第3章 JSP

2016年5月18日 17:49

第一部分

1. jsp是简化Servlet的一种技术，将html和java代码语句混合在同一个文件中编写，只对网页中动态产生的内容采用java代码进行编写，而对固定部分使用静态html页面的方式编写
2. 创建jsp步骤 ：
   a. 新建一个.jsp文件
   b. 在其中<% %>标签中书写java代码
3. jsp可以放置在出WEB-INF以外的任何路径
4. jsp运行原理 ： jsp本质上是一个servlet，web容器接收到jsp后通过jsp引擎编译成servlet文件，再将servlet文件通过servlet容器编译为.class文件
5. jsp编译后的java文件 ：

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.yang.demo.Person;

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

  private static final javax.servlet.jsp.JspFactory _jspxFactory =
          javax.servlet.jsp.JspFactory.getDefaultFactory();

  private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;

  private javax.el.ExpressionFactory _el_expressionfactory;
  private org.apache.tomcat.InstanceManager _jsp_instancemanager;

  public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
    return _jspx_dependants;
  }

  public void _jspInit() {
    _el_expressionfactory =
_jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
    _jsp_instancemanager =
org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());
  }

  public void _jspDestroy() {
  }

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

      //在jsp中可以直接使用的变量
    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
    final java.lang.Object page = this;

    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;


    try {
      response.setContentType("text/html; charset=ISO-8859-1");
      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("\r\n");
```

```
      out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"
\"http://www.w3.org/TR/html4/loose.dtd\">\r\n");
      out.write("<html>\r\n");
      out.write("<head>\r\n");
      out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ISO-8859-1\">\r\n");
      out.write("<title>Insert title here</title>\r\n");
      out.write("</head>\r\n");
      out.write("<body>\r\n");

      //使用<% %>编写的java代码全都在_jspService方法的这里
      Person p = new Person();
      out.print(p.getPersonInfo());
      … …

        out.write("\r\n");
        out.write("</body>\r\n");
        out.write("</html>");
      } catch (java.lang.Throwable t) {
        if (!(t instanceof javax.servlet.jsp.SkipPageException)){
          out = _jspx_out;
          if (out != null && out.getBufferSize() != 0)
            try {
              if (response.isCommitted()) {
                out.flush();
              } else {
                out.clearBuffer();
              }
            } catch (java.io.IOException e) {}
          if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
          else throw new ServletException(t);
        }
      } finally {
        _jspxFactory.releasePageContext(_jspx_page_context);
      }
    }
  }
```

6. 为jsp在web.xml中配置Servlet

```
  <servlet>
      <servlet-name>testJsp</servlet-name>
      <jsp-file>/index.jsp</jsp-file>
      <init-param>
          <param-name>password</param-name>
          <param-value>1234</param-value>
      </init-param>
  </servlet>
  <servlet-mapping>
      <servlet-name>testJsp</servlet-name>
      <url-pattern>/testJsp</url-pattern>
  </servlet-mapping>
```

7. jsp隐含变量 ： （可以不用声明直接使用的变量）
   a. **request** ： HttpServletRequest的一个对象
   b. **response** : HttpServletResponse的一个对象，在jsp中几乎不会调用
   c. **pageContext**：页面的上下文，是PageContext的一个对象，可以通过pageContext获取其他的8个隐含对象，也可以获得当前页面的其他信息，自定义标签时需要使用到pageContext
      e.g. `username = pageContext.getRequest().getParameter("username");`
   d. **session**：代表客户端与服务器的一次会话，是HttpSession的一个对象
   e. **application**：代表当前web应用，是ServletContext一个对象，可以获取当前web应用初始化参数
      `String username = application.getInitParameter("username");`
   f. **config**： 当前jsp对应Servlet的ServletConfig对象，平常几乎不适用，若需要访问当前jsp配置的Servlet初始化参数，需要通过映射的地址才可以
   g. **out** ： JspWriter对象，调用out.println()方法可以直接把字符串打印到浏览器上
   h. **page**：指向当前jsp对应的Servlet对象的引用，类型为Object，只能调用Object的方法，平常几乎不适用
   i. **exception**：在声明了错误页面后可以使用 ---> `<%@ page isErrorPage="true" %>`
   - 对属性作用于的范围 : pageContext < request < session < application

8. jsp语法 :
   a. jsp模板元素：jsp中的静态html部分成为模板元素
   b. jsp表达式 ： **<% out.print(test) %>** 和 **<%= test %>**功能相同，实现原理也相同，都是向页面输出
   c. jsp声明 : jsp声明将java代码封装到**<%! %>**之中，它里边的方法被插进_jspServlic()方法的外面，在jsp中几乎不去使用
   d. jsp注释 : **<%-- jsp注释 --%>** **<!-- html注释 -->** ， 区别:html注释不可以阻挡java代码的执行

9. 和属性相关的方法
   a. 相关方法
      - `Object getAttribute(String name);` //获取指定属性
      - `Enumeration getAttributeNames();` //获取所有属性的名字组成的Enumeration对象
      - `removeAttribute(String name);` //移除指定属性
      - `void setAttribute(String name, Object o);` //设置指定属性
   b. 域对象
      - 域对象可以调用与属性相关的方法

- 域对象包括：pageContext，request， session， application

c. 域对象的作用域
- pageContext ： 属性的作用范围仅限于当前jsp页面，在Servlet中无法获取属性值
- request ： 属性的作用范围仅限于同一个请求，在Servlet中无法获取属性值
- session ： 属性的作用范围仅限于一次会话(浏览器打开-关闭称之为一次会话，会话不失效的前提下)，在Servlet中可以获取属性值
- application ： 属性的作用范围限于当前web应用，在Servlet中可以获取属性和值

10. 请求的转发与重定向
a. 本质区别 ： 请求的转发只发送一次请求，重定向发送两次请求
b. 转发 ：
- 地址栏是初次发送请求的地址
- 最终的Servlet中，request和中转的request是同一个request
- 只能转发给当前web资源
- /代表当前web应用的根目录
- 实现步骤
  1. 调用HttpServletRequest的getRequestDispatcher(path)方法获取RequestDispatcher对象
  2. 使用RequestDispatcher对象的forward(request, response)方法进行转发

e.g.
```
String path = "testServlet";
RequestDispatcher requestDispatcher = request.getRequestDispatcher("/" + path);
requestDispatcher.forward(request, response);
```

c. 重定向 ：
- HttpServletResponse状态码

1xx	信息状态码：表示该请求已经接受，正在被处理
2xx	正确状态码：表示该请求已经正确接收并处理，没有发生错误，200表示一切正确
3xx	重定向状态码：客户端需要重新定向到新的资源，301表示永久重定向，302表示临时重定向
4xx	请求错误码：401表示没有权限访问，404表示资源不存在，405表示访问方式错误
5xx	服务器错误码：500表示程序出现异常而中途停止运行

- 地址栏为最后相应的地址
- 最终的Servlet中，request和中转的request不是同一个request
- 可以重定向到任何资源
- /代表当前web站点的根目录
- 实现步骤
  直接调用response的sendRedirect(Path)方法
  e.g.
```
String path = "testServlet";
response.sendRedirect(path);
```

第二部分
11. jsp指令
a. 为jsp引擎设计的，并不产生任何输出，而是告诉引擎如何处理jsp页面的其余部分
b. jsp指令的基本语法格式
<%@ 指令 属性名="值"%>

12. page指令
a. 用于定义jsp页面的各种属性，无论出现在哪里都作用于整个jsp页面
b. page指令语法
```
<%@ page
    [language="java"]   ->   当前页面可以使用的合法语言
    [extends="package.class"]   ->   当前jsp被翻译Servlet后可以继承的类
    [import="{package.class | package.* | …}"]   ->   导入所使用的包
    [session="true|false"]   ->   当前页面是否可以使用session
    [buffer="none | 8kb | sizekb"]  ->  buffer 属性指定 out 变量(类型为 JspWriter )使用的缓冲区的大小
    [autoFlush="true | false"]   ->   当缓冲区满后：true->清空缓冲区 / false->抛出异常
    [isThreadSafe="true | false"]   ->   控制jsp生成的servlet是否允许并行访问
    [info="text"]   ->   定义一个通过getServletInfo()方法获取的字符串
    [errorPage="relative_url"]   ->   指定当前页面发生错误后转发的url地址，/代表当前web应用的根目录
    [isErrorPage="true | false"]   ->   制定当前当前页面为发生错误后转发的页面，定义后可以使用exception对象获取错误信息，若指定isErrorPage为true，并使用exception方法，不建议直接访问该页面
    [contentType="mimeType[;charset=charactorSet]" | "text/html; charset=ISO-8859-1"]   ->
    制定jsp相应类型，实际上调用的是response.setContentType()方法，一般来说该属性值为text/html
    [pageEncoding="characterSet | ISO-8859-1"]  ->  指定当前jsp页面的字符编码
    [isELlgnored="true | false"]   ->   指定当前jsp页面是否可以使用EL表达式，通常取值为true
%>
```

13. 如何使用户不能直接访问某个页面 ：
对于Tomcat服务器中，WEB-INF下的文件不能通过浏览器直接访问，只有通过请求的转发的方式才可以进行访问

14. 处理发生错误的页面 ：
a. 方法1 ： 通过jsp指令处理错误页面
i. 在可能发生错误的页面中定义 ： <%@ page errorPage="/WEB-INF/error.jsp" %>
ii. 在WEB-INF中定义对应的错误页面
iii. 在错误页面中定义：<%@ page isErrorPage="true"%>，并在jsp代码片段中调用exception.getMessage()获取错误信息
b. 方法2 ： 通过配置web.xml处理错误页面
i. 在web.xml中配置错误页面处理信息

```
<error-page>
    <!-- 发生指定错误代码的时候转发到错误处理页面 -->
    <error-code>500</error-code>
    <!-- 制定相应错误页面的位置-->
    <location>/WEB-INF/error.jsp</location>
</error-page>
<error-page>
    <!-- 发生指定异常的时候转发到错误处理页面 -->
    <exception-type>java.lang.ArithmeticException</exception-type>
    <location>/WEB-INF/error.jsp</location>
</error-page>
```

i. 在WEB-INF中定义对应的错误页面
ii. 在错误页面中定义：<%@ page isErrorPage="true"%>，并在jsp代码片段中调用exception.getMessage()获取错误信息

15. include指令
    a. include指令用于通知jsp引擎在翻译当前jsp的时候**将其他文件的内容合并入当前jsp转换成的Servlet中**，也称之为**静态引入，可以共享变量**
    b. 语法 ：

```
<%-- 在testIncludeA.jsp中包含testIncludeB.jsp --%>
<%@ include file="testIncludeB.jsp" %>
```

    c. 生成的html代码 ：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>testIncludeA.jsp</title>
</head>
<body>
<h1>testIncludeA</h1>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>testIncludeB.jsp</title>
</head>
<body>
<h1>testIncludeB</h1>
</body>
</html>
</body>
</html>
```

    d. file属性必须指定需要引入的文件的相对路径
16. jsp标签
    a. **jsp:include page="" 动态包含jsp文件，会生成两个Servlet，通过一个包含方法包含进来，无法共享变量**
        - testIncludeA中的Servlet中的包含方法 ：
          org.apache.jasper.runtime.JspRuntimeLibrary.include(request, response, "testIncludeB.jsp", out, false);
    b. **jsp:forward page="" 对页面进行转发**
        - 等价于<%request.getRequestDispatcher("testIncludeB.jsp").forward(request, response); %>
        - 有时会使用jsp:param子标签传递参数，通常jsp:include也可以使用jsp:param自标签，使用request.getParam()可以获取传来的参数

```
e.g.
<jsp:forward page="testIncludeB.jsp">
    <jsp:param value="this is testIncludeA.jsp send value" name="test"/>
</jsp:forward>
```

17. 解决中文乱码
    a. 在页面中输入中文不出现乱码 ：
    保证contentType="text/html; charset=utf-8"和pageEncoding="utf-8"的编码一致且都支持中文
    b. 获取中文参数值不出现乱码 ：
    - post请求 -> request.setCharacterEncoding("UTF-8"); //在获取相应参数之前
    - get请求 ->

```
String val = request.getParameter("username");  //获取参数值
//将iso-8859-1的字符流转换为utf-8
String username = new String(val.getBytes("iso-8859-1"), "utf-8");
```

    - 在apache中配置 ->

```
server.xml
<Connector port="8989" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"
           useBodyEncodingForURI="true"
           />
```