



## 第2章 Servlet基础

本节内容:

**2.3 处理HTTP请求(2)**

**2.4 发送HTTP响应**

**2.5 Servlet生命周期**



## 2.3 处理HTTP请求

### 2.3.3 检索请求参数

### 2.3.4 使用请求对象存储数据

### 2.3.5 请求转发

### 2.3.6 其他请求处理方法



### 2.3.3 检索请求参数

- 请求参数（request parameter）是随请求一起发送到服务器的数据，它以“名/值”对的形式发送。下面是与检索请求参数有关的方法。

**String getParameter(String name)**

- 该方法必须保证指定的参数值只有一个。

**String[] getParameterValues(String name)**

- 返回值是一个String数组，该方法适用于参数有多个值的情况。



## 2.3.3 检索请求参数

- 如何传递请求参数？

- 方法1：通过表单指定请求参数，每个表单域可以传递一个请求参数，这种方法适用于**GET**请求和**POST**请求。

- 方法2：通过查询串指定请求参数，将参数名和值附加在请求的**URL**后面，这种方法只适用于**GET**请求。

- 下面是一个登录页面login.jsp，通过表单提供请求参数，然后在Servlet中检索参数并验证，最后向用户发送验证消息。

- 【例2-2】 login.jsp页面。

- 【例2-3】 LoginServlet.java程序。

&lt;%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%&gt;

&lt;html&gt;

```
<head><title>登录页面</title></head>
```

<body>

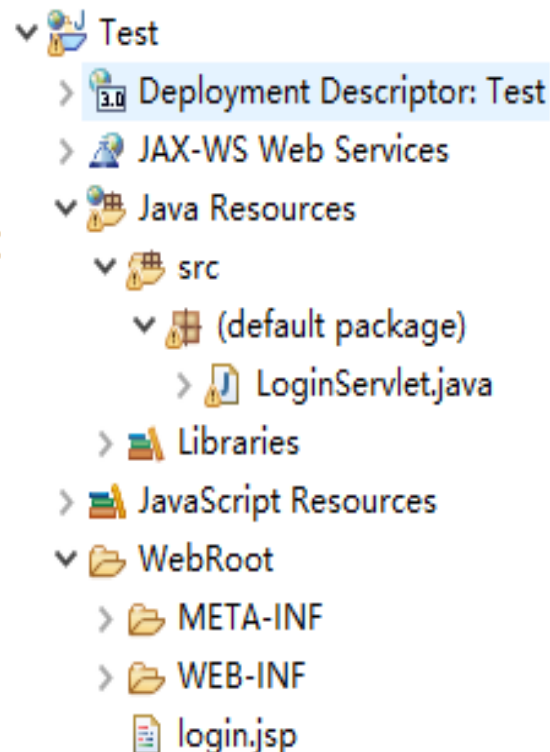

```
<form action="LS" method="post">
```

| 用户名: |
 <input type="text" name="username"/></td></tr> || 密   码: |
 <input type="password" name="password"/></td></tr> ||  |
  |

&lt;/form&gt;

&lt;/body&gt;

&lt;/html&gt;



## 2.3.3 检索请求参数— LoginServlet.java主要代码

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/LS")
public class LoginServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        // 检索客户端传来的请求参数
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        out.println("<html><body>");
        // 用户名和口令均为admin, 认为登录成功
        if("admin".equals(username)&& "admin".equals(password)){
            out.println("登录成功! 欢迎您, "+username);
        }else{
            out.println("对不起! 您的用户名或密码不正确. ");
        }
        out.println("</body></html>");
    }
}
```

← → □ ↻ http://localhost:8080/Test/LS

登录成功! 欢迎您, admin

← → □ ↻ http://localhost:8080/Test/LS

对不起! 您的用户名或密码不正确.



### 2.3.3 检索请求参数

- 在LoginServlet类中仅覆盖了doPost()方法，这样该Servlet只能处理POST请求，不能处理GET请求。
- 如果将login.jsp中form元素的method属性修改为“get”，该程序不能正常运行。
- 如果希望该Servlet既能处理POST请求，又能处理GET请求，可以添加下面的doGet()方法，并在其中调用doPost()方法。

```
public void doGet(HttpServletRequest request,  
                  HttpServletResponse response)  
    throws ServletException, IOException {  
    doPost(request,response); }
```



### 2.3.3 检索请求参数

- 如果向服务器发送GET请求，还可以将请求参数附加在请求URL的后面。例如，可以直接使用下面的URL访问LoginServlet，而不需要通过表单提供参数。

<http://localhost:8080/Test/LS?username=admin&password=admin>

- 问号后面内容称为查询串（query string），内容为请求参数的“名/值”对，参数名和参数值之间用等号（=）分隔，若有多个参数，中间用“&”符号分隔。





### 2.3.3 检索请求参数

在超链接中也可以传递请求参数，例如：

`<a href="/Test/LS?username=admin&password=admin">`  
登录`</a>` 。

■使用查询串提供请求参数的方法只能用在**GET**请求中，不能用在**POST**请求中，并且请求参数将显示在浏览器的地址栏中。



## 2.3.4 使用请求对象存储数据

- 可以使用请求对象存储数据。请求对象是一个作用域（**scope**）对象，可以在其上存储属性实现数据共享。
- 属性（**attribute**）包括属性名和属性值。属性名是一个字符串，属性值是一个对象。
- 有关属性存储的方法有
  - **public void setAttribute(String name, Object obj)**
  - **public Object getAttribute(String name)**



## 2.3.4 使用请求对象存储数据

- 下列代码创建一个购物车对象，并将其存储在请求作用域（request）中。

```
ShoppingCart cart = new ShoppingCart();
```

```
request.setAttribute("name", cart);
```

- 下列代码从请求作用域中检索出购物车对象。

```
ShoppingCart cart =
```

```
    (ShoppingCart )request.getAttribute("name");
```



## 2.3.5 请求转发

- 在实际应用中可能需要将控制转发(forward)到其他资源。
- 例如，对登录系统，如果用户输入了正确的用户名和口令，**LoginServlet**应该将请求转发到欢迎页面，否则应将请求转发到登录页面或错误页面。



## 2.3.5 请求转发

- 通过请求对象的`getRequestDispatcher()`方法得到`RequestDispatcher`对象，该对象称为请求转发器对象。
- 该方法的格式如下。

`RequestDispatcher getRequestDispatcher(String path)`

- `path`用来指定要转发到的资源路径。
  - 它可以是绝对路径，即以“/”开头，它被解释为相对于当前应用程序的文档根目录。
  - 也可以是相对路径，即不以“/”开头，它被解释为相对于当前资源所在的目录。



## 2.3.5 请求转发

- `RequestDispatcher`接口定义了下面两个方法：
- 将控制转发到服务器上的另一个动态或静态资源

**`public void forward (ServletRequest request,  
ServletResponse response)`**

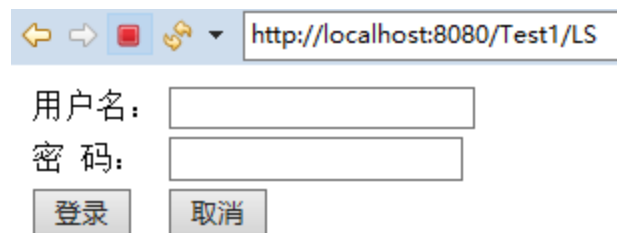
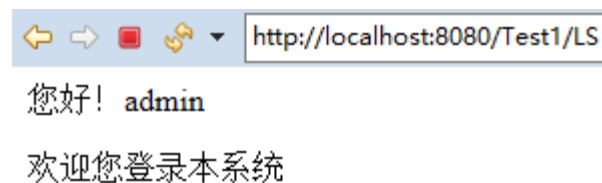
- 将控制转发到指定的资源，将其输出包含到当前输出中

**`public void include (ServletRequest request,  
ServletResponse response)`**

## 2.3.5 请求转发

### ■ 修改LoginServlet程序，例2-4

- 实现当用户登录成功将请求转发到welcome.jsp,
- 当登录失败将请求转发到login.jsp,





## 2.3.5 请求转发

**LoginServlet**转发部分核心代码:

```
if(username.equals("admin")&&password.equals("admin"))
{
    // 将用户名作为属性存储在请求作用域中
    request.setAttribute("username", username);
    // 将请求转发到welcome.jsp页面
    RequestDispatcher rd =
        request.getRequestDispatcher("/welcome.jsp");
    rd.forward(request, response);
}else{
    // 将请求转发到login.jsp页面
    RequestDispatcher rd =
        request.getRequestDispatcher("/login.jsp");
    rd.forward(request, response);
}
```





## 2.3.5 请求转发

welcome.jsp核心代码。

```
<%@ page contentType="text/html;charset = UTF-8"  
pageEncoding="UTF-8"%>
```

```
<html><head><title>欢迎页面</title></head>
```

```
<body>
```

```
<p>您好!
```

```
<%= (String)request.getAttribute("username")%> </p>
```

```
<p>欢迎您登录本系统</p>
```

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



## 2.3.6 其他请求处理方法

- 在`HttpServletRequest`接口中还定义了下面常用的方法来检索客户端有关信息

```
out.println("<tr><td>请求方法</td>");
```

```
out.println("<td>" + request.getMethod() + "</td></tr>");
```

```
out.println("<tr><td>请求URI</td>");
```

```
out.println("<td>" + request.getRequestURI() + "</td></tr>");
```

```
out.println("<tr><td>请求协议</td>");
```

```
out.println("<td>" + request.getProtocol() + "</td></tr>");
```



## 2.3.6 其他请求处理方法

```
out.println("<tr><td>客户主机名</td>");
```

```
out.println("<td>" + request.getRemoteHost() + "</td></tr>");
```

```
out.println("<tr><td>客户IP地址</td>");
```

```
out.println("<td>" + request.getRemoteAddr() + "</td></tr>");
```

```
out.println("<tr><td>端口</td>");
```

```
out.println("<td>" + request.getRemotePort() + "</td></tr>");
```



## 2.4 发送HTTP响应

### 2.4.1 HTTP响应结构

### 2.4.2 输出流与内容类型

### 2.4.3 响应重定向

### 2.4.4 设置响应头

### 2.4.5 发送状态码和错误消息



## 2.4.1 HTTP响应结构

- 由服务器向客户发送的HTTP消息称为HTTP响应（HTTP response）
- HTTP响应也由三部分组成：状态行、响应头和响应的数据。
- 如图所示为一个典型的HTTP响应消息。

## 2.4.1 HTTP响应结构



HTTP响应的状态行由三部分组成，各部分由空格分隔：

HTTP版本      请求结果的响应状态码以及描述状态码

状态行之后的头行称为响应头（**response header**）。响应头是服务器向客户端发送的消息。在图响应消息中包含了三个响应头。 **Date**响应头表示消息发送的日期，

响应头后面是一空行，空行的后面是响应的数据。

**Length**指示响应内容的长度。



## 2.4.2 输出流与内容类型

- **Servlet**可以使用输出流直接向客户发送响应。
- 调用响应对象的 **getWriter()** 方法可以得到 **PrintWriter**对象，使用它可向客户发送**文本数据**。
- 调用响应对象的**getOutputStream()**方法可以得到 **ServletOutputStream**对象，使用它可向客户发送**二进制数据**。
- 通常，在发送响应数据之前还需通过响应对象的 **setContentType()**方法设置响应的内容类型。



## 2.4.2 输出流与内容类型

- **PrintWriter**对象被**Servlet**用来动态产生页面。调用响应对象的**getWriter()**方法返回**PrintWriter**类的对象，它可以向客户发送文本数据。

**PrintWriter pw = response.getWriter();**

- 如果要向客户发送二进制数据（如JAR文件），应该调用响应对象的**getOutputStream()**方法，返回**ServletOutputStream**类对象。

**ServletOutputStream sos =  
response.getOutputStream();**





## 2.4.2 输出流与内容类型

- 在向客户发送数据之前，一般应该设置发送数据的**MIME**（Multipurpose Internet Mail Extensions）**内容类型**。
- MIME**是描述消息内容类型的因特网标准。**MIME**消息包含文本、图像、音频、视频以及其他应用程序专用的数据。
- 在客户端，浏览器根据响应消息的**MIME**类型决定如何处理数据。**默认的响应类型是text/html**，对这种类型的数据，浏览器解释执行其中的标签，然后在浏览器中显示结果。如果指定了其他**MIME**类型，浏览器可能打开文件下载对话框或选择应用程序打开文件。



## 2.4.2 输出流与内容类型

- 使用响应对象的`setContentType()`方法设置响应数据内容类型，如果没有调用该方法，内容类型将使用默认值`text/html`，即HTML文档。
- 使用响应对象的`setContentType()`方法还可以设置客户端显示所使用的字符集，例如：

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

- 如果没有指定响应的字符编码，`PrintWriter`将使用`ISO-8859-1`编码。



### 2.4.3 响应重定向

- **Servlet**在对请求进行分析后，可能不直接向浏览器发送响应，而是向浏览器发送一个**Location**响应头，告诉浏览器访问其他资源，这称为响应重定向。
- 响应重定向是通过响应对象的**sendRedirect()**方法实现。

**public void sendRedirect(String location)**

- **location**为资源的**URI**，若以“/”开头，则相对于服务器根目录（如，**/Test/login.html**），若不以“/”开头，则相对于**Web**应用程序的文档根目录（如，**login.jsp**）。



### 2.4.3 响应重定向

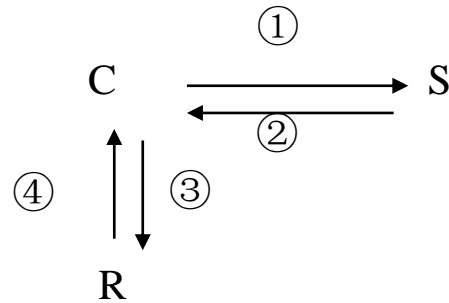
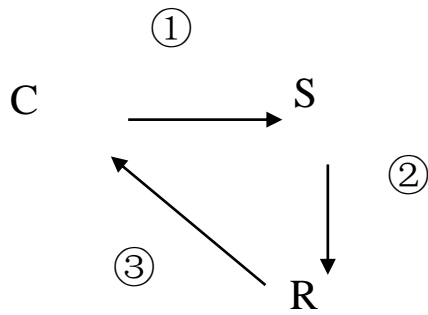
- 下面程序是一个使用`sendRedirect()`方法重定向响应的例子。

- **【例2-5】** `RedirectServlet.java`程序。

```
if((userAgent!=null)&&  
    (userAgent.indexOf("admin")!= -1)){  
    response.sendRedirect("welcome.jsp");  
}else{  
    response.sendRedirect("http://localhost:8080/");  
}
```

### 2.4.3 响应重定向

- 请求转发与响应重定向不同，区别。





## 2.4.5 发送状态码和错误消息

- 服务器向客户发送的响应的第一行是状态行，它由三部分组成：**HTTP版本**、**状态码**和**状态码**的描述信息，如下是一个典型的状态行：

**HTTP/1.1 200 OK**

- 状态码**200**是系统自动设置的，**Servlet**一般不需要指定该状态码。对于其他状态码，可以由系统自动设置，也可以使用响应对象的**setStatus()**方法设置，该方法的格式为：

**public void setStatus (int sc)**

- 该方法可以设置任意的状态码。参数**sc**表示要设置的状态码，它可以用整数表示。



## 2.4.5 发送状态码和错误消息

- 在HTTP协议1.1版中定义了若干状态码，这些状态码由3位整数表示，一般分为5类，如下表所示。

状态码范围	含义	示例
100~199	表示信息	100表示服务器同意处理客户的请求
200~299	表示请求成功	200表示请求成功，204表示内容不存在
300~399	表示重定向	301表示页面移走了，304表示缓存的页面仍然有效
400~499	表示客户的错误	403表示禁止的页面，404表示页面没有找到
500~599	表示服务器的错误	500表示服务器内部错误，503表示以后再试



## 2.5 Servlet生命周期

- 1 类加载
- 2 Servlet实例化
- 3 Servlet初始化
- 4 为客户提供服务
- 5 Servlet销毁





## 2.5 Servlet生命周期

- **Servlet**是一种在**Web**容器中运行的组件，有一个从创建到销毁的过程，这个过程被称为**Servlet生命周期**，包括**5**个阶段，如图所示。

## 2.5 Servlet生命周期

Web容器

① 加载到内存

当容器接收到对Servlet的请求时，容器根据请求URL找到  
当容器接收到对Servlet的请求时，容器根据请求URL找到  
当容器决定不再需要Servlet实例时，它将在Servlet实例上  
调用destroy()方法，Servlet在该方法中释放资源，如它在  
init()中获得的数据库连接。一旦该方法被调用，Servlet实  
例不能再提供服务。

容器通常使用Class类的newInstance()方法创建Servlet实例。

回收

⑤ 销毁



## 2.6 小结

(1) 处理HTTP请求(2)

(2) 发送HTTP响应

(3) **Servlet**生命周期