

## 第 4 章 JSP 技术基础知识

JSP (Java Server Page) 是 SUN 公司开发的一种服务器端的脚本语言, 自从 1999 年推出以来, 逐步发展为开发 Web 应用一项重要技术。JSP 可以嵌套在 HTML 中, 而且支持多个操作系统平台, 一个用 JSP 开发的 Web 应用系统, 不用做什么改动就可以在不同的操作系统中运行。

在本章接下来的内容中, 首先将简单介绍 JSP 的运行原理和基本语法, 然后重点介绍在实际开发过程中技巧和方法。

### 4.1 JSP 简介

JSP 本质上就是把 Java 代码嵌套到 HTML 中, 然后经过 JSP 容器的编译执行, 可以根据这些动态代码的运行结果生成对应的 HTML 代码, 从而可以在客户端的浏览器中正常显示。在这个小节中将介绍 JSP 的运行原理、JSP 的优点和其运行环境的搭建。

#### 4.1.1 运行原理

因为本书讲述的主要内容就是基于 JSP 的 Java Web 应用程序开发技术, 为了是读者对 JSP 有一个更加深入的了解, 在这里有必要对 JSP 的运行机制、原理进行深入的介绍。

如果 JSP 页面是第一次被请求运行, 服务器的 JSP 编译器会生成 JSP 页面对应的 Java 代码, 并且编译成类文件。当服务器再次收到对这个 JSP 页面请求的时候, 会判断这个 JSP 页面是否被修改过, 如果被修改过就会重新生成 Java 代码并且重新编译, 而且服务器中的垃圾回收方法会把没用的类文件删除。如果没有被修改, 服务器就会直接调用以前已经编译过的类文件。

下面就是一个简单的 JSP 页面, 在这个页面中没有任何动态代码, 其功能是在页面显示一句话, 这个 JSP 页面的具体代码如下。

```
//-----文件名: Simple.jsp-----
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<html>
  <head>
    <title>简单 JSP 页面示例</title>
  </head>
  <body>
    这是一个简单的 JSP 页面示例 <br>
  </body>
</html>
```

上面这个 JSP 页面在被请求的时候, Web 服务器中的 JSP 编译器会生成对应的 Java 文件, 上面这个 JSP 程序对应的 Java 代码如下所示。

```
//-----文件名: Simple_jsp.java-----
package org.apache.jsp;
```

```

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.util.*;

public final class Simple_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    private static java.util.Vector _jspx_dependants;

    public java.util.List getDependants() {
        return _jspx_dependants;
    }

    public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {

        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        JspWriter _jspx_out = null;
        PageContext _jspx_page_context = null;

        try {
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html;charset=gb2312");
            pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;

            out.write("\r\n");
            out.write("<html>\r\n");
            out.write("    <head>\r\n");
            out.write("        <title>简单 JSP 页面示例</title>\r\n");
            out.write("    </head>\r\n");
            out.write("<body>\r\n");
            out.write("    这是一个简单的 JSP 页面示例 <br>\r\n");
            out.write("    </body>\r\n");
            out.write("</html>\r\n");
        } catch (Throwable t) {

```

```

    if (!(t instanceof SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
            out.clearBuffer();
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
    }
} finally {
    if (_jspxFactory != null) _jspxFactory.releasePageContext(_jspx_page_context);
}
}
}

```

上面这段代码就是 Simple.jsp 所对应的 Java 代码，假如你的 Tomcat 安装路径为 C:\Tomcat5.0，Web 应用程序发布的名称为 chap4，那么可以在 C:\Tomcat 5.0\work\Catalina\localhost\chapt4\org\apache\jsp 这个目录下找到 JSP 所对应的 Java 文件和编译出来的类文件。

上面这段程序再本质上就是一个 Servlet，把所有页面的显示的内容都用 out 对象打印出来，包括每个 HTML 标签，所以说 JSP 页面本质上就是 Servlet 的一个种化身，在 JSP 程序种离不开 Servlet 的影子。这段代码的具体语法可以不必深究，这些工作都是有服务器中的 JSP 编译器来完成，这个过程是自动完成的，无需手工干预。

注意：只用被请求过的页面才能生成对应的 Java 文件，当 JSP 页面被修改后，再次对这个页面进行请求才会重新生成对应的 Java 文件。

#### 4.1.2 选择 JSP 的原因

在 Web 应用开发中，可供选择的动态页面语言技术有很多，例如 PHP，ASP，JSP 等，在这些动态页面语言中，JSP 凭借其自身的优点成为开发人员最喜欢的语言之一。下面列出的几条就是开发人员钟爱 JSP 的重要原因。

- ❑ JSP 就是在 HTML 中嵌入 Java 代码，所以在本质上 JSP 程序就是 Java 程序，JSP 程序继承了 Java 的一切优点。JSP 程序有严格的 Java 语法和丰富的 Java 类库支持。
- ❑ JSP 页面在服务器中都会被 JSP 编译器编译成对应的 Servlet，所以就拥有 Java 跨平台的优点，所有的 JSP 程序，无需改动就可以方便地迁移到其他操作系统平台，这是在其他动态脚本语言中所无法想象的。
- ❑ JSP 中可以使用 JavaBean 进行逻辑封装，这样就可以实现逻辑功能代码的重用，从而大大提高系统的可重用性，同时也提高了程序的开发效率。
- ❑ JSP 程序容易上手，如果有 HTML 和 Java 的基本知识，那么学习 JSP 程序就没有任何难度。
- ❑ 在 Java 领域，开源的项目越来越多，这些开源项目是全世界 Java 爱好者的心血的结晶，在我们的 JSP 程序中可以非常方便地使用这些开源工具。在开源项目的支持上，JSP 更是其他动态语言不能相比的。

开发者从对 Java 的热爱延伸到对 JSP 的热爱，同时 JSP 又是 J2EE 体系中最重要，而且是最基础的一个组成部分，如果要体验 J2EE 带来的开发效率和优势，学习 JSP 是一个非常有效的入门方式。

在接下来的章节将开始进入 JSP 的精彩世界。

### 4.1.3 环境搭建

要运行 JSP 程序，必需为其提供一个 JSP 容器，也就是需要一个 Web 服务器。支持 JSP 的服务器非常多，Tomcat、Resin、Weblogic、WebSphere 等对 JSP 的支持都非常好，但是由于 Weblogic 和 WebSphere 都是功能非常强大的重量级服务器，而且价格昂贵，对计算机的硬件配置要求也比较高，所以在一般情况下，如果只用到 JSP 的技术，是没有必要选择这两个服务器的。

一般情况下我们可以选择 Tomcat 或者 Resin 作为 JSP 容器，这两个服务器对 JSP 的支持都非常好，而且都是开源的，可以免费使用，同时它们对计算机的硬件配置要求也是非常低的。这两个服务器任选其一都可以满足所有 JSP 开发的需要，在这里我们选择 Tomcat 作为 JSP 容器。

Tomcat 的运行需要 JDK 的支持，所以在安装 Tomcat 的时候需要提前安装 JDK，其中 JDK 的安装方法在本书第二章中有详细的介绍。

Tomcat 是 apache 的一个开源项目，可以在其官方网站 <http://tomcat.apache.org> 下载 Tomcat 的各个版本，在这个网站上提供 Tomcat 各个版本下载，下载的文件格式有 zip、tar.gz、exe 三种格式，其中 zip 格式只需要解压到某个目录，然后在环境变量中设置路径即可运行，这种方法比较灵活，但是环境变量设置错误 Tomcat 就不能运行。tar.gz 格式是 Unix、linux 系统上的压缩格式。exe 格式是 Windows 操作系统中的安装程序，这中格式的安装程序提供详细的安装向导，安装过程比较容易，在这里我们选择 exe 的安装个格式。

- (1) 下载 jakarta-tomcat-5.0.28.exe 安装文件，运行这个文件就会进入如图 4.1 所示的安装界面。
- (2) 在图 4.1 中选择“Next”，进入如图 4.2 所示的安装协议界面。

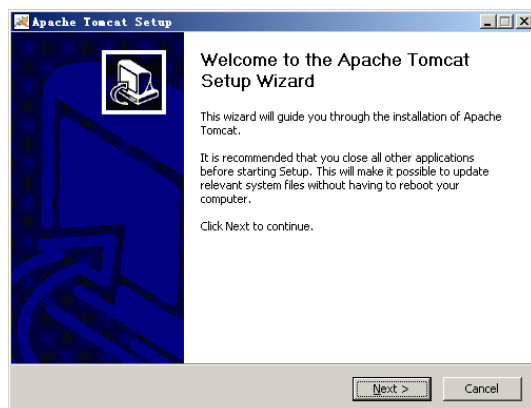


图 4.1 Tomcat 安装初始界面

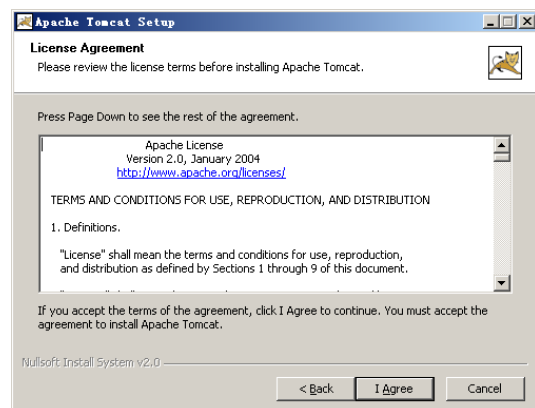


图 4.2 Tomcat 安装协议选择界面

(3) 在图 4.2 中选择“I Agree”进入如图 4.3 所示的组件选择界面，如果不接受协议安装程序就会退出。

(4) 在如图 4.3 的界面中，可以选择安装 Tomcat 的组件，其中安装类型选择 Normal 即可，这时候下面的几个组件会被自动选中，然后选择“Next”就可以进入如图 4.4 的安装路径选择界面。

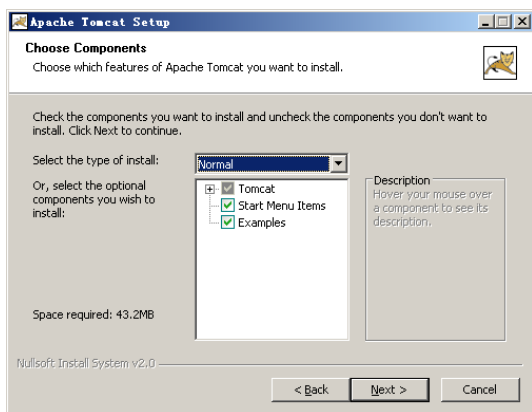


图 4.3 Tomcat 组件选择界面

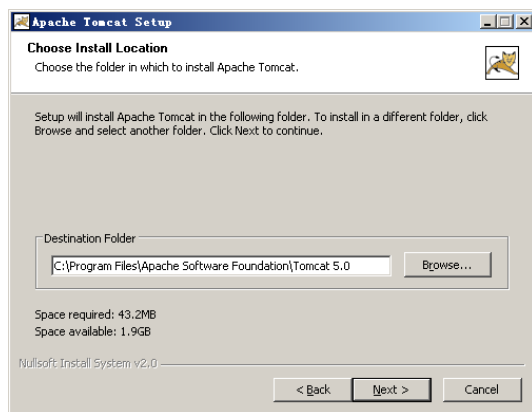


图 4.4 Tomcat 安装路径选择界面

(5) 在如图 4.4 中选择 Tomcat 将要安装到的目标路径，选择“Next”进入如图 4.5 所示界面。

(6) 在如图 4.5 的界面中，可以选择 Tomcat 的访问端口，可以采用默认端口是 8080，在上面这个界面中还可以设置 Tomcat 管理员的用户名和密码，利用这个管理员账号可以管理 Tomcat 的一些运行配置等信息。端口和管理员账号设置完成后选择“Next”进入如图 4.6 所示的 JVM 选择界面。

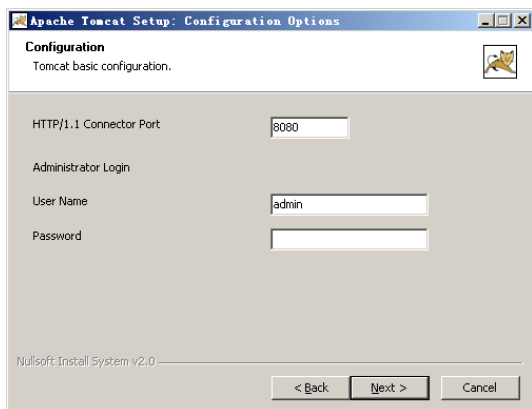


图 4.5 Tomcat 端口选择和管理员密码设置界面

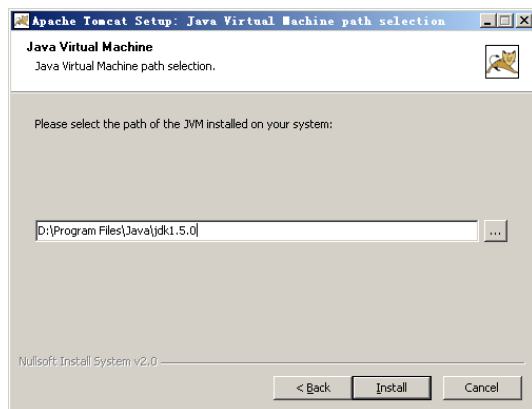


图 4.6 JVM 选择界面

(7) 在图 4.6 中的 JVM 路径不用选择，如果 JDK 已经正确安装配置的话，Tomcat 的安装程序可以自从系统变量中读出这个路径，无需手动配置。选择“Install”开始安装，安装完成以后会出现如图 4.7 所示的安装完成界面。

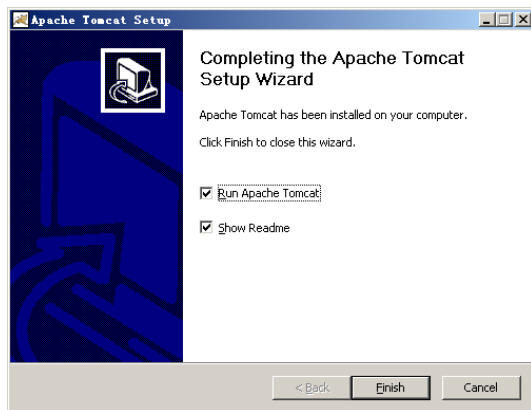


图 4.7 Tomcat 完成安装提示界面

(8) 在图 4.7 的界面中选择“Finish”完成安装。

(9) Tomcat 的启动方式后好多中，最基本的是在 Tomcat 安装目录下的 bin 文件夹中的 startup.bat、shutdown.bat，其中 startup.bat 可以启动 Tomcat 服务器，shutdown.bat 可以关闭 Tomcat 服务器。运行 startup.bat，如果可以看到如图 4.8 所示的启动信息说明 Tomcat 启动成功，

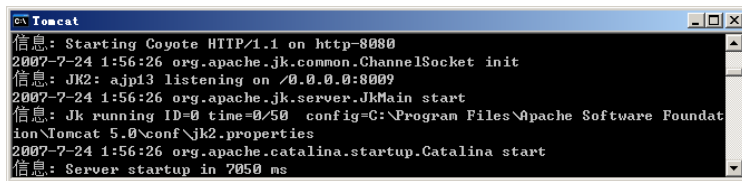


图 4.8 Tomcat 启动信息

(10) 当看到如图 4.8 所示“Server startup in \*\*\*\*\* ms”的时候说明 Tomcat 已经成功启动，打开浏览器，在地址栏中输入 <http://localhost:8080/>，如果出现如图 4.9 所示的界面说明 Tomcat 已经成功安装。

(11) Tomcat 还有一个图形界面的管理工具，选择“程序”|“所有程序”|“Apache Tomcat 5.0”|“Monitor Tomcat”，就会出现如图 4.10 所示的图形管理界面。

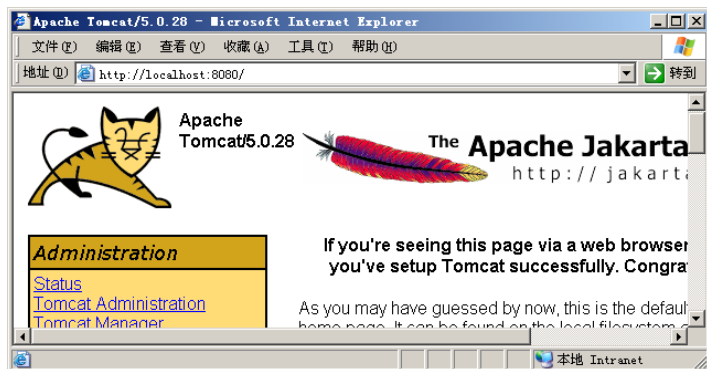


图 4.9 Tomcat 默认欢迎界面

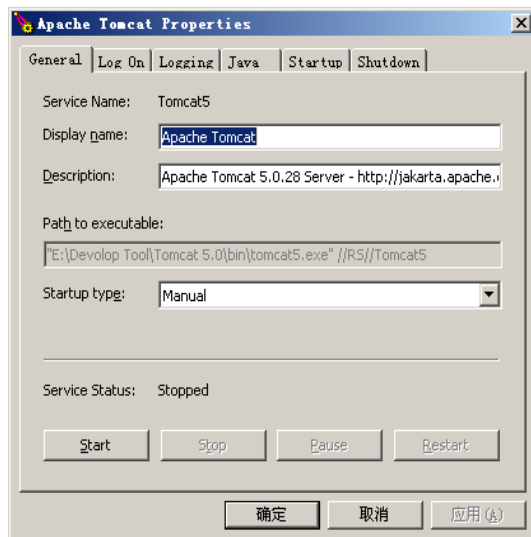


图 4.10 Tomcat 图形管理界面

(12) 如图 4.10 的管理界面中，“General”选项卡中，“Start”可以启动 Tomcat，“Stop”可以关闭 Tomcat，“Pause”可以暂停 Tomcat，“Restart”可以重新启动“Tomcat”。

(13) 在“logging”选项卡中可以对 Tomcat 的日志信息进行配置，在“Java”选项卡中可以对 JVM 进行配置。

注意：如果在启动 Tomcat 的过程中出现了“Address already in use: JVM\_Bind”这个错误，则错误的解决方法有两种，一种是关闭其他正在运行的 Tomcat 程序，另一种解决方法就是修改 Tomcat 的端口，可以在 Tomcat 安装目录下的 conf 目录下找到 server.xml 的文件，在这个文件中搜索“<Connector port=“8080””，然后把这个 8080 端口改为其他端口，重新启动 Tomcat 即可，如果这时候还有其他端口冲突，按照同样的方法修改相应端口即可。

## 4.2 JSP 基本语法

本书的重点内容是介绍基于 JSP 的 Web 开发技术，对于 Java 的语法在此不做详细的介绍，这里所涉及 JSP 语法指的是在 JSP 中所特有的语法规则，在接下来的章节中将假设读者已经了解 Java 的基本语法，只介绍 JSP 的结构、变量声明、表达式、动作、指令等 JSP 的特有语法。如对 Java 语法有疑问的读者可以参考相关语法书籍。

## 4.3 程序结构

JSP 就是把 Java 代码嵌套在 HTML 中，所以 JSP 程序的结构可以分为两大部分：一部分是静态的 HTML 代码；另一部分是动态的 Java 代码和 JSP 自身的标签和指令；当 JSP 页面第一次被请求的时候，服务器的 JSP 编译器会把 JSP 页面编译成对应的 Java 代码，根据动态 Java 代码执行的结果，生成对应的纯 HTML 的字符串返回给浏览器，这样就可以把动态程序的结果展示给用户。下面就是一个简单的 JSP 程序的代码，现在分析其基本结构。

```
//-----文件名: Add.jsp-----
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%
    int first = 0;
    int second = 0;
    if(request.getParameter("first")!=null&&request.getParameter("first").length()>0)
    {
        first = Integer.parseInt(request.getParameter("first"));
    }
    if(request.getParameter("second")!=null&&request.getParameter("second").length()>0)
    {
        second = Integer.parseInt(request.getParameter("second"));
    }
%>
<html>
<head>
<title>求和 JSP 程序示例</title>
<script type="text/javascript">
    function check()
    {
        if(this.document.forms[0].first.value.length==0)
            alert("请输入第一个整数!");
        else if(this.document.forms[0].second.value.length==0)
            alert("请输入第二个整数!");
        else if (isNaN(this.document.forms[0].first.value))
            alert("输入的的第一个数字必须是整型数据");
        else if (isNaN(this.document.forms[0].second.value))
            alert("输入的第二个数字必须是整型数据");
        else
            this.document.forms[0].submit();
    }
}
```

```

</script>
</head>
<body>
  <form action="Add.jsp" method="post">
    <font size="2">
      这个 JSP 页面的功能是求两个整数的和: <br>
      请输入第一个数: <input type="text" name="first"/><br>
      请输入第二个数: <input type="text" name="second"/><br>
      这两个数的和为: <%= (first+second) %><br>
      <input type="button" value="求和" onclick="check()"/><br>
    </font>
  </body>
</html>

```

在上面这个示例程序中，代码的结构是十分清晰的，至于这个程序的具体实现现在不必深究，在本章内容结束之后自然就会明白，现在只分析这个 JSP 程序的基本代码结构。

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
```

上面这行代码是一个 JSP 的 page 指令，是 JSP 特有的指令。

```

<%
  int first = 0;
  int second = 0;
  if(request.getParameter("first")!=null&&request.getParameter("first").length()>0)
  {
    first = Integer.parseInt(request.getParameter("first"));
  }
  if(request.getParameter("second")!=null&&request.getParameter("second").length()>0)
  {
    second = Integer.parseInt(request.getParameter("second"));
  }
%>

```

上面这段代码是典型的 Java 代码片段，如果要在 JSP 页面中用到 Java 代码，只需要在<% %>这个标签中嵌入 Java 代码片段即可。

```

<script type="text/javascript">
  function check()
  {
    if(this.document.forms[0].first.value.length==0)
      alert("请输入第一个整数!");
    else if(this.document.forms[0].second.value.length==0)
      alert("请输入第二个整数!");
    else if (isNaN(this.document.forms[0].first.value))
      alert("输入的第二个数字必须是整型数据");
    else if (isNaN(this.document.forms[0].second.value))
      alert("输入的第二个数字必须是整型数据");
    else
      this.document.forms[0].submit();
  }
</script>

```

上面这段代码是典型的表单输入验证，这是 JavaScript 代码片段，和 JSP 无关，是属于客户端的语言。

```
这两个数的和为: <%= (first+second) %><br>
```



在上面这行代码中用到了 JSP 的输出标签，在 HTML 页面中直接输出`<%= %>`中表达式的值。这个程序中其他剩余的其他部分就是单纯的静态 HTML 代码。上面这个示例程序的运行结果如图 4.11 所示。



图 4.11 求和 JSP 示例程序运行效果

## 4.4 JSP 动作指令

在 Web 程序涉及中经常需要用到 JSP 的动作指令，例如在使用 JavaBean 的时候就离不开 userBean 的指令，JSP 的强大功能和它丰富的动作指令标签是分不开的。

在接下来的章节中将对这些指令进行详细的介绍，读者可以仔细体会每个动作的示例程序，在示例程序中掌握这些动作指令的基本用法。

### 4.4.1 include 动作指令

`include` 动作指令可以在 JSP 页面中动态包含一个文件，这与 `<include>` 指令不同，前者可以动态包含一个文件，文件的内容可以是静态的文件也可以是动态的脚本，而且当包含的动态文件被修改的时候 JSP 引擎可以动态对其进行编译更新。而 `<include>` 指令仅仅是把一个文件简单的包含在一个 JSP 页面中，从而组合成一个文件，仅仅是简答的组合的作用。其功能没有 `<jsp:include>` 动作指令强大。

`include` 动作指令的具体使用方法可以参考下面的示例程序。

```
//-----文件名: IncludeAction.jsp-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>  
<html>  
  <head>  
    <title>include 动作指令使用示例程序</title>  
  </head>  
  <body>  
    <font size="2">  
      <jsp:include flush="true" page="header.txt"></jsp:include>  
      这是一个 JSP 动作标签 include 的使用示例程序。<br>  
      <jsp:include flush="true" page="footer.jsp"></jsp:include>  
    </font>  
  </body>  
</html>
```

在上面这个程序中，用了两次 `include` 动作指令，第一次包含了一个静态的文本文件，第二次用 `include` 指令包含了一个动态的 JSP 文件。`include` 动作标签的使用格式如下。

```
<jsp:include flush="true" page="header.txt"></jsp:include>
```

在上面这行代码中，只需要指定 page 的路径即可，即指明包含的文件为 header.txt 其中 header.txt 的内容如下。

```
//-----文件名: header.txt-----
```

这里用 include 动作指令包含一个静态的文本文件。<br>

在第二次使用 include 的指令时包含了一个 JSP 文件。

```
<jsp:include flush="true" page="footer.jsp"></jsp:include>
```

上面这行代码同样需要指明 page 的值，即指明包含的文件为 footer.jsp，其中 footer.jsp 的内容如下。

```
//-----文件名: footer.jsp-----
```

```
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
```

```
<html>
```

```
<body>
```

```
<%
```

```
out.print("这里用 include 动作指令包含一个动态的 JSP 页面!");
```

```
%>
```

```
</body>
```

```
</html>
```

上面这个示例程序的运行结果如图 4.12 所示。

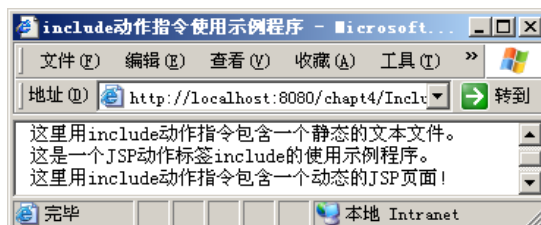


图 4.12 include 动作指令示例程序运行效果

注意：在被包含的 JSP 程序中，如果有所修改，JSP 引擎会及时发现，并重新编译。这就是 include 动作指令与 include 指令的最大不同。

## 4.4.2 forward 动作指令

forward 动作指令可以用来控制网页的重定向。即从当前页面跳转到另一个页面。

forward 动作的使用方法非常简单，具体使用格式如下。

```
<jsp:forward page="http://www.sohu.com"></jsp:forward>
```

在上面这行代码中，只要指明 page 的值，当 JSP 执行到这行代码的时候就可以直接跳转到对应的网页。forward 动作指令的具体用法可以参考下面的例子。

```
//-----文件名: Forward.jsp-----
```

```
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312" %>
```

```
<html>
```

```
<head>
```

```
<title>forward 动作指令使用示例</title>
```

```
</head>
```

```
<body>
```

```
<jsp:forward page="http://www.sohu.com"></jsp:forward>
```

```
</body>
```

```
</html>
```

上面这个页面运行以后就直接跳转到 [www.sohu.com](http://www.sohu.com) 的主页面。

注意：forward 动作指令和 HTML 中的<a></a>超链接标签是不同的，在<a></a>中只有单击链接才能实现页面跳转，在 forward 动作指令中一切都是可以用 Java 的代码进行控制，可以在程序中直接决定页面跳转的方向和时机。

### 4.4.3 param 动作指令

在上面 forward 动作指令中可以用程序控制页面的跳转，如果需要在跳转的时候同时传递参数，这时候就需要用到 param 动作指令。param 动作指令的具体使用方法可以参考下面的示例程序。

下面这个页面是初始页面，参数从这里开始传递。

```
//-----文件名: PassParam.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>页面跳转并且传递参数示例</title>
  </head>
  <body>
    从这个页面传递参数:
    <jsp:forward page="GetParam.jsp">
      <jsp:param name="param" value="test"/>
    </jsp:forward>
  </body>
</html>
```

在上面这段程序中，把页面跳转到 GetParam.jsp 这个页面，同时向 GetParam.jsp 这个页面传递了一个名称为 param 的参数，这个参数的值为 test。具体设置代码就是上面这段代码。

```
<jsp:forward page="GetParam.jsp">
  <jsp:param name="param" value="test"/>
</jsp:forward>
```

其中 GetParam.jsp 页面的代码如下。

```
//-----文件名: GetParam.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>页面跳转并且传递参数示例</title>
  </head>
  <body>
    <font size="2">
      这个页面接受传递过来的参数: <br>
      前一个页面传递过来的参数为: <%out.print(request.getParameter("param")); %>
    </font>
  </body>
</html>
```

在上面这个页面中，接受从 PassParam.jsp 这个页面传递过来的参数，并且把参数的值打印在页面上。上面这个跳转并且传递参数的示例程序运行效果如图 4.13 所示。



图 4.13 页面跳转并且传递参数示例程序运行效果

注意：在这个示例程序运行的过程中，我们可以发现，在 forward 跳转并且传递参数的过程中，浏览器地址栏中的地址始终是不变的，传递的参数也不会浏览器的地址栏中显示，这也是 forward 动作指令与 HTML 中 `<a>`/`</a>` 超链接的另一个区别。

#### 4.4.4 plugin 动作指令

`<jsp:plugin>` 元素用于在浏览器中播放或显示一个对象（典型的的就是 applet 和 bean），而这种显示需要在浏览器的 java 插件。当 jsp 文件被编译，送往浏览器时，`<jsp:plugin>` 元素将会根据浏览器的版本替换成 `<object>` 或者 `<embed>` 元素。

#### 4.4.5 useBean 动作指令

useBean 动作指令可以在 JSP 中引用 JavaBean，这个动作指令在实际开发过程中经常会用到。在第六章 JavaBean 的讲解过程中将对这个动作指令做详细的介绍。在这里我们仅仅知道其基本用法即可，而且在这里不在用示例程序说明。

useBean 的使用格式如下。

```
<jsp:useBean id=" " class=" " scope=" "></jsp:useBean>
```

其中 id 为所用到的 JavaBean 的实例对象名称，class 是 JavaBean 对应类的包路径，包括包名和类名。scope 是这个 JavaBean 的有效范围，共有 page、request、session、application 四个值可以选择。

#### 4.4.6 setProperty 动作指令

setProperty 一般情况下是和 JavaBean 配合使用的，用来给 JavaBean 的实例对象进行赋值操作，setProperty 的基本方法有以下两种。

```
<jsp:setProperty name="JavaBean 的实例名称" property="属性名" value="属性值"/>
```

上面这种方法是 setProperty 动作指令最基本的用法，用来给 JavaBean 实例对象的某一个属性赋值。

```
<jsp:setProperty name="JavaBean 的实例名称" property="*" />
```

上面这种 JavaBean 的赋值方法也是经常用到的，这个方法可以给 JavaBean 实例对象的所有属性进行赋值，其中在 JSP 的页面请求中有对应与所有属性的输入，这些输入的名称与 JavaBean 的属性名一一对应，所以 JSP 引擎可以根据名称给对应的属性进行赋值。

setProperty 这个动作指令的具体使用方法将在第六章中详细介绍，在这里仅仅知道基本的使用形式即可。

#### 4.4.7 getProperty 动作指令

getProperty 一般情况下也是和 JavaBean 配合使用的，用来取出 JavaBean 实例对象的属性值。这个动作指令的基本使用方法如下。

```
<jsp:getProperty name="JavaBean 的实例名称" property="属性名" value="属性值"/>
```

上面这行代码就可以取出 JavaBean 实例对象的一个属性值，至于 getProperty 这个动作指令更详细的使用方法，在第六章 JavaBean 的讲解中将有详细的实例介绍。

### 4.5 JSP 指令

JSP 的指令虽然没有动作指令那么丰富，但是其作用却是不容忽视的，例如 page 指令，在设置显示编码、引入类的包路径、设置错误页面等方面都是必不可少的。在接下来的章节中将介绍 JSP 的两个指令标签。

#### 4.5.1 page 指令

page 指令可以用来定义 JSP 页面的全局属性。例如编码、错误页面等。page 指令的属性很多，下面来具体介绍它的各个属性。

##### 1. language

这个属性用来设置页面所使用的语言，对于 JSP 来说当然要选 Java。具体设置方法如下。

```
<%@ page language="java" %>
```

##### 2. import

import 用来引入用到的包或者是类，这个属性的设置方法如下。

```
<%@ page import="java.util.*" %>
```

在上面这个行代码中，以引入 java.util.\* 包为例展示了在 JSP 中引入包或者类的方法。

##### 3. contentType

这个属性设置了 JSP 页面的 MIME 类型，对于还有中文的 JSP 页面可以按照下面这种方式设置。

```
<%@ page contentType="text/html; charset=gb2312" %>
```

经过这样的设置，页面显示编码方案设置为 gb2312，这种编码格式可以正确显示中文。

##### 4. session

设置在 JSP 页面中是否可以使用 session 对象，默认为 true。

##### 5. buffer

用来设置 out 对象缓冲区的大小，可以选择 none、也可以设置为指定的大小，单位为 KB。

##### 6. autoFlush

当在 JSP 页面设置了可以使用缓冲区的时候，才可以设置这个属性，这个属性设置为 true 的时候，缓冲区一旦满了就会自动刷新。如果设置为 false，缓冲区就满了以后就会报溢出错误。

##### 7. isThreadSafe

设置当前 JSP 页面是否是线程安全的，默认是 true，可以同时相应多个请求。

## 8. info

此属性设置当前 JSP 页面的描述信息，不常用。

## 9. errorPage

此属性设置错误处理页面，当页面出错的时候可以跳转到这个错误处理页面。

## 10. isErrorPage

设置当前页面是否为错误处理页面，默认为 false。

### 4.5.2 include 指令

include 指令可以在当前的 JSP 页面中包含一个文件，从而和当前页面组成一个整体的文件。这中包  
含仅仅是静态包含。

include 指令的具体使用方法实例如下。

```
//-----文件名: Include.jsp-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>  
<html>  
  <head>  
    <title>include 指令使用示例程序</title>  
  </head>  
  <body>  
    <font size="2">  
      这是 include 指令的使用示例程序。<br>  
      <%@ include file="inc.txt"%>  
    </font>  
  </body>  
</html>
```

在这个页面中静态引入了一个文件 inc.txt，其中 inc.txt 的内容如下。

```
//-----文件名: inc.txt-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>  
这是一个简单文本文件
```

上面这个程序的运行结果如图 4.14 所示。



图 4.14 include 指令实例程序运行效果

## 4.6 JSP 内置对象简介

JSP 内置对象即无需声名就可以直接使用的对象实例，在实际的开发过程中，比较常用的 JSP 内置

对象有 request、response、session、out、application 等，在接下来的章节中将详细介绍这几个 JSP 内置对象的使用方法。JSP 其他的几个内置对象在实际的开发中并不十分常用，在这里不做具体介绍。

## 4.7 request 对象

request 对象代表这从用户发送过来的请求，从这个对象中间可以取出客户端用户提交的数据或者是参数。这个对象只有接受用户请求的页面才可以访问。

### 4.7.1 request 对象使用场合

如果要与用户的互动，必须要知道用户的需求，然后根据这个需求生成用户期望看到的结果。这样才能实现与用户的互动。在 Web 应用中，用户的需求就抽象成一个 request 对象，这个对象中间包括用户所有的请求数据，例如通过表单提交的表单数据，或者是通过 URL 等方式传递的参数，这些就是用户的需求。request 正是用来收集类似这些用户的输入数据和参数。

同时，request 对象中还包括一些服务器的信息，例如端口、真实路径、访问协议等信息，通过 request 对象可以取得服务器的这些参数。

### 4.7.2 request 对象主要方法

request 对象的方法非常多，在这里我们只介绍其中最常用的几种方法，其他方法可以参考相关类库的介绍。

#### 1. getAttribute(String name)

这个方法可以取出指定名称的这个属性的值，这个属性可以用 setAttribute (String name,Object o) 这个方法进行赋值，如果没有对这个属性赋值的话取值的操作返回 null。

#### 2. getContextPath ()

这个方法可以获取的服务器上下文路径。

#### 3. getCookies ()

这个方法可以取出客户端的 Cookies。

#### 4. getHeader (String name)

这个方法可以取得指定名称的 HTTP 报头的属性值。

#### 5. getParameter (String name)

这个方法可以取出客户端提交到服务器的参数。

#### 6. getServerName ()

这个方法可以取得服务器的名称

#### 7. getServerPort ()

这个方法可以取得服务器的访问端口。

#### 8. setAttribute (String name,Object o)

这个方法对指定名称的属性进行赋值。

### 9. removeAttribute (String name)

这个方法可以移除指定名称的一个属性。

### 10. getRemoteAddr ()

这个方法返回客户端机器的 IP 地址。

下面是这些方法的调用实例。

```
//-----文件名: Request.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>request 主要方法调用示例</title>
  </head>
  <body>
    <font size="2">
      request 主要方法调用示例: <br>
    <%
      request.setAttribute("attr","Hello!");
      out.println("attr 属性的值为: "+request.getAttribute("attr")+"<br>");
      out.println("上下文路径为: "+request.getContextPath()+"<br>");
      out.println("Cookies:"+request.getCookies()+"<br>");
      out.println("Host:"+request.getHeader("Host")+"<br>");
      out.println("ServerName:"+request.getServerName()+"<br>");
      out.println("ServerPort:"+request.getServerPort()+"<br>");
      out.println("RemoteAddr:"+request.getRemoteAddr()+"<br>");
      request.removeAttribute("attr");
      out.println("属性移除操作以后 attr 属性的值为: "+request.getAttribute("attr")+"<br>");
    %>
    </font>
  </body>
</html>
```

在这个程序中对 request 常用的方法进行展示，其他方法的使用和这些基本的方法类似。可以参考这些例子进行尝试。上面这个示例程序的运行结果如图 4.15 所示。



图 4.15 request 对象主要方法调用示例运行效果



### 4.7.3 request 对象使用示例

#### 1. 使用 request 对象取得表单数据

request 获取用户数据的一个主要方式就是获取表单数据，下面是一个简单的表单，在这里为了方便说明，我们在把表单提交到这个页面自身，并在这个页面中取出提交的表单的数据。

```
//-----文件名: Form.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>request 获取表单数据示例</title>
  </head>
  <body>
    <font size="2">
      下面是表单内容:
      <form action="Form.jsp" method="post">
        用户名: <input type="text" name="userName" size="10"/>
        密 码: <input type="password" name="password" size="10"/>
        <input type="submit" value="提交">
      </form>
      下面是表单提交以后用 request 取到的表单数据: <br>
      <%
        out.println("表单输入 userName 的值:"+request.getParameter("userName")+"<br>");
        out.println("表单输入 password 的值:"+request.getParameter("password")+"<br>");
      %>
    </font>
  </body>
</html>
```

在上面这个程序中，当然可以把表单提交给其他的页面，那样只需把接受参数的方法放在对应的页面即可。在这里只是为了方便展示问题的才提交给页面自身。上面这个程序的运行结果如图 4.16 所示。



图 4.16 request 获取表单数据示例程序运行效果

#### 2. 使用 request 对象取得页面传递的参数

在实际的开发过程中，经常遇到这样的情景，在页面跳转中需要添加相对应的参数。例如在在论坛

中需要参看一条帖子的时候，单击链接会进入一个新页面，在新页面中显示帖子的内容，这个操作过程即需要把帖子的编号传到新页面。

对与参数的处理我们还是采用 request 来处理，下面的例子就展示了如何使用 request 对象来获取参数的值。

```
//-----文件名: URL.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>request 对象取得页面传递参数示例</title>
  </head>
  <%
    String param = request.getParameter("param");
  %>
  <body>
    <font size="2">
      <a href="URL.jsp?param=Hello">请单击这个链接</a><br>
      你提交的参数为: <%=param%>
    </font>
  </body>
</html>
```

在这个页面中我们同样把参数传递给这个页面自身，道理和前面表单的处理是一样的。读者也可以尝试参考这个例子把参数传递到一个不同的页面。这个程序运行的结果如图 4.17 所示。



图 4.17 request 对象取得页面传递参数示例程序运行效果

## 4.8 response 对象

response 对象是服务器端向客户端返回的数据，从这个对象中间可以取出一部分与服务器互动的数据和信息。只有接受这个对象的页面才可以访问这个对象。

### 4.8.1 response 对象使用场合

既然用户可以对服务器发出请求，服务器就需要对用户的请求做出反应。这里服务器就可以使用 response 对象向用户发送数据。response 是对应 request 的一个对象。

如果需要获取服务器返回的处理信息，就可以对 response 进行操作，同时当服务器需要再客户端进行某些操作的时候也需要用到 response 对象，例如服务器要在客户端生成 Cookies，那么这时候 response

对象就是一个很好的选择。

## 4.8.2 response 对象主要方法

response 的方法也很多，但是常用的也就其中的几个，下面介绍比较常用的几个方法。

### 1. addCookie (Cookie cookie)

这个方法可以添加一个 Cookie 对象，用来保存客户端的用户信息。

### 2. containsHeader (String name)

这个方法判断指定的头信息是否存在。

### 3. encodeRedirectURL (String url)

这个方法可以对 URL 进行进行编码。

### 4. encodeURL (String url)

这个方法可以对 URL 进行进行编码。

### 5. flushBuffer ()

这个方法可以清空缓存的内容。

### 6. sendError (int error)

这个放发可以向客户端浏览器发送错误代码。如 500 为服务器内部错误。

### 7. sendRedirect (String location)

这个方法可以把当前页面转发到其他的页面，实现页面的跳转。

## 4.8.3 response 对象使用示例

response 的用法很多，在这里我们用 response 来实现一个页面的重定向，具体的代码如下。

```
//-----文件名: Redirect.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html;charset=gb2312"%>
<html>
  <head>
    <title>response 对象页面重定向示例</title>
  </head>
  <body>
    <%
      response.sendRedirect("http://www.sohu.com");
    %>
  </body>
</html>
```

上面这个页面运行的时候直接重定向到 [www.sohu.com](http://www.sohu.com) 的首页。

## 4.9 session 对象

session 对象维护着客户端用户和服务器端的状态，从这个对象中间可以取出用户和服务器交互的过程中的数据和信息。这个对象在用户关闭浏览器离开 Web 应用之前一直有效。

### 4.9.1 session 对象使用场合

session 对象中保存的内容是用户与服务器整个交互过程中的信息，如果是想在整个交互的过程中都可以访问到的信息，就可以选择存放在 session 对象中。

例如在用户登录的过程中，可以在 session 中记录用户的登录状态，这样用户就不必在每个页面都重新登录，只要用户没有离开当前的 Web 应用系统，就可以一直保存登录的状态。

### 4.9.2 session 对象主要方法

session 所提供的方法并没有前面几个内置对象那么多，但是基本都是非常常用的。

#### 1. `getAttribute (String name)`

这个方法可以获取指定属性的值。

#### 2. `getCreationTime ()`

这个方法可以获取 session 对象创建的时间。

#### 3. `getLastAccessedTime ()`

这个方法可以获取 session 对象上次被访问的时间。

#### 4. `invalidate ()`

这个方法可以失 session 对象失效。

#### 5. `removeAttribute (String name)`

这个方法可以移除指定的属性。

#### 6. `setAttribute (String name, Object value)`

这个方法可以给指定名称的属性赋值。

### 4.9.3 session 对象使用示例

在这里我们模拟一个简单的用户登录动作，在这个示例程序中，我们不对提交的登录信息做具体的验证，只要用户名和密码都不为空就可以登录系统，这样处理只是为了方便展示 session 的使用方法，在具体的开发中必须要对登录信息进行验证的。

这个登录注册的示例程序代码如下。

```
//-----文件名: Login.jsp-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>  
<html>  
  <head>  
    <title>用户登录界面</title>
```

```

</head>
<body>
    <font size="2">
        <form action="LoginCheck.jsp" method="post">
            用户名: <input type="text" name="userName" size="10"/>
            密 码: <input type="password" name="password" size="10"/>
            <input type="submit" value="提交">
        </form>
    </font>
</body>
</html>

```

上面这个 JSP 页面向 LoginCheck.jsp 提交了一个登录表单，表单中有用户名和密码。下面是 LoginCheck.jsp 的内容。

```

//-----文件名: LoginCheck.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
    <head>
        <title>用户登录验证页面</title>
    </head>
    <%
        String userName = request.getParameter("userName");
        String password = request.getParameter("password");

        if(userName.length()>0&&password.length()>0)
        {
            session.setAttribute("status","Login");
            response.sendRedirect("Main.jsp");
        }else
            response.sendRedirect("Login.jsp");

    %>
    <body>
    </body>
</html>

```

在上面这个页面中，从表单中取出用户名和密码，如果用户名和密码都不为空就允许登录，否则就重定向到登录页面，让用户重新登录。如果用户登录成功，就在 session 中设置一个 status 属性，然后把用户重定向到系统的主页面。在主页面中可以访问 session 中的值。下面是 Main.jsp 的具体代码。

```

//-----文件名: Main.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
    <head>
        <title>系统主界面</title>
    </head>
    <body>
        <font size="2">
            <%
                Object status = session.getAttribute("status");
                if(session.getAttribute("status")!=null)
                    out.print("用户已经登录! ");
            %>
        </font>
    </body>
</html>

```

```
        else
            response.sendRedirect("Login.jsp");
    %>
</font>
</body>
</html>
```

在上面这个页面中，对用户的状态进行判断，如果从 `session` 中可以取出对应的属性值就说明用户已经登录，如果没有取到指定属性的值，说明用户没有登录，这时就重定向到登录页面，让用户重新登录。其中 `session` 的值在用户离开系统之前的任何页面都可以访问。

注意：当用户打开系统的一个网页，服务器就生成一个 `session` 对象，在用户离开之前，这个对象一直伴随用户。在这个过程中可以在这个 `session` 中间存储用户的信息，这些信息在所有页面中都是可以访问的。只有当用户关闭浏览器的时候，这个 `session` 对象就会被注销。

## 4.10 out 对象

这个对象是在 Web 应用开发过程中使用最多的一个对象，其功能就是动态的向 JSP 页面输出字符流，从而把动态的内容转化成 HTML 形式来展示。这个对象在任何 JSP 页面中都可以任意访问。

### 4.10.1 out 对象使用场合

`out` 对象的功能就是向 JSP 页面输出数据信息。所以当有动态信息要展示给用户的时候就要用到 `out` 对象。在前面的很多示例中已经多次用到这个对象，读者从中可以很清楚的看到，`out` 对象就是用来输入动态内容信息的。

### 4.10.2 out 对象主要方法

在这里只介绍 `out` 对象最常用的方法。

#### 1. `clear ()`

这个方法可以清除缓冲区的数据，但是仅仅是清除，并不向用户输出。

#### 2. `clearBuffer ()`

这个方法可以清除缓冲区的数据，同时把这些数据向用户输出。

#### 3. `close ()`

这个方法可以关闭 `out` 输出流。

#### 4. `flush ()`

这个方法可以输出缓冲区的内容。

#### 5. `isAutoFlush ()`

这个方法可以判断是否为自动刷新。

#### 6. `print (String str)`

这个方法在 out 对象中最常用的，可以向 JSP 页面输出数据，其中数据格式可以是 int、boolean、Object 等。用法都是类似的。

### 4.10.3 out 对象使用示例

out 对象在前面的示例中已经多次使用到，在这里就不再针对这个对象举例说明。

## 4.11 application 对象

application 对象保存着整个 Web 应用运行期间的全局数据和信息，从 Web 应用开始运行开始，这个对象就会被创建，在整个 Web 应用运行期间可以在任何 JSP 页面中访问这个对象。

### 4.11.1 application 对象使用场合

application 中保存的信息可以在整个应用的任何地方访问，这个 session 对象类似，但和 session 对象还是有所区别的。只要 Web 应用还在正常运行，application 对象就可以访问，而 session 对象在用户离开系统就被注销。

所以如果要保存在真个 Web 应用运行期间都可以访问的数据，这时候就要用到 application 这个对象。

### 4.11.2 application 对象主要方法

下面介绍 application 对象的最常用的主要方法。

#### 1. `getAttribute (String name)`

这个方法可以获取指定属性的值。

#### 2. `getServerInfo ()`

这个方法可以取得服务器的信息。取出的值是类似 Apache Tomcat/5.0.28 这样的形式。

#### 3. `removeAttribute (String name)`

这个方法可以移除指定的属性。

#### 4. `setAttribute (String name, Object o)`

这个方法可以给一个指定名称的属性赋值。

### 4.11.3 application 对象使用示例

在这里我们要实现一个简单的计数器，这个计数器就是利用 application 对象来储存计数器的值，用来统计服务器开始运行以来的访问量。

```
//-----文件名: Counter.jsp-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
```

```

<html>
<head>
<title>利用 application 对象实现的计数器示例</title>
</head>
<body>
<font size="2">
<%
    int count=0;
    if(application.getAttribute("count")==null)
    {
        count = count + 1;
        application.setAttribute("count",count);
    }else
    {
        count = Integer.parseInt(application.getAttribute("count").toString());
        count = count + 1;
        application.setAttribute("count",count);
    }

    out.println("您是本系统的第"+count+"访问者！");
%>
</font>
</body>
</html>

```

在上面这个程序中，当第一次访问时候把 count 的初始值设置为 1，以后每次刷新的时候累加 count 的值。上面这个计数器的运行过程中，多个页面之间共享计数器的值，而且关闭浏览器然后再新开口的时候，以前计数器的值还保留。这就是 application 与 session 最大的区别。

## 4.12 JSP 中文问题完全解决方案

在 Java 开发中，中文乱码是一个最让人头疼的问题，如果不对中文做特殊的编码处理，这些中文字符就会变成乱码或者是问号。而在不同情况下对这些乱码的处理方法又各不相同，这就导致很多初学者对中文乱码问题束手无策。其实造成这种问题的根本原因是 Java 中才用的默认编码方式是 Unicode，而中文的编码方式一般是 GB2312，因为编码格式的不同，导致在中文不能正常显示。

对于中文乱码问题，在不同的 JDK 版本和不同的应用服务器中的处理方法是不同的。但是其本质上都是一样的，就是把中文字符转化成合适的编码方式，或者是把在显示中文的环境中声名采用 GB2312 的编码。统一编码方案之后自然可以正常显示。

在接下来的章节中，将对 JSP 开发过程中的中文乱码问题进行详细的介绍，对各种乱码提供对应的解决方法，在各种编码方案中，UTF-8、GBK、GB2312 都是支持中文显示的，在本章中我们统一采用 GB2312 的编码格式来支持中文。

### 4.12.1 JSP 页面中文乱码

在 JSP 页面中，中文显示乱码有两种情况：一种是 HTML 中的中文乱码，另一中是在 JSP 中动态



输出的中文乱码。下面来看这样一个 JSP 程序。

```
//-----文件名: PageCharset.jsp-----
<%@ page language="java" import="java.util.*"%>
<html>
  <head>
    <title>中文显示示例</title>
  </head>
  <body>
    这是一个中文显示示例:
    <%
      out.print("这里是用 JSP 输出的中文。");
    %>
  </body>
</html>
```

上面这个 JSP 程序看起来好像是在页面显示几句中文，而且标题栏也是中文，但是在浏览器中的运行结果却如图 4.18 所示。

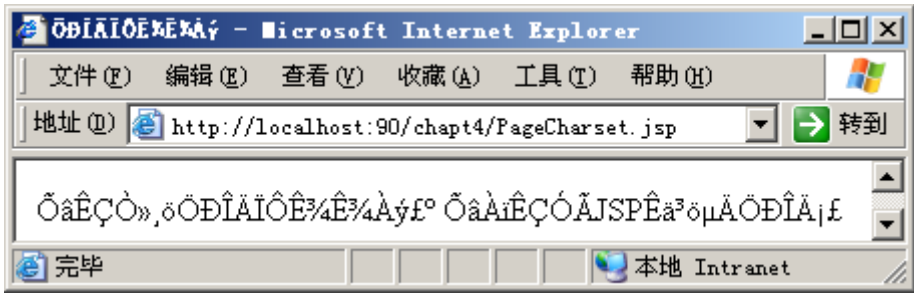


图 4.18 JSP 页面运行乱码效果

显然，图 4.18 中所示并非我们预期的效果，在我们 JSP 源代码中清清楚楚看到的是中文，在这里为什么就成了乱码，造成这种原因的可能就是出在浏览器端的字符显示设置上，所以我们可以把上面的程序进行如下改进。

```
<%@ page language="java" import="java.util.*"%>
```

在上面这行代码中就是对整个页面的起作用的 page 指令，我们可以在这个指令中间设置支持中文显示的编码方式，具体修改方法如下所示。

```
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
```

上面这行代码中，向 page 指令中添加了页面内容和显示方式的设置，其中采用 gb2312 的编码方式来显示 HTML 页面的内容，所以中文可以正常显示，改进以后的程序运行效果如图 4.19 所示。



图 4.19 JSP 页面正常运行效果

## 4.12.2 URL 传递参数中文乱码

在一般情况下，可以用类似 `http://localhost:8080/chapt4/URLCharset.jsp?param='中文'` 这种形式来传递参数，而且 HTML 在处理表单的时候，当表单的 method 采用 get 方法的时候，传递参数的形式与 URL 传递参数的形式基本一样。下面就是采用 URL 传递参数的一个示例程序。

```
//-----文件名: URLCharset.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>URL 传递参数中文处理示例</title>
  </head>
  <%
    String param = request.getParameter("param");
  %>
  <body>
    <a href="URLCharset.jsp?param='中文'">请单击这个链接</a><br>
    你提交的参数为: <%=param%>
  </body>
</html>
```

上面这个 JSP 程序的功能就是通过一个 URL 链接向自身传递一个参数，这个参数是中文字符串，这个程序的运行效果如图 4.20 所示。

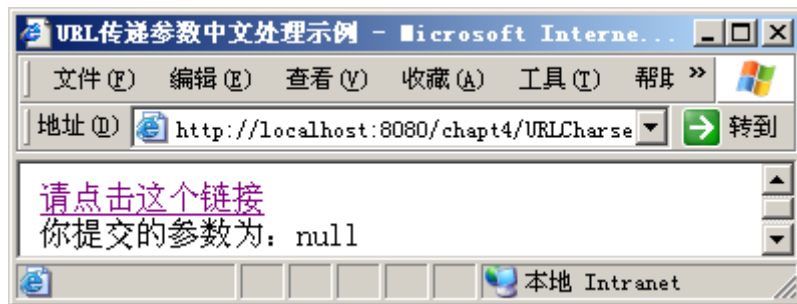


图 4.20 URL 传递中文参数初始界面

在图 4.21 所示的界面中单击链接，可以得到如图 4.21 所示的运行效果。

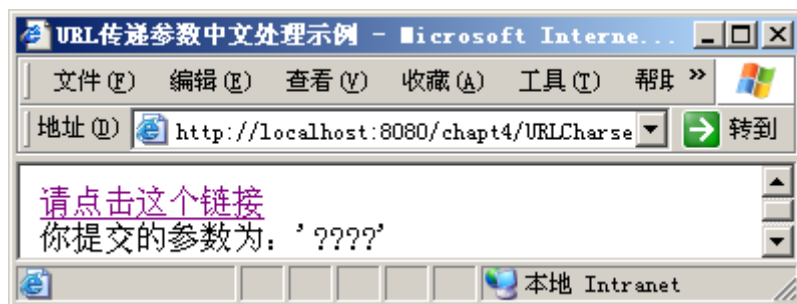


图 4.21 URL 传递中文参数出现乱码

对于 URL 传递中文参数乱码这个问题，其处理方法比较独特，仅仅转换这个中文字符串的编码，或者设施 JSP 页面显示编码都是不能解决问题的。在这里需要多 Tomcat 服务器的配置文件进行修改才可以解决问题。在这里需要修改 Tomcat 的 conf 目录下的 server.xml 配置文件。具体修改方法如下。

```
<Connector port="8080" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
```

```
enableLookups="false" redirectPort="8443" acceptCount="100"
debug="0" connectionTimeout="20000"
disableUploadTimeout="true" />
```

在上面的这段代码中间添加 URI 编码的设置即可，即在 `port="8080"` 后面添加 URI 编码设置 `URIEncoding="gb2312"` 即可。重新启动 Tomcat 服务器，可以得到如图 4.22 所示的界面。



图 4.22 URL 传递中文参数正常运行效果

注意: URL 中文乱码的处理方法比较特殊，后续章节中的转码和过滤器的方法对这个问题都不起作用，这个问题的解决只有更改 Tomcat 的配置文件。

### 4.12.3 表单提交中文乱码

对于表单中提交的数据，可以使用 `request.getParameter("")` 的方法获取。但是当表单中如果出现中文数据的时候就会出现乱码。请看下面这个简单的表单。

```
//-----文件名: FormCharset.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
<head>
<title>Form 中文处理示例</title>
</head>
<body>
<font size="2">
下面是表单内容:
<form action="AcceptFormCharset.jsp" method="post">
    用户名: <input type="text" name="userName" size="10"/>
    密 码: <input type="password" name="password" size="10"/>
    <input type="submit" value="提交">
</form>
</font>
</body>
</html>
```

在上面这个表单中，向 `AcceptFormCharset.jsp` 这个页面提交两项数据，下面是 `AcceptFormCharset.jsp` 的内容。

```
//-----文件名: AcceptFormCharset.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
<head>
<title>Form 中文处理示例</title>
</head>
```

```

<body>
<font size="2">
  下面是表单提交以后用 request 取到的表单数据: <br>
  <%
    out.println("表单输入 userName 的值:"+request.getParameter("userName")+"<br>");
    out.println("表单输入 password 的值:"+request.getParameter("password")+"<br>");
  %>
</font>
</body>
</html>

```

在上面这个程序中，如果表单中输入的内容没有中文，则可以正常显示，当输入的数据中有中文的时候，假如有如图 4.23 所示的数据输入。



图 4.23 用户名为中文的表单输入

在上面这个表单中，用户名的输入为中文字符，当提交这个表单的时候，就会得到如图 4.24 所示的结果。

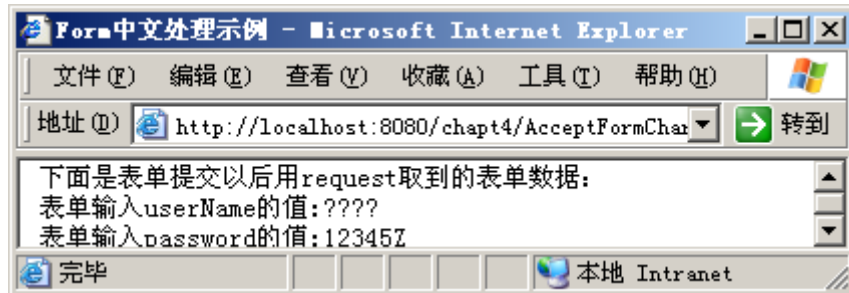


图 4.24 表单输入中文乱码

在图 4.25 所示的界面中可以清楚看到，输入的中文用户名在用 request 取出以后全部变成了乱码，造成这个问题的原因是：在 Tomcat 中，对于以 POST 方法提交的表单采用的默认编码为 ISO-8859-1，而这种编码格式不支持中文字符。对于这个问题，可以采用转换编码格式的方法来解决，现在对 AcceptFormCharset.jsp 改造如下。

```

<%
  out.println("表单输入 userName 的值:"+request.getParameter("userName")+"<br>");
  out.println("表单输入 password 的值:"+request.getParameter("password")+"<br>");
%>

```

上面这两行是对表单数据的处理，可以在这里进行数据的转码，具体的转换编码的方法如下。

```

<%
  String userName = request.getParameter("userName");
  String password = request.getParameter("password");
  out.println("表单输入 userName 的值:"+new String(userName.getBytes("ISO-8859-1"),"gb2312")+"<br>");

```

```
out.println("表单输入 password 的值:"+new String(password.getBytes("ISO-8859-1"),"gb2312")+"<br>");  
%>
```

经过这样的转换编码以后，所有的中文输入都可以用 request 对象正常取出。在上面这个程序中，new String(userName.getBytes("ISO-8859-1"),"gb2312")这句代码是转换编码格式的关键，先从 ISO-8859-1 格式的字符串中取出字节内容，然后再用 gb2312 的编码格式重新构造一个新的字符串。这样就可以支持中文的表单输入的正常取值和显示。改进以后的程序运行结果如图 4.25 所示。

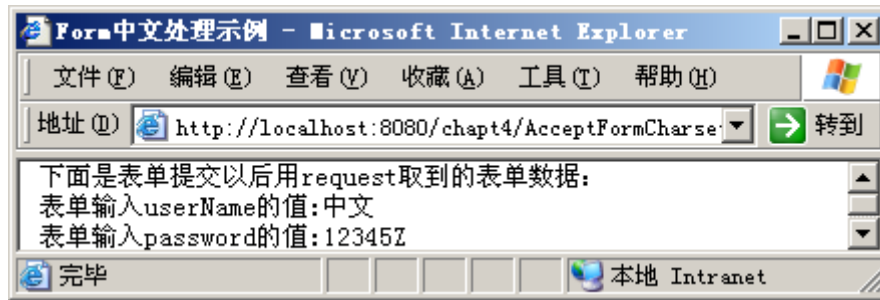


图 4.25 表单中文输入正常取值并显示

注意：new String(userName.getBytes("ISO-8859-1"),"gb2312")这个转码的方法中，是从编码格式为 ISO-8859-1 的字符串中取出字节内容，然后转换成 gb2312，如果原来字符的编码格式不是 ISO-8859-1 的时候这个方法就失效。例如在下面的过滤器处理中，把所有的编码都改成了 gb2312，这个时候如果再使用 new String(userName.getBytes("ISO-8859-1"),"gb2312")这个方法进行转码就会发生错误。

经过上面的更改编码格式的处理，表单的中文输入乱码问题已经解决。但是同时暴露了另一个问题，在上面这个表单中，输入项只有两项，所以对每个输入项都进行编码的转化也不是很麻烦，但是试想一下，现在如果有一个很大的表单，表单的输入项有几十个之多，这个时候对每个输入项都进行转码就很麻烦了，同时，在一个应用系统中，表单的个数也是非常多的，那么对于每个表单都得做同样的转码处理，这样的重复工作是最让人头疼的。

在这种情况下，有一种比较简便的方法，那就是使用过滤器 filter，过滤器的原理和使用方法在介绍 servlet 的时候会做详细的介绍，在这里由于需要用它处理中文问题，提前知道基本的用法即可。

过滤器的基本原理就是对于每一个用户请求，都必须经过过滤器的处理才能继续发送到目的页面，在 JSP 中，以 POST 方式提交的表单在本质上就是封装在 request 对象中，而 request 对象是必须要经过过滤器的处理的，所以，对于表单的中文问题来说，可以在 filter 中对所有的请求进行编码格式的处理，这样就不必在每个表单中都做转码处理，节省大量的时间和精力。

下面的代码就是中文处理过滤器的具体代码。

```
//-----文件名: SetCharacterEncodingFilter.java-----  
import java.io.IOException;  
import javax.servlet.Filter;  
import javax.servlet.FilterChain;  
import javax.servlet.FilterConfig;  
import javax.servlet.ServletException;  
import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;  
import javax.servlet.UnavailableException;  
public class SetCharacterEncodingFilter implements Filter {  
  
    protected FilterConfig filterConfig;  
    protected String encodingName;
```

```

protected boolean enable;

public SetCharacterEncodingFilter() {
    this.encodingName = "gb2312";
    this.enable = false;
}

public void init(FilterConfig filterConfig) throws ServletException {
    this.filterConfig = filterConfig;
    loadConfigParams();
}

private void loadConfigParams() {
    this.encodingName = this.filterConfig.getInitParameter("encoding");
    String strIgnoreFlag = this.filterConfig.getInitParameter("enable");
    if (strIgnoreFlag.equalsIgnoreCase("true")) {
        this.enable = true;
    } else {
        this.enable = false;
    }
}

public void doFilter(ServletRequest request, ServletResponse response,
                    FilterChain chain) throws IOException, ServletException {
    if(this.enable) {
        request.setCharacterEncoding(this.encodingName);
    }
    chain.doFilter(request, response);
}

public void destroy() {
}
}

```

filter 本质上就是一个 servlet，如果这个 servlet 要起作用的话就需要在 web.xml 中进行配置，对于这个 filter 可以在 web.xml 中添加下面的内容。

```

<filter>
    <filter-name>SetCharacterEncoding</filter-name>
    <filter-class>
        SetCharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>gb2312</param-value>
    </init-param>
    <init-param>
        <param-name>enable</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>SetCharacterEncoding</filter-name>
    <url-pattern>/*</url-pattern>

```

```
</filter-mapping>
```

经过这样的处理，所有的请求都会被转换成 gb2312 的编码格式，这样表单中的中文输入就可以正常获取并显示，不必对每个输入项再做转换编码的处理。

注意：web.xml 是每个 Web 应用系统必需的一个配置文件，在这里可以配置应用系统中的一些基本信息，还可以配置应用系统中的 servlet 等。这个文件改动以后，需要重新启动 Tomcat 以后才能生效。

#### 4.12.4 数据库操作中文乱码

在建立数据库的时候，应该选择支持中文的编码格式，最好能和 JSP 页面的编码格式保持一致，这样就可以尽可能减少数据库操作的中文乱码问题。同时在 JDBC 连接数据库的时候可以使用类似下面这种形式的 URL。

```
jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=pubs;useUnicode=true;characterEncoding=gb2312
```

在上面这行代码中，指定了访问数据所使用的编码策略。在这里我们选择的数据库驱动程序为微软提供的官方驱动，连接的数据库为 SqlServer 2000 自带的 pubs 数据库，选择的编码为 gb2312。

上面所说的方法仅仅适用于创建数据库的时候已经选择支持中文的编码，但是当数据库已经创建，而且编码格式已经是 ISO-8859-1 的时候，这种情况下通过重新创建数据库显然是不显示的，尤其是当数据库中已经有大量数据的时候，这时候用上面介绍的方法可以正常向数据库中写入中文数据，但是读出的数据格式是 ISO-8859-1，如果有中文内容就会显示乱码。这时候只有在读取数据库的时候进行转码，使用的方法还是上一节中的转码方法，在这里把它整理成一个转码的函数。

```
public String encoder(String str) throws UnsupportedEncodingException
{
    String result = new String(str.getBytes("ISO-8859-1"), "gb2312");
    return result;
}
```

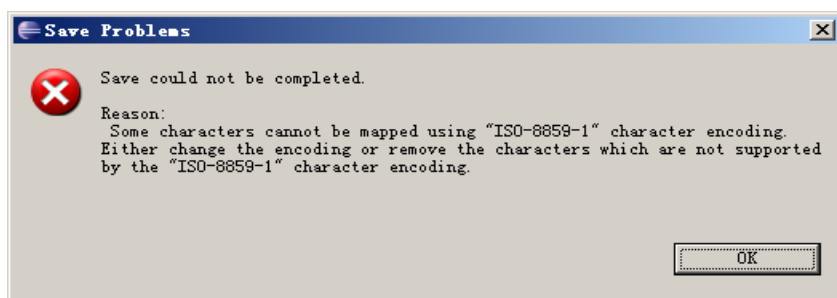
有了这个转码函数，在读取数据库的时候就可以使用 encoder(rs.getString("列名"))，这样取得中文字符串的内容就可以正常显示。

#### 4.12.5 Eclipse 开发工具中 JSP 文件中文不能保存

在 Eclipse 中，JSP 文件默认的编码格式为 ISO-8859-1，所以在 JSP 代码中间如果出现中文就不能保存，像下面这段 JSP 代码就不能保存。

```
<%@ page language="java" import="java.util.*" %>
<html>
<head>
<title>中文显示示例</title>
</head>
<body>
这是一个中文显示示例:
</body>
</html>
```

这段代码在保存的时候会有如图 4.26 所示的提示。



对与这个问题，只要在 JSP 页面中指明页面编码即可。

```
<%@ page language="java" import="java.util.*" %>
```

把上面这行代码添加支持中文的页面编码就能保存，具体代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
```

其中 `pageEncoding="gb2312"` 指明了 JSP 页面编码采用 `gb2312`, 这样就可以正常保存 JSP 的源文件。

#### 4.12.6 Eclipse 开发工具中中文显示乱码

在 Eclipse 中，由于默认的 JSP 编码格式为 ISO-8859-1，所以当打开由其他编辑器编辑的 JSP 页面就会出现乱码，下面这个 JSP 程序就是用 UltraEdit 编辑的页面，中文完全可以显示。

```
<%@ page language="java" import="java.util.*"%>
<html>
<head>
<title>中文显示示例</title>
</head>
<body>
这是一个中文显示示例:
<%
    out.print("这里是用 JSP 输出的中文。");
%>
</body>
</html>
```

但是在 Eclipse 中打开以后却成为下面这样。

```
<%@ page language="java" import="java.util.*"%>
<html>
<html>
    <head>
        <title>D??????"?"?"ay</title>
    </head>
    <body>
        ?a"??"???D?????"?"?"yjiêo
    <%
        out.print("?a"??"?"®?JSP"?3?|]??D???jê");
    %>
    </body>
</html>
```

在上面这个 JSP 中，所有的中文都变成乱码。造成这个问题的原因是两个编辑器保存源文件的编码



格式不同，在 UltraEdit 中可以支持中文，但是在 Eclipse 中对 JSP 文件的保存方式为 ISO-8859-1，而这种编码格式对中文不支持，所以就出现了乱码的问题。

(1) 对于这个问题的解决方法是更改 Eclipse 默认的编码方案。在 Eclipse 中选择“Window”|“Preferences”会弹出如图 4.27 所示的窗口。

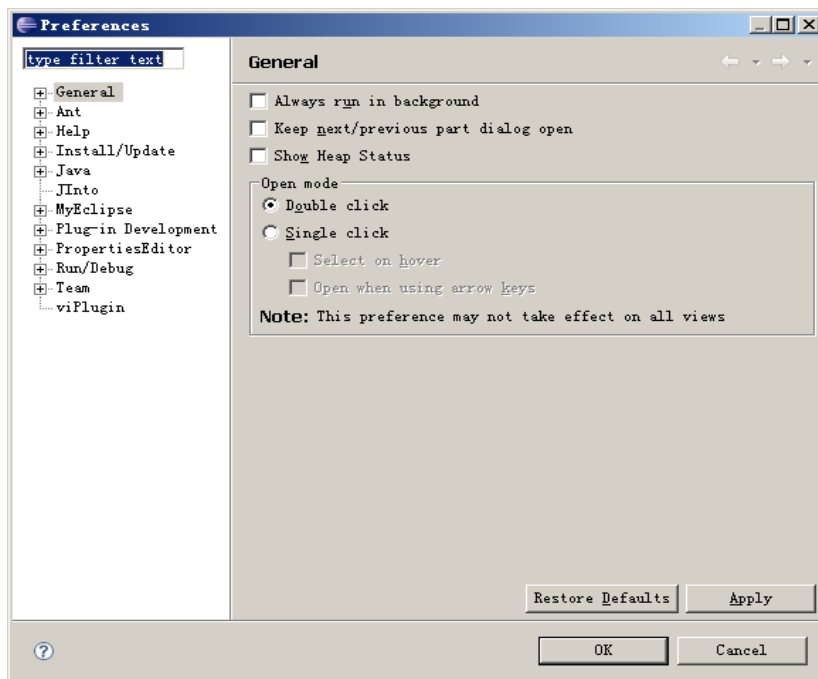


图 4.27 Eclipse 选项定制窗口

(2) 在上面这个窗口中选择“General”|“Content Types”会切换到如图 4.28 所示的界面。

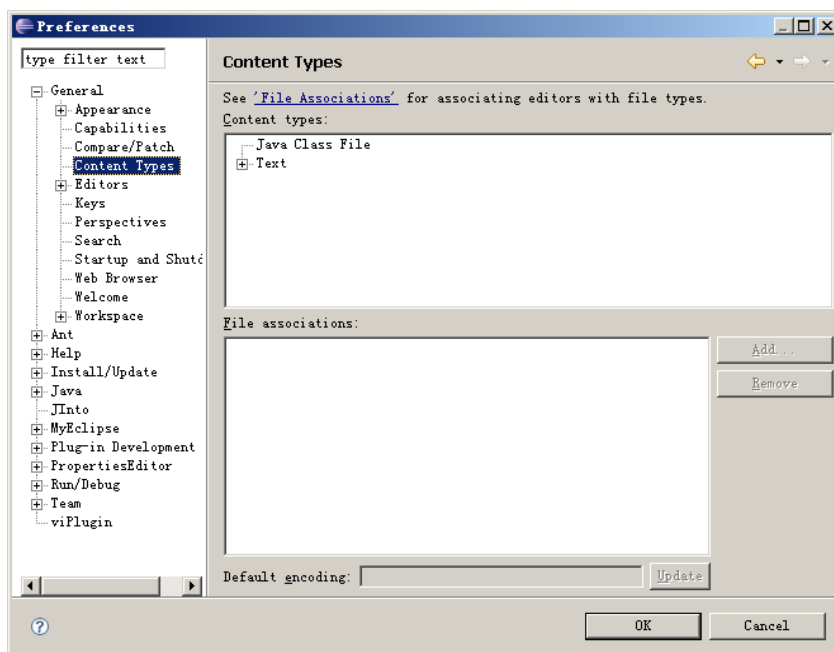


图 4.28 Eclipse 内容类型设置窗口

(3) 在上面的窗口的右上部分，选择“Text”|“JSP”在上面窗口的右下部分就会出现各种 JSP

文件的列表，选择\*.jsp，然后按照图 4.29 设置，然后单击“Update”更新设置，最后重新启动 Eclipse 中文就可以在 Eclipse 中正常显示。

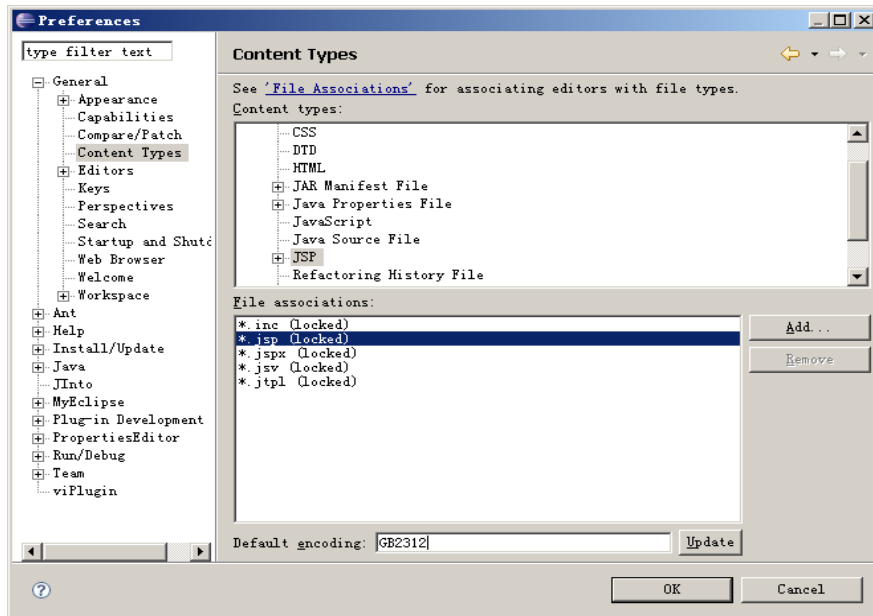


图 4.29 Eclipse 中 JSP 文件编码设置

#### 4.12.7 JSP 下载中文文件名乱码

在实现文件下载功能的时候，如果出现中文文件名，如果不进行特殊的处理，下载下来的中文文件名会变成乱码，在下载前，就需要对这个文件名进行处理，然后才能正常显示中文的文件名，具体处理方法示例如下。

```
//-----文件名: FileNameCharset.jsp-----
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%@ page import="java.io.*" %>
<%
    String fname = "中文";
    OutputStream os = response.getOutputStream();//取得输出流
    response.reset();//清空输出流

    //下面是对中文文件名的处理
    response.setCharacterEncoding("UTF-8");
    fname = java.net.URLEncoder.encode(fname, "UTF-8");
    response.setHeader("Content-Disposition", "attachment; filename="+ new String(fname.getBytes("UTF-8"),
"gb2312") + ".xls");
    response.setContentType("application/msexcel");//定义输出类型
    os.close();

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

经过上面的处理，下载下来的中文文件名称可以正常显示。

总之，在 JSP 中出现中文乱码的问题，在不同的服务器中有不同的原因，但最根本的原因都是相通的，那就是编码格式的冲突，只要知道使用的服务器采用的编码格式，然后再根据这个编码格式把中文字符串转码即可。读者要在上面的这些例子中理解中文处理的关键所在，要做到以不变应万变。

## 4.13 其他 JSP 开发技巧

### 4.13.1 自定义错误页面

在 JSP 中，如果出现在代码的错误，就会直接在页面上打印类似如图 4.30 所示的错误信息。

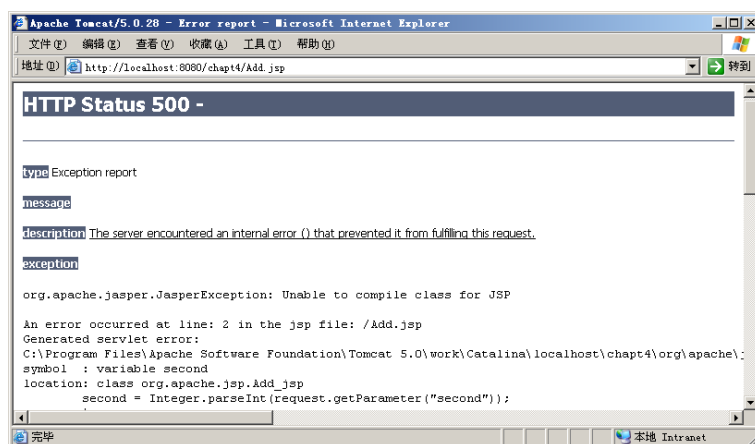


图 4.30 JSP 代码错误的页面

当用户看到这样的页面，肯定会不知所措，用户不会明白出现了什么问题，所以有必要在 JSP 页面中统一指定错误处理页面。例如在 JSP 页面中的 page 指令中，可以设置错误页面，具体设置方法如下。

```
<%@page language="java" contentType="text/html; charset=gb2312" errorPage="error.jsp"%>
```

上面这行代码中 errorPage="error.jsp"指定了这个页面出错以后会转向 error.jsp，我们可以在 error.jsp 中详细描述出错信息，让用户可以明白发生了什么情况。

上面介绍的这种方法当然可以很好的解决问题，但是由于在每个 JSP 页面都要添加错误页面的设置，当页面比较多时，工作量就相当大。所以我们提供一种更方便的解决办法。

在前面章节中介绍过，每个 Web 应用都需要一个 web.xml 文件，这个文件中是 Web 应用的基本配置信息，同样我们可以在这个文件中统一配置错误处理页面，具体操作方法就是在 web.xml 中添加如下内容。


```
<error-page>
  <error-code>500</error-code>
  <location>error.jsp</location>
</error-page>
```

上面这段代码指定了 500 错误时的错误处理页面，其中 500 错误即服务器内部错误，读者可以根据这个示例尝试配置其他错误类型的处理页面，例如 400 没有找到请求页面的错误处理页面。

### 4.13.2 在 MyEclipse 中快速部署项目

在 Web 应用开发的过程中，部署项目往往十分麻烦，虽然在后续章节中介绍的 Ant 可以非常方便的完成这个任务，但是 Ant 复杂的操作不适合初学者，在这里我们使用前面推荐的 MyEclipse 这个集成开发工具来部署项目。

MyEclipse 的安装在前面第二章中已经详细介绍，在这里直接开始介绍如何发布 Web 应用项目。要想发布部署一个项目，首要任务就是把 MyEclipse 和服务器 Tomcat 联系起来，下面就是详细的配置步骤。

- (1) 在 MyEclipse 的工具栏中单击这个服务器的标志，就会看到如图 4.31 所示的界面。
- (2) 在图 4.31 中可以清楚的看到，这时候还没有可用的服务器，所以需要进行服务器的设置。单击 “No enabled servers available.....” 这个区域，就会出现如图 4.32 所示的界面。

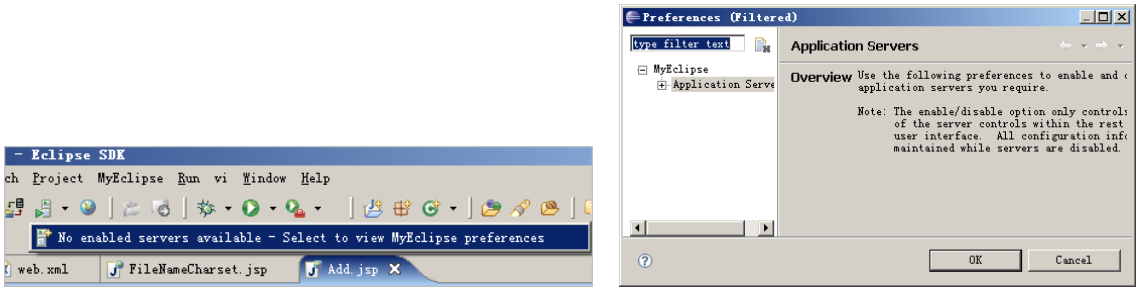


图 4.31 MyEclipse 中没有配置服务器情况下的界面      图 4.32 MyEclipse 服务器配置页面

- (3) 在如图 4.32 所示的界面中，选择 “Application Servers” | “Tomcat 5” 可以看到如图 4.33 所示的界面。

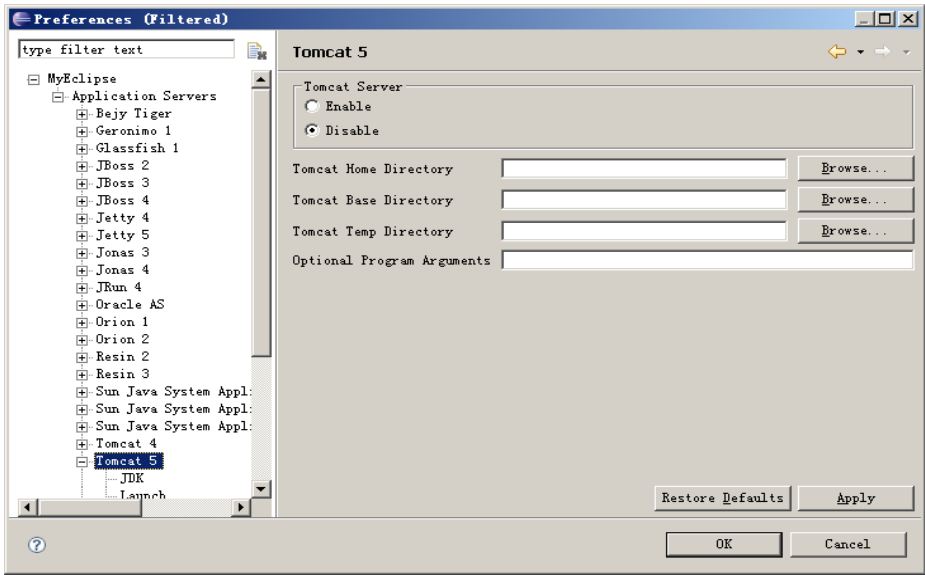


图 4.33 MyEclipse 中的 Tomcat 配置界面

- (4) 按照如图 4.34 所示的详细配置进行设置。

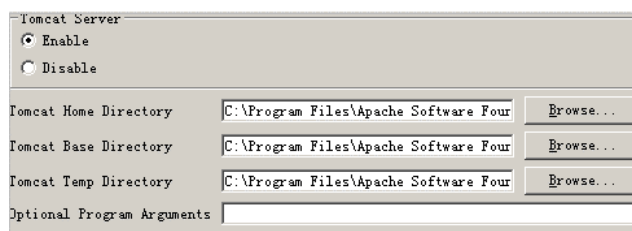



图 4.34 MyEclipse 中 Tomcat 的详细配置

在图 4.34 中，需要设置的只有两处，第一需要选择 Tomcat Server 下面的 Enable，这个选项可以激活对应的 Tomcat 服务器。第二处设置是选择 Tomcat Home Directory，即选择 Tomcat 在硬盘中的安装位置，这个属性选择以后，其他两个属性都可以自动生成。

(5) 按照图 4.34 设置完成之后，单击如图 4.33 所示的“Apply”，到这里在 MyEclipse 中配置 Tomcat 的设置就结束了。

### 4.13.3 测试配置是否成功

下面来测试下配置是否成功，在 MyEclipse 的工具栏中单击  图标，会出现如图 4.35 所示的界面。在图 4.35 所示的界面中，选择 Tomcat 5，会出现如图 4.36 所示的界面。

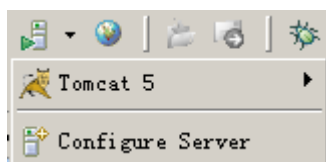


图 4.35 MyEclipse 中的 Tomcat 菜单

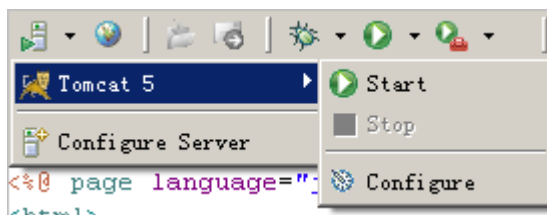


图 4.36 MyEclipse 中的 Tomcat 启动菜单

在如图 4.36 所示的界面中选择“Start”就可以启动 Tomcat 服务器，如果在 MyEclipse 的控制台看到如图 4.37 所示的界面就说明 Tomcat 在 MyEclipse 中配置成功。

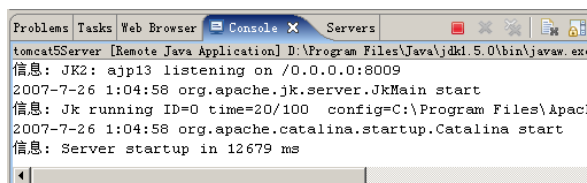



图 4.37 MyEclipse 中 Tomcat 的启动信息

通过上面一系列的设置，现在已经可以在 MyEclipse 开发环境中调用 Tomcat 服务器。接下来将介绍如何在 MyEclipse 中快速部署项目。

- (1) 在 MyEclipse 的工具栏中选择  这个图标，就会得到如图 4.38 所示的界面。
- (2) 在如图 4.38 所示的界面中选择“Add”，就可以得到如图 4.39 所示的界面。

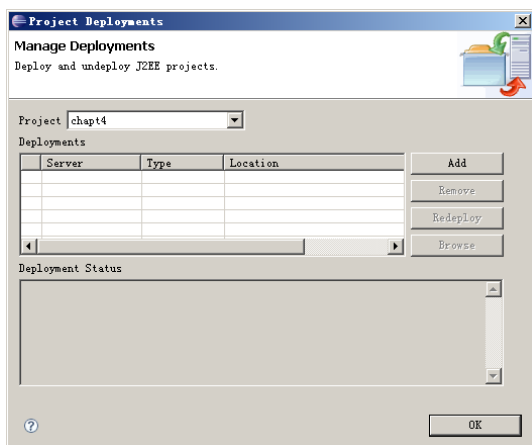


图 4.38 MyEclipse 中的项目部署界面

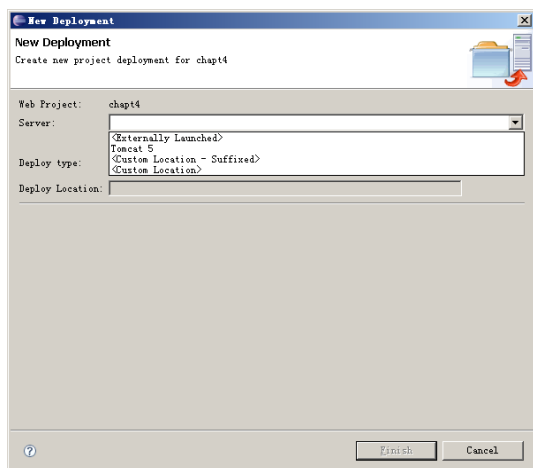


图 4.39 创建新的项目部署界面

在图 4.39 所示的界面中，从 Server 的下拉列表中选择 Tomcat 5，这个 Tomcat 服务器就在前面的设置中已经成功配置。选择 Tomcat 以后，上面这个界面中的 Finish 按钮就被激活，选择“Finish”就结束了部署，然后启动 Tomcat 就可以访问部署成功的应用。

注意：在使用 MyEclipse 快速部署项目的时候，默认把工程名称当作应用部署时候的名称。

### 4.13.3 在 MyEclipse 中调试 Web 应用程序

在 MyEclipse 中，对 JSP 页面进行调试也是非常方便的，如果需要调试 JSP 页面，只需要在 JSP 页面源代码的左侧双击鼠标左键，当出现如图 4.40 所示的小圆点的时候说明断点设置成功。

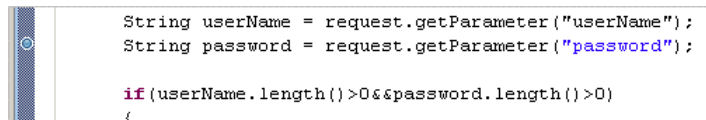


图 4.40 在 MyEclipse 中设置断点

断点设置成功以后，再次访问这个页面的时候，程序运行到断点位置就会停止，并且把 MyEclipse 切换到如图 4.41 所示的 Debug 视图。

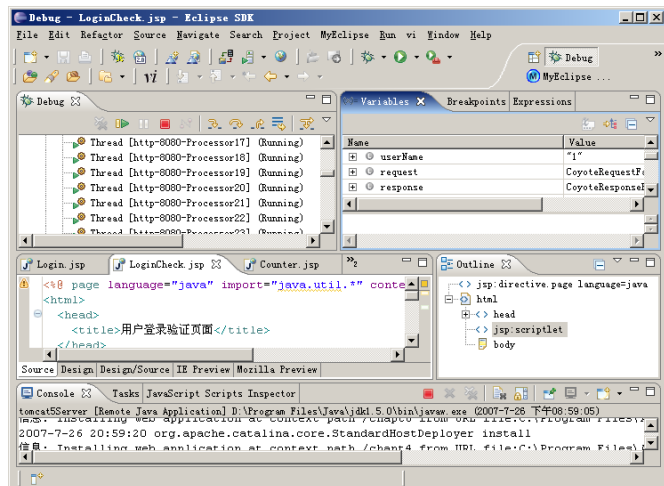


图 4.41 MyEclipse 调试界面

在上面这个界面中，右上角可以查看变量值，断点位置，表达式等信息，中间的两个窗口是源代码窗口源代码大纲，最下面的窗口是控制台信息输出窗口。

当然，我们可以在程序中设置多个断点，不仅可以为 JSP 程序设置断点，而且还可以给 Java 代码设置断点，MyEclipse 中 Debug 有几个常用的快捷键 F8 恢复，F5 进入方法，F6 单步执行跳过语句，F7 跳出方法。

#### 4.13.4 学习使用日志 Log4j

在一般情况下，我们可以把错误信息，或者是调试的信息输入到控制台，这时候要用 `System.out.println()`这个方法，这个方法固然可行，但是带来了很多不便，第一个问题就是在每个需要输入信息，以便调试维护的地方都需要这样的语句，这就增加了很多工作量，第二个问题是，当系统正式交付的时候需要把这些调试信息去掉，这个时候就需要一个一个找到这些输出语句，并且删除，这样带来的还是不必要的额外工作量。

在 JSP Web 开发中，有很多方便的日志工具可供选择，利用这些日志工具可以很方便的对系统中的错误信息进行管理，在这里我们选择使用 Log4j，Log4j 是目前 JSP 开发中使用最多的日志工具。Log4j 按照严重程度给日志风味 5 个等级：DEBUG（调试）、INFO（提示）、WARN（警告）、ERROR（错误）、FATAL（严重错误）。我们只需要简单的配置文件 `log4j.properties` 来就 Log4j 进行简单的配置，然后再把 `log4j.jar` 添加到项目中 WEB-INF/lib 文件夹中，就可以在程序中调用其强大的日志功能，下面是一个示例的 Log4j 的配置代码。

```
//-----文件名: log4j.properties-----  
log4j.rootLogger=ERROR,console,file  
log4j.appender.console=org.apache.log4j.ConsoleAppender  
log4j.appender.file=org.apache.log4j.FileAppender  
log4j.appender.file.File=system.log  
log4j.appender.console.layout=org.apache.log4j.SimpleLayout  
log4j.appender.file.layout=org.apache.log4j.SimpleLayout
```

上面这个配置文件配置了日志的级别为错误级别，日志信息在控制台和文件中输出，其中输入到文件的名称为 `system.log`。经过上面这样简单的配置，就可以在程序中使用类似下面的方法使用这个日志工具。

```
public class Hello{  
    private static Log logger = LogFactory.getLog(Hello.class);  
    public sayHello ( ){  
        logger.error(" sayHello method error!");  
    }  
}
```

### 4.14 小结

在本章中，对 JSP 的基本语法和对象等知识进行了系统的介绍，而且对于其中大部分的知识点都给出了具体示例，这些示例在具体的开发过程中都有很大的参考价值，读者可以在这些示例程序的基础上进行尝试，试着修改其中的功能，只有这样才能对其运行原理有更深入的了解和体会，这就是学习程序语言的最基本最有效的方法。