第 3 章

JSP 视图

JSP 与静态网页 HTML 对应，表示 Jakarta 服务器页面。需要注意的是，它不是指运行在 Web 服务器的页面，而是在服务器端定义页面的样式，然后由 Servlet 容器解析成 HTML 后，回应给客户端浏览器显示。

Jakarta EE 10 中包含的 JSP 版本是 3.1。

参见 hello.jsp 的代码，看一下 JSP 的页面组成，其基本组成就是 Java 代码＋HTML＋JS。

```
<%@ page language="java" contentType="text/html;
            charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%
    String msg = (String)request.getAttribute("msg");
 %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="ISO-8859-1">
        <title>hello</title>
        <script type="text/javascript"></script>
    </head>
    <body>
        <% =msg %>
    </body>
</html>
```

视频讲解

3.1 JSP 与 Servlet 的关系

所有 JSP 页面的默认父类是 org.apache.jasper.runtime.HttpJspBase，而 HttpJspBase 又继承了 HttpServlet，因此所有 JSP 页面的本质就是 Servlet。它与自定义 Servlet 一样，由 Servlet 容器统一管理。

```
public abstract class HttpJspBase extends HttpServlet
                                implements HttpJspPage { … }
```

JSP 容器就是 Servlet 容器，它提供了 JSP 的生命期管理、JSP 运行期支持和 Servlet 组件管理。HTTP 请求通过 JSP 容器发送给 JSP 页面，JSP 容器管理 JSP 页面生命期，二次访问 JSP 分为如下几个阶段（见图 3-1 和图 3-2）。

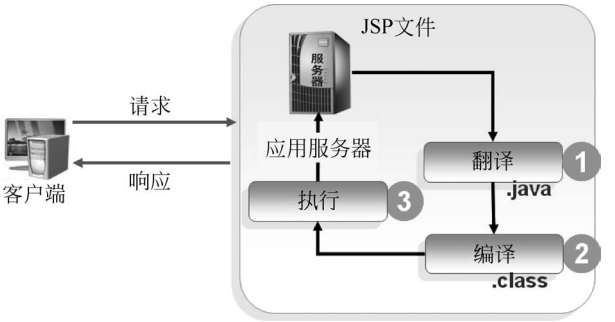

图 3-1 第一次访问 JSP

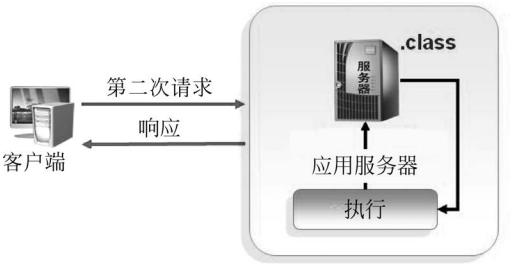

图 3-2 第二次访问 JSP

（1）翻译阶段：JSP 容器校验 JSP 页面、标签文件、Java 脚本等语法拼写，然后把 JSP 页面自动翻译成 Java 代码，如 hello.jsp 就会翻译成 hello_jsp.java。翻译后的文件保存在 Tomcat 的 work 文件夹下（\work\Catalina\localhost\Hello\org\apache\jsp\main\hello_jsp.java），翻译后的代码如下：

```
public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
                        implements org.apache.jasper.runtime.JspSourceDependent,
                        org.apache.jasper.runtime.JspSourceImports { … }
```

（2）编译阶段：JSP 容器把翻译好的 Java 文件，自动编译成 class 文件，如 hello_jsp.java 编译成 hello_jsp.class。JSP 页面的翻译与编译处理，在第一次访问 JSP 页面时进行。如果项目运行期间 JSP 页面没有变化，则第二次访问 JSP 页面无须再次翻译和编译。

（3）执行阶段：JSP 容器调用编译好的 JSP 组件，完成创建、初始化、服务、释放等操作，JSP 组件的生命期管理与前面讲的 Servlet 生命期管理一致。JSP 的运行结果为 HTML，即返回给客户端浏览器的是 HTML 文本信息（见图 3-3）。

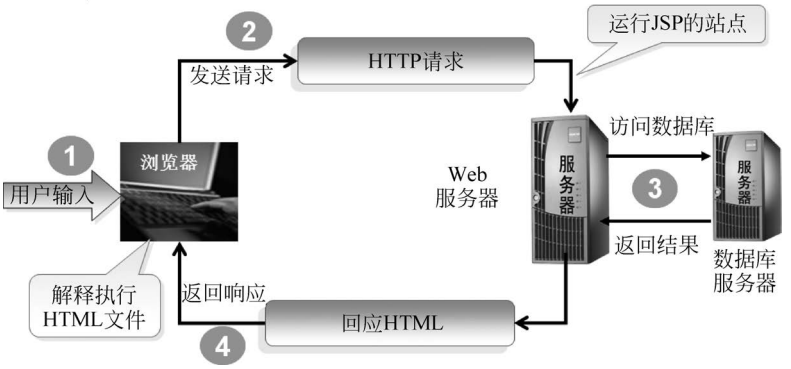

图 3-3 JSP 输出 HTML

视频讲解

3.2 JSP 指令

3.2.1 Page 指令

每个 JSP 页面的头部必须使用 Page 指令声明当前页面特性。

Page 指令定义了一系列依赖于 JSP 容器交互的属性信息(见表 3-1)。

表 3-1 Page 指令属性

属 性	描 述
language	指定当前页面使用的服务器脚本语言 <%@ page language="java" %>
contentType	设置 HTTP 的 Content-Type 响应头,指明将发送到客户程序的文档的 MIME 类型 <%@ page contentType="text/html" %>
pageEncoding	设置当前 JSP 页面的编码格式 <%@ page pageEncoding="utf-8" %>
import	指明服务器脚本中使用的类所在包,与业务类中的 import 含义相同,如 <%@ page import="java.util.*, com.icss.biz.*" %> 默认情况下,servlet 导入 java.lang.*、jakarta.servlet.*、jakarta.servlet.jsp.*、jakarta.servlet.http.* 等包
buffer	设置 out 变量(类型为 JspWriter)使用的缓冲区大小 <%@ page buffer="none" %> <%@ pagebuffer="16Kb" %>
autoFlush	用来控制当缓冲区充满之后,是应该自动清空输出缓冲区(默认),还是在缓冲区溢出后抛出一个异常 <%@ pageautoFlush ="false" %>

续表

属　性	描　述
errorPage	errorPage 属性用来指定一个 JSP 页面，由该页面来处理当前页面中抛出但未被捕获的任何异常 <%@ page errorPage = "/oops.jsp" %>
isErrorPage	表示当前页是否可以作为其他 JSP 页面的错误页面 <%@ page isErrorPage = "false" %>
…	其他不常用属性

例如：

```
<%@ page language = "java" import = "com.icss.biz.*,com.icss.util.*"
         contentType = "text/html"  pageEncoding = "utf-8" %>
```

3.2.2 taglib 指令

taglib 简称标签库，是一套 Java EE 已经定义好的标签库，根据需要引用即可。如果 Java EE 标签库不能满足你的需要，也可以根据业务定义自己的标签库。

JSTL 即 JSP 标准标签库，具体使用见第 5 章。

例如：

```
<%@ taglib uri = "jakarta.tags.sql" prefix = "sql" %>
<%@ taglib uri = "jakarta.tags.core" prefix = "c" %>
<%@ taglib uri = "jakarta.tags.fmt" prefix = "fmt" %>
```

3.2.3 include 指令

include 指令用于在当前 JSP 页面中替换文本或代码数据。include 指令的作用与<jsp:include>元素的作用基本一致。

例如：

```
<%@ include file = "../public/head.html" %>
```

等效于

```
<jsp:include page = "../public/head.jsp"></jsp:include>
```

如果 include 文件发生了变化，JSP 容器可以接到变化通知，然后会重写编译变化的 JSP 文件。JSP 文件允许在项目运行期间动态修改 JSP 文件，修改后 Tomcat 无须重新启动。

注意：include 指令与<jsp:include>元素中都要使用相对路径，不要使用绝对路径。

include 指令与<jsp:include>元素的区别见表 3-2。

表 3-2 include 指令与<jsp:include>元素对比

语　法	嵌入内容	对　象	描　述
<%@include file=... %>	相对 file 的路径	静态	JSP 容器解析嵌入内容
<jsp:include page= />	相对 page 的路径	静态或动态	内容不被提前解析，在 Servlet 运行期动态嵌入


视频讲解

3.3 JSP 中的 Java 元素

用一对<%　%>包裹的内容，不属于 HTML 和 JavaScript 数据，可以统称为服务器 Java 元素信息。这些元素分别为：

- 指令：<%@ 指令 %>，见 3.2 节。
- 服务端注释：<% --……--%>，注释后的内容 JSP 容器不做处理。
- Java 表达式输出：<% =Java 表达式 %>，在 JSP 中输出表达式结果，如

```
<% = msg %>
<% = (new java.util.Date()).toLocaleString() %>
```

- Java 脚本：<% Java 代码 %>，如同在 Servlet 中编写代码一样。
- Java 声明：<% !Java 方法或变量 %>，这种模式尽量不要使用，JSP 的定位是视图，业务方法应该抽取到服务层。

3.3.1 Java 脚本与表达式

Java 脚本元素(<% Java 代码 %>)与 Java 表达式元素(<% =Java 表达式 %>)经常协同操作，即在 Java 脚本中进行动态计算，在 Java 表达式中输出运算结果。

示例 3-1

动态获取 Web 站点绝对路径，并在<img>中使用绝对路径显示图片信息。

```
<%@ page language="java" import="java.util.*"
         contentType="text/html; charset=utf-8" %>
<%
    String path = request.getContextPath();
    String basePath = request.getScheme()+"://"+request.getServerName()
            +":"+request.getServerPort()+path+"/";
%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="ISO-8859-1">
        <title>Insert title here</title>
```

```
    </head>
    <body>
        <img src="<%=basePath%>main/flex.png">
    </body>
</html>
```

示例 3-2

在 Java 脚本中进行简单计算，并在 JSP 页面中输出计算结果。

```
<%@ page language="java" import="java.util.*"
         contentType="text/html; charset=utf-8" %>
<html>
    <head><title>计算圆的面积</title></head>
    <body>
        <%
            double r = 5;
            double PAI = 3.1415926;
            double area = PAI*r*r;
        %>
        圆的周长为 <%=r%>  <br>
        圆的面积为 <%= area %>
    </body>
</html>
```

3.3.2 Java 声明

Java 声明元素：

```
<%!Java 方法或变量 %>
```

可以声明变量或方法，在 JSP 的页面范围均可调用声明的变量或方法。

声明信息不会在当前输出流中产生任何输出。

声明在 JSP 页面初始化时被初始化，在其他声明、Java 脚本、Java 表达式中都有效。

示例 3-3

示例中声明了一个整数变量 i，这个变量在 page 页面范围内全局有效。

```
<%! int i=0; %>
<!DOCTYPE html>
<html>
    <head>
        <title>hello</title>
    </head>
```

```
    <body>
        i= <%= i=i+5 %>
    </body>
</html>
```

示例 3-4

```
<%@ page language="java" contentType="text/html"
    pageEncoding="utf-8" import="java.util.*,java.text.*" %>
<%!
    public String today(Date date){
        SimpleDateFormat sd = new SimpleDateFormat("yyyy-MM-dd");
        return sd.format(date);
    }
%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <%= today(new Date()) %>
    </body>
</html>
```

3.3.3 JSP 中使用注释

1. HTML 与 XML 中的注释

在 HTML 和 XML 中的注释语法是：

```
<!-- comments … -->
```

使用这种注释，注释信息会随着 response 输出流与 JSP 其他信息一起输出到客户端。在 JSP 中也支持这种注释的使用。

示例 3-5

```
<body>
    <!-- 显示打招呼信息 -->
    <%= msg %>
</body>
```

在注释中，使用 Java 脚本、表达式、声明等，JSP 容器仍然会处理，语法如下：

```
<!-- 注释信息 <% = 表达式 %> 其他注释 … -->
```

示例 3-6

```
<%@ page language = "java" import = "com.icss.biz. * ,com.icss.util. * "
    contentType = "text/html"  pageEncoding = "utf - 8" %>
<%
    String msg = (String)request.getAttribute("msg");
    String uname = request.getParameter("uname");
 %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf - 8">
        <title> hello </title>
    </head>
    <body>
        <!-- 显示<% = uname %>打招呼信息  -->
        <% = msg %>
    </body>
</html>
```

2. JSP 中的注释

在 JSP 页面中设计了专门的元素用于注释，这种注释与 HTML 注释的区别：注释内容不会被 response 输出流写到客户端。

JSP 注释语法如下：

```
<% --  注释信息 -- %>
```

用 Java 脚本可以替换上面的 JSP 注释：

```
<% /**  注释信息 … **/ %>
```

示例 3-7

```
<body>
        <!-- 显示<% = uname %>打招呼信息  -->
        <% -- 显示<% = uname %>打招呼信息  -- %>
        <% /** 显示打招呼信息 … **/ %>
        <% = msg %>
</body>
```


视频讲解

3.4 JSP 的 9 个内置对象

JSP 页面中可以使用 9 个内置对象，分别为 request、response、session、application、out、pageContext、config、page、exception(见表 3-3)。

表 3-3 JSP 内置对象

内 置 对 象	Java 类型	描 述
request	jakarta. servlet. http. HttpServletRequest	接收 HTTP 请求数据
response	jakarta. servlet. http. HttpServletResponse	回应 HTTP 请求
session	jakarta. servlet. http. HttpSession	会话对象
application	jakarta. servlet. ServletContext	存储全局数据
out	jakarta. servlet. jsp. JspWriter	输出文本
pageContext	jakarta. servlet. jsp. PageContext	当前页面的上下文对象
page	java. lang. Object	page = this，表示当前页面对象
config	jakarta. servlet. ServletConfig	用于读取初始化配置参数
exception	java. lang. Throwable	在<%@ page isErrorPage="true"%>的页面中才能使用，用于输出异常信息

注：本节只重点演示几个常用内置对象的使用，其他内容请参考 JSP 3.1 规范。

JSP 内置对象在页面翻译阶段，会转换成相应的 Java 类型。这些对象的创建与释放，都是由 JSP 容器自动管理，开发人员无须关注。

3.4.1 request 与 response 对象

request 对象是 jakarta. servlet. http. HttpServletRequest 接口的实例，参见 2.5 节 ServletRequest 接口。它是 JSP 中使用次数最多的内置对象，常用于获取 Web 站点名称、站点端口号、主机名、HTTP 请求信息、分发器等。

response 对象是 jakarta. servlet. http. HttpServletResponse 接口的实例，参见 2.6 节 ServletResponse 接口。response 对象在 JSP 页面应用并不多，主要应用于 Servlet 中。

示例 3-8

获取 Hello 项目的站点名。

```
<%
    out.println("path = "   +   request.getContextPath());
    out.println("path2 = "  +  application.getContextPath());
%>
```

输出结果：path=/Hello，path2=/Hello。

总结

参见 HttpServletRequest 和 ServletContext 的 API，分别调用 request.getContextPath() 和 application.getContextPath()可以获得当前站点名称。

示例 3-9

获得 Web 站点绝对地址。

```
<%
    String path = request.getContextPath();
    String basePath = request.getScheme() + "://" + request.getServerName()
                    + ":" + request.getServerPort() + path + "/";
%>
<body>
    <img src="<%=basePath%>pic/flex.png">
</body>
```

总结

request 对象是 HTTP 请求的封装，根据 HTTP 请求，可以动态解析当前站点的站点名称、IP、主机名、端口、HTTP 方法名、HTTP 参数、HTTP 的头信息、HTTP 的 ContentType 等很多信息。

示例 3-10

默认主页设置。

(1) 在 Hello 项目的 WebContent 根下新建 index.jsp。

(2) 在 web.xml 中配置项目启动时的欢迎页。

```
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

(3) 默认页 index.jsp 不显示任何信息，转向真正的主页 main.jsp。

```
<%@ page language="java" contentType="text/html;" pageEncoding="GBK"%>
<%
    request.getRequestDispatcher("/main/main.jsp").forward(request, response);
%>
```

(4) 浏览器访问 http://localhost:8080/Hello/,会自动转到主页 main.jsp。

总结

默认主页的设置,在 Jakarta Web 项目很常见,如上代码使用 request 对象获取了 RequestDispatcher 对象,然后调用分发器的 forward()方法实现了服务器端页面的跳转。


视频讲解

3.4.2 pageContext 对象

pageContext(jakarta.servlet.jsp.PageContext)对象继承 JspContext 类,给 JSP 组件在 Servlet 环境下运行提供了上下文环境支持。

通过 pageContext 可以获取访问当前页面的其他 JSP 内置对象:request 对象、response 对象、session 对象、application 对象、config 对象、out 对象、exception 对象、page 对象。

- public abstract jakarta.servlet.ServletRequest getRequest()
  返回 request 对象。
- public abstract jakarta.servlet.ServletResponse getResponse()
  返回 response 对象。
- public abstract jakarta.servlet.http.HttpSession getSession()
  返回 session 对象。
- public abstract jakarta.servlet.ServletContext getServletContext()
  返回 application 对象。
- public abstract jakarta.servlet.ServletConfig getServletConfig()
  返回 config 对象。
- public abstract java.lang.Exception getException()
  返回 exception 对象。
- public abstract java.lang.Object getPage()
  返回 page 对象。

示例 3-11

修改索引页 index.jsp,实现页面转向。

```
<%@ page language="java" contentType="text/html" pageEncoding="gbk" %>
<%
    pageContext.forward("/main/main.jsp");
%>
```

示例 3-12

在 EL 表达式或 Java 表达式中获取绝对路径。

```
<body>
    <img src="${pageContext.request.contextPath}/pic/flex.png">
```

等效于

```
    <img src="<%=request.getContextPath()%>/pic/flex.png">
</body>
```

3.4.3 session 与 application 对象

session 和 application 对象的使用详细介绍见第 6 章。

3.5 <jsp:>标准动作

在 JSP 页面中使用<jsp:>开头的 XML 元素，统称 JSP 标准动作(Action)。

JSP 标准动作有<jsp:useBean>、<jsp:setProperty>、<jsp:getProperty>、<jsp:include>、<jsp:forward>、<jsp:param>、<jsp:plugin>、<jsp:params>、<jsp:fallback>、<jsp:attribute>、<jsp:body>、<jsp:invoke>、<jsp:doBody>、<jsp:element>、<jsp:text>、<jsp:output>、<jsp:root>、<jsp:declaration>、<jsp:scriptlet>、<jsp:expression>等，下面着重介绍常用的几个标准动作，其他内容参见 JSP 3.1 规范。

3.5.1 <jsp:useBean>


视频讲解

<jsp:useBean>的作用如下：

- 用 ID 值在指定的 scope 范围查找指定类型的 Java 对象。
- 如果对象存在，直接返回对象引用，否则实例一个新对象。
- 在 Java 脚本中声明一个 ID 值的变量，指向找到的对象。

简化的基本语法如下：

```
<jsp:useBean  id="变量名"
              scope="page|request|session|application"
              class="包名.类名"  />
```

示例 3-13

(1) 在 com.icss.entity 下定义实体类 User。

```
public class User {
    private String uname;
    private String pwd;
    private int role;
    private double score;
    public User() {
    }
    public User(String uname) {
        this.uname = uname;
    }
}
```

(2) 在 JSP 页面使用<jsp:useBean>创建 bean 对象。

注意 User 必须有一个无参构造器,否则调用时报错。

```
<body>
    <jsp:useBean id="myUser" class="com.icss.entity.User"/>
    <jsp:setProperty name="myUser" property="score" value="85"/>
    ${myUser.score + 3}
</body>
```

输出结果:88.0。

JavaBean 有如下特点:

- 不能是非静态内部类。
- 必须拥有一个无参的构造器或者@Inject 注解的构造器。

示例 3-14

修改示例 3-13 的代码,首先在 Java 脚本中创建 User 对象,然后存储于 request 域对象中。<jsp:useBean>必须指明查找范围是 request,否则默认从 page 中查找。

```
<body>
    <%
        User user = new User("tom");
        user.setScore(85);
        request.setAttribute("myUser", user);
    %>
    <jsp:useBean id="myUser" class="com.icss.entity.User" scope="request"/>
    ${myUser.score + 3}
</body>
```

输出结果：88.0。

如果没有设置 scope="request"，从 page 中找不到 myUser，输出结果为 3.0。


视频讲解

3.5.2 <jsp:setProperty>与<jsp:getProperty>

<jsp:setProperty>设置 bean 对象的属性值，简化的基本语法如下。

```
<jsp:setProperty  name = "bean 的名字"
                  property = "bean 的属性名"
                  value = "属性值"
                  param = "请求参数名" />
```

value 用于给属性直接赋值，param 从请求中提取参数值给属性赋值，value 与 param 不能同时使用。

<jsp:getProperty>为读取 bean 对象的属性值，简化的基本语法如下：

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

注意：<jsp:getProperty>没有 scope 属性，它调用 pageContext 的 findAttribute()方法进行范围查找，即按照 page、request、session、application 的顺序查找。如果未找到指定名称的 bean 对象，则抛出异常。

示例 3-15

(1) 在 page 域中查找 myUser，没找到就实例化一个 User 对象。

(2) 设置 myUser 对象的属性 score=85。

(3) 读取名字为 myUser 的 bean 对象，并在页面输出。

```
<body>
    <jsp:useBean id = "myUser" class = "com.icss.entity.User"/>
    <jsp:setProperty name = "myUser" property = "score" value = "85"/>
    <jsp:getProperty name = "myUser" property = "score"/>
</body>
```

输出结果：85.0。

3.5.3 <jsp:include>

在 JSP 当前页面，嵌入静态或动态资源(必须是当前 Web 应用的资源，不能是外部资源)。嵌入资源时，应该使用相对路径，这会被翻译成基于当前 context 的绝对路径。

<jsp:include>在 JSP 页面翻译后，在 Servlet 运行过程动态导入资源。<jsp:include>属性见表 3-4。

表 3-4 <jsp:include>属性

属 性 名	描 述
page	嵌入 URL 资源的相对路径
flush	可选属性，boolean 值，如果 flush 为真，缓冲区马上刷新。默认值是 false

<jsp:include>与<c:import>功能相似，区别见 5.2.4 节中<c:import>部分内容。

<jsp:include>的基本语法如下：

```
<jsp:include  page=" 相对路径指向嵌入页面"  flush="true|false"/>
```

示例 3-16

在当前 JSP 页面动态嵌入一个共享的头文件。

```
<body>
    <jsp:include page="/main/head.jsp"></jsp:include>
</body>
```

3.5.4 <jsp:forward>

<jsp:forward>与 RequestDispatcher 的 forward()方法功能一致，就是在 Web 应用运行期，分发请求到当前 Web 应用的静态资源、JSP 页面或 Servlet 类。

基本语法如下：

```
<jsp:forward  page="相对路径" />
```

示例 3-17

从项目索引页 index.jsp 转向主页。

```
<body>
    <jsp:forward page="/main/main.jsp"></jsp:forward>
</body>
```

3.6 本章习题

(1)(　　)可在 JSP 页面出现该指令的位置处静态插入一个文件。

A. page 指令标签　　B. page 指令的 import 属性

C. include 指令标签　　D. include 动作标签

(2) 在JSP中,使用<jsp:useBean>动作可以将JavaBean嵌入JSP页面,对JavaBean的访问范围不能是(　　)。

A. page　　B. request　　C. response　　D. application

(3) Page指令的作用是(　　)。

A. 用来定义整个JSP页面的一些属性和这些属性的值

B. 用来在JSP页面内某处嵌入一个文件

C. 使该JSP页面动态包含一个文件

D. 指示JSP页面加载Java plugin

(4) 关于include指令,下面表述错误的是(　　)。

A. include指令用于把资源文件嵌入当前页面中

B. include指令可以嵌入动态页面和静态页面

C. file属性是目标资源的路径和文件名

D. file属性可以使用相对路径,也可以使用绝对路径

(5) 下列关于<jsp:useBean>说法,错误的是(　　)。

A. <jsp:useBean>用于定位或实例化一个JavaBeans组件

B. <jsp:useBean>首先会试图定位一个Bean实例,如果这个Bean不存在,那么<jsp:useBean>就会从一个class或模板中进行实例化

C. <jsp:useBean>元素的主体通常包含有<jsp:setProperty>元素,用于设置Bean的属性值

D. 如果这个Bean已经存在,<jsp:useBean>能够定位它,那么主体中的内容将不会起作用

(6) 下面关于jsp:setProperty说法,错误的是(　　)。

A. jsp:setProperty用来设置已经实例化的Bean对象的属性

B. name属性表示要设置属性的是哪个Bean

C. property属性表示要设置哪个属性

D. Param指定用哪个请求参数作为Bean属性的值

E. value属性用来指定Bean属性的值,value和Param参数可以同时使用

(7) 下面对out对象的说法,错误的是(　　)。

A. out对象用于输出字符型数据

B. out对象的作用域范围是application

C. out.newLine()方法用来输出一个换行符

D. out.close()方法用来关闭输出流

(8) 下面关于request对象的说法,错误的是(　　)。

A. request对象是ServletRequest的一个实例

B. 当客户端请求一个JSP网页时,JSP引擎会将客户端的请求信息包装在一个request对象中

C. 在 Servlet 中用 request.setAttribute()存储的数据，在 JSP 页面只能用 request.getAttribute()读取

D. request.getParameter()方法返回指定参数的值

(9) 在 JSP 文件中加载动态页面可以用(　　)。

A. <%@ include file="fileName" %>指令

B. <jsp:include>动作

C. page 指令

D. <jsp:forward>动作

E. taglib 指令

(10) 当要在 JSP 页面中使用自定义标签时，下面错误的是(　　)。

A. 在 tld 文件中定义标签

B. 创建一个标签处理器

C. 在 JSP 页面中使用 page 指令，引入这个标签的标签库

D. 在 JSP 页面中使用 taglib 指令，引入这个标签的标签库

(11) 对于预定义<%! 声明 %>的说法，错误的是(　　)。

A. 一次可声明多个变量和方法，只要以“;”结尾就行

B. 一个声明仅在一个页面中有效

C. 声明的变量将作为局部变量使用

D. 声明信息不会在当前输出流中产生任何输出

(12) 在 JSP 中使用<jsp:getProperty>标记时，不会出现的属性是(　　)。

A. name　　B. value　　C. property　　D. scope

(13) 在 JSP 中调用 JavaBean 时，不会用到的标记是(　　)。

A. <jsp:Javabean>　　B. <jsp:useBean>

C. <jsp:setProperty>　　D. <jsp:getProperty>

(14) 在 JSP 中如果要导入 java.io.* 包，应该使用(　　)指令。

A. page　　B. taglib　　C. include　　D. forward

(15) 如果当前 JSP 页面出现异常，需要转到一个异常页，设置 page 指令的(　　)属性。

A. Exception　　B. isErrorPage　　C. error　　D. errorPage

(16) JSP 中的隐式注释为(　　)。

A. // 注释内容　　B. <! --注释内容-->

C. <%--注释内容--%>　　D. /* 注释内容 */

(17) 下列(　　)指令定义在 JSP 编译时包含所需要的资源。

A. include　　B. page　　C. taglib　　D. forward

(18) 重定向应该使用(　　)方法。

A. response.sendRedirect("login.jsp")

B. request.sendRedirect("login.jsp")

C. <jsp :forward page= "login.jsp"/>

D. <forward page= "login.jsp"/>

(19) <jsp:useBean >声明对象的默认有效范围为(　　)。

A. page　　B. session　　C. application　　D. request

(20) 某 JSP 程序中声明使用 javaBean 的语句如下：

```
<jsp:useBean id = "user" class = "mypackage.User" scope = "page"/>
```

要取出该 javaBean 的 loginName 属性值，以下语句正确的是(　　)。

A. <jsp:getProperty name="user" property="loginName"/>

B. <jsp:getProperty id="user" property="loginName" />

C. <%=user.getLoginName()%>

D. <%=user.getProperty("loginName")%>

(21) 以下(　　)选项不属于浏览器的功能。

A. 发送 HTTP 请求，并接收 HTTP 响应

B. 解析并展现 JSP 代码样式

C. 解析并运行 JavaScript 代码

D. 解析并展现 HTML 代码样式

(22) 以下(　　)选项不属于 Web 服务器的功能。

A. 接收 HTTP 请求，并回应 HTTP　　B. 动态编译 JSP 代码

C. 运行并展示 JSP 页面　　D. 管理 Servlet 和 JSP 组件生命期

(23) 下面(　　)选项属于解释型语言。

A. HTML　　B. JSP　　C. Java　　D. JavaScript