# JSP

## JSP简介

* JSP全称Java Server Pages，顾名思义就是运行在java服务器中的页面，也就是在我们JavaWeb中的动态页面，其本质就是一个Servlet。
* 其本身是一个动态网页技术标准，它的主要构成有HTML网页代码、Java代码片段、JSP标签几部分组成，后缀是.jsp。
* JSP相比HTML页面来说，最直观的功能就是可以在页面中使用变量，这些变量一般都是从域对象中获取。有了变量的好处就是我们的页面可以动态的显示信息。
* 相比于Servlet，JSP更加善于处理显示页面，而Servlet跟擅长处理业务逻辑，两种技术各有专长，所以一般我们会将Servlet和JSP结合使用，Servlet负责业务，JSP负责显示。

## JSP的基本语法

### 2.1 基本格式

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  </head>  <body>    </body>  </html> |

* JSP的基本格式和HTML页面相似，不同之处就是使用JSP时页面的顶部必须使用JSP指令声明一些JSP相关的信息。如上图JSP文件，首行使用一条page指令声明了JSP的相关信息，关于JSP指令我们在这里先不过多讲解，目前可以把它当成一种固定格式。
* 在首行的JSP指令下边就可以来编辑我们的HTML代码了，从上边的文件中也可以看出实际上就是原封不动的HTML代码。
* 如上这面的JSP文件，我们如果不需要加入动态代码，就可以直接来编写HTML代码，语法是一模一样的。
* 代码编辑完成后启动服务器，访问JSP页面（和访问HTML页面一样，直接输入地址），会看到页面正常显示和普通HTML一样。
* 注意：**JSP文件的运行依赖于WEB服务器**，也就是说如果不是通过WEB服务器，浏览器是不能直接打开JSP文件的。

### 2.2 JSP脚本元素

* 在JSP中我们主要通过脚本元素来编写Java代码，这些Java代码一般会在页面被访问时调用。
* JSP脚本元素主要分三种：脚本片段、表达式还有声明。

#### 2.2.1 JSP脚本片段

* 脚本片段是嵌入到JSP中Java代码段，格式以<%开头，%>结尾，两个%号之间就可以编写Java代码了。
* 如：

|  |
| --- |
| <% System.out.println("Hello World");%> |

* 上边就是一个JSP的脚本片段，片段中的Java代码使我们非常熟悉的内容，这条语句会在JSP页面被访问时向页面中打印一条“Hello World”。
* 通过这种方式我们可以在JSP中完成大量的Java代码，甚至写一些业务逻辑，但是并不建议这么做。
* 这种方式编写的Java代码，会放到Servlet的service方法中执行，既然是写在一个方法中的代码那就对我们就不能随便的去写。比如：不能定义成员变量、不能定义方法、不能定义类。

#### 2.2.2 JSP表达式

* JSP表达式用来直接将Java变量输出到页面中，格式以<%=开头，以%>结尾，中间是我们要输出的内容。
* 如：

|  |
| --- |
| <%=str %> |

* 上边语句中的str是JSP中的一个String型的变量，通过这种方式可以将该变量输出到页面中。

#### JSP声明（了解）

* JSP声明中的内容会被直接写到类中，格式以<%!开头，以%>结尾，中间是Java代码
* 如：

|  |
| --- |
| <%! **private** **int** a = 0; %> |

* 上边这条语句相当于在类中声明了一个成员变量，由于JSP声明中的代码会被写在类中，所以在类中可以编写的内容在JSP声明中都可以编写。如：定义成员变量、定义方法、构造器、构造代码块、静态代码块。
* JSP声明使用的机会并不是很多，所以知道即可。

#### 2.2.4 注释

* JSP注释和其他注释功能一样，注释的内容只有在当前JSP页面中可以看到，但是在转换后的Servlet中以及浏览器端显示的页面中都是不可见的。
* 语法：

|  |
| --- |
| <%-- 注释内容 --%> |

* JSP中个中注释的比较：

|  |  |  |  |
| --- | --- | --- | --- |
|  | JSP注释 | Java注释 | HTML注释 |
| JSP页面 | 可见 | 可见 | 可见 |
| Java代码 | 不可见 | 可见 | 可见 |
| 浏览器 | 不可见 | 不可见 | 可见 |

### 2.3 JSP运行原理

* 上边我们演示了JSP中的几种脚本元素，这几种脚本元素都是可以运行的Java代码，大家一定会有一个疑问，为什么在一个页面中可以运行Java代码呢？
* 实际上Tomcat在运行JSP时，并不是直接显示的我们所编写的JSP页面，而是将JSP页面转换成了一个Java类，这个Java类是什么，我想大家也能猜到了，它实际上就是一个Servlet。
* 这个Servlet在哪呢？还记得我们说过的Tomcat的work目录吗？在那个目录下保存着Tomcat自动生成的一些内容，下面让我们来找到那个目录。
  + 对于Eclipse来说是在:
    - 工作空间下的.metadata\.plugins\org.eclipse.wst.server.core\tmp0
  + 对于MyEclipse来说就可以直接去Tomcat的安装目录去查找
* 在Work目录下的Catalina\localhost\07\_WEB\_SERVLET\org\apache\jsp文件夹中我们可以发现两个文件index\_jsp.java和index\_jsp.class，前者就是Tomcat自动生成的Servlet的源码，后者是编译后的.class文件。打开java文件内容如下：

|  |
| --- |
| package org.apache.jsp;  import javax.servlet.\*;  import javax.servlet.http.\*;  import javax.servlet.jsp.\*;  public final class index\_jsp extends org.apache.jasper.runtime.HttpJspBase  implements org.apache.jasper.runtime.JspSourceDependent {  private int a = 0; //JSP声明生成的代码  private static final JspFactory \_jspxFactory = JspFactory.getDefaultFactory();  private static java.util.List \_jspx\_dependants;  private javax.el.ExpressionFactory \_el\_expressionfactory;  private org.apache.AnnotationProcessor \_jsp\_annotationprocessor;  public Object getDependants() {  return \_jspx\_dependants;  }  public void \_jspInit() {  \_el\_expressionfactory = \_jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();  \_jsp\_annotationprocessor = (org.apache.AnnotationProcessor) getServletConfig().getServletContext().getAttribute(org.apache.AnnotationProcessor.class.getName());  }  public void \_jspDestroy() {  }  public void \_jspService(HttpServletRequest request, HttpServletResponse response)  throws java.io.IOException, ServletException {  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 {  response.setContentType("text/html; charset=UTF-8");  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("<!DOCTYPE html>\r\n");  out.write("<html>\r\n");  out.write("<head>\r\n");  out.write("<meta charset=\"UTF-8\">\r\n");  out.write("<title>Insert title here</title>\r\n");  out.write("</head>\r\n");  out.write("<body>\r\n");  out.write("\t");    System.out.println("Hello World"); //脚本片段生成的代码  int x = 20394; //脚本片段生成的代码    out.write("\r\n");  out.write("\t\r\n");  out.write("\t\r\n");  out.write("\t<!-- JSP表达式生成的代码 -->\r\n");  out.write("\t");  out.print(x );  out.write("\r\n");  out.write("\t\r\n");  out.write("\t");  out.write("\r\n");  out.write("</body>\r\n");  out.write("</html>");  } catch (Throwable t) {  if (!(t instanceof SkipPageException)){  out = \_jspx\_out;  if (out != null && out.getBufferSize() != 0)  try { out.clearBuffer(); } catch (java.io.IOException e) {}  if (\_jspx\_page\_context != null) \_jspx\_page\_context.handlePageException(t);  else log(t.getMessage(), t);  }  } finally {  \_jspxFactory.releasePageContext(\_jspx\_page\_context);  }  }  } |
|  |

* 通过观察发现index\_jsp名字和我们创建的jsp文件名字类似，只是把index.jsp中的点换成的\_，实际上他就是Tomcat根据我们编写的JSP文件自动生成的类。
* index\_jsp这个类继承了org.apache.jasper.runtime.HttpJspBase，而HttpJspBase又继承了HttpServlet。由此证明，index\_jsp就是一个Servlet。而在我们访问JSP时服务器就是调用了该Servlet来响应请求。
* 有同学可能会有疑问，**Servlet是需要在web.xml中配置的**，而我们并没有配置JSP的serlvet映射，那他是如何访问的呢？实际在conf目录中的**web.xm**l早已配置好了JSP的映射信息，具体内容如下：

|  |
| --- |
| <servlet>  <servlet-name>jsp</servlet-name>  <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>  <init-param>  <param-name>fork</param-name>  <param-value>false</param-value>  </init-param>  <init-param>  <param-name>xpoweredBy</param-name>  <param-value>false</param-value>  </init-param>  <load-on-startup>3</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>jsp</servlet-name>  <url-pattern>\*.jsp</url-pattern>  </servlet-mapping>  <servlet-mapping>  <servlet-name>jsp</servlet-name>  <url-pattern>\*.jspx</url-pattern>  </servlet-mapping> |

* 既然已经证明其就是一个Servlet，那我们已知Servlet是调用service方法来处理请求的，在我们的index\_jsp中并没有我们熟悉的service()方法，但是经仔细观察发现有如下方法\_jspService(HttpServletRequest request, HttpServletResponse response)，该方法就相当于我们JSP中service()方法。
* service方法中声明了如下几个局部变量：

|  |
| --- |
| PageContext pageContext = null;  HttpSession session = null;  ServletContext application = null;  ServletConfig config = null;  JspWriter out = null;  Object page = this; |

* + 这几个对象在方法下边进行了赋值操作，再加上参数中的request和response，以及出异常的时候还有一个exception。这些是我们JSP中的九大隐含对象，后边我们还要在讲解。这些对象除了exception比较特殊外，其他都可以直接在JSP中直接使用。
* 注意观察该方法，是如何将JSP中的代码转换为Java代码的：
  + Html代码：out.write("<!DOCTYPE html>\r\n");
    - JSP中的HTML代码会变成字符串通过out.write()方法输出。
  + <%%>中的代码：System.out.println("Hello World"); //脚本片段生成的代码。
    - 脚本片段中的代码会直接复制到对应的位置。
  + <%=x%>中的代码：out.print(x );
    - 表达式中的变量，会变成out.print()的参数输出到页面中。
  + <%! %>中的代码：private int a = 0; //JSP声明生成的代码
    - 声明中的代码，会被原封不动的写到类中。
* 理解了JSP的运行原理对我们理解JSP是非常重要的，也就是说在我们编写JSP代码的时候，在脑海里应该可以想象出编译好的servlet的样子。

### 2.4 JSP生命周期

### 2.5 JSP隐含对象

* 隐含对象指在JSP中无需创建可以直接使用的对象，包括：
  + out（JspWriter）：相当于response.getWriter()获取的对象，用于在页面中显示信息。
  + config（ServletConfig）：对应Servlet中的ServletConfig对象。
  + page（Object）：对应当前Servlet对象，实际上就是this。
  + pageContext（PageContext）：当前页面的上下文，也是一个域对象。
  + exception（Throwable）：错误页面中异常对象
  + request（HttpServletRequest）：HttpServletRequest对象
  + response（HttpServletResponse）：HttpServletResponse对象
  + application（ServletContext）：ServletContext对象
  + session（HttpSession）：HttpSession对象

#### 2.5.1 域对象

* 在JavaWeb中总共有四个域，页面、请求、会话和整个应用。域对象主要作用就是在这四个域中传递数据的。
* 每个域对象的内部实际上都有一个map用来存储数据，数据以键值对的结构存放，key是String类型的，value使用Object类型。
* 我们可以在一个域对象中放入数据。然后，在当前域中的其他JSP页面或Servlet中获取该数据。以达到一个共享数据的目的。
* 在JSP中可以获得全部四个域对象，而Servlet中只能获取三个域对象request、session、application。
* 四个域对象
  + pageContext
    1. 类型：PageContext
    2. 范围：当前JSP页面
    3. 注意：该对象只能在JSP中获取，Servlet中没有
  + Request
    1. 类型：HttpServletRequest
    2. 范围：当前请求
  + Session
    1. 类型：HttpSession
    2. 范围：当前会话
  + Application
    1. 类型：ServletContext
    2. 范围：当前应用
* 域对象都有三个操作数据的主要方法：
  + public void setAttribute(String name, Object o);
    1. 在当前域中放入数据
  + public Object getAttribute(String name)
    1. 根据名字获取当前域中的数据
  + public void removeAttribute(String name);
    1. 根据名字删除当前域中的数据
* 四个范围
  + 页面：页面范围内的数据，只能在当前页面中获取，一旦转到其他页面当前域中的数据便失效，不能获取。
  + 请求：请求范围内的数据，和页面范围类似，它表示的是一次请求范围。区分一次请求主要是看是不是同一个request。比如：转发是表示一个请求，重定向是多个请求。
  + 会话：会话比请求更高一级。简单来说，就是打开浏览器到关闭浏览器，这一个完整的上网过程叫做一个会话。只要没有关闭浏览器或设置session失效，就可以在域中获取到Session中的数据。
  + 应用：应用是最高级的域对象，他代表整个WEB应用，在这个域对象中设置的数据在所有的域中都能获取。
* PageContext
  + PageContext和其他域对象还不太一样，还有一些特有的功能。
  + PageContext是整个页面的上下文，可以获取页面相关的内容。
    1. 作为页面域对象。
    2. 可以获取指定域中的数据。
       - getAttribute(String name, int scope)
    3. 可以向指定域中设置数据。
       - setAttribute(String name, Object value, int scope)
       - 上述两个方法中int scope是域类型的常量值，PageContext为每个域对象设置了一个整形常量分别为：
         * PageContext.PAGE\_SCOPE 值为 1
         * PageContext.REQUEST\_SCOPE 值为 2
         * PageContext.SESSION\_SCOPE 值为 3
         * PageContext.APPLICATION\_SCOPE 值为 4
    4. 全域查找
       - Object findAttribute(String name)
    5. 可以获取其他隐含对象。
       - HttpSession getSession()
       - Object getPage()
       - ServletRequest getRequest()
       - ServletResponse getResponse()
       - Exception getException()
       - ServletConfig getServletConfig()
       - ServletContext getServletContext()
       - JspWriter getOut()

#### 2.5.2 其他隐含对象

* out（JspWriter）
  + 赋值：out = pageContext.getWriter();
  + 作用：向页面中输出内容。
  + 本质：JSP的字符输出流。
* config（ServletConfig）：对应Servlet中的ServletConfig对象。
  + 赋值：config = pageContext.getServletConfig();
  + 作用：获取配置信息。
  + 本质：ServletConfig对象。
* page（Object）：对应当前Servlet对象，实际上就是this。
  + 赋值：Object page = this;
  + 本质：当前Servlet对象的引用。
* exception（Throwable）：错误页面中异常对象
  + 赋值：Throwable exception =

org.apache.jasper.runtime.JspRuntimeLibrary.getThrowable(request);

* + - * 作用：获取异常信息。
      * 本质：Throwable对象。
* response（HttpServletResponse）：HttpServletResponse对象
  + 赋值：service()方法的参数。
  + 作用：同Servlet中的response。

### 2.6 JSP指令

* JSP指令用来设置与整个jsp页面相关的属性，它并不直接产生任何可见的输出,而只是告诉引擎如何处理其余JSP页面。
* 指令格式：<%@指令名 属性名1=”属性值1” 属性名2=”属性值2” %>
* JSP中有三种指令page、include、taglib。

#### 2.6.1 page指令

* page指令是我们最常用的指令，属性非常多。
  + **import** 导包
  + isThreadSafe 是否单线程模式
  + contentType 响应的文件类型
  + isELIgnored 是否忽略EL表达式
  + **isErrorPage** 是否是一个错误页面
  + **errorPage** 发生错误后转发的页面
  + language JSP使用的语言，目前只有java
  + extends 继承父类
  + session 页面中是否具有session对象
  + buffer 定义out对象如何处理缓存
  + autoFlush 缓存是否自动刷新
  + info 定义转换后放到页面中的串
  + pageEncoding 定义JSP页面的字符编码

#### 2.6.2 include指令

* include是静态包含指令，主要是用于在当前页面中引入其他页面。
* 用法：<%@ include file=”页面地址”%>
* 例如，有如下两个页面
  + index.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"* errorPage=*"error.jsp"*%>  <%@ include file=*"in.html"* %>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  </head>  <body>    </body>  </html> |

* in.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <h1>Hello I'm in.html Page</h1>  </body>  </html> |

* 这种写法就相当于在index.jsp中的include标签的位置，将in.html的代码复制一遍

|  |  |
| --- | --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"* errorPage=*"error.jsp"*%>   |  | | --- | | <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <h1>Hello I'm in.html Page</h1>  </body>  </html> |   <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  </head>  <body>    </body>  </html> |

* 也就是说include的所引入页面的代码会在JSP对应的Servlet文件中生成

#### 2.6.3 taglib指令

* 定义JSP可以使用的标签库，这部分我们目前还用不到，等到JSTL时我们在详细讲解

### 2.7 JSP动作标签

* JSP动作标签与HTML标签不同，HTML标签由浏览器来解析，而**JSP动作标签需要服务器（Tomcat）来运行**。
* 常用的JSP动作标签。
  + <jsp:forward>:
    - 作用：在页面中用于转发操作
    - 实例：

|  |
| --- |
| <jsp:forward page=*"target.jsp"*></jsp:forward> |

* + - 子标签：<jsp:param value="paramValue" name="paramName"/>
      * 作用：在转发时设置请求参数，通过request.getParameter()在目标页面获取请求参数。
      * 实例：

|  |
| --- |
| <jsp:forward page=*"target.jsp"*>  <jsp:param value=*"paramValue"* name=*"paramName"*/>  </jsp:forward> |

* <jsp:include>:
  + 作用：动态包含，将其他页面包含到当前页面中。
  + 实例：

|  |
| --- |
| <jsp:include page=*"target.jsp"*></jsp:include> |

* + 原理：当使用动态包含时，Tomcat会在生成的Servlet中加入如下代码：

org.apache.jasper.runtime.JspRuntimeLibrary.include(request, response, "target.jsp", out, false);

* + 与静态包含的区别：
    - 静态包含使用include指令，动态包含使用<jsp:inclued>标签
    - 静态包含会直接将目标页面复制到生成的Servlet中，动态包含是在生成的servlet中使用include()方法来引入目标页面。
    - 当目标页面发生改变时，静态包含不能体现，动态包含可以体现