## 开发第一个servlet

**Servlet 本身不能独立运行，需要在一个web应用中运行的**

**而一个web应用是部署在tomcat中的**

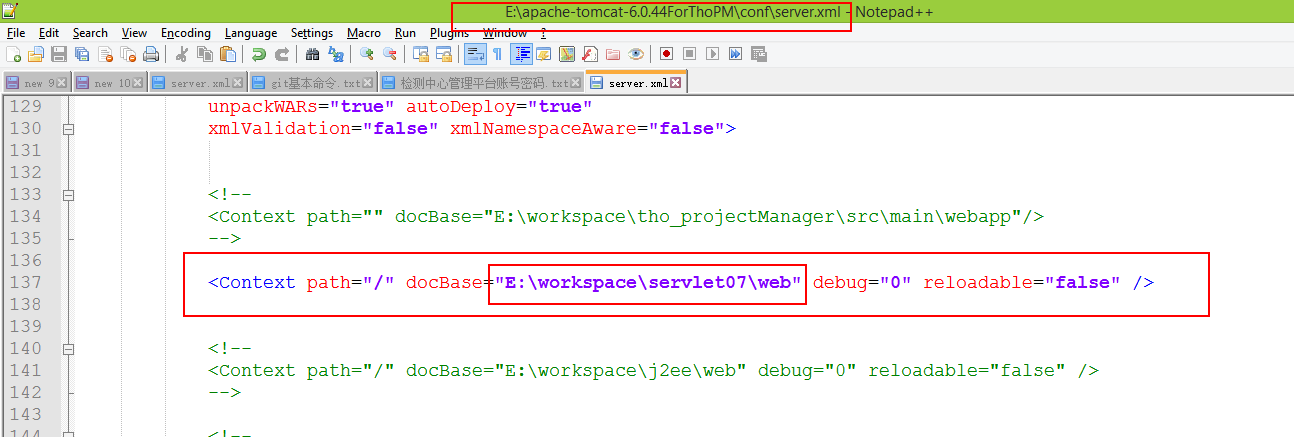
**所以开发一个servlet需要如下几个步骤**

**创建web应用项目**

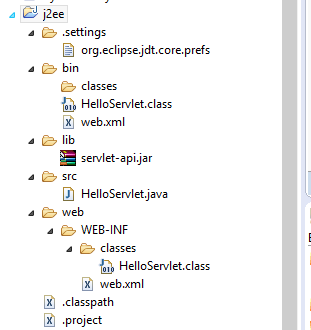
**编写servlet代码**

**部署到tomcat中**

**每次部署新项目，记得在tomcat的server.xml中修改要运行的项目名称**

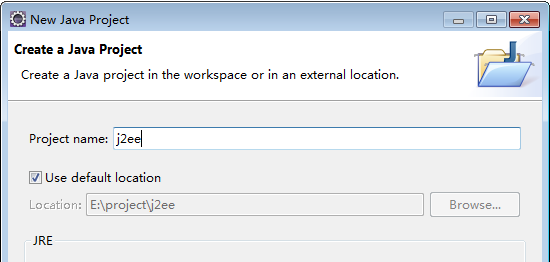


### 0、整体项目目录结构



### 创建一个java项目j2ee

File->New->Other->Java->Java Project  
输入名称j2ee  
这样就创建一个单纯的基于java的项目  
注意： 必须 放在****e:\project\j2ee****目录下，后续配置是基于这个目录进行  
注意：****不要使用如下方式****  
File->New->Other-Web->Dynamic Web Project  
这是eclipse ee创建项目的方式，它会把各种tomcat,相关配置文件的生成都自动做掉了。 而这些细节，恰恰就是你需要学习和掌握的。 所以对于初学者不建议使用这种方式。



### 导入必须的servlet-api.jar包

因为是开发Servlet所以需要用到 servlet-api.jar 这个包里的相关类。  
  
servlet-api.jar 包位于 D:\tomcat\lib\servlet-api.jar  
  
或者从本网页的右侧下载   
  
导入办法:(复制到在lib下，再add to build path)  
  
右键点击项目 -> properties -> Java Build Path ->Libraries -> Add External JARs

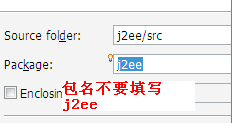
### 编写HelloServlet

创建 HelloServlet  
****注意：****eclipse创建的时候会自带一个****包名j2ee****，****不要使用这个包名，让包名置空****  
HelloServlet继承了 HttpServlet  
并且提供了一个doGet方法  
  
后续测试该类的时候，会直接在浏览器输入地址

<http://127.0.01/hello> 或

http://127.0.01:8080/hello

在浏览器中输入地址提交数据的方式是GET ([参考GET和POST的区别](http://how2j.cn/k/html/html-form/190.html" \l "step1235))  
所以该Servlet需要提供一个对应的doGet方法  
  
在doGet中，输出html  
分别是 标题元素h1 和 当前时间的字符串



import java.io.IOException;

import java.util.Date;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet{

public void doGet(HttpServletRequest request, HttpServletResponse response){

try {

response.getWriter().println("<h1>Hello Servlet!</h1>");

response.getWriter().println(new Date().toLocaleString());

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

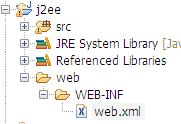
}

}

}

### 4、配置web.xml

首先在j2ee下创建目录web  
接着再创建目录web/WEB-INF  
然后在WEB-INF目录中创建 ****web.xml****  
  
web.xml提供路径与servlet的映射关系  
把/hello这个路径，映射到 HelloServlet这个类上  
  
<servlet> 标签下的 <servlet-name>  
与 <servlet-mapping> 标签下的 <servlet-name> 必须一样  
  
<servlet-name>与<servlet-class>可以不一样，但是为了便于理解与维护，一般都会写的一样。 一目了然



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

<web-app>

<servlet>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

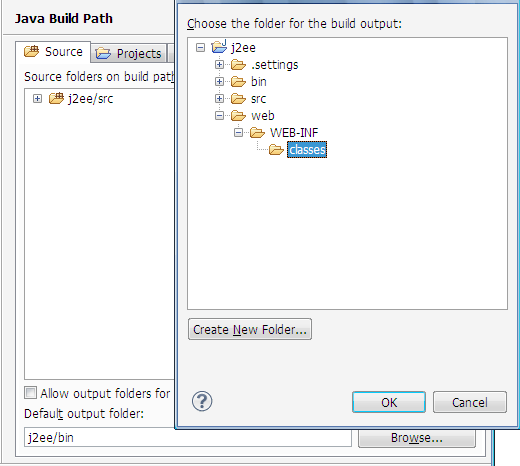
<url-pattern>/hello</url-pattern>

</servlet-mapping>

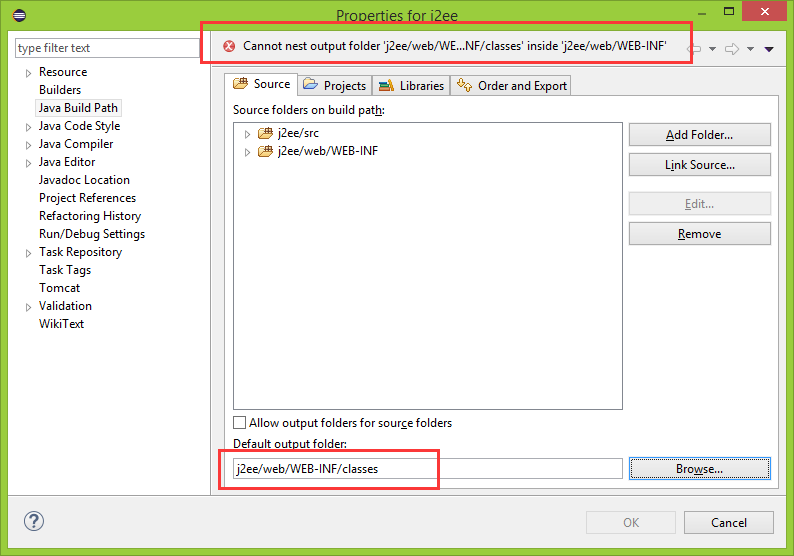
</web-app>

### 5、指定项目输出到classes目录

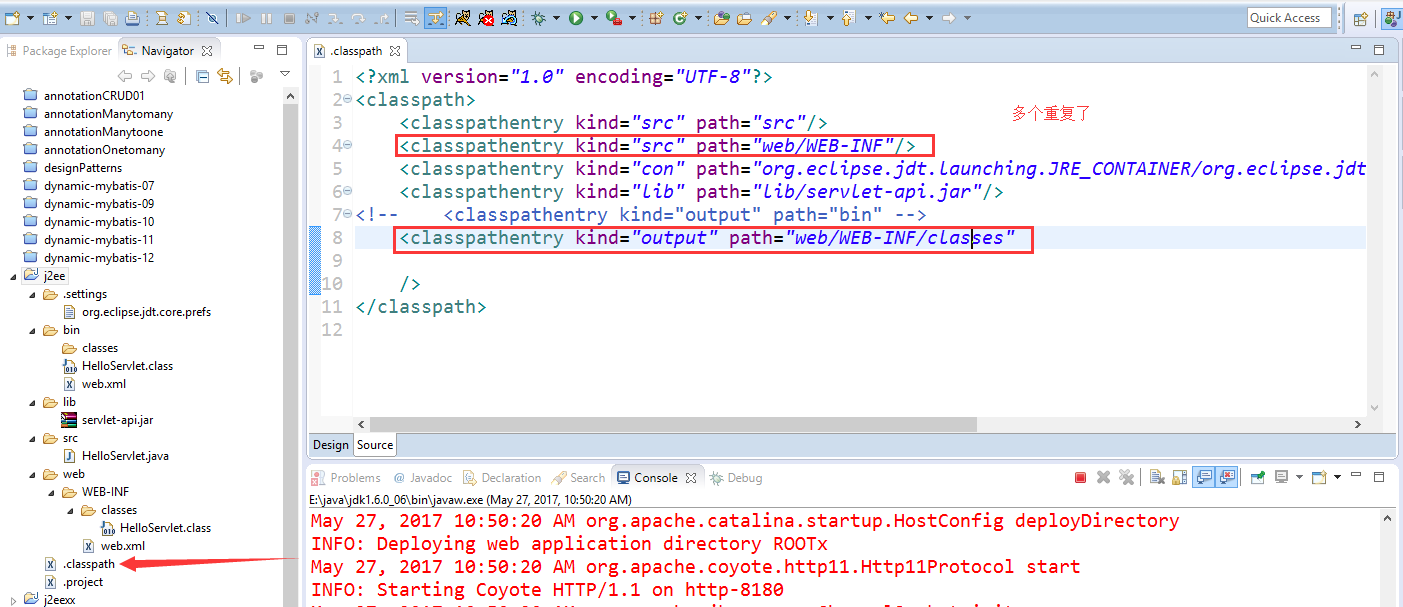
首先在WEB-INF下创建classes目录  
  
把项目的class文件输出由原来的 j2ee/bin 设置到 j2ee/web/WEB-INF/classes下  
  
步骤:   
项目右键->properties->Java Build Path->Source->右下角的 Brower-> 指定位置是 j2ee/web/WEB-INF/classes。  
  
****为什么要有这一步？**** 在ecilpse中默认输出的class是在bin目录下，但是tomcat启动之后，在默认情况下，不会去bin目录找这些class文件，而是到WEB-INF/classes这个目录下去寻找。 所以通过这一步的配置，使得eclipse的class文件输出到WEB-INF/classes目录下，那么这样就和tomcat兼容了。



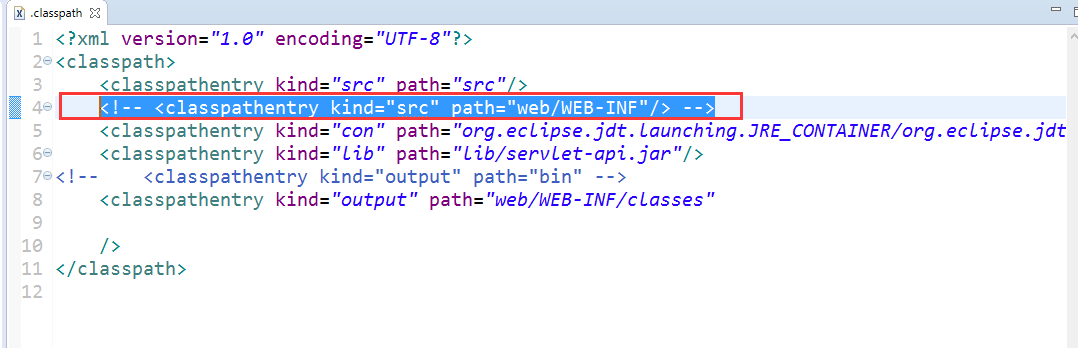
如果：报错，不能指定



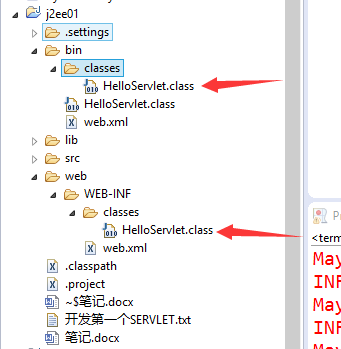
则：是因为：



注释掉一个就OK了：



或把 bin->classes下的xxx.class文件剪切到web->WEB-INF->classes下：



### 6、配置tomcat的server.xml 中的<context

Tomcat 请使用 [80端口的，纯净版tomcat](http://how2j.cn/k/tomcat/tomcat-download/1130.html" \l "step4455)

不是80端口的，默认是8080，此时访问时不能省略8080端口号  
  
为了把 j2ee/web 这个目录部署到tomcat中，进行如下操作：  
  
打开tomcat/conf/sever.xml  
  
在 <Host name="localhost" 。。。 下面加一句

<Context path="/" docBase="e:\\project\\j2ee\\web" debug="0" reloadable="false" />

path="/" 就表示直接通过 http://127.0.0.1/hello 就可以访问网页了。  
  
如果设置为

<Context path="/j2ee" docBase="e:\\project\\j2ee\\web" debug="0" reloadable="false" />

那么表示需要通过 http://127.0.0.1/****j2ee****/hello才能访问呢  
  
  
如果这一步老是失败，请直接在右侧下载修改好的server.xml

<Context path="/" docBase="e:\\project\\j2ee\\web" debug="0" reloadable="false" />

### 7、删除tomcat webapps下的ROOT目录

在上一步部署web的时候server.xml中的path 配置为 "/" 与 webapps下的ROOT目录冲突了  
  
所以要****删除ROOT目录****

### 8、重启tomcat，访问http://127.0.0.1/hello

关闭tomcat后，启动startup.bat 访问下面地址就可以看到servlet的输出

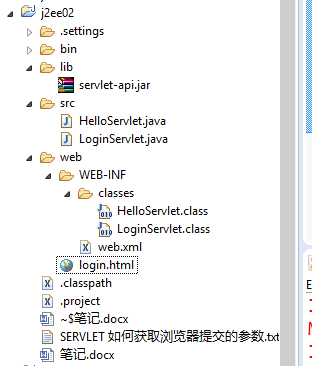
http://127.0.0.1/hello

并且每一次刷新，都能看到输出新的时间

## 获取参数

通过登录行为，演示servlet如何获取从浏览器提交的账号密码

### 0、整体项目目录结构



### 1、创建 login.html

在web上右键 -> New ->File  
创建一个 login.html文件  
然后添加一个form元素  
action="login" 标题会提交到login路径，login路径在后续步骤会映射到LoginServlet  
method="post" post方式表示提交的密码信息在浏览器地址栏看不到  
  
接着准备账号和密码的input元素  
因为要提交两个数据，在servlet端为了区分哪个是账号，哪个是密码，要给这两个input元素的name属性分别叫做name和password。

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>登录页面</title>

</head>

<body>

<form action="login" method="post">

账号: <input type="text" name="name"> <br>

密码: <input type="password" name="password"> <br>

<input type="submit" value="登录">

</form>

</body>

</html>

### 创建 LoginServlet

创建一个LoginServlet  
  
因为浏览器中的****form的method是post****,所以LoginServlet需要提供一个****doPost****方法  
  
在doPost方法中，通过****request.getParamter****根据name取出对应的账号和密码  
  
然后用System.out.println() 打印在控制台  
  
****注**** 这里是打印在控制台，并没有在网页上输出

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

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

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

}

}

### 映射LoginServlet到路径login

在web.xml中新增映射

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

<web-app>

<servlet>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

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

</servlet-mapping>

<servlet>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

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

</servlet-mapping>

</web-app>

### 在页面提交数据

****首先重启tomcat****  
然后访问页面

http://127.0.0.1/login.html

输入账号密码，提交  
然后****在tomcat的窗口****，就可以看到提交的账号和密码了。

### 出现的问题：HTTP Status 404 (The requested resource is not available)

原来字母前面有一个空格符。试着去掉**空格**，然后运行，正常了！

一个英文页面里面提到的解决方法就是反复强调检查拼写。

注意检查拼写，还有文件名。切记！切记！

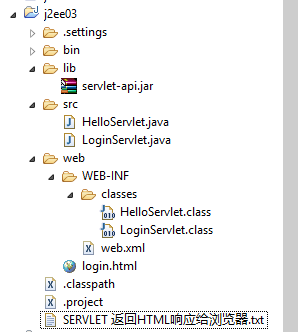
## 三.返回响应

根据浏览器提交的账号密码返回登录成功或者失败

这一步本来应该通过访问数据库来实现，这里简化一下，直接在内存中进行校验

如果账号是 admin,密码是123, 就返回登录成功，否则返回登录失败

### 整体项目目录结构



### 1、LoginServlet

判断账号密码是否为 admin 123，如果是就打印  
success 否则就打印 fail  
  
根据账号密码，创建对应的html字符串。  
  
然后通过response.getWriter().println() 发送到浏览器。  
  
  
****注:**** 比较的时候把常量字符串"admin" "123"放前面，因为用户可能没有输入账号密码就提交，servlet会获取到null。 这样就规避了空指针异常的问题。  
  
****注:**** 这里输出的响应是success和fail，不是中文。 如果使用中文浏览器有可能看到乱码。  
  
关于中文乱码问题，统一在[中文问题](http://how2j.cn/k/servlet/servlet-gbk/554.html)章节解决，咱们一个知识点一个知识点的掌握

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

String html = null;

if ("admin".equals(name) && "123".equals(password))

html = "<div style='color:green'>success</div>";

else

html = "<div style='color:red'>fail</div>";

PrintWriter pw = response.getWriter();

pw.println(html);

}

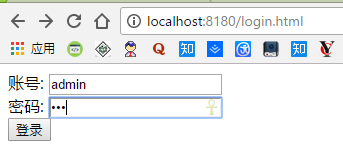
}

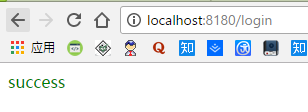
### 重启tomcat，提交数据

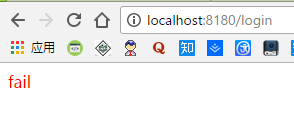
访问页面

http://127.0.0.1/login.html

http://localhost:8180/login.html





  
  
账号输入admin  
密码输入123  
提交可以看到success

## 四.调用流程

### 0、整体项目目录结构

### 1、login.html

首先访问

http://127.0.0.1/login.html

打开一个静态的html页面，在这个页面中可以通过form，以post的形式提交数据

### /login路径

在上一步的login.html中，用form，把账号和密码，提交到/login这个路径，并且附带method="post"

### 找到对应的Servlet

tomcat接受到一个新的请求：

http://127.0.0.1/login

其路径是/login，接着就到[配置文件web.xml](http://how2j.cn/k/servlet/servlet-paramter/547.html" \l "step1588)进行匹配，发现/login，对应的Servlet类是 LoginServlet。  
接下来的工作，就会基于这个LoginServlet进行。

<servlet>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

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

</servlet-mapping>

### 实例化Servlet对象

Tomcat 定位到了LoginServlet后，发现并没有LoginServlet的实例存在，于是就调用LoginServlet的public无参的构造方法LoginServlet()实例化一个LoginServlet对象以备后续使用

### 5、调用doGet或者doPost     Tomcat从上一步拿到了LoginServlet的实例之后，根据页面login.html提交信息的时候带的method="post"，去调用对应的doPost方法。

### 6、request获取参数

接着流程进入了doPost方法中，

protected void doPost(HttpServletRequest request, HttpServletResponse response){

...

}

在这个方法中，通过参数request，把页面上传递来的账号和密码信息取出来

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

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

### response设置响应

接着，根据账号和密码是否正确(判断是否是admin和123)， 创建不同的html字符串。  
  
然后把html字符串通过如下方式，设置在了response对象上。

PrintWriter pw = response.getWriter();

pw.println(html);

到这里，Servlet的工作就做完了。

### 8、tomcat把html传递给浏览器    在Servlet完成工作之后，tomcat拿到被Servlet修改过的response，根据这个response生成html 字符串，然后再通过[HTTP协议](http://how2j.cn/k/http/http-tutorials/568.html)，这个html字符串，回发给浏览器，浏览器再根据HTTP协议获取这个html字符串，并****渲染****在界面上。 这样在效果上，浏览器就可以看到Servlet中生成的字符串了。

## 五.service() 方法

Servlet 需要提供对应的doGet() 与 doPost()方法

### 1、doGet()

当浏览器使用get方式提交数据的时候，servlet需要提供doGet()方法  
  
哪些是get方式呢？  
  
form默认的提交方式  
如果通过一个超链访问某个地址  
如果在地址栏直接输入某个地址  
ajax指定使用get方式的时候

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

}

}

### 2、doPost()

当浏览器使用post方式提交数据的时候，servlet需要提供doPost()方法  
  
哪些是post方式呢？  
在form上显示设置 method="post"的时候  
ajax指定post方式的时候

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

}

}

### 3、service()

LoginServlet继承了HttpServlet,同时也继承了一个方法

service(HttpServletRequest , HttpServletResponse )  
  
**实际上，在执行doGet()或者doPost()之前，都会先执行service()**  
由service()方法进行判断，到底该调用doGet()还是doPost()  
  
可以发现，service(), doGet(), doPost() 三种方式的参数列表都是一样的。  
  
所以，有时候也会****直接重写service()****方法，在其中提供相应的服务，就不用区分到底是get还是post乐。  
  
比如把前面的登录的LoginServlet，改为提供service方法，也可以达到相同的效果

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

String html = null;

if ("admin".equals(name) && "123".equals(password))

html = "<div style='color:green'>success</div>";

else

html = "<div style='color:red'>fail</div>";

PrintWriter pw = response.getWriter();

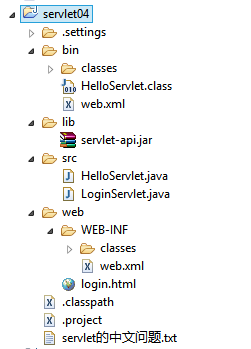
pw.println(html);

}

}

## 六.Servlet的中文问题

### 0、整体项目目录结构



### 获取中文的参数

为了成功获取中文参数，需要做如下操作  
1. login.html中加上

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

这句话的目的是告诉浏览器，等下发消息给服务器的时候，使用UTF-8编码  
  
2. login.html  
form的method修改为post  
  
3. 在servlet进行解码和编码

byte[] bytes= name.getBytes("ISO-8859-1");

name = new String(bytes,"UTF-8");

先根据ISO-8859-1解码，然后用UTF-8编码  
这样就可以得到正确的中文参数了  
  
这样需要对每一个提交的数据都进行编码和解码处理，如果觉得麻烦，也可以使用一句话代替：

request.setCharacterEncoding("UTF-8");

并且把这句话放在request.getParameter()之前  
  
以上是使用UTF-8的方式获取中文呢。 也可以使用GBK。把所有的UTF-8替换为GBK即可。 GB2312同理。

**Login.html**

<!DOCTYPE html>

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

<form action="login" method="post">

账号 ： <input type="text" name="name"> <br> 密码: <input

type="password" name="password"> <br> <input

type="submit" value="登录">

</form>

**LoginServlet.java**

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

request.setCharacterEncoding("UTF-8");

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

// byte[] bytes = name.getBytes("ISO-8859-1");

// name = new String(bytes, "UTF-8");

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

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

}

}

### 2、返回中文的响应

在Servlet中，加上

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

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

String html = null;

if ("admin".equals(name) && "123".equals(password))

html = "<div style='color:green'>登录成功</div>";

else

html = "<div style='color:red'>登录失败</div>";

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

PrintWriter pw = response.getWriter();

pw.println(html);

}

}

## 七.Servlet的生命周期

### 1、生命周期

一个Servlet的生命周期由 [实例化](http://how2j.cn/k/servlet/servlet-lifecycle/550.html" \l "step1594)，[初始化](http://how2j.cn/k/servlet/servlet-lifecycle/550.html" \l "step1595)，[提供服务](http://how2j.cn/k/servlet/servlet-lifecycle/550.html" \l "step1596)，[销毁](http://how2j.cn/k/servlet/servlet-lifecycle/550.html" \l "step1597)，[被回收](http://how2j.cn/k/servlet/servlet-lifecycle/550.html" \l "step1597) 几个步骤组成



### 2、实例化

当用户通过浏览器输入一个路径，这个路径对应的servlet被调用的时候，该Servlet就会被实例化  
  
为LoginServlet显示提供一个构造方法 LoginServlet()  
  
然后通过浏览器访问，就可以观察到  
"LoginServlet 构造方法 被调用"  
  
无论访问了多少次LoginServlet  
LoginSerlvet构造方法 ****只会执行一次****，所以Serlvet是****单实例的****

****import java.io.IOException;****

****import javax.servlet.ServletException;****

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

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

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

****public class LoginServlet extends HttpServlet {****

****public LoginServlet(){****

****System.out.println("LoginServlet 构造方法 被调用");****

****}****

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

****throws ServletException, IOException {****

****//略****

****}****

****}****

### 3、初始化

LoginServlet 继承了HttpServlet，同时也继承了init(ServletConfig) 方法  
  
init方式是一个实例方法，所以会在构造方法执行后执行。  
  
无论访问了多少次LoginSerlvet  
init初始化 ****只会执行一次****

import java.io.IOException;

import javax.servlet.ServletConfig;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

public LoginServlet() {

System.out.println("LoginServlet 构造方法 被调用");

}

public void init(ServletConfig config) {

System.out.println("init(ServletConfig)");

}

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// 略

}

}

### 4、提供服务

接下来就是执行service()方法，然后通过浏览器传递过来的信息进行判断，是调用doGet()还是doPost()方法  
  
在service()中就会编写我们的****业务代码****，在本例中就是判断用户输入的账号和密码是否正确

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletConfig;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

public LoginServlet() {

System.out.println("LoginServlet 构造方法 被调用");

}

public void init(ServletConfig config) {

System.out.println("init(ServletConfig)");

}

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

String html = null;

if ("admin".equals(name) && "123".equals(password))

html = "<div style='color:green'>success</div>";

else

html = "<div style='color:red'>fail</div>";

PrintWriter pw = response.getWriter();

pw.println(html);

}

}

### 5、销毁

接着是销毁destroy()  
在如下几种情况下，会调用destroy()  
1. 该Servlet所在的web应用重新启动  
在serverl.xml中配置该web应用的时候用到了

<Context path="/" docBase="e:\\project\\j2ee\\web" debug="0" reloadable="false" />

如果把 reloadable="****false****" 改为reloadable="****true****" 就表示有任何类发生的更新，web应用会****自动重启****  
当web应用自动重启的时候，destroy()方法就会被调用  
  
2. 关闭tomcat的时候 destroy()方法会被调用，但是这个一般都发生的很快，不易被发现。

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

public void destroy() {

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

}

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// 略

}

}

### 6、被回收

当该Servlet被销毁后，就满足垃圾回收的条件了。 当下一次垃圾回收GC来临的时候，就有可能被回收。  
  
这个。。。也是不容易被观察到的现象。

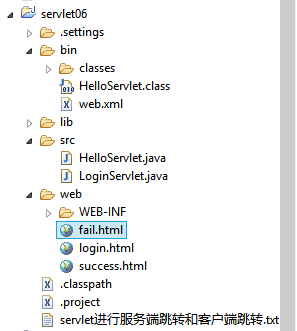
## 八.servlet进行服务端跳转和客户端跳转

页面跳转是开发一个web应用经常会发生的事情。

比如登录成功或是失败后，分别会跳转到不同的页面。

跳转的方式有两种，服务端跳转和客户端跳转

### 0、整体项目目录结构



### 1、准备两个页面 success.html fail.html

首先在web目录下准备两个页面 success.html,fail.html  
分别用于显示登录成功 或者登录失败  
  
如果登录成功了，就****服务端跳转****到success.html  
  
如果登录失败了，就****客户端跳转****到fail.html

**success.html**

<div style="color:green">login success</div>

**fail.html**

<div style="color:red">login fail</div>

### 2、服务端跳转

在Servlet中进行服务端跳转的方式：

request.getRequestDispatcher("success.html").forward(request, response);

服务端跳转可以看到浏览器的地址依然是/login 路径，并不会变成success.html

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

if ("admin".equals(name) && "123".equals(password)) {

request.getRequestDispatcher("success.html").forward(request, response);

}

}

}

### 3、客户端跳转

在Servlet中进行客户端跳转的方式：

response.sendRedirect("fail.html");

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

if ("admin".equals(name) && "123".equals(password)) {

request.getRequestDispatcher("success.html").forward(request, response);

}

else{

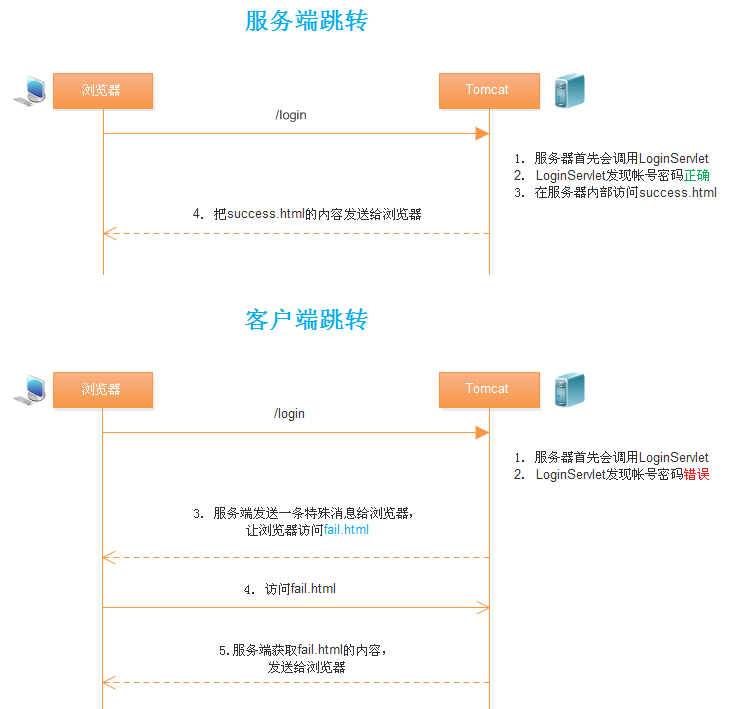
response.sendRedirect("fail.html");

}

}

}

### 4、服务端跳转与客户端跳转图示



## 九.servlet自启动配制

有的时候会有这样的业务需求：

tomcat一启动，就需要执行一些初始化的代码，比如校验数据库的完整性等。

但是Servlet的生命周期是在用户访问浏览器对应的路径开始的。如果没有用户的第一次访问，就无法执行相关代码。

这个时候，就需要Servlet实现自启动 即，伴随着tomcat的启动，自动启动初始化，在初始化方法init()中，就可以进行一些业务代码的工作了。

### 1、load-on-startup

在web.xml中，配置Hello Servlet的地方，增加一句

<load-on-startup>10</load-on-startup>

取值范围是1-99  
  
即表明该Servlet会随着Tomcat的启动而初始化。  
  
同时，为HelloServlet提供一个init(ServletConfig) 方法，验证自启动  
  
如图所示，在tomcat完全启动之前，就打印了****init of HelloServlet****  
<load-on-startup>****10****</load-on-startup> 中的****10****表示启动顺序  
如果有多个Servlet都配置了自动启动，小的数字先启动

**web.xml**

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

<web-app>

<servlet>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>HelloServlet</servlet-class>

<load-on-startup>10</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

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

</servlet-mapping>

<servlet>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

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

</servlet-mapping>

</web-app>

**HelloServlet.java**

import java.io.IOException;

import java.util.Date;

import javax.servlet.ServletConfig;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet{

public void init(ServletConfig config){

System.out.println("init of Hello Servlet");

}

public void doGet(HttpServletRequest request, HttpServletResponse response){

try {

response.getWriter().println("<h1>Hello Servlet!</h1>");

response.getWriter().println(new Date().toLocaleString());

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

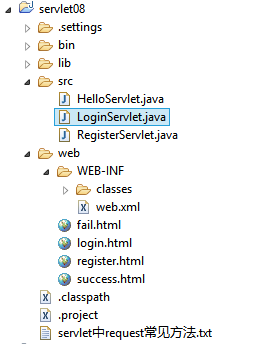
}

}

## 十.servlet中request常见方法

request对象的类是HttpServletRequest，提供了很多有实用价值的方法

### 0、整体项目目录结构



### 1、request的常见方法

request.getRequestURL(): 浏览器发出请求时的完整URL，包括协议 主机名 端口(如果有)" +

request.getRequestURI(): 浏览器发出请求的资源名部分，去掉了协议和主机名" +

request.getQueryString(): 请求行中的参数部分，只能显示以get方式发出的参数，post方式的看不到

request.getRemoteAddr(): 浏览器所处于的客户机的IP地址

request.getRemoteHost(): 浏览器所处于的客户机的主机名

request.getRemotePort(): 浏览器所处于的客户机使用的网络端口

request.getLocalAddr(): 服务器的IP地址

request.getLocalName(): 服务器的主机名

request.getMethod(): 得到客户机请求方式一般是GET或者POST

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

System.out.println("浏览器发出请求时的完整URL，包括协议 主机名 端口(如果有): " + request.getRequestURL());

System.out.println("浏览器发出请求的资源名部分，去掉了协议和主机名: " + request.getRequestURI());

System.out.println("请求行中的参数部分: " + request.getQueryString());

System.out.println("浏览器所处于的客户机的IP地址: " + request.getRemoteAddr());

System.out.println("浏览器所处于的客户机的主机名: " + request.getRemoteHost());

System.out.println("浏览器所处于的客户机使用的网络端口: " + request.getRemotePort());

System.out.println("服务器的IP地址: " + request.getLocalAddr());

System.out.println("服务器的主机名: " + request.getLocalName());

System.out.println("得到客户机请求方式: " + request.getMethod());

String html = null;

if ("admin".equals(name) && "123".equals(password))

html = "<div style='color:green'>登录成功</div>";

else

html = "<div style='color:red'>登录失败</div>";

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

PrintWriter pw = response.getWriter();

pw.println(html);

}

}

### 2、获取参数

request.getParameter(): 是常见的方法，用于获取单值的参数

request.getParameterValues(): 用于获取具有多值得参数，比如注册的时候提交的爱好，可以使多选的。

request.getParameterMap(): 用于遍历所有的参数，并返回Map类型。

本例准备了一个注册的例子

包括一个注册页面 register.html

服务端的 RegisterServlet

分别演示了获取单值参数，多值参数以及遍历所有的参数

**register.html**

<!DOCTYPE html>

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

<form action="register" method="get">

账号 ： <input type="text" name="name"> <br>

爱好 ： LOL<input type="checkbox" name="hobits" value="lol">

DOTA<input type="checkbox" name="hobits" value="dota"> <br>

<input type="submit" value="注册">

</form>

### 3、获取头信息

request.getHeader() 获取浏览器传递过来的头信息。

比如getHeader("user-agent") 可以获取浏览器的基本资料，这样就能判断是firefox、IE、chrome、或者是safari浏览器

request.getHeaderNames() 获取浏览器所有的头信息名称，根据头信息名称就能遍历出所有的头信息

在本例，修改HelloServlet,使其获取头信息

访问HelloServlet获取如下头信息:

host: 主机地址

user-agent: 浏览器基本资料

accept: 表示浏览器接受的数据类型

accept-language: 表示浏览器接受的语言

accept-encoding: 表示浏览器接受的压缩方式，是压缩方式，并非编码

connection: 是否保持连接

cache-control: 缓存时限

import java.io.IOException;

import java.util.Date;

import java.util.Enumeration;

import javax.servlet.ServletConfig;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet{

public void init(ServletConfig config){

System.out.println("init of Hello Servlet");

}

public void doGet(HttpServletRequest request, HttpServletResponse response){

Enumeration<String> headerNames= request.getHeaderNames();

while(headerNames.hasMoreElements()){

String header = headerNames.nextElement();

String value = request.getHeader(header);

System.out.printf("%s\t%s%n",header,value);

}

try {

response.getWriter().println("<h1>Hello Servlet!</h1>");

response.getWriter().println(new Date().toLocaleString());

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

### 4、服务端传参

setAttribute和getAttribute可以用来在进行服务端跳转的时候，在不同的Servlet之间进行数据共享

## 十一.servlet中response常见方法

response是HttpServletResponse的实例，用于提供给浏览器的响应信息

### 1、设置响应内容

通过response设置响应已经用得比较多了，在前面的Servlet学习中都是用到  
PrintWriter pw= ****response.getWriter();****  
通过response.getWriter(); 获取一个PrintWriter 对象  
可以使用println(),append(),write(),format()等等方法设置返回给浏览器的html内容。

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet{

public void doGet(HttpServletRequest request, HttpServletResponse response){

try {

PrintWriter pw= response.getWriter();

pw.println("<h1>Hello Servlet</h1>");

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

### 2、设置响应格式

response.setContentType("text/html");

"text/html" 是即格式 ，在[request获取头信息](http://how2j.cn/k/servlet/servlet-request/555.html" \l "step1609) 中对应的request.getHeader("accept").   
"text/html" 是存在的，表示浏览器可以识别这种格式，如果换一个其他的格式， 比如 "text/lol" ，浏览器不能识别，那么打开此servlet就会弹出一个下载的对话框。  
  
这样的手段也就常常用于实现下载功能

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) {

try {

PrintWriter pw = response.getWriter();

pw.println("<h1>Hello Servlet</h1>");

response.setContentType("text/lol");

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

### 3、设置响应编码

设置响应编码有两种方式

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

2. response.setCharacterEncoding("UTF-8");

这两种方式都需要在response.getWriter调用之前执行才能生效。  
  
他们的区别在于

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

不仅发送到浏览器的内容会使用UTF-8编码，而且还通知浏览器使用UTF-8编码方式进行显示。所以总能正常显示中文

2. response.setCharacterEncoding("UTF-8");

仅仅是发送的浏览器的内容是UTF-8编码的，置于浏览器是用哪种编码方式显示不管。 所以当浏览器的显示编码方式不是UTF-8的时候，就会看到乱码，需要手动再进行一次设置。

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) {

try {

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

// response.setCharacterEncoding("UTF-8");

PrintWriter pw = response.getWriter();

pw.println("<h1>第一次 使用 Servlet</h1>");

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

### 4、301或者302客户端跳转

客户端有两种跳转  
302 表示临时跳转  
301 表示永久性跳转  
  
302就是前面在[客户端跳转](http://how2j.cn/k/servlet/servlet-jump/551.html" \l "step1567)章节用到过的

response.sendRedirect("fail.html");

301要使用另外的手段：

response.setStatus(301);

response.setHeader("Location", "fail.html");

用户感受不出这两种跳转的区别，但是可以借助[火狐的调试工具](http://how2j.cn/k/http/http-debug/569.html)看到响应的头信息是:  
301 Moved Permanently。  
301和302的区别主要在搜索引擎对页面排名的时候有影响，这是属于SEO范畴的概念，在此就不展开了。

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

if ("admin".equals(name) && "123".equals(password)) {

request.getRequestDispatcher("success.html").forward(request, response);

} else {

response.setStatus(301);

response.setHeader("Location", "fail.html");

}

}

}

### 5、设置不使用缓存

使用缓存可以加快页面的加载，降低服务端的负担。但是也可能看到过时的信息，可以通过如下手段通知浏览器不要使用缓存

response.setDateHeader("Expires",0 );

response.setHeader("Cache-Controll","no-cache");

response.setHeader("pragma","no-cache");

import java.io.IOException;

import java.util.Date;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) {

try {

response.setDateHeader("Expires", 0);

response.setHeader("Cache-Controll", "no-cache");

response.setHeader("pragma", "no-cache");

response.getWriter().println("<h1>Hello Servlet!</h1>");

response.getWriter().println(new Date().toLocaleString());

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

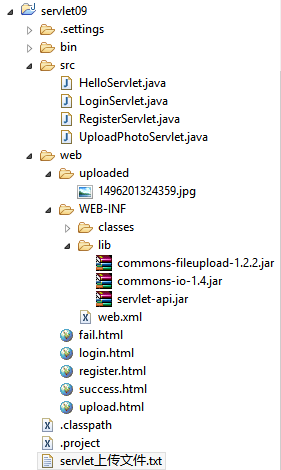
}

}

}

## 十二.servlet上传文件

### 0、整体项目目录结构



### 1、准备上传页面 upload.html

上传页面有两点需要注意

1. form 的method必须是post的，get不能上传文件。 还需要加上enctype="multipart/form-data" 表示提交的数据是二进制文件

**<form action="/uploadPhoto" method="post" enctype="multipart/form-data">**

2. 需要提供****type="file"****的字段进行上传

<!DOCTYPE html>

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

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

英雄名称:<input type="text" name="heroName" /> <br>

上传头像 : <input type="file" name="filepath" /> <br>

<input type="submit" value="上传">

</form>

### 2、准备UploadPhotoServlet

在UploadPhotoServlet进行上传的功能开发。  
1. 需要用到两个第三方的jar包，****commons-io-1.4.jar****和****commons-fileupload-1.2.2.jar****。 在右侧下载，并放在WEB-INF/lib下  
  
2. 前部分代码是固定写法，用来做一些准备工作。 直到遍历出Item,一个Item就是对应一个浏览器提交的数据，通过item.getInputStream可以打开浏览器上传的文件的输入流。  
  
3. 客户提交的文件名有可能是一样的，所以在服务端保存文件的时候，不能使用客户提交的文件名。这里使用的是一种粗糙的解决文件名重复的办法，即使用时间戳。   
  
4. 读取输入流中的数据，保存在服务端的目录下 e:/project/j2ee/web/uploaded  
****注：**** 为什么要放这里？ 因为后续网页上显示的时候是通过http://127.0.0.1/uploaded/xxx.jpg 路径来查找图片  
  
5. 根据临时生成的文件名，创建一个html img元素，然后通过response返回浏览器

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.InputStream;

import java.io.PrintWriter;

import java.util.Iterator;

import java.util.List;

import javax.servlet.ServletException;

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.FileUploadException;

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

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

public class UploadPhotoServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response) {

String filename = null;

try {

DiskFileItemFactory factory = new DiskFileItemFactory();

ServletFileUpload upload = new ServletFileUpload(factory);

// 设置上传文件的大小限制为1M

factory.setSizeThreshold(1024 \* 1024);

List items = null;

try {

items = upload.parseRequest(request);

} catch (FileUploadException e) {

e.printStackTrace();

}

Iterator iter = items.iterator();

while (iter.hasNext()) {

FileItem item = (FileItem) iter.next();

if (!item.isFormField()) {

// 根据时间戳创建头像文件

filename = System.currentTimeMillis() + ".jpg";

String photoFolder = "e:\\project\\j2ee\\web\\uploaded";

File f = new File(photoFolder, filename);

f.getParentFile().mkdirs();

// 通过item.getInputStream()获取浏览器上传的文件的输入流

InputStream is = item.getInputStream();

// 复制文件

FileOutputStream fos = new FileOutputStream(f);

byte b[] = new byte[1024 \* 1024];

int length = 0;

while (-1 != (length = is.read(b))) {

fos.write(b, 0, length);

}

fos.close();

} else {

System.out.println(item.getFieldName());

String value = item.getString();

value = new String(value.getBytes("ISO-8859-1"), "UTF-8");

System.out.println(value);

}

}

String html = "<img width='200' height='150' src='uploaded/%s' />";

response.setContentType("text/html");

PrintWriter pw= response.getWriter();

pw.format(html, filename);

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

### 3、配置web.xml

配置UploadPhotoServlet

<servlet>

<servlet-name>UploadPhotoServlet</servlet-name>

<servlet-class>UploadPhotoServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>UploadPhotoServlet</servlet-name>

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

</servlet-mapping>

### 4、复制jar包

把右侧的两个jar包commons-io-1.4.jar，commons-fileupload-1.2.2.jar 复制到WEB-INF/lib 目录下

### 5、在项目中导入jar包

把jar包导入到项目中，导包办法：右键 project->properties->java build path->libaries->add external jars

### 6、上传测试

选中某个图片点击上传，就可以看到上传的图片了

### 7、如何处理其他非File字段

因为浏览器指定了以二进制的形式提交数据，那么就不能通过常规的手段获取非File字段：

request.getParameter("heroName")

在遍历Item时(Item即对应浏览器提交的字段)，可以通过

item.isFormField

来判断是否是常规字段还是提交的文件。 当item.isFormField返回true的时候，就表示是常规字段。  
  
然后通过item.getFieldName()和item.getString()就知道分别是哪个字段，以及字段的值了。

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.InputStream;

import java.io.PrintWriter;

import java.util.Iterator;

import java.util.List;

import javax.servlet.ServletException;

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.FileUploadException;

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

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

public class UploadPhotoServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response) {

String filename = null;

try {

DiskFileItemFactory factory = new DiskFileItemFactory();

ServletFileUpload upload = new ServletFileUpload(factory);

// 设置上传文件的大小限制为1M

factory.setSizeThreshold(1024 \* 1024);

List items = null;

try {

items = upload.parseRequest(request);

} catch (FileUploadException e) {

e.printStackTrace();

}

Iterator iter = items.iterator();

while (iter.hasNext()) {

FileItem item = (FileItem) iter.next();

if (!item.isFormField()) {

// 根据时间戳创建头像文件

filename = System.currentTimeMillis() + ".jpg";

String photoFolder = "e:\\project\\j2ee\\web\\uploaded";

File f = new File(photoFolder, filename);

f.getParentFile().mkdirs();

// 通过item.getInputStream()获取浏览器上传的文件的输入流

InputStream is = item.getInputStream();

// 复制文件

FileOutputStream fos = new FileOutputStream(f);

byte b[] = new byte[1024 \* 1024];

int length = 0;

while (-1 != (length = is.read(b))) {

fos.write(b, 0, length);

}

fos.close();

} else {

System.out.println(item.getFieldName());

String value = item.getString();

value = new String(value.getBytes("ISO-8859-1"), "UTF-8");

System.out.println(value);

}

}

String html = "<img width='200' height='150' src='uploaded/%s' />";

PrintWriter pw= response.getWriter();

pw.format(html, filename);

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}