JSP

1 JSP的概念

JSP 是 Java Server Page 的缩写，是一种动态网页技术标准。在 HTML 文件中加入 Java 程序段和 JSP 标签等，就构成了 JSP 网页。

为什么使用 JSP：

1. HTML 语言书写的 Web 页面只能实现静态的网页，无法实现动态可变的页面要求，处理能力弱，无法完成复杂的功能。
2. Servlet 具有强大的处理能力，但是在页面的组织上繁琐、复杂，给 Web 页面的开发带来了极大的不便。
3. JSP 正是结合了 HTML 页面设计方便和 Servlet 强大处理能力而产生的一种动态网页设计技术。

例如，下面的代码在页面上显示一个正弦函数表。

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>一个简单的JSP</title>
    </head>
    <body>
        <table frame="border" bordercolor="black"> <!-- 设置表有黑色边框 -->
            <!-- 第一行 表头 -->
            <tr>
                <!-- 第1列 列头,设置列有黑色边框 -->
                <td style="border: 1px solid black;"></td>
                <!-- 通过循环产生余下的列 -->
                <%
                for(int i = 0; i <= 360; i = i + 30) {
                %>
                <td style="border: 1px solid black;"><%= i%></td>
                <%
                }
                %>
            </tr>
            <!-- 第二行 正弦函数值 -->
            <tr>
                <!-- 第1列 列头 -->
                <td style="border:1px solid black;">sin</td>
                <!-- 通过循环产生余下的列 -->
                <%
                for(int i = 0; i <= 360; i = i + 30) {
                %>
                <td style="border: 1px solid black;"> <!-- 设置列有黑色边框 -->
                    <!-- 计算正弦值，并保留三位小数，在对应的列中进行显示 -->
                    <%= String.format("%.3f", Math.sin(i/180.0*Math.PI)) %>
                </td>
                <%
                }
                %>
            </tr>
        </table>
    </body>
</html>
```

代码大部分是由 HTML 书写，多出了一些以 `<%` 开头的片段，片段中包含着 Java 代码。

JSP 页面就是将 Java 代码和一些 JSP 元素嵌入到 HTML 中构成的页面编码，这样既能方便快捷地进行页面格式的设计，又能够提升页面的处理能力，增强页面功能。

2 JSP的工作原理

JSP 的工作过程如图 2.1 所示。

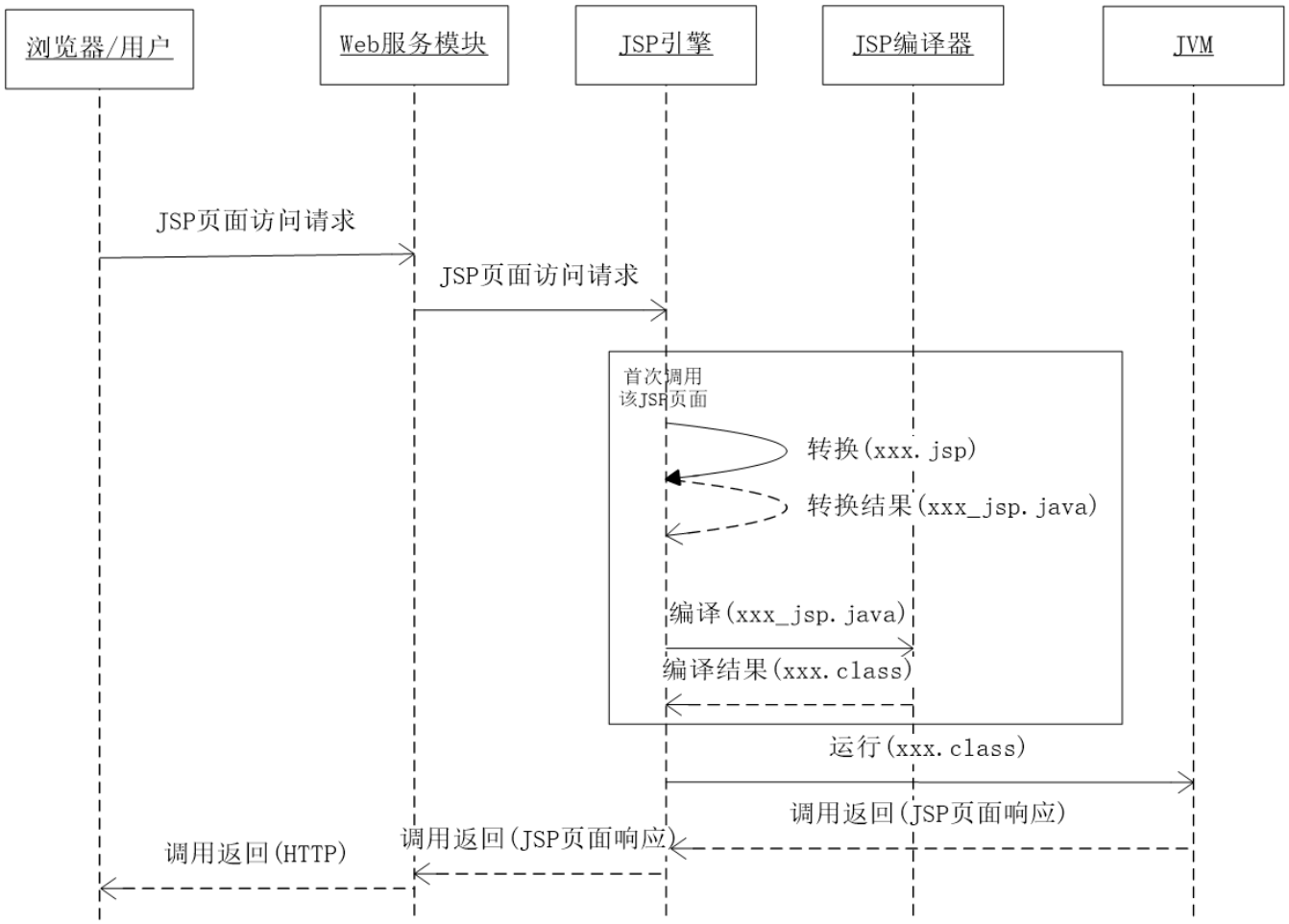

图 2.1　JSP 的工作过程

JSP 与 Servlet 一样，是在服务器端执行的程序。

JSP 在执行之前首先被转换成 Servlet。这个转换工作在 JSP 页面第一次被请求执行时由服务器的 JSP 引擎自动完成，转换成的 Java 源代码被编译成 .class 文件。而后 JSP 引擎加载运行对应的 .class 文件生成相应的结果页面，并将输出结果发送到浏览器进行显示。

JSP 中的页面显示内容都会用`out`对象进行打印输出。

3 JSP的基本构成

JSP 页面的基本构成除了 HTML 元素之外，还包括了以下一些 JSP 元素：

1. JSP 声明
2. JSP 程序块
3. JSP 表达式
4. JSP 指令
5. JSP 动作

6. JSP 注释

3.1 JSP声明

在 JSP 页面中可以声明变量和方法。这些声明的变量和方法的作用域是声明该变量和方法的 JSP 页面。

声明通常是供 JSP 页面中其它 Java 程序段所使用，本身不会产生任何输出，对页面不会造成直接影响。

JSP 中声明是通过 JSP 声明标签完成的，具体的语法格式为：`<%! 声明1;声明2; %>`。

例如：

```
<%! int i = 1; %>
<%!
    int j = 0;
    String str = new String("Hello");
%>

<%
    out.println(i + j);
    out.println(str);
%>
```

在 JSP 页面被转换成 Servlet 时，`<%! ... %>`中的声明代码将作为 Servlet 类的全局内容被放到生成出的 Servlet 类中，即被转换为类的属性和方法。

3.2 JSP程序块

JSP 中可以通过插入 Java 代码来实现所需的功能，这一能力大大增强了 JSP 的灵活性。

JSP 程序块以`<%`为起始，以`%>`为结束，在期间允许插入任何合法的 Java 代码。

`<%...%>`可以出现在 JSP 页面的任何地方，可以出现任意多次。每个`<%...%>`中可以包含多条 Java 语句，每条语句必须使用`;`号结尾。每个`<%...%>`中的代码可以是不完整的，但是一个 JSP 页面中所有`<%...%>`中的语句组合在一起必须是完整的。

在 JSP 页面被转换成 Servlet 时，`<%...%>`中的代码被默认放到生成出的 Servlet 的`service()`方法中。

3.3 JSP表达式

在 JSP 中可以通过表达式标签简单快捷的将 Java 变量或 Java 表达式的值输出到 HTML 页面中相应的位置。

JSP 表达式的语法格式为：`<%= Java变量或表达式 %>`。

JSP 表达式以`<%=`开头，以`%>`结尾，中间是符合 Java 语言语法的变量或者表达式，此处的变量或者表达式中包含的变量必须是已声明的。结尾不能有`;`号。变量或表达式的值会被转换成字符串，并按照先后顺序依

次将结果输出到页面上 JSP 表达式所在的位置。

在 JSP 页面被转换成 Servlet 时，`<%= ... %>`中的代码转换为`print()`方法的参数加入到 Servlet 类中。

3.4 JSP指令

JSP 指令用来设置 JSP 页面的相关信息，如页面的各种属性、文件包含及标签信息等。在 JSP 规范中定义了三种指令：`page`、`include`和`taglib`。

JSP 指令同样会被 JSP 引擎转换为 Java 代码，它对应的 Java 代码是 Servlet 中的引用部分（如`import`语句）和一些设置方法的调用。这些代码不会向客户端发送任何信息，但是它将决定 Servlet 引擎如何来处理该 JSP 页面。

JSP指令的格式为：`<%@ 指令名 属性名="属性值" %>`

JSP 指令的名称和属性名称都是大小写敏感的。指令名的所有字母都是小写的。属性名符合 Java 的命名规范，第一个单词的首字母小写，其余单词的首字母大写。

每种指令都定义了若干属性。根据具体的编程需要，开发人员可以选择一个或多个属性进行设置。属性的设置可以放到一条 JSP 指令中，也可以放到多条 JSP 指令中。

3.4.1 `page`指令

`page`指令用来对 JSP 页面的各种属性进行设置，它将作用于整个页面，一般放在 JSP 页面的起始位置。

`page`指令在 JSP 规范中的定义如下：

```
<%@ page
    [ language="java" ]
    [ import="{package.class | package.*}, ... " ]
    [ contentType="{mimeType [; charset=characterSet] | text/html; charset=ISO-8859-1}" ]
    [ pageEncoding="{characterSet | ISO-8859-1}" ]
    [ session="true|false" ] [ extends="package.class" ]
    [ buffer="none|8kb|sizekb" ][ autoFlush="true|false" ]
    [ isThreadSafe="true|false" ][ info="text" ]
    [ erroPage="erroPageURL" ][ isErroPage="true|false" ]
    [ isELIgnored="true|false" ]
%>
```

`page`指令的常用属性如下表所示。

属性	取值	描述
`language`	`java`	该 JSP 程序块中使用的语言，默认为 Java
`import`	包名|类名	引入该 JSP 页面中需要用到的包和类等。 `import`是`page`指令中唯一可以出现多次的属性。

属性	取值	描述
`contentType`	文档类型	指定 JSP 页面的 MIME 类型和字符编码。常见的 MIME 类型包括： 1. `text/plain`：纯文本文件 2. `text/html`：纯文本的 HTML 页面 3. `image/jpeg`：JPG 图片 4. `image/gif`：GIF 图片 5. `application/msword`：Word 文档 6. `application/x-msexel`：Excel 文档 字符编码的默认值为 ISO-8859-1，可以通过`charset`修改字符编码。
`pageEncoding`	字符集	指定该 JSP 页面使用的字符编码。默认为 ISO-8859-1。

`page`指令示例：

```
<%@ page language="java" pageEncoding="UTF-8" %>
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*"%>
```

注意：`pageEncoding`设置保存当前 .jsp 文件所用的编码，`contentType`中的`charset`设置浏览器处理页面时采用的编码。

3.4.2 `include`指令

JSP 通过`include`指令来包含其他文件，被包含文件将在 JSP 页面转换成 Servlet 之前被插入到`include`指令出现的位置，这些文件可以是纯文本、HTML 或 JSP 文件。其语法格式为：`<%@ include file="url" %>`。

`file`属性指定被包含的文件，可以是绝对路径或相对路径。

3.4.3 `taglib`指令

JSP 允许开发人员自己定义 JSP 标签，并在 JSP 页面中进行使用。`taglib`指令在 JSP 页面中指明需要使用的自定义标签库并定义其标签前缀。语法格式为：`<%@ taglib uri="URIToTagLibrary" prefix="tagPrefix" %>`。

`uri`属性指明标签库描述符的位置，`uri`可以是 Uniform Resource Locator（URL）、Uniform Resource Name（URN）或者一个相对或绝对的路径。

`prefix`指定自定义标签的前缀。该前缀是自定义的字符串，但是不能使用 jsp、jspx、java、javax、servlet、sun 和 sunw。

例如：

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>利用taglib使用标准JSP标签库</title>
    </head>
    <body>
        <c:forEach var="i" begin="1" end="10" step="1">
            <c:out value="${i}" />
        </c:forEach>
    </body>
</html>
```

3.5 JSP动作

JSP 动作利用 XML 语法格式的标记来控制 Servlet 引擎的行为，为请求处理阶段提供信息。它们影响 JSP 运行时的行为和对客户端请求的响应。

动作元素由 JSP 引擎实现，利用 JSP 动作可以动态地插入文件、重用 Java Bean 组件、把用户重定向到另外的页面、为 Java 插件生成 HTML 代码等。

JSP 动作与 JSP 指令的区别：

- JSP 指令是静态的，是在 JSP 编译之前完成的。
- JSP 动作是在执行阶段的指令。

JSP 2.0 规范中定义了 20 个标准的动作元素，可以分为五大类：

1. 与存取 Java Bean 有关，包括：`<jsp:useBean>`、`<jsp:setProperty>`、`<jsp:getProperty>`。
2. JSP 1.2 开始就有的基本元素，包括 6 个动作元素：`<jsp:include>`、`<jsp:forward>`、`<jsp:param>`、`<jsp:plugin>`、`<jsp:params>`、`<jsp:fallback>`。
3. JSP 2.0 新增加的元素，主要与 JSP Document 有关，包括 6 个元素：`<jsp:root>`、`<jsp:declaration>`、`<jsp:scriptlet>`、`<jsp:expression>`、`<jsp:text>`、`<jsp:out`。
4. JSP 2.0 新增的动作元素，主要是用来动态生成 XML 元素标签的值，包括 3 个动作：`<jsp:attribute>`、`<jsp:body>`、`<jsp:element>`。
5. JSP 2.0 新增的动作元素，主要是用在 TagFile 中，有 2 个元素：`<jsp:invoke>`、`<jsp:dobody>`。

常用的元素有 7 个，下面分别进行具体介绍。

3.5.1 `jsp:include` 动作

该动作能够把指定文件的输出结果插入正在生成的页面中。语法如下：

```
<!-- 第一种 -->
<jsp:include page="path" flush="true" />

<!-- 第二种 -->
<jsp:include page="path" flush="true">
    <jsp:param name="paramName" value="paramValue"/>
</jsp:include>
```

`page` 设置需要包含的文件，可以是相对路径或绝对路径。

`flush="true"` 设置被包含页面是否使用缓冲，一般必须设置为 `true` 。

`<jsp:param>` 子句能传递一个或多个参数给动态文件，可以在一个页面中使用多个 `<jsp:param>` 来传递多个参数给动态文件。

`jsp:include` 动作的作用与 `include` 指令类似，但不同的是 `include` 指令所包含的文件在 JSP 文件被转换成 Servlet 的时候被引入，而该动作所包含的文件在该 JSP 页面被请求的时候才被插入。这使得该指令的执行效率要稍微差一点，但是它的灵活性却要好得多。

另外， `include` 指令是将被包含文件原封不动地插入到包含文件当中，而 `jsp:include` 动作则是将被包含文件执行之后输出的结果插入到包含文件。

3.5.2 `jsp:param` 动作

该动作能够以 “名-值” 对的方式为其它动作提供附加信息。在目标页面可以通过 `request.getParameter("参数名")` 方式取出对应值。该动作的语法格式为： `<jsp:param name="参数名" value ="参数值">` 。

它有两个必须的属性参数， `name` 表示参数的名称， `value` 表示参数的取值，取值可以以表达式的形式出现。

`<jsp:param>` 子句能传递一个或多个参数给动态文件，也可在一个页面中使用多个 `<jsp:param>` 来传递多个参数给动态文件。

该动作常与 `<jsp:include>` 、 `<jsp:forward>` 、 `<jsp:plugin>` 动作一起使用，为它们提供所需要传递的参数信息。例如：

```
<jsp:include page="包含页面的url" >
    <jsp:param name="参数名1" value="参数值1">
    <jsp:param name="参数名2" value="参数值2">
    ...........
</jsp:include>

<jsp:forward page="转向页面的url" >
    <jsp:param name="参数名1" value="参数值1">
    <jsp:param name="参数名2" value="参数值2">
    ...........
</jsp:forward>
```

3.5.3 `jsp:forward` 动作

该动作把请求转到另外的页面。其语法格式如下：

```
<!-- 第一种 -->
<jsp:forward page="path" />

<!-- 第二种 -->
<jsp:forward page="path" >
    <jsp:param name="paramName" value="paramValue" />
</jsp:forward>
```

该动作只有一个属性`page`。`page`属性的值可以是目的页面的相对路径，也可以是一个表示目的页面相对路径的表达式。

在转到别的页面时，还可以通过`<jsp:param>`子句向目的页面传递一个或多个参数，也可在一个页面中使用多个`<jsp:param>`来传递多个参数给目的页面。

3.5.4 `jsp:useBean` 动作

该动作用来加载一个将在 JSP 页面中使用的 Java Bean。该动作会创建一个 Bean 实例并指定它的名字以及作用范围。

该动作使得我们既可以发挥 Java 组件重用的优势，同时也避免了损失 JSP 区别于 Servlet 的方便性。

该动作的语法如下：

```
<jsp:useBean  id="name"
              scope="page|request|session|application"
              typeSpec />
```

`id`表示 bean 对象的引用名。

`scope`表示 bean 对象的作用域，默认为`page`。

`typeSpec`表示 bean 的类型，可以是以下几种形式：

1. `class="className"`
2. `class="className" type="typeName"`
3. `beanName="beanName" type="typeName"`
4. `type="typeName"`

其中，`class`和`beanName`都是 Bean 类的名字，并且必须为全名（即包含包名）。`type`指定引用该对象的变量的类型，它必须是 Bean 类的名字、超类名字、该类所实现的接口名字之一。

假如使用`class`，JSP 引擎先判断是否存在该 Bean 类的实例，如果不存在就使用`new`关键字实例化一个；而使用的是`type`时，它只是查找指定的范围中是否存在该 Bean 类的对象，如果不存在并且又没使用`class`或`beanName`，就会抛出异常。

3.5.5 `jsp:setProperty` 动作

该动作用来设置 Bean 中的属性值。其语法为：`<jsp:setProperty name="beanName" prop_expr />`

其中 `prop_expr` 有以下几种可能的情形：

1. `property="*"`
2. `property="propertyName"`
3. `property="propertyName" param="parameterName"`
4. `property="propertyName" value="propertyValue"`

`name` 属性是必需的，它表示要设置属性的是哪个 Bean。`name` 值应当和 `<jsp:useBean>` 中的 `id` 值相同。

`property` 属性也是必需的，它表示要设置哪个属性。

有一个特殊用法：如果 `property` 的值是 `"*"`，表示所有名字和 Bean 属性名字匹配的请求参数都将被传递给相应的属性 set 方法。

`value` 属性是可选的，该属性用来指定 Bean 属性的值。字符串数据会在目标类中通过标准的 `valueOf()` 方法自动转换成数字、`boolean`、`Boolean`、`byte`、`Byte`、`char`、`Character`。例如，`boolean` 和 `Boolean` 类型的属性值（比如 `"true"`）通过 `Boolean.valueOf()` 转换，`int` 和 `Integer` 类型的属性值（比如 `"42"`）通过 `Integer.valueOf()` 转换。

`param` 也是可选的，它指定用哪个请求参数作为 Bean 属性的值。如果当前请求没有参数，则什么事情也不做，系统不会把 `null` 传递给 Bean 属性的 set 方法。因此，你可以让 Bean 自己提供默认属性值，只有当请求参数明确指定了新值时才修改默认属性值。

`value` 和 `param` 不能同时使用，但可以使用其中任意一个。

使用 `jsp:setProperty` 来为一个 Bean 的属性赋值，可以使用两种方式来实现。

1. 在 `jsp:useBean` 后使用 `jsp:setProperty`。在这种方式中，不管 `jsp:useBean` 是找到了一个现有的 Bean，还是新创建了一个 Bean 实例，`jsp:setProperty` 都会执行。例如：

```
<jsp:useBean id="myUser" ... />
...
<jsp:setProperty name="myUser" property="user" ... />
```

2. `jsp:setProperty` 出现在 `jsp:useBean` 标签内。在这种方式中，`jsp:setProperty` 只会在新的对象被实例化时才将被执行。例如：

```
<jsp:useBean id="myUser" ... >
    ...
    <jsp:setProperty name="myUser" property="user" ... />
</jsp:useBean>
```

3.5.6 `jsp:getProperty` 动作

该动作提取指定 Bean 属性的值，并转换成字符串，然后输出。其语法如下：

```
<jsp:getProperty name="name" property="propertyName" />
```

该动作有两个必需的属性：

1. `name`： 表示 Bean 的名字；
2. `property`： 表示要提取哪个属性的值。

注意：

1. 在使用`<jsp:getProperty>`之前，必须用`<jsp:useBean>`来创建/获得 Bean 实例。
2. 不能使用`<jsp:getProperty>`来检索一个已经被索引了的属性。
3. 能够和 Java Bean 组件一起使用`<jsp:getProperty>`，但是不能与 Enterprise Java Bean 一起使用。

3.5.7 `jsp:plugin` 动作

该动作用来执行一个 applet 或 Bean，如果需要的话还要下载一个 Java 插件用于执行它。 其语法如下：

```
<jsp:plugin>
    type="bean | applet"
    code="classFileName"
    codebase="classFileDirectoryName"
    [ name="instanceName" ] [ archive="URIToArchive, ..." ]
    [ align="bottom | top | middle | left | right" ]
    [ height="displayPixels" ] [ width="displayPixels" ]
    [ hspace="leftRightPixels" ] [ vspace="topBottomPixels" ]
    [ jreversion="JREVersionNumber | 1.1" ]
    [ nspluginurl="URLToPlugin" ] [ iepluginurl="URLToPlugin" ] >
    [ <jsp:params>
    [ <jsp:param name="parameterName"
                value="{parameterValue | <%= expression %>}" /> ]
      </jsp:params> ]
    [ <jsp:fallback>text message for user</jsp:fallback> ]
</jsp:plugin>
```

3.6 JSP注释

在 JSP 页面中可以包含三种类型的注释：HTML 注释、JSP 代码注释和 Java 代码注释。

3.6.1 HTML注释

HTML 注释出现在 JSP 页面的 HTML 代码部分，它是 HTML 语言语法规定的原版注释，遵循 HTML 的语法规则。

在使用浏览器浏览 JSP 页面时，HTML 注释的内容不会在页面上显示，但是通过浏览器查看页面源代码时能够看到，也就是说 HTML 注释实质上是被发送到客户端，但是在呈现时被客户端屏蔽过滤了。

3.6.2 JSP代码注释

JSP 代码注释是在 JSP 页面中使用 JSP 注释标记描述的注释，语法格式为：`<%-- 注释内容 --%>`

这种注释不会被 JSP 编译器转换到 JSP 页面对应的 Java 源代码中，因此也不可能发送到客户端，所以在使用浏览器浏览 JSP 页面以及通过浏览器查看页面源代码时都不能看到这种注释。

3.6.3 Java代码注释

Java 代码注释出现在 JSP 页面的 Java 代码段中，也就是`<%`与`%>`之间的区域，遵循 Java 语言的注释规则。

这种注释能够被 JSP 编译器转换到 JSP 页面对应的 Java 源代码中，但是同样不会被发送到客户端，因此在使用浏览器浏览 JSP 页面以及通过浏览器查看页面源代码时都不能看到这种注释。

4 JSP内置对象

内置对象是 JSP 规范所定义的、由 Web 容器实现和管理的一些在 JSP 页面中都能使用的公共对象。这些对象只能在 JSP 页面的表达式或代码段中才可使用，在使用时不需要编写者进行实例化。

常见的内置对象包括：

1. 输出输入对象：`request`、`response`、`out`
2. 通信控制对象：`pageContext`、`session`、`application`
3. Servlet 对象：`page`、`config`
4. 错误处理对象：`exception`

4.1 `out`对象

`out`对象是`javax.servlet.jsp.jspWriter`类型的一个实例，它是一个输出流对象，作用域是 page，即当前 JSP 页面。用来向客户端输出数据。

`out`对象的方法说明如下：

方法	说明
`print()`或`println()`	输出数据
`newLine()`	输出换行字符
`flush()`	输出缓冲区数据
`close()`	关闭输出流
`clear()`	清除缓冲区中数据，但不输出到客户端

方法	说明
`clearBuffer()`	清除缓冲区中数据，并输出到客户端
`getBufferSize()`	获得缓冲区大小
`getRemaining()`	获得缓冲区中没有被占用的空间
`isAutoFlush()`	是否为自动输出

4.2 `request` 对象

`request` 对象是 `javax.servlet.http.HttpServletrequest` 类型的一个实例，它是一个用来表示请求信息的对象，作用域为 request。该对象封装了用户提交的信息，通过调用该对象相应的方法可以获取封装的信息，即使用该对象可以获取用户提交信息。

`request` 对象的方法说明如下：

方法	说明
`isUserInRole()`	判断认证后的用户是否属于某一成员组
`getAttribute()`	获取指定属性的值，如该属性不存在则返回 `null`
`getAttributeNames()`	获取所有属性名的集合
`getCookies()`	获取所有 `Cookie` 对象
`getCharacterEncoding()`	获取请求的字符编码
`getContentLength()`	返回请求正文的长度，如不确定则返回 -1
`getHeader()`	获取执行名字的报头值
`getHeaders()`	获取指定名字报头的所有值，以枚举的形式返回
`getHeaderNames()`	获取所有报头的名字，以枚举的形式返回
`getInputStream()`	返回请求输入流，获取请求中的数据
`getMethod()`	获取客户端向服务器端传送数据的方法
`getParameter()`	获取指定名字的参数值
`getParameterNames()`	获取所有参数的名字，以枚举的形式返回
`getParameterValues()`	获取指定名字参数的所有值
`getProtocol()`	获取客户端向服务器端传送数据的协议名称
`getQueryString()`	获取以 GET 方法向服务器端传送的查询字符串

方法	说明
`getRequestURI()`	获取发出请求字符串的客户端地址
`getRemoteAddr()`	获取客户端的 IP 地址
`getRemoteHost()`	获取客户端的名字
`getSession()`	获取和请求相关的会话
`getServerName()`	获取服务器的名字
`getServerPath()`	获取客户端请求文件的路径
`getServerPort()`	获取服务器的端口号
`removeAttribute()`	删除请求中的一个属性
`setAttribute()`	设置执行名字的参数值

4.3 `response` 对象

`response` 对象是 `javax.servlet.http.HttpServletResponse` 类型的一个实例，它是一个用来表示对客户端响应的对象，作用域是 page。此对象封装了返回到 HTTP 客户端的输出，向页面作者提供设置响应头标和状态码的方式。

`response` 对象的方法说明如下：

方法	说明
`addCookie()`	添加一个 `Cookie` 对象
`addHeader()`	添加 HTTP 文件指定名字头信息
`containsHeader()`	判断指定名字的 HTTP 文件头信息是否存在
`encodeURL()`	使用 sessionid 封装 URL
`flushBuffer()`	强制把当前缓冲区内容发送到客户端
`getBufferSize()`	返回缓冲区大小
`getOutputStream()`	返回到客户端的输出流对象
`sendError()`	向客户端发送错误信息
`sendRedirect()`	把响应发送到另一个位置进行处理
`setContentType()`	设置响应的 MIME 类型
`setHeader()`	设置指定名字的 HTTP 文件头信息

4.4 session 对象

session 对象是 javax.servlet.http.HttpSession 类型的一个实例，它是代表一个会话的对象，作用域是 session。此对象用来跟踪会话，它存在于 HTTP 请求之间，可以存储任何类型的命名对象。

从一个客户打开浏览器并连接到服务器开始，到客户关闭浏览器离开这个服务器结束，被称为一个会话。当一个客户访问一个服务器时，可能会在这个服务器的几个页面之间反复连接，反复刷新一个页面，服务器应当通过某种办法知道这是同一个客户，这就需要 session 对象。

当一个客户首次访问服务器上的一个 JSP 页面时，JSP 引擎产生一个 session 对象，同时分配一个 String 类型的 ID 号，JSP 引擎同时将这个 ID 号发送到客户端，存放在 Cookie 中，这样 session 对象和客户之间就建立了一一对应的关系。当客户再访问连接该服务器的其他页面时，不再分配给客户新的 session 对象，直到客户关闭浏览器后，服务器端该客户的 session 对象才取消，并且和客户的会话对应关系消失。当客户重新打开浏览器再连接到该服务器时，服务器为该客户再创建一个新的 session 对象。

如果不需要在请求之间跟踪会话对象，可以通过在 page 指令中指定 session="false" 。

session 对象的方法说明如下：

方法名	说明
getAttribute()	获取指定名字的属性
getAttributeNames()	获取 session 中全部属性名字，返回一个枚举
getCreationTime()	返回 session 的创建时间
getId()	获取会话标识符
getLastAccessedTime()	返回最后发送请求的时间
getMaxInactiveInterval()	返回 session 对象的生存时间，单位千分之一秒
invalidate()	销毁 session 对象
isNew()	该 session 对象是否是这次请求新产生的
removeAttribute()	删除指定名字的属性
setAttribute()	设定指定名字的属性值

4.5 pageContext 对象

pageContext 对象是 javax.servlet.jsp.PageContext 类型的一个实例，它代表的是页面上下文，作用域是 page。使用该对象可以访问页面中的共享数据。

pageContext 对象的方法如下所示：

方法名	说明
`forward()`	重定向到另一页面或 Servlet 组件
`getAttribute()`	获取某范围中指定名字的属性值
`findAttribute()`	按范围搜索指定名字的属性
`removeAttribute()`	删除某范围中指定名字的属性
`setAttribute()`	设定某范围中指定名字的属性值
`getException()`	返回当前异常对象
`getRequest()`	返回当前请求对象
`getResponse()`	返回当前响应对象
`getServletConfig()`	返回当前页面的 `ServletConfig` 对象
`getServletContext()`	返回所有页面共享的 `ServletContext` 对象
`getSession()`	返回当前页面的会话对象

4.6 `application` 对象

`application` 对象是 `javax.servlet.ServletContext` 类型的一个实例，它表示的是该 JSP 页面所在应用的上下文环境信息，作用域是 application。通过该对象能够获得应用在服务器中运行时的一些全局信息。

服务器启动后就产生了这个 `application` 对象，当客户在所访问的网站的各个页面之间浏览时，这个 `application` 对象都是同一个，直到服务器关闭。与 `session` 不同的是，所有客户的 `application` 对象都是同一个，即所有客户共享这个内置的 `application` 对象。

`application` 对象的方法如下所示：

方法名	说明
`getAttribute()`	获取应用对象中指定名字的属性值
`getAttributeNames()`	获取应用对象中所有属性的名字，返回一个枚举
`getInitParameter()`	返回应用对象中指定名字的初始参数值
`getServletInfo()`	返回 Servlet 编译器中当前版本信息
`setAttribute()`	设置应用对象中指定名字的属性值

4.7 `config` 对象

`config` 对象是 `javax.servlet.ServletConfig` 类型的一个实例，它表示的是该 JSP 页面的配置信息，作用域是 page。

事实上， JSP 页面通常无须配置，也就不存在配置信息。因此，该对象更多地在 Servlet 中有效。

`config` 对象的方法如下：

方法名	说明
`getServletContext()`	返回所执行的 Servlet 的环境对象
`getServletName()`	返回所执行的 Servlet 的名字
`getInitParameter()`	返回指定名字的初始参数值
`getInitParameterNames()`	返回该 JSP 中所有的初始参数名，返回一个枚举

4.8 `page` 对象

`page` 对象是 `java.lang.Object` 类型的一个实例，它代表该 JSP 页面被编译成 Servlet 类的实例对象，作用域是 page。它与 Servlet 类中的 `this` 关键字相对应，可以使用它来调用 Servlet 类中所定义的方法，但是一般很少使用。

4.9 `exception` 对象

`exception` 对象是 `java.lang.Throwable` 类型的一个实例，它代表其他页面中的异常和错误，作用域是 page。只有当页面是错误处理页面，即编译指令 `page` 的 `isErrorPage` 属性为 `true` 时，该对象才可以使用。

5 JSP页面调用Servlet

从 JSP 页面调用 Servlet 可以通过 form 表单的提交、 `jsp:include` 动作、 `jsp:forward` 动作以及使用 `a` 标签的 `href` 属性等方法来实现。

1. 通过 `form` 标签的 `action` 属性： `<form action="url" method="POST">`
2. 通过 `jsp:include` 动作： `<jsp:include page="url" />`
3. 通过 `jsp:forward` 动作： `<jsp:forward page="url" />`
4. 使用 `a` 标签的 `href` 属性： `<a href="url">`

6 JSP页面调用Java Bean

用户可以使用 Java Bean 将功能、处理、值、数据库访问和其他任何可以用 Java 代码创造的对象进行打包，并且其他的开发者可以通过内部的 JSP 页面、Servlet、其他 Java Bean、applet 程序或者应用来使用

这些对象。

一个 Java Bean 和一个 Java applet 相似，是一个非常简单的遵循某种严格协议的 Java 类。每个 Java Bean 的功能都可能不一样，但它们都必须支持以下特征：

1. 如果类的成员变量的名字是 `xxx`，那么为了更改或获取成员变量的值，在类中可以使用两个方法：`getXxx()` 用来获取属性 `xxx`，`setXxx()` 用来修改属性 `xxx`。
2. 对于 `boolean` 类型的成员变量，允许使用 "is" 代替上面的 "get"。
3. 类中方法的访问属性都必须是 `public` 的。
4. 类中如果有构造方法，那么这个构造方法也是 `public` 的并且没有参数。

Java Bean 的简单示例：

```
public class SimpleBean
{
    // 无参构造方法
    SimpleBean (){}

    private String name; // 定义 String 类型的简单属性 name
    private boolean info;

    // 简单属性的 getXxx() 方法
    public String getName()
    {
        return name;
    }

    // 简单属性的 setXxx() 方法
    public void setName(String name)
    {
        this.name = name;
    }

    // 布尔类型的取值方法
    public boolean isInfo()
    {
        return info;
    }

    //布尔类型的 setXxx() 方法
    public void setInfo(boolean info)
    {
        this.info = info;
    }
}
```

在 JSP 中可以使用 `<jsp:useBean>`、`<jsp:setProperty>`、`<jsp:getProperty>` 这三个动作来完成对 Java Bean 的调用。

`<jsp:useBean>` 动作用来将一个 Java Bean 的实例引入到 JSP 中，并且使得这个实例具有一定生存范围，在这个范围内还具有一个唯一的 id。这样 JSP 通过 id 来识别 Java Bean，并通过 `id.method` 类似的语句来调

用 Java Bean 中的公共方法。 在执行过程中，`<jsp:useBean>`首先会尝试寻找已经存在的具有相同`id`和`scope`值的 Java Bean 实例，如果没有就会自动创建一个新的实例。

`<jsp:setProperty>`动作主要用于设置 Bean 的属性值。

`<jsp:getProperty>`动作用来获得 Java Bean 实例的属性值，并将他们转换为`java.lang.String`类型，最后放置在隐含的`out`对象中。

注意：Java Bean 的实例必须在`<jsp:getProperty>`前面定义。

下面的代码展示了 JSP 调用 Java Bean 实现用户登录功能的示例。

```
<!-- index.html -->
<form action="login.jsp" method="post">
    用户名：<input type="text" name="userName"><br>
    密　码：<input type="password" name="passWord"><br>
    <input type="submit" value="登陆" name="login">
</form>
```

```
<!-- login.jsp -->
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<jsp:useBean id="userInfo" scope="page" class="javaee.jsp.UserInfo">
</jsp:useBean>
<jsp:setProperty name="userInfo" property="*"/>
......
<body>
    用户："<%= userInfo.getUserName() %> "进行登陆操作<br>
    <%
    if(userInfo.isAdmin())
        out.println("成功登陆");
    else
        out.println("登陆失败");
    %>
</body>
</html>
```

```java
// UserInfo.java
public class UserInfo
{
    private String userName;
    private String passWord;

    public String getUserName()
    {
        return userName;
    }

    public void setUserName(String userName)
    {
        this.userName = userName;
    }

    public String getPassWord()
    {
        return passWord;
    }

    public void setPassWord(String passWord)
    {
        this.passWord = passWord;
    }

    public boolean isAdmin()
    {
        if(userName.equals("admin") && passWord.equals("admin"))
            return true;
        else
            return false;
    }
}
```

7 EL表达式

EL 是 Expression Language 的缩写。在 EL 表达式出现之前，开发 Java Web 应用程序时，经常需要将大量的 Java 代码片段嵌入 JSP 页面中，这样会使代码比较乱。而使用 EL 表达式则比较简洁。

EL 表达式的目的：使 JSP 写起来更加简单。

EL 表达式的语法：以 `${` 开头，以 `}` 结束，中间为合法的表达式。具体语法格式如下：

```
${expression}
```

大括号中的表达式指明了要输出的内容，有效的表达式可以包含字符串、操作符、变量（对象引用）和函数调用。

EL 表达式中的变量可以是任何存在于 JSP 作用范围的 Java Bean 对象，它们通常是使用`jsp:useBean`动作来定义的。

EL 存取变量数据的方法：例如`${username}`，表示取出某一范围中名称为`username`的变量。如果不指定范围，默认按照 page、request、session、application 的范围顺序查找。如果途中找到，就直接回传，不再继续找下去；如果全部的范围都没有找到，则回传`null`。

为了能够方便地完成取值的操作，EL 中定义了一些默认变量（隐含变量）：

1. `pageScope`、`requestScope`、`sessionScope`、`applicationScope`：包含相应作用范围的参数集合，相当于被保存在`java.util.Map`中的某个参数。使用参数名访问相应的参数，例如`${sessionScope.sampleValue}`表示获得 session 中的`sampleValue`属性的值。
2. `param`、`paramValues`：用于获取通过表单提交的请求中的参数，`param`表明请求包含的参数为单一控件，`paramValues`表明请求包含的参数为控件数组。
3. `header`、`headerValues`：包含请求参数头部信息的集合，`header`表示单一头部信息，`headValues`表示数组型的头部信息。
4. `cookie`：包含所有请求的 cookie 集合，集合中的每个对象对应`javax.servlet.http.Cookie`类型。
5. `initParam`：包含所有应用程序初始化参数的集合。
6. `pageContext`：等价于 page 环境类`javax.servlet.jsp.PageContext`的实例，用来提供访问不同的请求参数。

8 JSTL标签库

通过`taglib`指令来使用标签库，标签库中丰富的标签能够代替传统的 Java 片段语言来实现页面的显示逻辑。然而，自定义标签很容易造成重复定义和非标准的实现。鉴于此，出现了 JSTL（JSP Standard Tag Library，JSP 标准标签库）。

JSTL 1.1 标签库可以分为以下几类：

1. 核心标签库：包含 Web 应用的常见工作，如循环、表达式赋值、基本输入输出等。
2. 国际化标签库：用来格式化显示数据的工作，比如对不同区域的日期格式化等。
3. 数据库标签库：可以做访问数据库的工作。
4. XML 标签库：用来访问 XML 文件的工作。这是 JSTL 标签库的一个特点。
5. 函数标签库：用来读取已经定义的某个函数。

标签库	标签库的标识符（URI）	前缀
Core	http://java.sun.com/jsp/jstl/core	c
I18N formatting	http://java.sun.com/jsp/jstl/fmt	fmt
Database access	http://java.sun.com/jsp/jstl/sql	sql
XML processing	http://java.sun.com/jsp/jstl/xml	x

标签库	标签库的标识符（URI）	前缀
Functions	http://java.sun.com/jsp/jstl/functions	fn

在使用标签之前，首先需要使用标签库的标识符和前缀，将标签所在的标签库引入 JSP 页面中。例如，下面的代码中声明了将使用 Core 标签库，之后页面中 `<c:forEach>` 标签就是使用了 JSTL 的标签进行了工作。

```
<%@ page contextType="text/html; charset=UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html>
    <body>
        <c:forEach var="i" begin="1" end="10" step="1">
            ${i}<br>
        </c:forEach>
    </body>
</html>
```

8.1 核心标签库

Core 标签库，又称为核心标签库，对 JSP 页面一般处理的封装。

多用途核心标签：`<c:out>`、`<c:set>`、`<c:remove>`、`<c:catch>`

条件控制标签：`<c:if>`、`<c:choose>`、`<c:when>`、`<c:otherwise>`

循环控制标签：`<c:forEach>`、`<c:forTokens>`

URL 相关标签：`<c:import>`、`<c:url>`、`<c:redirect>`、`<c:param>`

标签	描述
`<c:out>`	用于在 JSP 中显示数据，就像 `<%=...%>`
`<c:set>`	用于保存数据
`<c:remove>`	用于删除数据
`<c:catch>`	用来处理产生错误的异常情况，并且将错误信息储存起来
`<c:if>`	与一般程序中的 `if` 一样
`<c:choose>`	只作为 `<c:when>` 和 `<c:otherwise>` 的父标签
`<c:when>`	`<c:choose>` 的子标签，用来判断条件是否成立
`<c:otherwise>`	`<c:choose>` 的子标签，接在 `<c:when>` 之后，当 `<c:when>` 判断为 `false` 时被执行

标签	描述
`<c:forEach>`	基础迭代标签，接受多种集合类型
`<c:forTokens>`	根据指定的分隔符来分隔内容并迭代输出
`<c:import>`	检索一个绝对或相对 URL，然后将其内容暴露给页面
`<c:url>`	使用可选的查询参数来创造一个 URL
`<c:redirect>`	重定向至一个新的 URL
`<c:param>`	用来给包含或重定向的页面传递参数

8.2 国际化标签库

JSTL 国际化标签用来格式化并输出文本、日期、时间、数字。

标签	描述
`<fmt:formatNumber>`	使用指定的格式或精度格式化数字
`<fmt:parseNumber>`	解析一个代表数字、货币或百分比的字符串
`<fmt:formatDate>`	使用指定的风格或模式格式化日期和时间
`<fmt:parseDate>`	解析一个代表日期或时间的字符串
`<fmt:bundle>`	绑定资源
`<fmt:setLocale>`	指定地区
`<fmt:setBundle>`	绑定资源
`<fmt:timeZone>`	指定时区
`<fmt:setTimeZone>`	指定时区
`<fmt:message>`	显示资源配置文件信息
`<fmt:requestEncoding>`	设置 `request` 的字符编码

8.3 数据库标签库

JSTL 数据库标签库提供了与关系型数据库进行交互的标签。对于早期纯 JSP 开发的应用以及小型的开发有着重大意义，但对于 MVC 模型来说却是违反规范的。

标签	描述
`<sql:setDataSource>`	指定数据源
`<sql:query>`	运行 SQL 查询语句
`<sql:update>`	运行 SQL 更新语句
`<sql:param>`	将 SQL 语句中的参数设定为指定值
`<sql:dateParam>`	将 SQL 语句中的日期参数设为指定的`java.util.Data`对象值
`<sql:transaction>`	在共享数据库连接中提供嵌套的数据库行为元素， 将所有语句以一个事务的形式来运行

例如：

```
<sql:setDataSource
    var="dataSrc"
    url="jdbc:postgresql://localhost:5432/myDB"
    driver="org.postgresql.Driver"
    user="admin"
    password="1111" />

<sql:query var="queryResults" dataSource=${dataSrc}>
    select * from table1
</sql:query>

<c:forEach var="row" items="${queryResults.rows}">
    <tr>
        <td>${row.userName}</td>
        <td>${row.passWord}</td>
    </tr>
</c:forEach>
```

8.4 XML标签库

JSTL XML 标签库提供了创建和操作 XML 文档的标签。

标签	描述
`<x:out>`	与`<%=...>`类似，不过只用于 XPath 表达式
`<x:parse>`	解析 XML 数据
`<x:set>`	设置 XPath 表达式
`<x:if>`	判断 XPath 表达式，若为真则执行本体中的内容，否则跳过本体
`<x:forEach>`	迭代 XML 文档中的节点

标签	描述
`<x:choose>`	`<x:when>`和`<x:otherwise>`的父标签
`<x:when>`	`<x:choose>`的子标签，用来进行条件判断
`<x:otherwise>`	`<x:choose>`的子标签，当`<x:when>`判断为`false`时被执行
`<x:transform>`	将 XSL 转换应用在 XML 文档中
`<x:param>`	与`<x:transform>`共同使用，用于设置 XSL 样式表

8.5 函数标签库

JSTL 包含一系列标准函数，大部分是通用的字符串处理函数。在 JSP 2.0 规范下出现的函数标签库为 EL 表达式语句提供了许多更为有用的功能。

函数	描述
`fn:contains()`	测试输入的字符串是否包含指定的子串
`fn:containsIgnoreCase()`	测试输入的字符串是否包含指定的子串，大小写不敏感
`fn:endsWidth()`	测试输入的字符串是否以指定的后缀结尾
`fn:escapeXml()`	跳过可以作为 XML 标记的字符
`fn:indexOf()`	返回指定字符串在输入字符串中出现的位置
`fn:join()`	将数组中的元素合成一个字符串然后输出
`fn:length()`	返回字符串长度
`fn:replace()`	将输入字符串中指定的位置替换为指定的字符串然后返回
`fn:split()`	将字符串用指定的分隔符分隔，组成一个子字符串数组并返回
`fn:startsWidth()`	测试输入的字符串是否以指定的前缀开始
`fn:substring()`	返回字符串的子串
`fn:substringAfter()`	返回字符串在指定子串之后的子集
`fn:substringBefore()`	返回字符串在指定子串之前的子集
`fn:toLowerCase()`	将字符串中的字符转为小写
`fn:toUpperCase()`	将字符串中的字符转为大写
`fn:trim()`	移出首位的空白符