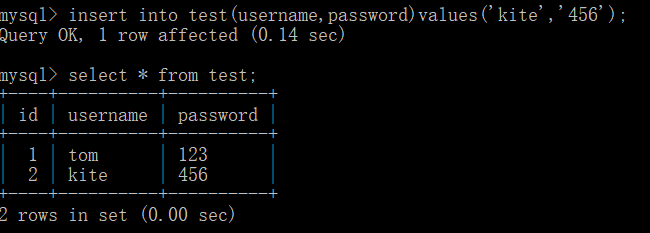
进行对表的更改

如：插入



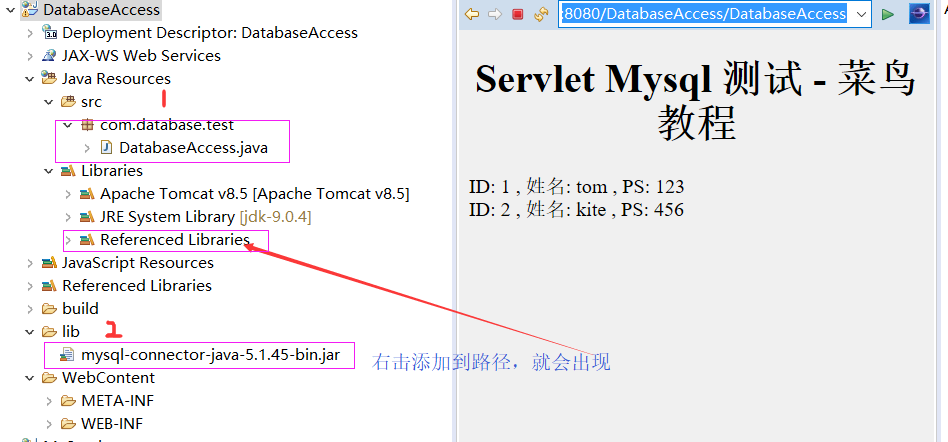
如需查看某数据库的某个表。





（退出MySQL命令：exit; ）

1. 数据库创建好后，进行Servlet与数据库的连接。



DatabaseAccess.java代码

package com.runoob.test;

//改成自己的包

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.\*;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class DatabaseAccess

\*/

@WebServlet("/DatabaseAccess")

public class DatabaseAccess extends HttpServlet {

    private static final long serialVersionUID = 1L;

    // JDBC 驱动名及数据库 URL

    static final String JDBC\_DRIVER = "com.mysql.jdbc.Driver";

    static final String DB\_URL = "jdbc:mysql://localhost:3306/RUNOOB";

    // 数据库的用户名与密码，需要根据自己的设置

    static final String USER = "root";

    static final String PASS = "123456";

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public DatabaseAccess() {

super();

// TODO Auto-generated constructor stub

}

    /\*\*

     \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

     \*/

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        Connection conn = null;

        Statement stmt = null;

        // 设置响应内容类型

        response.setContentType("text/html;charset=UTF-8");

        PrintWriter out = response.getWriter();

        String title = "Servlet Mysql 测试 - 菜鸟教程";

        String docType = "<!DOCTYPE html>\n";

        out.println(docType +

        "<html>\n" +

        "<head><title>" + title + "</title></head>\n" +

        "<body bgcolor=\"#f0f0f0\">\n" +

        "<h1 align=\"center\">" + title + "</h1>\n");

        try{

            // 注册 JDBC 驱动器

            Class.forName("com.mysql.jdbc.Driver");

            // 打开一个连接

            conn = DriverManager.getConnection(DB\_URL,USER,PASS);

            // 执行 SQL 查询

            stmt = conn.createStatement();

            String sql;

            sql = "SELECT id, name, url FROM websites";

            ResultSet rs = stmt.executeQuery(sql);

            // 展开结果集数据库

            while(rs.next()){

                // 通过字段检索

                int id = rs.getInt("id");

                String name = rs.getString("name");

                String url = rs.getString("url");

                // 输出数据

                out.println("ID: " + id);

                out.println(", 站点名称: " + name);

                out.println(", 站点 URL: " + url);

                out.println("<br />");

            }

            out.println("</body></html>");

            // 完成后关闭

            rs.close();

            stmt.close();

            conn.close();

        } catch(SQLException se) {

            // 处理 JDBC 错误

            se.printStackTrace();

        } catch(Exception e) {

            // 处理 Class.forName 错误

            e.printStackTrace();

        }finally{

            // 最后是用于关闭资源的块

            try{

                if(stmt!=null)

                stmt.close();

            }catch(SQLException se2){

            }

            try{

                if(conn!=null)

                conn.close();

            }catch(SQLException se){

                se.printStackTrace();

            }

        }

    }

    /\*\*

     \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

     \*/

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        // TODO Auto-generated method stub

        doGet(request, response);

    }

}

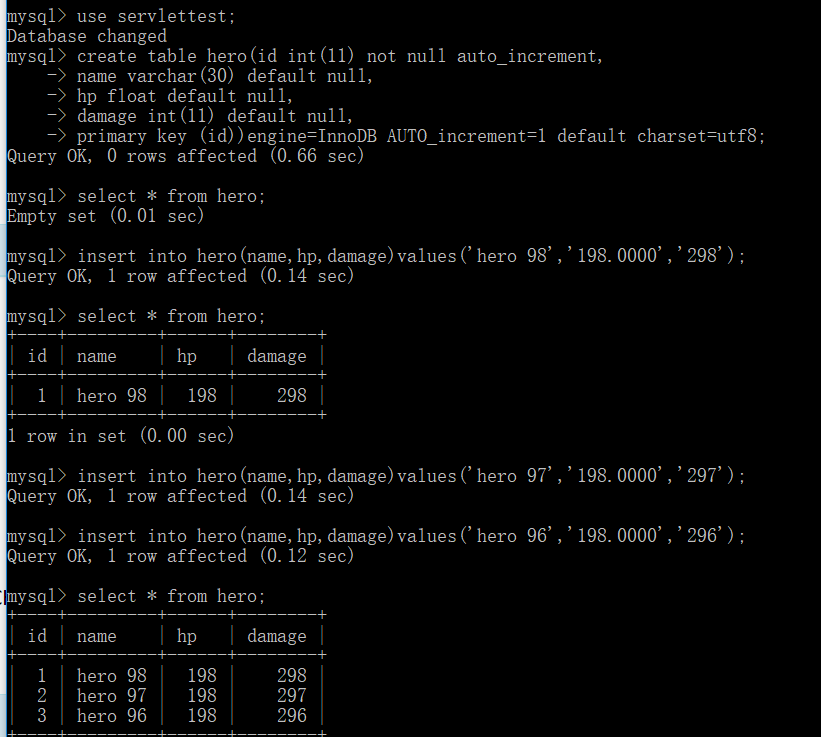
若出现无法连接到数据库的错误提示，检查是否将第二步的jar包引入路径，再者就是运行时重新启动Tomcat

maven3.0+；eclipse EE；jdk 9；Tomcat8.5，MySQL-**--动态web之数据库的使用。**

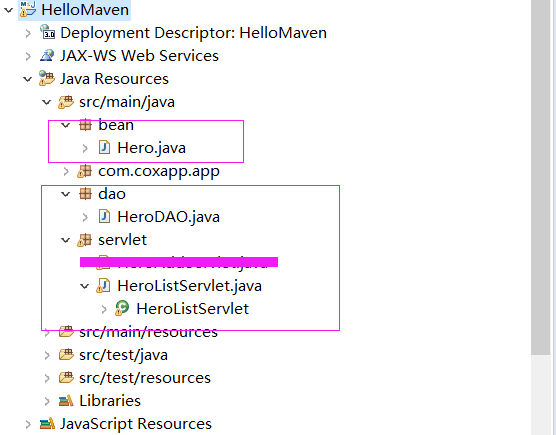
**学习网址：**[**http://how2j.cn/k/servlet/servlet-query/563.html**](http://how2j.cn/k/servlet/servlet-query/563.html)

数据库CRUD（C - Creation 增加 ；R - Retrieve 查询 ；U - Update 修改 ；D - DELETE 删除 ）的综合学习使用。

1. 查询数据库的信息并显示在网页上。
2. 先建立数据库，创建表hero，并添加一些数据，方法同上。（连接数据库）



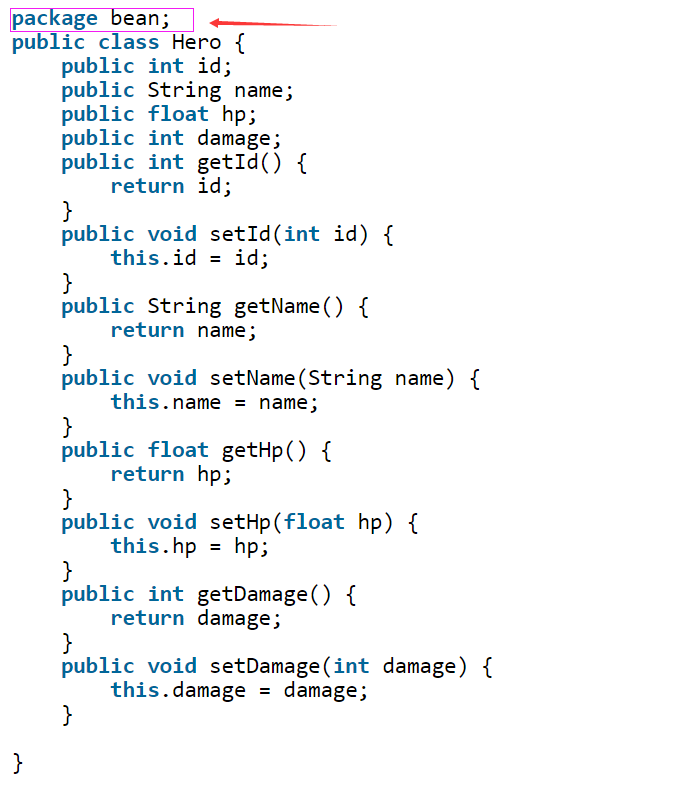
1. 准备实体类Hero
2. 准备DAO类和HeroDAO
3. 编写HeroListServlet
4. Run as Servlet。访问<http://localhost:8080/HelloMaven/HeroListServlet>
5. 在工程中的位置



1. 代码如下

Hero类有id,name,hp,damage等属性。

并且为每一个属性提供public的getter和setter。



准备一个HeroDAO，提供增加，删除，修改，查询等常规数据库操作方法（注意引进的包）

package dao;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import bean.Hero;

public class HeroDAO {

    public HeroDAO() {

        try {

            Class.forName("com.mysql.jdbc.Driver");

        } catch (ClassNotFoundException e) {

            e.printStackTrace();

        }

    }

    public Connection getConnection() throws SQLException {

        return DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/how2java?characterEncoding=UTF-8","root",

                "admin");

    }

    public int getTotal() {

        int total = 0;

        try (Connection c = getConnection(); Statement s = c.createStatement();) {

            String sql = "select count(\*) from hero";

            ResultSet rs = s.executeQuery(sql);

            while (rs.next()) {

                total = rs.getInt(1);

            }

            System.out.println("total:" + total);

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return total;

    }

    public void add(Hero hero) {

        String sql = "insert into hero values(null,?,?,?)";

        try (Connection c = getConnection(); PreparedStatement ps = c.prepareStatement(sql);) {

            ps.setString(1, hero.name);

            ps.setFloat(2, hero.hp);

            ps.setInt(3, hero.damage);

            ps.execute();

            ResultSet rs = ps.getGeneratedKeys();

            if (rs.next()) {

                int id = rs.getInt(1);

                hero.id = id;

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    public void update(Hero hero) {

        String sql = "update hero set name= ?, hp = ? , damage = ? where id = ?";

        try (Connection c = getConnection(); PreparedStatement ps = c.prepareStatement(sql);) {

            ps.setString(1, hero.name);

            ps.setFloat(2, hero.hp);

            ps.setInt(3, hero.damage);

            ps.setInt(4, hero.id);

            ps.execute();

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    public void delete(int id) {

        try (Connection c = getConnection(); Statement s = c.createStatement();) {

            String sql = "delete from hero where id = " + id;

            s.execute(sql);

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    public Hero get(int id) {

        Hero hero = null;

        try (Connection c = getConnection(); Statement s = c.createStatement();) {

            String sql = "select \* from hero where id = " + id;

            ResultSet rs = s.executeQuery(sql);

            if (rs.next()) {

                hero = new Hero();

                String name = rs.getString(2);

                float hp = rs.getFloat("hp");

                int damage = rs.getInt(4);

                hero.name = name;

                hero.hp = hp;

                hero.damage = damage;

                hero.id = id;

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return hero;

    }

    public List<Hero> list() {

        return list(0, Short.MAX\_VALUE);

    }

    public List<Hero> list(int start, int count) {

        List<Hero> heros = new ArrayList<Hero>();

        String sql = "select \* from hero order by id desc limit ?,? ";

        try (Connection c = getConnection(); PreparedStatement ps = c.prepareStatement(sql);) {

            ps.setInt(1, start);

            ps.setInt(2, count);

            ResultSet rs = ps.executeQuery();

            while (rs.next()) {

                Hero hero = new Hero();

                int id = rs.getInt(1);

                String name = rs.getString(2);

                float hp = rs.getFloat("hp");

                int damage = rs.getInt(4);

                hero.id = id;

                hero.name = name;

                hero.hp = hp;

                hero.damage = damage;

                heros.add(hero);

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return heros;

    }

}

一般会这样命名,以查询为例 HeroListServlet ： [表][行为]Servlet 这样一种命名规则。  
  
所以对于Hero而言就会如此命名：  
增加 HeroAddServlet  
删除 HeroDeleteServlet  
编辑 HeroEditServlet  
修改 HeroUpdateServlet  
查询 HeroListServlet  
  
  
在HeroListServlet中，会使用HeroDAO把数据查询出来，然后拼接成一个table用于显示其内容

package servlet;

import java.io.IOException;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import bean.Hero;

import dao.HeroDAO;

public class HeroListServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html; charset=UTF-8");

List<Hero> heros = new HeroDAO().list();

StringBuffer sb = new StringBuffer();

sb.append("<table align='center' border='1' cellspacing='0'>\r\n");

sb.append("<tr><td>id</td><td>name</td><td>hp</td><td>damage</td></tr>\r\n");

String trFormat = "<tr><td>%d</td><td>%s</td><td>%f</td><td>%d</td></tr>\r\n";

for (Hero hero : heros) {

String tr = String.format(trFormat, hero.getId(), hero.getName(), hero.getHp(), hero.getDamage());

sb.append(tr);

}

sb.append("</table>");

response.getWriter().write(sb.toString());

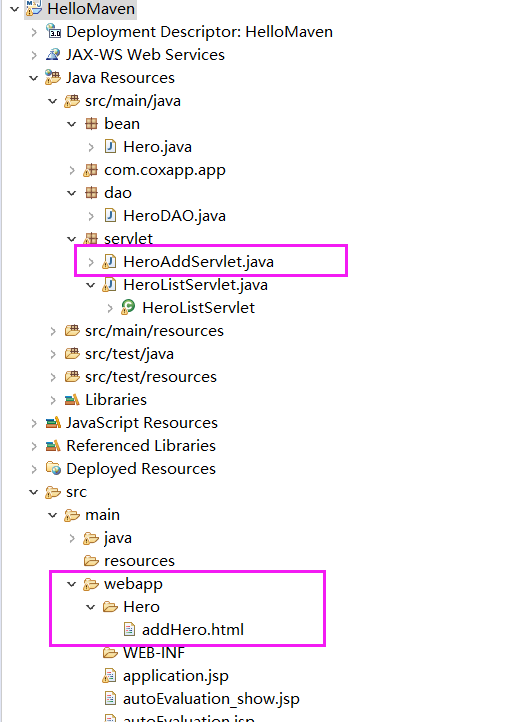
}

}

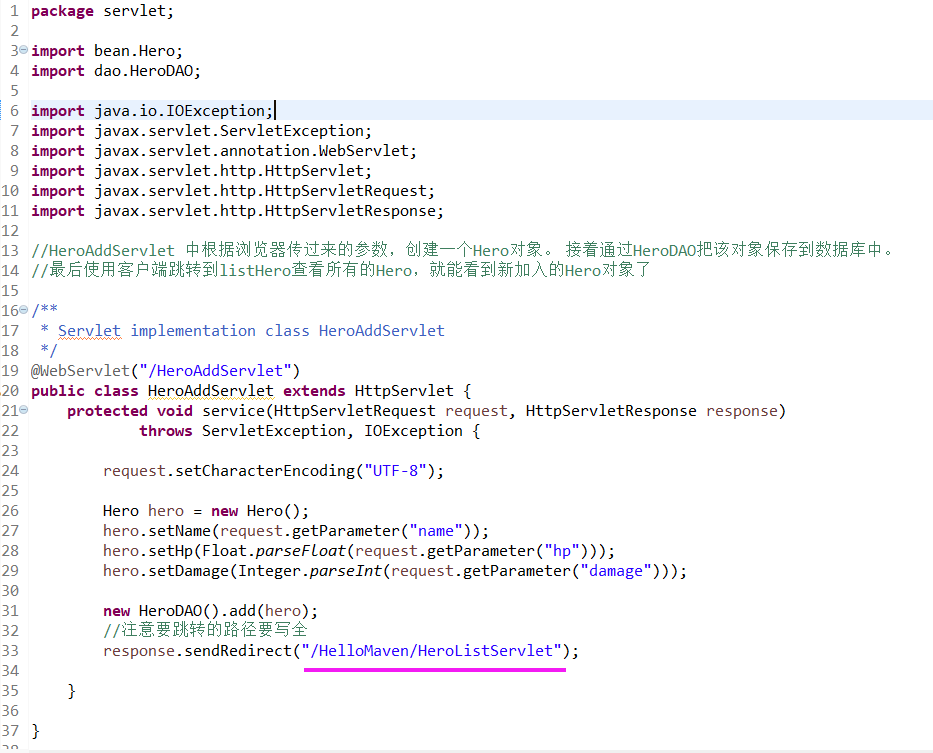
1. 通过在网页上添加数据，将数据存储到数据库中

在“一”的基础上，进行“二”的操作。

1. 准备增加的页面 addHero.html
2. 编写 HeroAddServlet（HeroAddServlet 中根据浏览器传过来的参数，创建一个Hero对象。 接着通过HeroDAO把该对象保存到数据库中。  
   最后使用客户端跳转到listHero查看所有的Hero，就能看到新加入的Hero对象了）
3. 在工程中的位置



1. 代码

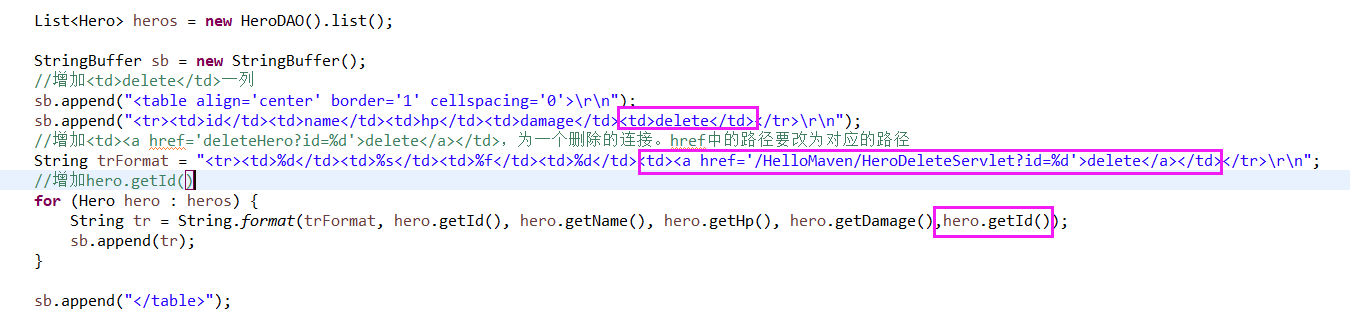
访问网址：<http://localhost:8080/HelloMaven/Hero/addHero.html>

添加后，也可在数据库中查询，可以看到新添加的数据。

1. 删除

在“二”的基础上操作

1. 更改HeroListServlet.java,使表格多一列delete。



1. 新增HeroDeleteServlet.java.

**package** servlet;

**import** dao.HeroDAO;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.annotation.WebServlet;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class HeroDeleteServlet

\*/

@WebServlet("/HeroDeleteServlet")

**public** **class** HeroDeleteServlet **extends** HttpServlet {

**protected** **void** service(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

**int** id = Integer.*parseInt*(request.getParameter("id"));

**new** HeroDAO().delete(id);

response.sendRedirect("/HelloMaven/HeroListServlet");

}

}

1. 编辑

在“三”的基础上操作。（编辑后需要第“五”步的HeroUpdataServlet.java的程序将其更新）

1. 更改HeroListServlet.java,使表格多一列edit。



1. 新增HeroEditServlet.java。（注意表单中action的URL）

package servlet;

import bean.Hero;

import dao.HeroDAO;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class HeroEditServlet

\*/

@WebServlet("/HeroEditServlet")

public class HeroEditServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

int id = Integer.parseInt(request.getParameter("id"));

Hero hero = new HeroDAO().get(id);

StringBuffer format = new StringBuffer();

response.setContentType("text/html; charset=UTF-8");

format.append("<!DOCTYPE html>");

format.append("<form action='/HelloMaven/HeroUpdataServlet' method='post'>");

format.append("名字 ： <input type='text' name='name' value='%s' > <br>");

format.append("血量 ： <input type='text' name='hp' value='%f' > <br>");

format.append("伤害： <input type='text' name='damage' value='%d' > <br>");

format.append("<input type='hidden' name='id' value='%d'>");

format.append("<input type='submit' value='更新'>");

format.append("</form>");

String html = String.format(format.toString(), hero.getName(), hero.getHp(), hero.getDamage(), hero.getId());

response.getWriter().write(html);

}

}

1. 更新。（有缺陷，就是如果之前有删除，那么新添加的信息的序号会出现断号问题，即序号不衔接！！）

帮助“四”完成更新。只需添加一个HeroUpdataServlet.java程序。（！！注意response重定位的URL）

package servlet;

import bean.Hero;

import dao.HeroDAO;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class HeroUpdataServlet

\*/

@WebServlet("/HeroUpdataServlet")

public class HeroUpdataServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

request.setCharacterEncoding("UTF-8");

Hero hero = new Hero();

hero.setId(Integer.parseInt(request.getParameter("id")));

hero.setName(request.getParameter("name"));

hero.setHp(Float.parseFloat(request.getParameter("hp")));

hero.setDamage(Integer.parseInt(request.getParameter("damage")));

new HeroDAO().update(hero);

response.sendRedirect("/HelloMaven/HeroListServlet");

}

}

使用Servlet进行CRUD开发的弊端：

在Servlet编写html代码很痛苦，效率不高，可读性差，难以维护。  
  
最好可以在html文件里面写html代码，同时又能在里面调用java的变量，那么这样就需要学习[JSP](http://how2j.cn/k/jsp/jsp-tutorials/530.html)了。（jsp，可以在HTML中写java代码）

程序代码中：try……catch的用法。

首先要清楚，如果没有try的话，出现异常会导致程序崩溃。  
而try则可以保证程序的正常运行下去，比如说：  
try{  
int i = 1/0;  
}catch(Exception e){  
........  
}

try { name=dis.readUTF(); age=dis.readInt(); maths=dis.readFloat(); english=dis.readFloat(); physical=dis.readFloat(); chinese=dis.readFloat(); } catch(Exception e){  
//直接输出异常，帮助分析  
e.printStackTrace();  
//如果上面是语句发生异常，则可以自己处理  
name="" //default value  
age =0 ;  
....  
//或往外抛出，自己处理异常  
throw e;  
}

try catch 是捕捉try部分的异常，当你没有trycatch的时候，如果出现异常则程序报错，加上trycatch，出现异常程序正常运行，只是把错误信息存储到Exception里，所以catch是用来提取异常信息的，你可以在Catch部分加上一句System.out.println(e.ToString());，如果出现异常可以把异常打印出来。

# 问题：servlet重定向到jsp后，css样式和图片都没了，解决办法

servlet重定向到jsp后，css样式和图片都没了，原因是在servlet中转发时css和图片的路径变成相对于这个servlet的相对路径而非相对于web项目的路径了。

解决办法：导入css样式和图片时把css写成动态绝对路径，

<http://blog.csdn.net/sun9528/article/details/72935178>



# 问题：使用jsp遍历数据库中表的某列数据。

方法：<https://zhidao.baidu.com/question/378330059.html>



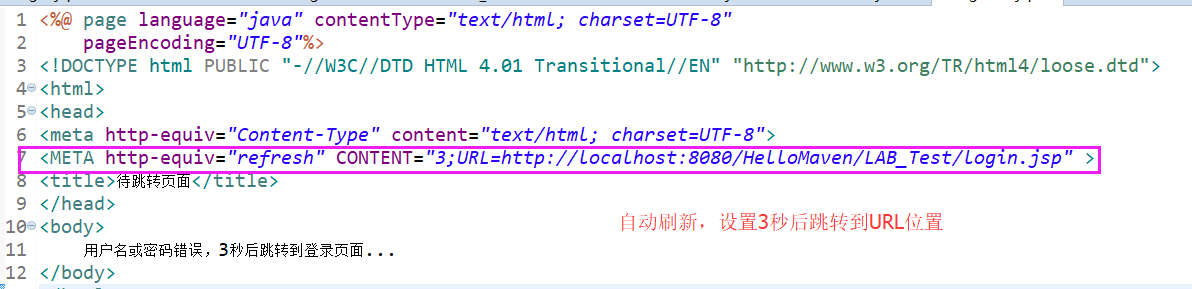


# 问题：访问数据库检查用户身份是否合法，不合法则。。。合法则。。。

可以使用一个跳转页面，在此页面输出错误信息并跳转到登录界面。



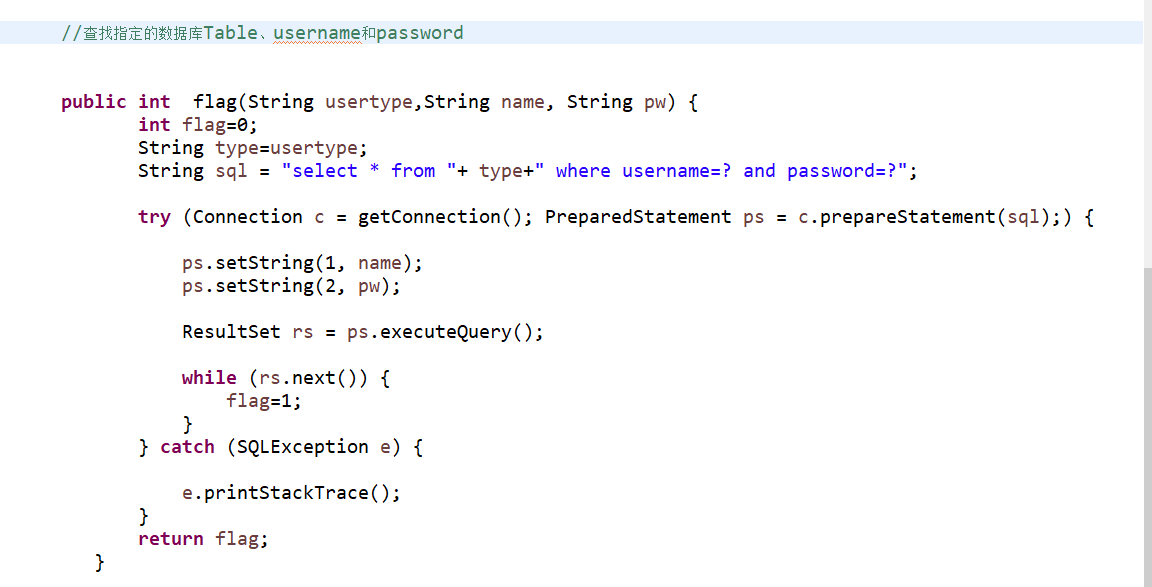
跳转页面如下：



**通过读取用户类型，将其值传送到DAO中，在Servlet中访问不同的数据库表**

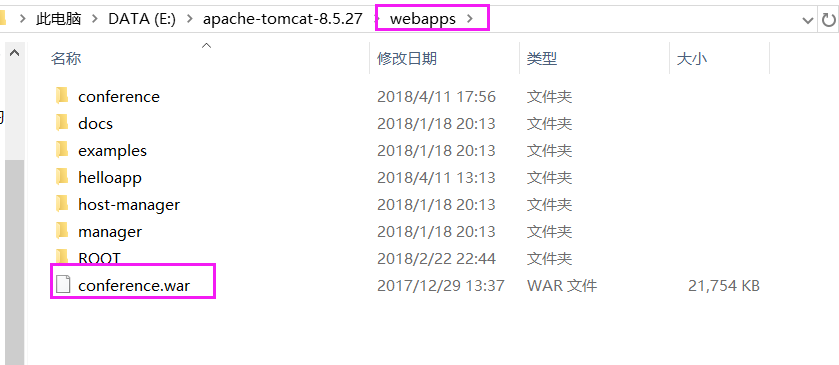


**DAO中的内容，如下图**

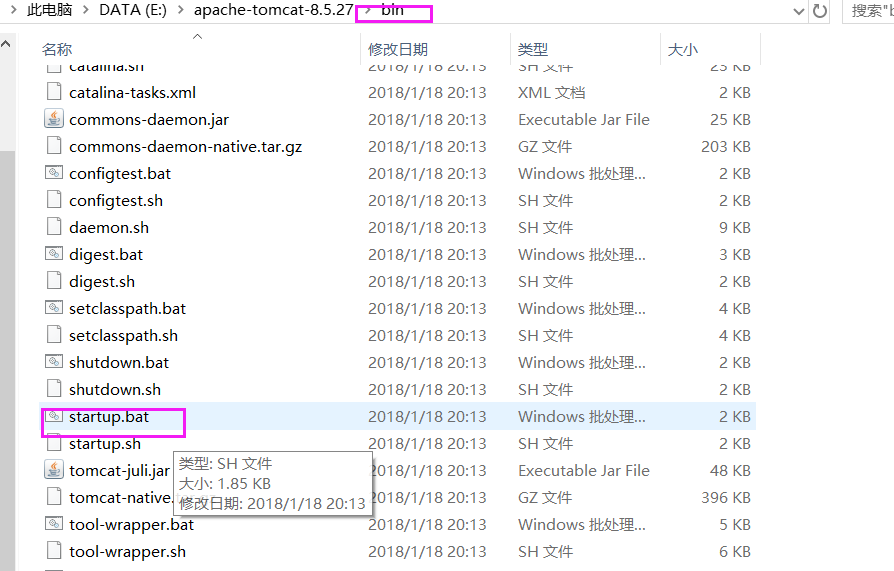


# 问题：如何把war包部署到Tomcat服务器？

1. 导出war包
2. 找到Tomcat安装目录下的webapps，将war包放在此目录



1. 然后在Tomcat的bin目录下找到startup.bat.启动Tomcat服务器，



1. 在浏览器中输入<http://localhost:8080/conference>即可



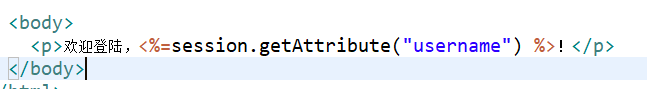
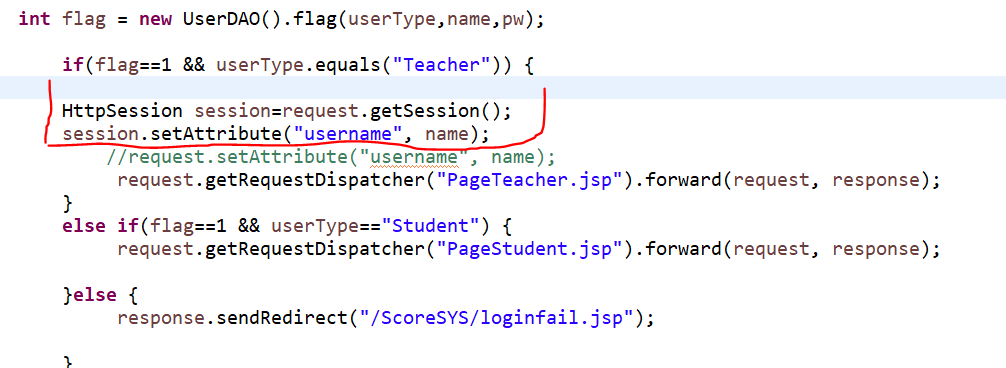
**错误代码：**

**Could not initialize class dbc.JdbcUtil**

**原因很大可能是数据库的****没添加进去，或者是连接数据库的代码写错了**

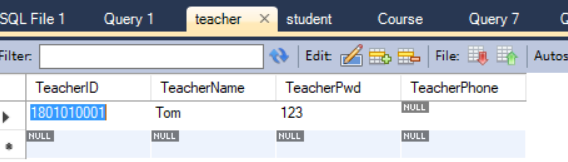
**注意：jsp页面传值不能超过两个页面，如果需要在多个页面中使用某个值，有两种方法。**

1. **在一个纯java代码的jsp页面做中转处理**
2. **通过使用session，session的使用方法**

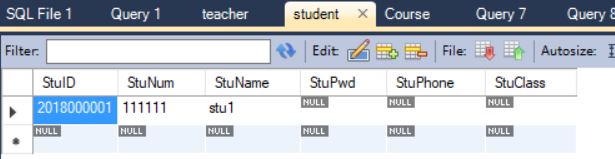


# 问题：MySQL workbench外键的使用

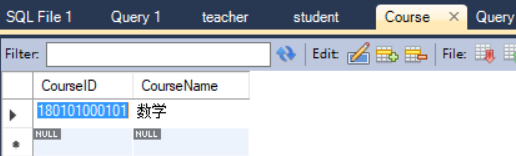
教师表



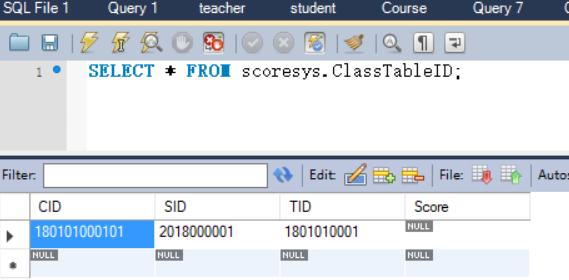
学生表



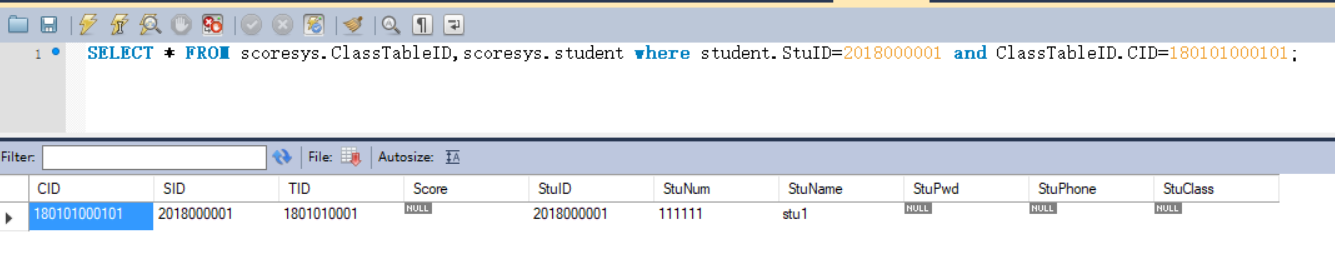
课程表



选课表



通过外键查询信息

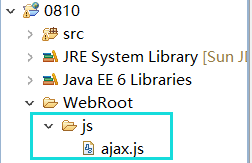


# 问题：MySQL workbench的数据库表中使用外键，容易造成数据插入失败，其原因是因为表间的约束关系。

如约束表中不存在的约束键数据，将导致数据插入失败。解决方法为，在外部使用SQL语句前，现在数据库中对此SQL语句进行测试。还有一个需要注意的点是，主键不能重复。

# 问题：ajax的使用—封装创建ajax到js，method方法为POST

**Ajax.js封装目录：**



**Ajax.js代码（ajax创建对象函数和发动请求函数）**

//声明XMLHttpRequest对象

**var** xmlHttpRequest=**null**;

//创建XMLHttpRequest对象实例的方法

**function** createXHR(){

**if** (window.XMLHttpRequest)

{

// IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码

xmlHttpRequest=**new** XMLHttpRequest();

}

**else**

{

// IE6, IE5 浏览器执行代码

xmlHttpRequest=**new** ActiveXObject("Microsoft.XMLHTTP");

}

**if**(xmlHttpRequest == **null**) {

alert("不能创建Ajax对象！");

}

}

//发送客户端的请求，该方法有4个参数，其中method取值为Post或get

**function** sendRequest(url,params,method,handler){

createXHR();

**if**(!xmlHttpRequest) **return** **false**;

xmlHttpRequest.onreadystatechange = handler;

**if**(method == "GET"){

xmlHttpRequest.open(method,url+ '?' + params,**true**);

xmlHttpRequest.send(**null**);

}

**if**(method == "POST"){

xmlHttpRequest.open(method,url,**true**);

xmlHttpRequest.setRequestHeader("Content-type","application/x-www-form-urlencoded");

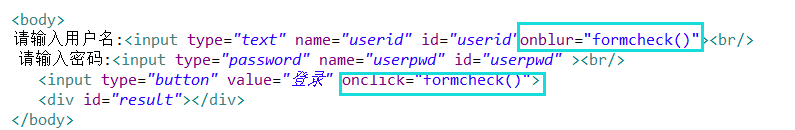
xmlHttpRequest.send(params);

}

}

**在页面中如何使用Ajax。**

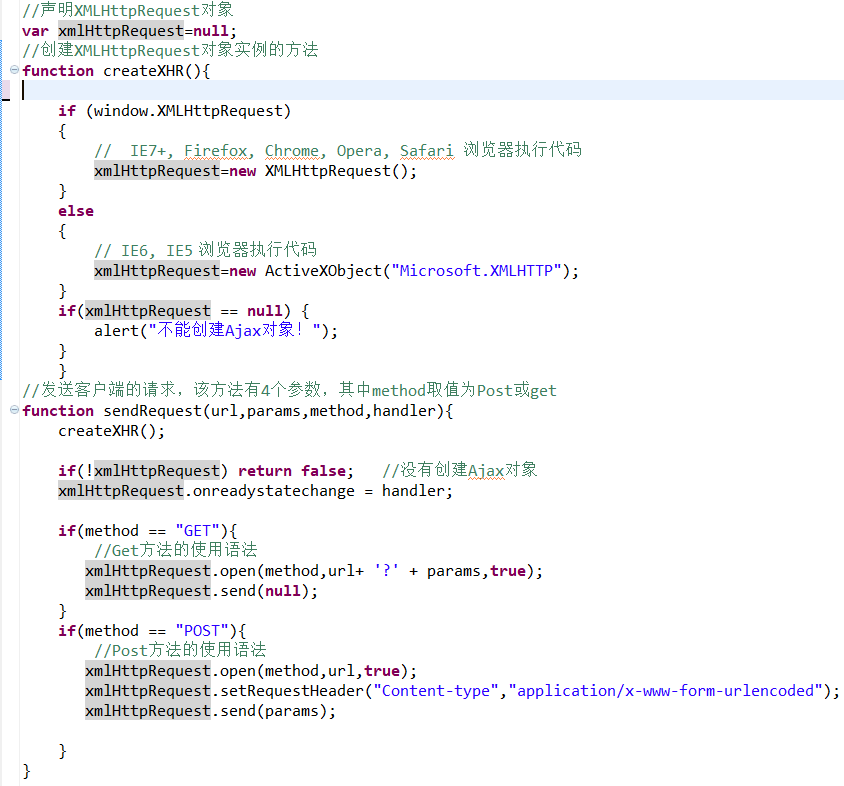
1. **页面内容**



1. **使用Ajax**



**对比HTML页面与封装的ajax间是如何传递参数与调用的。**

**MySQL基本语句的使用**

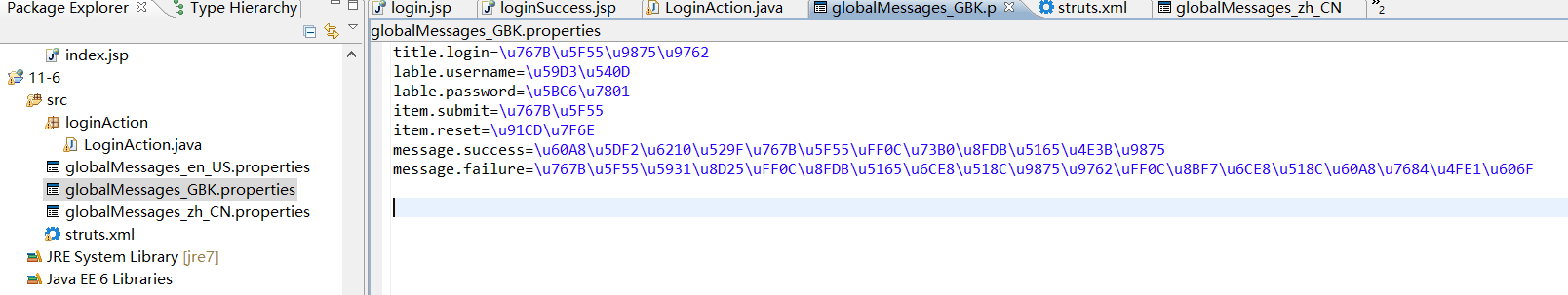
**插入：insert into classtable (TeacherID,StuID,SourceID) values(1514001,151002,151400101);**

**查询多表数据：select s.StuID,s.StuClass,s.StuName ,ct.Score from student s,classtable ct where s.StuID=ct.StuID and ct.TeacherID='1514001' and ct.SourceID='151400103' order by s.StuID asc ;**

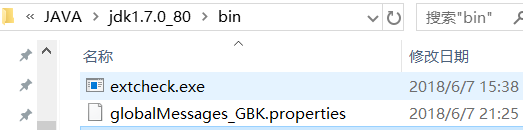
**更新数据：update classtable set Score= 88 where TeacherID=1514001 and StuID=151001 and SourceID=151400101;**

# 问题：将中文资源文件中的中文字符转换为Unicode编码

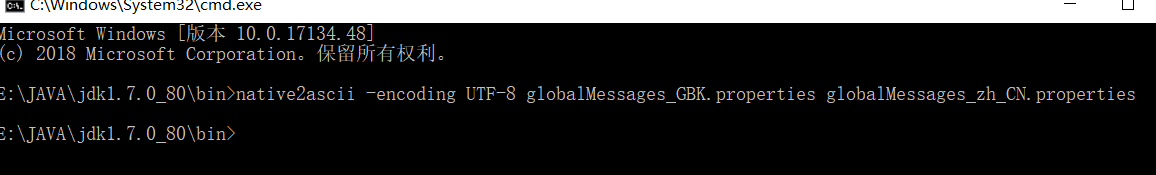
1. 编写源文件

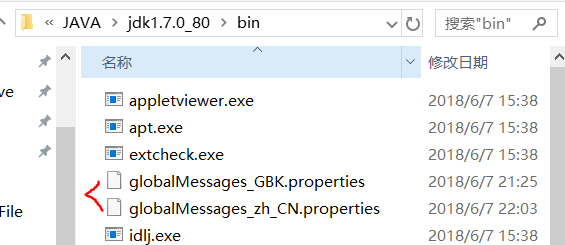


1. 将源文件复制到项目环境所以jdk的bin目录下



1. 在bin目录下进入cmd
2. 编译

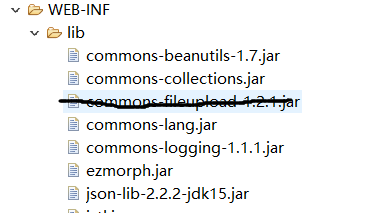




# 问题：异步加载Ajax与json数据实现登录

1. 在工程中通过build path加载使用json—>

所需要的.jar文件，并且将这些.jar文件添加到工程的lib目录下。所需文件（6个）如下：



登录表单：

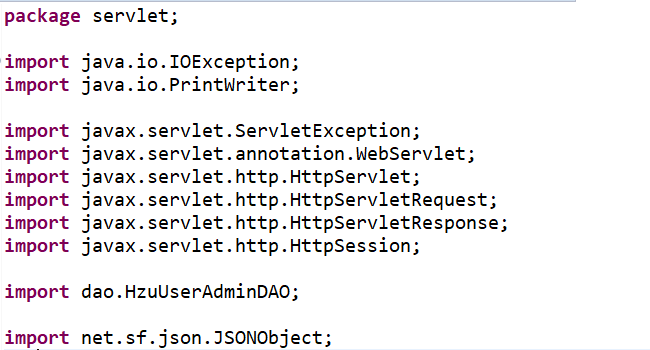


jQuery：



登录检查Servlet：

1. 引入包



1. 正文



# 问题：href="#"与href="javascript:void(0)"的区别

**#** 包含了一个位置信息，默认的锚是**#top** 也就是网页的上端。

而javascript:void(0), 仅仅表示一个死链接。

在页面很长的时候会使用 **#** 来定位页面的具体位置，格式为：**# + id**。

如果你要定义一个死链接请使用 javascript:void(0) 。

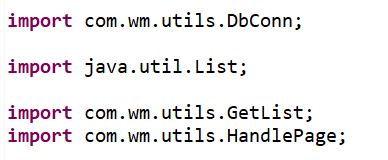
使用方法：

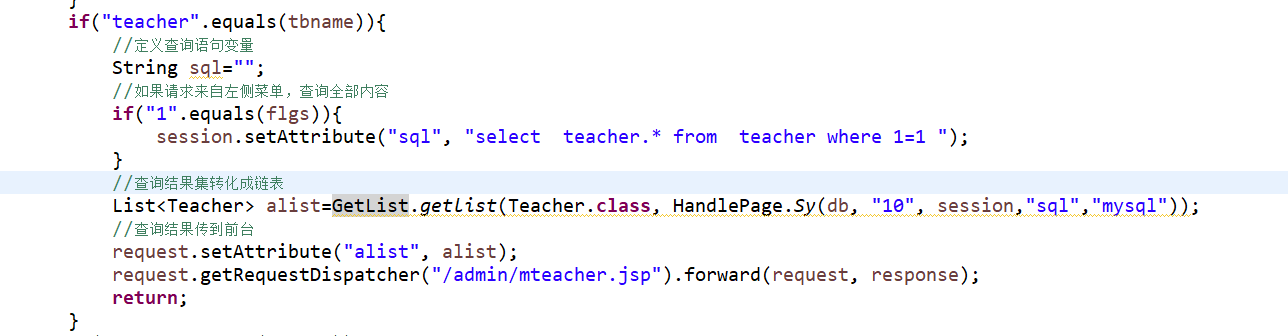
# javascript:void(0) 含义

我们经常会使用到 javascript:void(0) 这样的代码，那么在 JavaScript 中 javascript:void(0) 代表的是什么意思呢？

javascript:void(0) 中最关键的是 void 关键字， void 是 JavaScript 中非常重要的关键字，该操作符指定要计算一个表达式但是不返回值。

# 问题：import com.wm.utils.GetList;与 import com.wm.utils.HandlePage;的使用方法。





**List<Teacher> alist=GetList.*getlist*(Teacher.class, HandlePage.*Sy*(db, "10", session,"sql","mysql"));**

**分析？？Teacher.class为类；HandlePage.*Sy（参数1，参数2，参数3，参数4，参数5）***

***参数1,db 连接数据库***

***参数2，10 执行数据库语句为10类型？？***

***参数3，session 参数4为session类型***

***参数4，sql 要执行的具体SQL语句内容***

***参数5，mysql 数据库类型？？***

# 问题：报表的简单使用Chart.js

**下载js**

[**https://cdnjs.com/libraries/Chart.js**](https://cdnjs.com/libraries/Chart.js)

**Chart.js教程**

[**http://www.chartjs.org/docs/latest/getting-started/**](http://www.chartjs.org/docs/latest/getting-started/)

**相关连接**

[**https://www.cnblogs.com/xiyangbaixue/p/4034437.html**](https://www.cnblogs.com/xiyangbaixue/p/4034437.html)

**参考教程**

[**https://blog.csdn.net/kiss\_mark/article/details/47005651**](https://blog.csdn.net/kiss_mark/article/details/47005651)

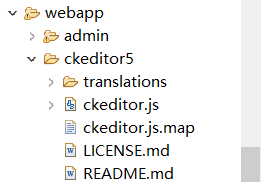
# 问题： ichart教程

[**http://www.bubuko.com/infodetail-759634.html**](http://www.bubuko.com/infodetail-759634.html)

# 问题： CKeditor5的使用

1. **在eclipse中使用**

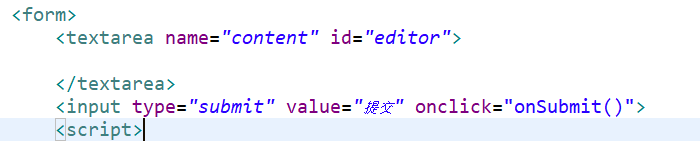
将压缩包解压后复制到根目录



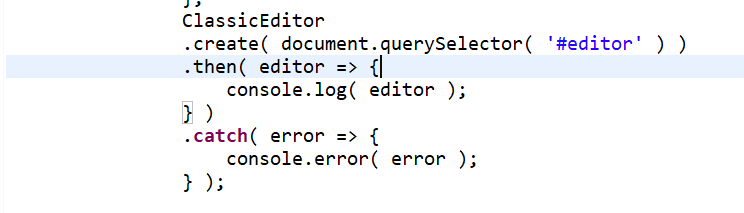
在页面中引用



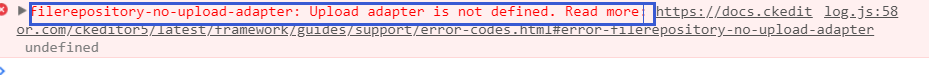
在页面中创建表单和textarea



用编辑器替换掉textarea



此时可以在页面显示编辑器，但上传图片及预览功能未实现，浏览器报错

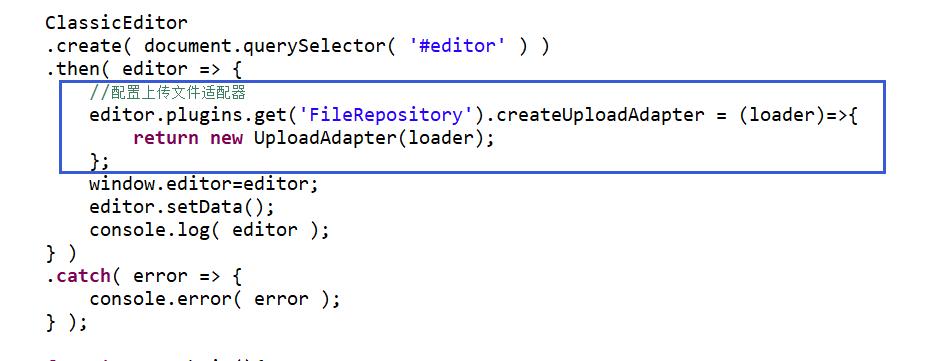


点击链接<https://docs.ckedit>可查看文档，寻找解决方法

1. **上传图片**
2. **加载适配器**



1. **为页面中的编辑器配置适配器**



1. **获取编辑器中的值**

参考资料<http://www.leftso.com/blog/457.html>



**获取上传图片的路径并将图片存入硬盘指定目录（参考Servlet中的文件上传，原理一样）**

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

// 上传文件存储目录

**private** **static** **final** String ***UPLOAD\_DIRECTORY*** = "uploadCommunityImage";

// 上传配置

**private** **static** **final** **int** ***MEMORY\_THRESHOLD*** = 1024 \* 1024 \* 3; // 3MB

**private** **static** **final** **int** ***MAX\_FILE\_SIZE*** = 1024 \* 1024 \* 40; // 40MB

**private** **static** **final** **int** ***MAX\_REQUEST\_SIZE*** = 1024 \* 1024 \* 50; // 50MB

/\*\*

\* 上传数据及保存文件

\*/

**protected** **void** doPost(HttpServletRequest request,

HttpServletResponse response) **throws** ServletException, IOException {

response.setCharacterEncoding("utf-8");

response.setContentType("text/html");

//创建json对象

JSONObject json=**new** JSONObject();

HttpSession session=request.getSession();

// 检测是否为多媒体上传

**if** (!ServletFileUpload.*isMultipartContent*(request)) {

// 如果不是则停止

PrintWriter writer = response.getWriter();

writer.println("Error: 表单必须包含 enctype=multipart/form-data");

writer.flush();

**return**;

}

// 配置上传参数

DiskFileItemFactory factory = **new** DiskFileItemFactory();

// 设置内存临界值 - 超过后将产生临时文件并存储于临时目录中

factory.setSizeThreshold(***MEMORY\_THRESHOLD***);

// 设置临时存储目录

factory.setRepository(**new** File(System.*getProperty*("java.io.tmpdir")));

ServletFileUpload upload = **new** ServletFileUpload(factory);

// 设置最大文件上传值

upload.setFileSizeMax(***MAX\_FILE\_SIZE***);

// 设置最大请求值 (包含文件和表单数据)

upload.setSizeMax(***MAX\_REQUEST\_SIZE***);

// 中文处理

upload.setHeaderEncoding("UTF-8");

// 构造临时路径来存储上传的文件

// 这个路径相对当前应用的目录

String uploadPath = request.getServletContext().getRealPath("./") + File.***separator*** + ***UPLOAD\_DIRECTORY***;

System.***out***.print("uploadPath="+uploadPath);

// 如果目录不存在则创建

File uploadDir = **new** File(uploadPath);

**if** (!uploadDir.exists()) {

uploadDir.mkdir();

}

**try** {

// 解析请求的内容提取文件数据

//@SuppressWarnings("unchecked")

List<FileItem> formItems = upload.parseRequest(request);

**if** (formItems != **null** && formItems.size() > 0) {

// 迭代表单数据

**for** (FileItem item : formItems) {

// 处理不在表单中的字段

**if** (!item.isFormField()) {

String fileName = **new** File(item.getName()).getName();

String filePath = uploadPath + File.***separator*** + fileName;

File storeFile = **new** File(filePath);

// 在控制台输出文件的上传路径

System.***out***.println("这是CKeditor的图片上传，filename="+fileName);

System.***out***.println(filePath);

String showpath="/images/"+fileName;

// 保存文件到硬盘

item.write(storeFile);

request.setAttribute("message",

"文件上传成功!");

session.setAttribute("filepath", fileName);

json.put("res", "1");

json.put("url", showpath);

}

}

}

} **catch** (Exception ex) {

request.setAttribute("message",

"错误信息: " + ex.getMessage());

}

String text=request.getParameter("htmlStr");

System.***out***.print("text="+text);

PrintWriter out=response.getWriter();

out.print(json.toString());

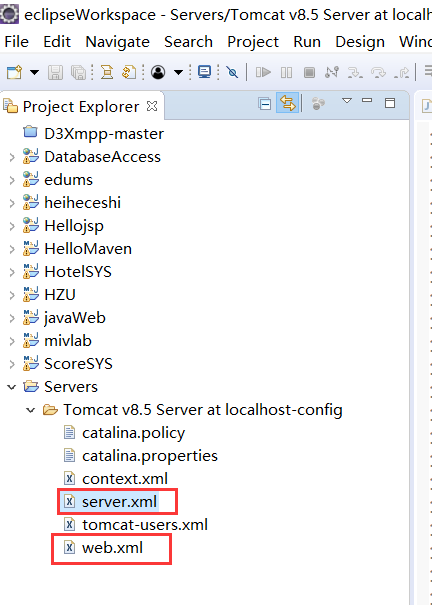
out.flush();

out.close();

# 问题： 如何配置Tomcat虚拟路径（浏览器无法直接访问盘符上的目录及文件）

1. **修改eclipse中的Tomcat中的配置文件web.xml与server.xml**

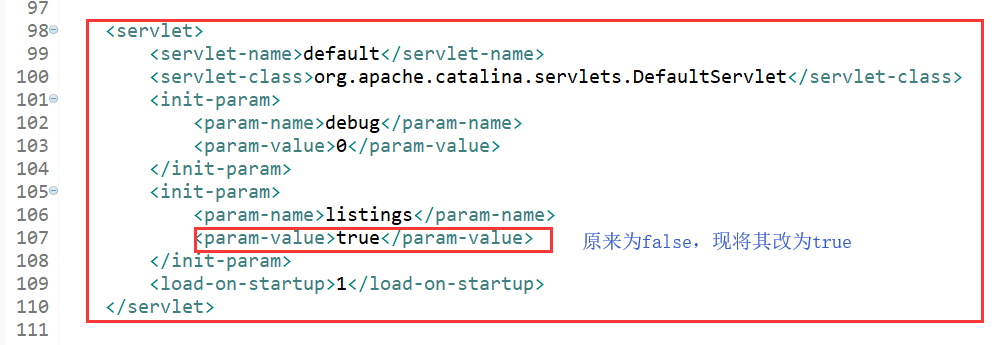
**结构目录**



**（1）在server.xml文件中找到<Host></Host>在此标签内添加如下语句，其中path为虚拟路径的名称，docBase为盘符目录的路径**



**（2）在web.xml中修改如下**



**（3）在HTML文件中，可以这样使用虚拟目录中的文件（注释部分）**

