第 1 章　搭建 Java Web 开发环境

正所谓“工欲善其事，必先利其器”，开发一个 Web 应用程序，首先必须搭建好开发环境，选择好开发工具，从而达到事半功倍的开发效果。现如今支持 Web 的应用服务器非常多，例如 WebSphere、WebLogic、Tomcat 等，配置方法各不相同，本书选择 Apache Tomcat 9.0.8 作为服务器开发平台、JDK 使用的是 10.0.1 版本。

本章主要涉及的知识点有：

- JSP 支持的网络协议
- Web 应用程序的运行环境和开发环境
- Tomcat 软件的安装和配置
- JSP 开发工具的选择

1.1　Web 开发背景知识

本节的重点是介绍 Web 开发的一些基础知识，首先简单介绍 Web 访问的基本原理，然后对 HTTP 超文本传输协议进行简单介绍，最后介绍静态网页与动态网页的区别以及各种 Web 服务器的优缺点。通过本节的学习，读者可以掌握 Java Web 开发所需的背景知识。

1.1.1　Web 访问的基本原理

Web 访问可以简单划分为两个过程：客户端请求、服务器端响应并显示结果。客户端的请求通过 Servlet 引擎传递给 Servlet 模块，Web 服务器接收客户的请求，并把处理的结果返回给客户。客户端与服务器之间的通信协议就是 HTTP 超文本协议。客户端与服务器之间的请求模式如图 1.1 所示。

图 1.1　Web 访问原理

1.1.2　超文本传输协议

超文本传输协议（Hypertext Transfer Protocol，HTTP）是一种互联网上应用最为广泛的网络协议，是一种无状态的协议。自 1990 年起，就已经被应用于 WWW 全球信息服务系统。所有的 WWW

文件都必须遵守这个标准。

HTTP 协议的主要特点如下。

- 简单、快速：客户端向服务器请求服务时，只需发送请求方法和路径 URL。通常请求的方法有 POST 和 GET。由于 HTTP 传输协议简单，使得 HTTP 服务器的程序规模相对较小，因此传输速度较快。
- 灵活：HTTP 允许传输任意类型的数据，例如普通文本、超文本、音频、视频等，主要由 Content-Type 控制。
- 无状态：无状态是指对于数据库事务处理没有记忆能力。后续的处理如果需要前面的信息，就需要重新发送。
- 无连接：无连接的含义是每次连接只处理一次请求，处理完当次请求后就断开连接。

1.1.3 静态网页和动态网页

在网站设计中，直接使用 HTML 标记语言编写的网页通常被称为“静态网页”。静态网页是标准的 HTML 文件，后缀名为.htm 或.html。它所展示的内容一般是固定不变的，早期的网站一般都是由静态网页制作的。静态网页更新起来比较麻烦，需要将更新的 HTML 页面重新上传到网站服务器。显然，这样的网站缺乏灵活性，同时网站的维护成本也比较高。动态网页技术的出现改变了如此不灵活的状态，用户在不同时间或不同地点访问同一动态网页时显示的内容可以是不同的。

注 意

所谓静态网页与动态网页，是基于访问网页时页面的内容有无变化而言的，与页面的视觉效果没有关系。因为动态的视觉效果大多是通过 JavaScript 或其他基于 JavaScript 的框架技术实现的，与动态网页技术没有直接必然的联系。

动态网页中的变化内容大部分来自于数据库中数据的变化，通过增加、删除、修改、查找数据库中存储的数据来显示内容的变化。例如，在微博中发布一条微博后，查看微博时，会将所发的微博即时显示出来，这在静态网页中是无法完成的。动态网页在被访问时，首先运行服务端脚本，通过它生成网页内容。显然，动态网页的显示内容是在访问该网页的时候动态生成的，而静态网页是提前做好放在服务器中的，因此，当前网络上的网页大多是动态网页，很少有静态网页，除非一些固定不变的内容，例如发布公告等新闻内容。

目前比较流行的动态网页技术主要包含 ASP、PHP 以及 JSP。

- ASP 更精确地说应该是一个中间件，它将 Web 上的请求转入到 IIS 解释器中，IIS 将 ASP 上的 Script 脚本全部解析执行。其缺点就是不能跨平台，只能在 Windows 平台下，开发受到诸多限制。其优点是微软提供了强大的 IDE，所以开发者容易上手且开发效率也较高。
- PHP 是当前比较流行的动态网页技术，是一种 HTML 内嵌式的语言，其语法融合了 Java、C 以及 Perl，能够比 CGI 更加快速地执行动态网页。其优点是开源、跨平台，正因为其具有开源和跨平台特性，所以很多网站都采用 PHP 编写自身的网页；其缺点是安装复杂，需要添加很多的外部库来支持，如图形需要 gd 库等。

CGI（Common Gateway Interface）也是一种动态网页技术，虽然功能强大，但是由于编程困难、效率低下、修复复杂等缺陷，逐渐被新技术淘汰。

- JSP（Java Server Pages）采用 Java 语言作为服务器端脚本，页面由 HTML 和嵌入 Java 代码组成。随着 Java 的广泛应用，JSP 的应用也越来越广泛。其优点是简单易用，完全面向对象，具有 Java 的平台无关性和安全可靠性。目前大多数的动态网站开发都采用 JSP 技术，它具有很高的市场占有率。

1.1.4 Web 浏览器和 Web 服务器

1．Web 浏览器

浏览器是指 Web 服务的客户端浏览程序。它可以向服务器发送各种请求，并对从服务器中返回的各种信息（包括文本、超文本、音频、视频等各种数据）进行解释、显示和播放。现如今，Web 浏览器遍地开花，国外的有 Internet Explorer（IE）、Chrome、Firefox、Safari，以及近几年逐渐步入公众视野的 Microsoft Edge 浏览器，国内的有 360 浏览器、QQ 浏览器、遨游浏览器、搜狗浏览器、猎豹浏览器等，它们都各自占据着一片江山，由于 IE 和 Edge 绑定在 Windows 操作系统中，因此 IE 和 Edge 在市场中占有明显高的份额，但随着互联网的不断进步，Chrome 浏览器大量普及，已逐渐展露超越 IE 的势头。

国内的浏览器从利用 IE 支持的内核逐步发展为支持多种内核，例如 360 浏览器 7.5 版本同时支持 Trident+Blink 内核，而国外的浏览器内核是自主研发的，例如 IE 的内核是 Trident、Chrome 由 Webkit 转为 Blink 内核、Firefox 的内核是 Gecko、Safari 的内核是 Webkit 等。

2．Web 服务器

浏览器与服务器的关系可谓是“唇齿相依”，浏览器发送请求，服务器处理请求并将结果返回给浏览器显示。Web 服务器的种类繁多，目前比较流行的有 WebSphere、WebLogic、Tomcat 等。它们的配置、启动方式各不相同，也各自有其优缺点。

- WebSphere 服务器是 IBM 的产品，是一款功能很完善的 Web 服务器，基于 Java 程序研发，用于建立、部署和管理 Web 应用程序。WebSphere 产品有开发版和商业版，但是 WebSphere 不开源。
- WebLogic 服务器是一款多功能、基于标准的 Web 服务器。它遵从开放的标准、支持基于组件的开发，因为它性能比较稳定，所以国内有很多大公司在使用。
- Tomcat 是一款开放源代码、基于 Java 的 Web 服务器，也是一款轻量型的 Web 服务器，是基于 Apache 许可证下开发的自由软件，是根据 JSP 和 Servlet 技术标准实行的。它安装简单、占有系统空间较小，却能支撑较大的 Web 应用系统，所以它是开发者、小型公司、学校网站建设者的首选软件。

1.2 JSP 简介

上一节简单介绍了 Web 开发的一些背景知识，读者已经了解了 Web 访问的基本原理、超文本传输协议、静态网页与动态网页的区别，以及主流的浏览器和 Web 服务器。本节将介绍 JSP 的基本概念、执行过程等，让读者了解 JSP 是什么、能做些什么。

1.2.1 什么是 JSP

JSP 技术是由 SUN 公司（现被 Oracle 收购）提出，多家公司参与的，于 1999 年推出的一款建设动态网页的方法。它基于 Java Servlet 技术来开发动态的、高性能的 Web 应用程序。JSP 的网页实际上是由在 HTML 文件中加入的 Java 代码片段和 JSP 的特殊标记构成的。

因为 JSP 是 Java 的成员，所以 JSP 具有平台无关性，即实现了跨平台功能，实现了用户界面和程序代码的解耦合，使得业务逻辑与代码的耦合度更低，开发人员可以在不更改 JSP 程序的情况下修改用户的界面。

JSP 页面实质上也是一个 HTML 页面，只不过它包含了用于产生动态网页内容的 Java 代码，这些 Java 代码可以是 Java Bean、SQL 语句、RMI（远程方法调用）对象等。例如：一个 JSP 页面包含了用于产生静态网页的 HTML 代码，同时也包含了连接数据库的 JDBC 代码，那么当网页在浏览器中显示时，它既包含了静态的 HTML 代码，又包含了从数据库中取得的动态内容，正因为如此才能称之为动态网页。

在 JSP 页面中，动态的内容与静态的内容可以相互分离，使得界面的设计者可以完全专注于界面的美化，而动态的部分则由 JSP 程序开发者负责，实现界面与业务逻辑的分离，从而实现 JSP 代码的高度复用。

1.2.2 JSP 的优势

JSP 可以看作是 Java Servlet 的一种扩展，JSP 在使用前必须被编译为 Servlet，也就是 Java 类，然后被调用执行，Servlet 所产生的 Web 页面是不能包含在 HTML 标签中的，因为它离不开 Java 类文件的支持。随着学习的深入，用户将体验到 JSP 的很多优势。

1. 开发简单方便

在 JSP 中的编辑跟编写 HTML 文件基本一样，在处理表单方面极为方便。对于设置 HTTP 报头，JSP 同样提供了丰富的方法，使得 JSP 开发者在编写通用功能时很便捷，从而花费更多的时间在业务逻辑上。

2. 跨平台

Java 本身就有跨平台的特性，因此 JSP 程序可以在支持 Java 的平台上开发运行，显然这对平台移植极为有利。当 JSP 在更换服务平台时，若不涉及数据库等相关操作，几乎不做任何变动就可以完成服务平台的迁移。当需要更换 Web 服务器时，JSP 同样可以做到不修改或者少量修改就在新的 Web 服务器中编译、运行。

3. 高效率和高性能

JSP 可以是 Servlet 的扩展，因此 Java 虚拟机为每一个请求创建一个单独的线程，而不是进程，如此系统就能很快地处理请求。同时 JSP 只会被编译一次，即在首次加载时需要编译，这样就加快了系统的响应速率。当一个请求处理结束之后，相关 JSP 映射的 Java 类并不会从内存中删除，而是被保留在内存中，当下次同样的请求发生时，系统会提供更快的响应速度。

4. 低成本

众所周知 Java 是开源的开发语言，JSP 也是基于 Java 的开源环境开发的动态网页技术，所以就省去了商业的付费项目。再有，开发者可以从众多的 Java IDE 中选择一款适合自己的开发工具来进行项目研发，当然，也可以直接利用文本编辑器直接编写，只是这样比较耗时而且容易出错。还有许多的商业软件可以使用，但是通常来说 JSP 的开发总成本比采用其他技术要低一些。

综上所述，采用 JSP 动态网页技术是目前 Web 开发者的最佳选择。

1.2.3 JSP 的执行顺序

在编写 JSP 程序时，我们要了解它的执行顺序，这对于后续的学习会有很大的帮助。JSP 程序的执行过程大致如下：首先客户端向 Web 服务器提出请求，然后 JSP 引擎负责将页面转化为 Servlet，此 Servlet 经过虚拟机编译生成类文件，再把类文件加载到内存中执行，最后由服务器将处理结果返回给客户端。整个流程如图 1.2 所示。

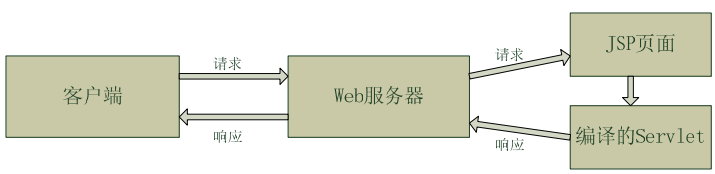

图 1.2 JSP 执行顺序

JSP 页面代码会被编译成 Servlet 代码，所以从执行效率上说肯定是没有 Servlet 快，但并不是每一次都需要编译 JSP 页面。当 JSP 第一次被编译成类文件后，重复调用该 JSP 页面时，JSP 引擎发现该 JSP 页面没有被改动过，那么就会直接使用编译后的类文件，而不会再次编译为新的 Servlet。当然，如果页面被修改过，则需要重新加载编译。

1.2.4 一个 JSP 的简单实例

以下是 JSP 网页的一个简单实例，功能是输出 1~10 的相加结果：

----------index.jsp----------

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
    <head>
        <title>JSP 简单例子</title>
    </head>
    <body>
```

```
08          <%
09              int count=0;
10              for(int i=1;i<10;i++)
11              {
12                  count+=i;
13              }
14              out.print("1 到 10 的相加结果："+count);
15           %>
16      </body>
17  </html
```

注 意

在第 1 行中，pageEncoding 标签可以设定字符类型，第 8~15 行为 Java 代码。

该程序的主要作用是利用 JSP 输出 1~10 的和，其中代码是由简单的 HTML 代码和 JSP 表达式构成的，JSP 表达式中包括一段 Java 程序段。

1.3 安装 Java 10 环境

上一节简单介绍了 JSP 中的相关概念，让读者对 JSP 有一个初步的了解。本节将介绍 Java 开发环境的安装和搭建，使读者能了解 JSP 所需的开发环境。

1.3.1 下载 JDK10 和配置环境变量

JDK 是由 SUN 公司（现被 Oracle 收购）提供的 Java 开发工具和 API，首先从 Oracle 公司的官网（http://www.oracle.com/technetwork/java/index.html）下载 Java SE。需要注意的是，JDK 的版本较多，各版本之间还存在不兼容问题，以及操作系统的兼容性问题。本书使用的是 Windows 7 的 64 位操作系统，所以下载了 jdk-10.0.1_windows-x64_bin.exe 版本，读者可以根据自身的操作系统下载不同的版本。

注 意

API 在 Java 世界中就是可以查找的参考内容，也就是 Java 方法的说明。

下载完成后双击执行 jdk-10.0.1_windows-x64_bin.exe，根据安装向导的提示完成安装，若用户选择的安装目录是 D:\Java\jdk-10.0.1，安装结束后应该在 D:\Java\jdk-10.0.1\bin 目录下出现如图 1.3 所示的文件。

该目录文件夹下包含了很多可执行文件，例如 java.exe、javac.exe、javadoc.exe 等。javac.exe 是 Java 的编译器，用来编译 Java 文件，将它变为字节码。在 DOC 环境下，利用命令“javac test.java”的形式编译一个 Java 文件。java.exe 是运行编译后的 Java Class 文件。java.exe 也可以运行.jar 文件，比如运行命令“java –jar test.jar”，就可以运行 test.jar 文件了。

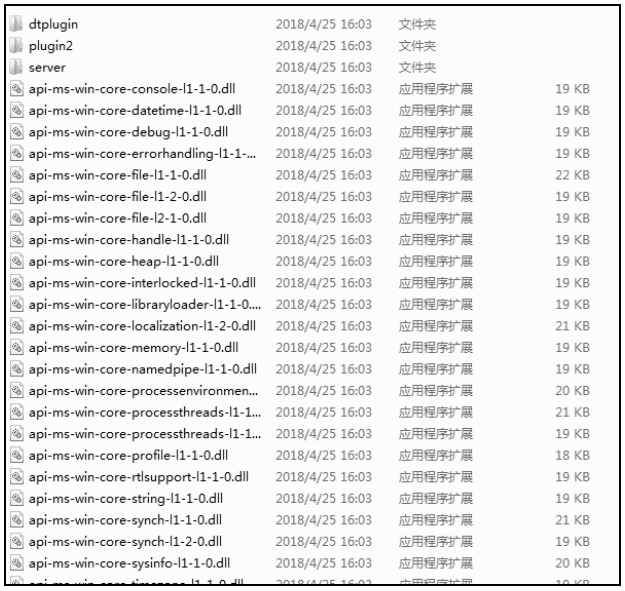

图 1.3 JDK 安装目录下的文件

安装完 JDK 后，接着就是配置 Java 的环境变量，配置的作用如下：

- 系统自动查找 Java 编译器路径。
- 服务器或其他需要依赖 Java 的程序安装时需要知道 Java 路径。
- 编译和执行时指定 Java 路径。

配置环境变量的步骤如下：

01 以 Windows 为例，右击“我的电脑”或“计算机”，在弹出的快捷菜单中选择“属性”命令，在打开的页面左侧单击“高级系统配置”选项，弹出“系统属性”对话框，在“系统属性”对话框中选择“高级”选项卡，单击“环境变量”按钮，如图 1.4 所示。

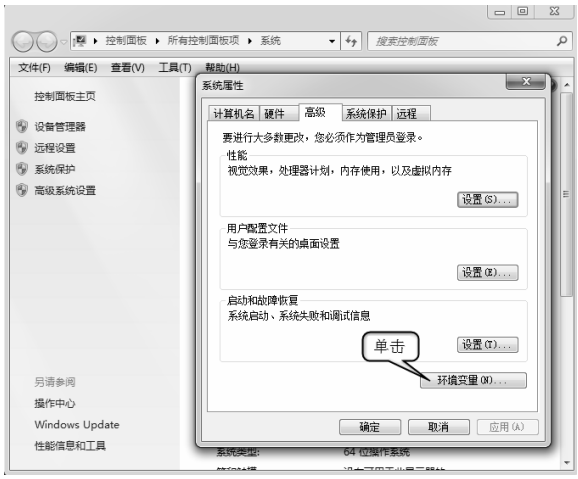

图 1.4 配置环境变量

02 在弹出的“环境变量”对话框中单击“新建”按钮，分别设定 JAVA_HOME、JRE_HOME、PATH 和 CLASSPATH 共 4 个新的环境变量。如果 PATH 变量已存在，只需要在最后添加下方内容即可。JDK9 及以后的版本，删除了 tools.jar 等 jar 包，不再进行相关配置。4 个变量值分别如下，其中%JAVA_HOME%、%JRE_HOME%表示引用值:

```
JAVA_HOME= D:\Java\jdk-10.0.1
JRE_HOME= D:\Java\jre-10.0.1
CLASSPATH= .;%JAVA_HOME%\lib;%JRE_HOME%\lib
PATH= ;%JAVA_HOME%\bin;%JRE_HOME%\bin
```

注 意

CLASSPATH 的值前面有一个符号“.”。

1.3.2 下载 Intellij IDEA 开发工具

Intellij IDEA 是当下流行的 Java IDE（Integrated Developing Environment）开发工具，深受开发者的青睐，它能支持目前主流的技术和框架，擅长于企业应用、移动应用和 Web 应用的开发，并且拥有丰富的插件。

从官网 https://www.jetbrains.com/idea/download/下载 Intellij IDEA 2018.1.4，或者使用搜索引擎搜索 Intellij IDEA 2018.1.4，然后选择合适的链接下载。下载完成后双击安装包，安装界面如图 1.5 所示。

图 1.5 安装 Intellij IDEA 的示例界面

安装完成后会生成桌面快捷方式，也可根据操作系统的位数双击安装目录内 bin 目录下的 idea.exe 或 idea64.exe 运行程序，例如 D:\Program Files\JetBrains\IntelliJ IDEA 2018.1.4\bin\idea.exe 目录。运行 Intellij IDEA 后会提示输入注册码，这时输入购买时的注册码就可以了，或者选择试用

30 天。运行后的界面如图 1.6 所示。

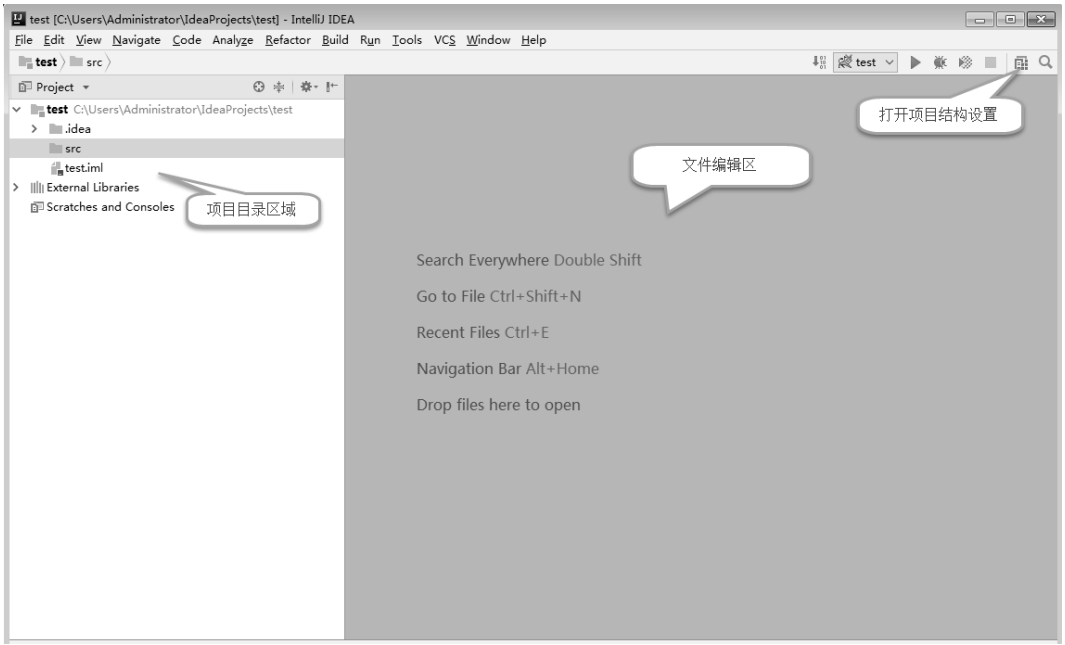

图 1.6　Intellij IDEA 的运行界面

Intellij IDEA 安装成功后，新建项目或导入项目时需要配置项目所需的 JDK。例如：配置已安装的 JDK 时，可选择 File | Project Structure| Platform Settings | SDKs 选项来增加 JDK，如图 1.7 所示。

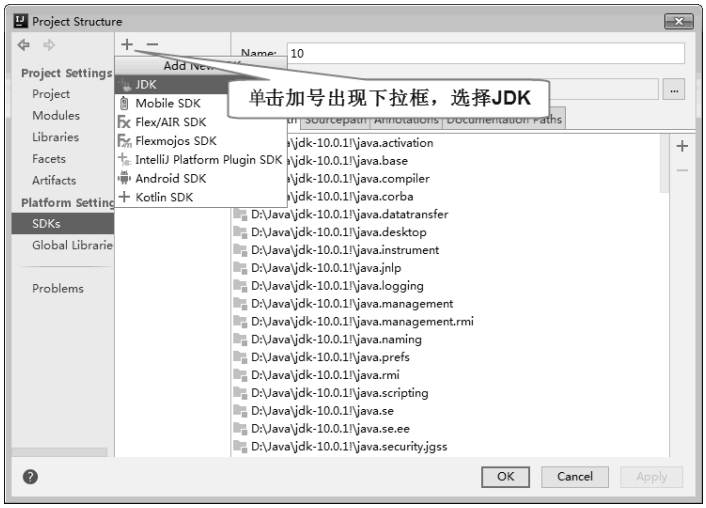

图 1.7　Intellij IDEA 查找 JDK 的选择界面

配置 Tomcat 时，可在 Run | Edit Configurations 中单击上方加号按钮，在左侧下拉框选择 Tomcat Server| Local ，如图 1.8 所示，在弹出的窗口中配置 Tomcat 的安装路径，并在已安装的 JDK 版本中选择 jdk10.0.1，如图 1.9 所示。

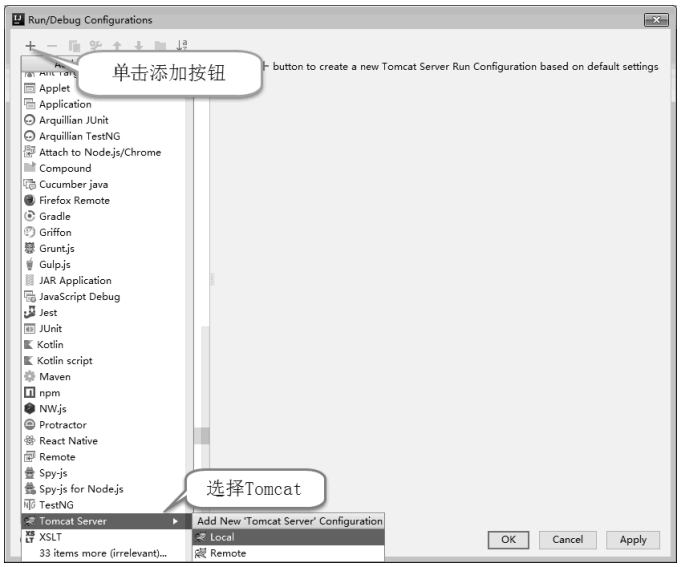

图 1.8　打开配置 Tomcat 的界面

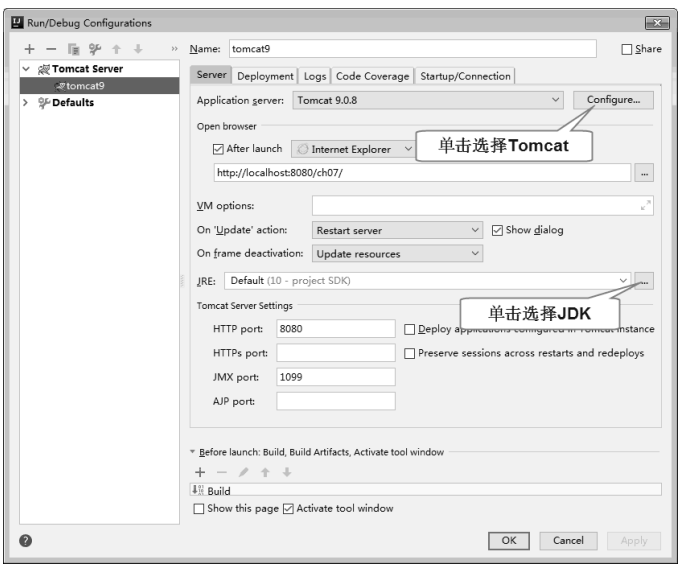

图 1.9　选择 Tomcat 和 JDK 的界面

当然，还有很多 Java IDE 工具，例如 MyEclipse、NetBeans、JCreator、JRun 等。编辑 JSP 页面的工具还有 Dreamveaver MX 等。由于篇幅问题，这里就不一一介绍了。单从学习的角度来说，建议选择轻便型的 IDE 工具，如果是企业开发，那么人性化、智能化的 Inellij IDEA 将是最佳选择。

1.3.3　下载安装 Tomcat 9 服务器

1. 目录结构

Tomcat 是轻量级的 Web 应用服务器，可以从官网 http://tomcat.apache.org/下载最新的 Tomcat 服务器版本，本书使用的是 Tomcat 9.0.8 版本。下载完成后直接解压 Tomcat 文件到指定的目录下，

例如 E:\Program Files\apache-tomcat-9.0.8 中。下面介绍 Tomcat 的目录结构：

- bin 文件夹，包含 Tomcat 服务器启动和终止的批处理文件。例如 startup.bat、startup.sh、shutdown.bat、shutdown.sh、catalina.bat、catalina.sh 等。其中 startup.bat、shutdown.bat、catalina.bat 是 Windows 中的批处理文件；startup.sh、shutdown.sh、catalina.sh 是 Linux 中的脚本文件。
- conf 文件夹，包含 Tomcat 的配置信息，主要有 server.xml 和 web.xml 两个配置文件。在 server.xml 中可以更改服务端口和改变 Web 默认的访问目录。
- lib 文件夹，存放 Tomcat 运行中需要的.jar 包文件，例如 catalina.jar、servlet-api.jar、tomcat-dbcp.jar 等.jar 包，正因为有这些包的支持， Tomcat 才可以运行 Web 应用程序。
- logs 文件夹，存放执行 Tomcat 的日志文件。
- temp 文件夹，存放 Tomcat 的临时文件信息。
- webapps 文件夹，是 Tomcat 默认的 Web 文件夹。本身自带两个 admin 应用和 manager 应用。开发人员可以直接将 Web 应用存放在该文件夹下。
- work 文件夹，存放 Tomcat 执行应用后的缓存。

2．配置修改

在 Tomcat 服务器中，需要经常修改其配置信息来满足系统的需求，例如在 server.xml 中可以更改服务端口和改变 Web 默认的访问目录。

（1）修改端口号

方法如下：

```
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
                redirectPort="8443" URIEncoding="UTF-8"/>
```

Tomcat 默认的端口号为 8080，可以自定义其他端口，修改完毕后，保存 server.xml，然后重启 Tomcat 服务器，这样服务器的端口就成功变更了。

（2）修改 Web 默认的访问目录

方法如下：

```
<Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true"
            xmlValidation="false" xmlNamespaceAware="false">
```

修改 appBase 中的文件夹地址。例如，将 appBase 的属性值 webapps 改为 D:\test，修改后的文件如下：

```
<Host name="localhost"  appBase="D:\test"
            unpackWARs="true" autoDeploy="true"
            xmlValidation="false" xmlNamespaceAware="false">
```

这样就可以将 Web 默认的访问目录更改为 D:\test，以后加载 Web 应用程序时就会在该目录下

创建目录。

（3）建立自身的 Web 目录

开发人员可以将应用部署在 Tomcat 服务器的默认 webapps 目录下，也可以部署在自己创建的目录下。方法如下：首先创建自身的目录 D:\test，其次配置 Web 目录，在 server.xml 文件的末尾</HOST>中加入如下语句：

```
<Context path="text" docBase="D:\test" debug="0" reloadable="true"></Context>
```

该语句的作用是将目录 D:\test 设置为 Tomcat 服务器的 Web 目录，将该文件的访问路径设置为“text”。属性 docBase 的值为“D:\test”，它是指应用的物理路径。修改后将 server.xml 文件进行保存。假设现在 test.jsp 页面位于 D:\test 目录下，那么页面的访问路径就为 http://localhost:8080/text/test.jsp。

在 bin 文件夹下，可以修改 catalina.bat 或者 catalina.sh 来更改 Tomcat 的启动配置信息。例如增加 Java 的运行内存：

```
set JAVA_OPTS=-XX:PermSize=512M -XX:MaxPermSize=512m -Xms512m -Xmx1024m
```

更多的修改内容请参见 Tomcat 官网说明。

1.4 小结

本章为读者介绍了搭建 Web 开发环境以及 Web 开发的基础知识，这些都是学习 JSP 开发技术之前必须掌握的知识。读者不仅需要理解并掌握本章内容，还应该亲自动手安装 JDK、Tomcat、Intellij IDEA 以及制作一个简单的 JSP 测试程序。

1.5 习题

1. JSP 动态网页技术有哪些优势？
2. JSP 引擎的作用有哪些？
3. JSP 页面执行的顺序是什么？
4. 如何修改 Tomcat 的服务端口？
5. 如何创建一个自己的 Web 目录？

第 2 章　JSP 基础语法：与编写 HTML 一样容易

上一章介绍了 Web 开发所需的基础知识以及如何安装 JDK、Intellij IDEA 以及 Tomcat 应用服务器，本章将要介绍 JSP 的基本语法、如何在 JSP 页面中嵌套 Java 以及 JSP 的指令等。从本章开始意味着读者将正式开始学习 JSP 技术。

通过本章的学习，读者可以了解以下内容：

- JSP 中的注释表达式
- JSP 中的声明表达式
- JSP 中指令标签的作用和使用方法
- 运用 HTML 页面的元素、Java 代码段、JSP 标签创建 JSP 实例

2.1　JSP 注释

在编写规范的代码时，合理和必要的注释是至关重要的，这也是对开发人员的一项最基本要求。编程语言的注释起到了说明、解释的作用，在实际执行的过程中并不执行。在 JSP 页面中，注释可以归纳为两种：一种是原有的 HTML 注释；另外一种是 JSP 的注释。

1. HTML 的注释

HTML 的注释如下：

```
<!--  注释的内容 -->
```

例如：

```
<!--  注释的内容会被照搬到浏览器中 -->
```

2. JSP 的注释

JSP 的注释语法如下：

```
<%-- 注释的内容 --%>
```

例如：

```
<%-- 注释的内容不会照搬到浏览器中 --%>
```

在实际应用中，JSP 通常会引入 Java 的注释，二者混着用。因为 Java 的注释在脚本中注释，而 JSP 的注释在脚本外注释。

【例 2.1】JSP 中不同的注释用法。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <meta content="text/html;charset=utf-8" http-equiv="Content-Type">
    <title>JSP 注释例子</title>
  </head>
      <body>
        <!-- 该 JSP 注释可以在浏览器文件中看到 -->
        <br/>
        <br/>
         <table align="center">
              <tr>
                   <th>
                        <b>JSP 注释</b>
                   </th>
              </tr>
         </table>
         <br/>
         <table>
              <tr>
                   <th>
                        <b>JSP 注释</b>
                   </th>
              </tr>
         </table>
         <table align="right">
              <tr>
                   <th>
                        <b>JSP 注释</b>
                   </th>
              </tr>
         </table>
        <%--该 JSP 注释无法在浏览器中看到 --%>
        <%
        //  这是脚本中的 Java 注释
        /*
              这也是脚本中的 Java 注释
        */
         %>
      </body>
    </html>
```

上述代码介绍了 JSP 的各种注释格式，其中代码中的第 9 行是“<!-- -->”格式，第 34 行是“<%-- --%>”格式。因为都是注释，所以界面运行的结果看不到注释中的内容，通过浏览器中的“查看源文件内容”功能可以看到 HTML 注释的内容。本例运行的结果如图 2.1 所示。

图 2.1 JSP 注释运行结果界面

注 意

第 36 行的代码说明在 JSP 页面中可以嵌套 Java 注释。

2.2 JSP 声明

JSP 声明用于定义 JSP 中的变量、方法以及静态方法，实际上它与在 Java 中定义一个全局变量、共用方法一样，JSP 声明部分的变量和方法是可以在 JSP 页面中被调用和使用的。

JSP 声明的基本语法有两种，分别是：

- <%! 变量定义/方法定义/类 %>
- <jsp:declaration>变量定义/方法定义/类</jsp:declaration>

注 意

第 2 种语法现在已经过时，一般采用第一种声明语法。

JSP 声明的结尾跟 Java 的结尾一样都以“;”结束，可以一次定义多个变量，利用“,”分割。声明部分的变量和方法只在当前的页面中有效。

【例 2.2】演示变量、方法和类的声明。

----------declar.jsp----------

```
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <!DOCTYPE HTML>
03  <html>
04    <head>
05      <meta content="text/html;charset=utf-8" http-equiv="Content-Type">
06      <title>JSP 声明例子</title>
07    </head>
08    <%!
09        int x,y=60,z;                    //多个声明以","分割
10        String name="John";
11        Date date = new Date();
12     %>
13     <%!
14        int add(int m,int n){            //计算两个数的和
15            int result=0;
16            result = m+n;
```

```
17                  return result;
18              }
19          %>
20          <%!
21              int chengji(int m,int n){       //计算两个数的乘积
22                  int result=0;
23                  result = m*n;
24                  return result;
25              }
26          %>
27          <%!
28              class Circle{                   //计算圆的面积
29                  double r;
30                  Circle(double r){
31                   super();                   //继承空的构造方法
32                   this.r = r;
33                  }
34                  double area(){              //取整
35                      return    Math.floor(Math.PI*r*r);
36                  }
37              }
38          %>
39      <body>
40          <%
41              out.println("我的名字："+name);
42              out.println("<br/><br/>");
43              out.println("x 的值为："+x);
44              out.println("<br/><br/>");
45              out.println("y 的值为："+y);
46              out.println("<br/><br/>");
47              out.println("z 的值为："+z);
48              out.println("<br/><br/>");
49              out.println("现在的时间为："+date);
50              out.println("<br/><br/>");
51              out.println("10 与 20 的和："+add(10,20));
52              out.println("<br/><br/>");
53              out.println("10 与 20 的积："+chengji(10,20));
54          %>
55          <br/>
56          <br/>
57          <%
58              Circle c = new Circle(6);
59              out.println("半径为 6 的圆面积为："+c.area());
60          %>
61      </body>
62  </html>
```

页面效果如图 2.2 所示。

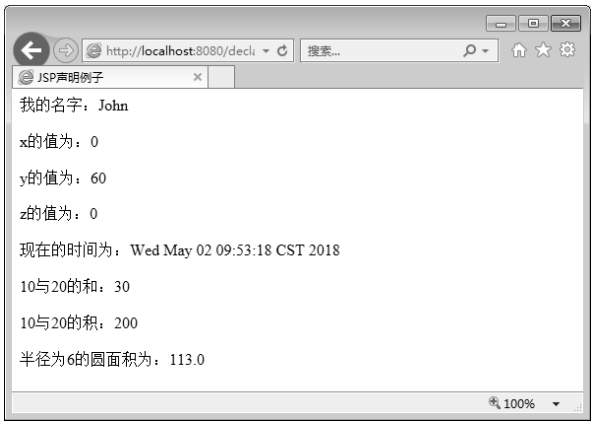

图 2.2　declar.jsp 页面的运行结果

2.3　JSP 表达式

JSP 表达式的作用是将动态信息显示在页面中，它的语法形式也有两种：

```
<%=变量或者表达式%>
```

或者

```
<jsp:expression>变量或者表达式</jsp:expression>
```

第 2 种形式已经很少使用，在一般的 IDE 工具中也不提供这种形式的表达式；第 1 种形式是目前主流的写法，本书实例也是利用该种形式书写的。

表达式的值由服务器负责计算，计算结果以字符串的形式发送到客户端。

下面看一个实例，JSP 页面利用 Date 类输出当前的时间。

```
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <!DOCTYPE HTML>
03  <html>
04    <head>
05      <meta content="text/html;charset=utf-8" http-equiv="Content-Type">
06      <title>JSP 表达式例子</title>
07    </head>
08
09    <body>
10                          当前的时间为：</br>
11          <%=new Date()%>
12    </body>
13  </html>
```

在上述代码中，第 1 行导入 Date 类库，第 11 行直接引用 Date 对象，页面效果如图 2.3 所示。从页面的显示结果来看，文本“当前的时间为：”被正常显示，其后的“</br>”（HTML 标记）

使得后面显示的内容换行，只有“<%=new Date()%>”一行被替换成当前的时间。

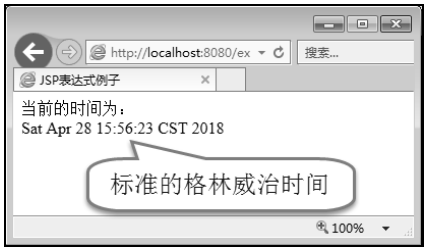

图 2.3　JSP 表达式运行结果

以下是上面 JSP 页面生成的源代码，与 JSP 页面代码对比可以发现，只有首行和第 11 行不同，其他的代码都一样。

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta content="text/html;charset=utf-8" http-equiv="Content-Type">
    <title>JSP 表达式例子</title>
  </head>

  <body>
                    当前的时间为：</br>
      Sat Apr 28 15:56:23 CST 2018
  </body>
</html>
```

通过对比可以发现，JSP 页面中的 HTML 元素在源代码中被原样保留，只有 JSP 代码会发生改变。因此，JSP 页面中的静态代码都是利用 HTML 模板来编写。

注　意

查看 JSP 页面生成的源代码时，一般是通过浏览器来查看的。对于 IE 浏览器或者是 IE 内核的浏览器，可以在待查看的页面上单击右键，然后在弹出的快捷菜单中选择“查看源文件”命令。

2.4　JSP 指令

上一节介绍了 JSP 表达式的用法以及示例，通过学习，读者已经可以应用前面章节的知识编写简单的 JSP 动态网页了，但要编写出功能强大、复杂的 JSP 页面还需要学习更多的内容。本节将介绍 JSP 的另一元素：指令。JSP 指令包括 page 指令、include 指令、taglib 指令。

2.4.1　与页面属性相关的 page 指令

page 指令用来设置 JSP 页面的属性和相关功能，基本语法形式有如下两种：

```
<%@ page attribute1="value1" […attribute n="value n"]%>
```

或者

```
<jsp:directive.page attribute1="value1" […attribute n="value n"] />
```

注 意

现在很少使用第 2 种形式，大多使用第 1 种表达形式。

page 指令有多种属性，但使用最多的是 language、import、pageEncoding 这 3 个，其中 language 是必须设置的，目前 JSP 页面使用 Java 语言，所以其默认值是 java；import 用来声明需要导入的包；pageEncoding 属性用于设置页面的编码。在所有的属性中除 import 可以声明多个外，其他的属性都只能出现一次。page 指令中的其他属性见表 2.1。

表 2.1 page 指令的常用属性

属性和属性值	说明
session="true|false"	限定 session 对象是否可用，默认为 true
autoFlush="true|false"	指明缓冲区域是否自动清除，默认为 true
info="text"	描述该 JSP 页面的相关信息
errorPage="URL"	当页面产生异常时跳转的路径
isErrorPage="true|false"	指定该 JSP 页面是否为处理异常错误的页面，当设定为 true 时才能使用 exception 对象，默认为 false
isThreadSafe="true|false"	是否允许多线程使用，默认为 true
buffer="8kb"	输出流是否有缓冲区，默认为 8KB
contentType="text/html; charset=UTF-8"	设定 MIME 类型和编码属性。编码属性一般设置为 UTF-8，MIME 类型还有很多，application/vnd.ms-excel 表示 Excel 电子表格，image/gif 表示 GIF 图像等
extends="class"	指明由该 JSP 页面产生的 Servlet 所继承的父类

2.4.2 引入文件的 include 指令

include 指令是在 JSP 页面生成 Servlet 时引入需要包含的页文件，既可以是 HTML 文件，也可以是 JSP 文件，还可以是其他文件（例如.js 文件）。include 指令的作用是在标签插入的位置插入静态的文件内容，使其与 JSP 文件组合成新的 JSP 页面，然后由 JSP 引擎翻译成 Servlet 文件，这样做有如下两个好处：

- 页面的代码可以复用，因为被引入的文件是静态文件，所以在其他的 JSP 页面中也可以导入。
- JSP 页面的代码结构清晰易懂，维护也比较简单。

include 指令的基本语法如下：

```
<%@include file="URL"%>
```

file 属性指向要包含的文件，一定要注意引入的路径是否正确，一旦路径出错，在编译的时候将不能通过。

include 指令经常用来包含网站中经常出现的相同页面。例如，一般情况下，网站为每个页面都设置导航栏，把它放在页面的顶端或者左边，这部分代码在每个页面都重复，可以用 include 来解决，为开发者省去重复动作。下面的实例将对 include 指令进行演示。

【例 2.3】演示 include 指令。

index_include.jsp 页面演示了 include 指令，其源代码如下：

```
---------- index_include.jsp ----------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
01      <head>
02          <meta content="text/html;charset=UTF-8" http-equiv="Content-Type">
03          <title>JSP include 指令演示</title>
04      </head>
05      <body>
06              <%@include file="John.html" %>
07              <br/>
08              <div align="center">JSP include 指令演示</div>
09              <%@include file="copyRight.jsp" %>
10      </body>
11  </html>
```

John.html 是内嵌页面，用来显示内容，其源代码如下：

```
---------- John.html ----------
01  <table align="center" border="1" width="600">
02          <tr>
03                  <td align="center" height="18" width="100%">
04                          <span>This is top page.</span>
05                  </td>
06          </tr>
07  </table>
```

copyRight.jsp 是显示版权信息的页面，其源代码如下：

```
---------- copyRight.jsp ----------
01  <%@ page language="java" import="java.util.*,java.text.SimpleDateFormat"
02                  pageEncoding="UTF-8"%>
03  <%
04          Date d = new Date();
05          SimpleDateFormat sdf = new SimpleDateFormat("yyyy");
06          String t = sdf.format(d);
07          String copyRightsMess = "John  版权所有  2010-"+t;
08  %>
09  <br/>
```

```
10    <div align="center" ><%=copyRightsMess%></div>
```

本例的页面效果如图 2.4 所示。

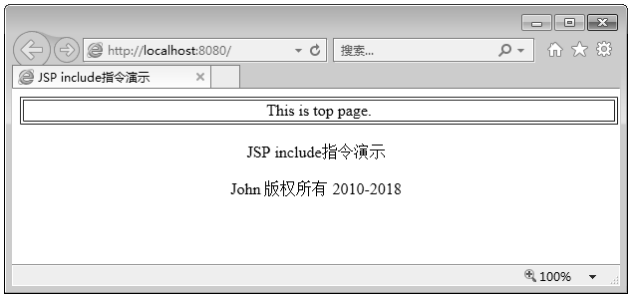

图 2.4　index_include.jsp 页面的运行结果

注 意

include 指令在 JSP 页面被转换成 Servlet 时才将文件导入，这与<jsp:include>动作不同。

2.4.3　与标签相关的 taglib 指令

taglib 指令（又名标签指令）是 JSP 新增的一个指令，用户可以自定义新的标签在页面中执行。taglib 指令的语法如下：

```
<%@taglib    uri="tagliburl" prefix="tagPre" %>
```

其中 uri 属性用来表示自定义标签库的存放位置。prefix 属性是为了区分不同标签库的标签名，在页面中引用标签也是以 prefix 开头的。

现在比较流行的 JSTL、EL 标签是如何在 JSP 中使用的呢？下面通过实例进行说明如何利用 JSTL 标签输出 1~10。

注 意

JSTL 标签库会在后续的章节中进行详细说明，这里先演示它的基本用法。

【例 2.4】演示 taglib 指令。

```
---------------- index.jsp ----------------
01    <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02    <%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c"%>
03    <!DOCTYPE HTML>
04    <html>
05       <head>
06          <meta content="text/html;charset=UTF-8" http-equiv="Content-Type">
07          <title>JSP taglib 指令演示</title>
08       </head>
09       <body>
10             <table>
11                   <tr>
```

```
                    <td>输出值</td>
                </tr>
                <c:forEach begin="1" end="10" var="num">
                    <tr>
                        <td><c:out value="${num}"></c:out></td>
                    </tr>
                </c:forEach>
            </table>
    </body>
</html>
```

从上述代码可以看出，JSTL 标签使得 JSP 页面十分简洁，它不需要定义或初始化对象、方法。但凡事都不能只看表面，应该注重本质，标签的定制在页面显示时是如此简单，但在编制时却是一个复杂的过程，它通过一定的编程步骤将 JSP 代码和 Java 代码联系起来。标签定制的最大好处就是使得开发者的职责分工更加明细：标签定制者无须关注业务逻辑的实现，页面编程人员直接使用标签即可。这样两者就不会冲突，分工明确。下面简单介绍定制标签的过程。

1．标签库

标签库是用于定义标签的 XML 文件。该文件中包含了标签属性、标签名称、JSP 在处理标签时所需的类文件等信息。在使用自定义标签时，必须指明标签库所在的目录；在执行标签功能时，通过定义标签属性来决定显示的内容。下面看一段标签的源代码，其中包括了标签名定义、属性定义以及标签对应的 Java 类。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE taglib
   PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
   "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
        <tlib-version>1.0</tlib-version>
        <jsp-version>1.2</jsp-version>
        <short-name>lms</short-name>
        <!--   页面引用的 url 地址  -->
       <uri>/lms-tags</uri>

        <tag>
             <!--   标签名称  -->
             <name>page</name>
             <!--   标签处理类  -->
             <tag-class>com.common.model.PageTag</tag-class>
             <body-content>JSP</body-content>
             <!--   标签描述  -->
             <description>分页</description>
             <!--   标签属性  -->
             <attribute>
                  <!--   标签属性名称  -->
                  <name>object</name>
                  <!--   标签属性是否为必需，true 是必需，false 不是必需  -->
```

```
25                      <required>true</required>
26                      <!--   标签属性对应的处理类 -->
27                      <type>com.common.model.PageObject</type>
28                      <!--   标签属性描述 -->
29                      <description>PageObject 对象</description>
30              </attribute>
31          </tag>
32
33      <tag>
34               <!--   标签名称 -->
35              <name>substring</name>
36              <!--   标签处理类 -->
37              <tag-class>com.gsta.common.tag.SubStringTag</tag-class>
38              <description>substring</description>
39              <attribute>
40                      <name>content</name>
41                      <required>true</required>
42                      <type>java.lang.String</type>
43              </attribute>
44          </tag>
45  </taglib>
```

在上述代码中，第 6~8 行代码分别定义了标签的版本、JSP 的版本、标签的简称，第 12~31 行代码定义了 page 标签（第 21~30 行代码定义 page 标签的属性）。在定义属性名称的时候，可以定义该标签的数据类型以及该属性是否为必需的。例如第 41 行，JSP 页面就会根据其定义在执行时检查相关的错误信息。

2．标签处理类

其实标签处理类就是一个 Java 类，该 Java 类实现标签接口（javax.servlet.jsp.Tag），当自定义标签被 JSP 处理时即可被执行。每个标签中都需要执行的方法是：

```
Public int doStartTag() throws JspException
```

在标签中定义属性后，在 Java 类中就必须有这些属性并且设定其 get/set 方法。例如，在上面标签的源代码中与 substring 标签对应的是一个用于返回结果的 Java 类，在 substring 标签定义时，必须定义其 content 属性并给出 get/set 方法，代码如下所示：

```
01  private String content;
02  public void setContent(String content) {
03          this.content = content;
04  }
05
06  public void setLength(int length) {
07          this.length = length;
08  }
```

上述代码给出了属性 content 的 get/set 方法。

2.5 JSP 动作

上一节介绍了 JSP 的 page、include、taglib 这 3 个指令，这些指令不仅可以帮助 JSP 编程人员编写出复杂的 JSP 页面，而且能简化 JSP 页面代码，为后期维护带来便利。本节将介绍 JSP 中的另外一组元素：JSP 动作。在 JSP 中一共有 13 个动作：<jsp:include>、<jsp:forward>、<jsp:plugin>、<jsp:param>、<jsp:useBean>、<jsp:getProperty>、<jsp:setProperty>、<jsp:output>、<jsp:attribute>、<jsp:element>、<jsp:body>、<jsp:params>、<jsp:fallback>。其中<jsp:include>、<jsp:forward>、<jsp:param>使用较多，下面将进行主要介绍。

2.5.1 <jsp:include>动作

<jsp:include>动作与 include 指令十分相似，它们的作用都是引入文件到目标页面。<jsp:include>的语法格式如下：

```
<jsp:include page="relative URL" flush="true"/>
```

例如：

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

其中，page 是<jsp:include>动作的一个必选属性，它指明了需要包含文件的路径，该路径可以是相对的，也可以是绝对的。flush 属性用于指定输出缓存是否转移到被导入文件中。如果指定为 true，则包含在被导入文件中；如果指定为 false，则包含在源文件中。

注 意

page 属性所指的绝对路径不是文件系统的绝对路径，而是以应用系统目录为根目录的绝对路径。通常情况下，Web 应用下的 WEB-INF 目录就是根目录，一般在设定 URL 时指定相对路径，这样简单一些。

使用<jsp:include>动作与 include 指令可以达到相同的界面效果。例如，例 2.3 中介绍的使用 include 指令演示界面页脚的例子，直接将 include 替换成<jsp:include>动作，它们在页面显示结果中没有什么不同。

使用 jsp:include 包含页脚信息的代码如下：

```
---------- index_include.jsp ----------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
    <head>
      <meta content="text/html;charset=UTF-8" http-equiv="Content-Type">
      <title>JSP include 动作演示</title>
    </head>
    <body>
        <jsp:include page=" John.html " />
        <br/>
```

```
11              <div align="center">JSP include 指令演示</div>
12              <jsp:include    page="copyRight.jsp"    flush=”true”/>
13          </body>
14  </html>
```

在上述示例代码中，将第 9 行代码换成 jsp:include 动作。可以看出，两种不同的包含方式达到了相同的效果。

但是，jsp:include 动作与 include 指令还是有些不同的：首先，jsp:include 动作是在页面被访问时导入的，而 include 指令是由 JSP 引擎在编译时导入的；其次，在 include 指令中，被包含的文件会同主页面一块被编译为一个 Servlet 类文件，而 jsp:include 动作包含的文件跟主页面会是相对独立的两个文件，在编译时会被编译成两个 Servlet 类文件，因此 jsp:include 在效率上稍微慢些。

【例 2.5】演示 jsp:include 动作。

```
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02   <!DOCTYPE HTML>
03  <html>
04     <head>
05        <title>新闻摘要</title>
06         <meta http-equiv="pragma" content="no-cache">
07         <meta http-equiv="cache-control" content="no-cache">
08         <meta http-equiv="expires" content="0">
09         <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
10         <meta http-equiv="description" content="This is my page">
11     </head>
12     <body>
13           <br/>
14           <table align="center">
15                <tr>
16                     <th>中华新闻网</th>
17                </tr>
18           </table>
19           <br/>
20           <p>最新新闻摘要：
21               <ul>
22                    <li><jsp:include page="news1.html" /></li>
23                    <li><jsp:include page="news2.html"/></li>
24                    <li><jsp:include page="news3.html"/></li>
25               </ul>
26           </p>
27     </body>
28  </html>
```

在上述代码中，第 22~24 行代码应用了 JSP 动作，页面运行效果如图 2.5 所示。

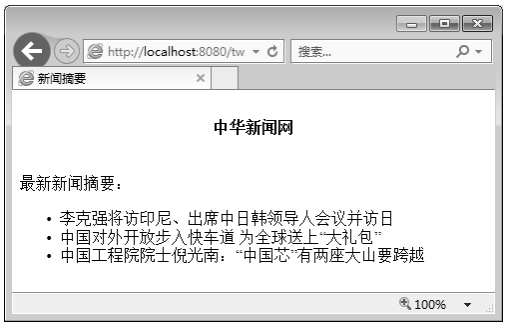

图 2.5　jsp:include 运行结果

2.5.2　<jsp:forward>动作

<jsp:forward>动作的作用是转发请求到另外一个页面中，在请求过程中会连同请求的参数数据一起被转发到目标页面中，目标页面通过 request.getParameter 方法获得参数值进行进一步处理。<jsp:forward>的基本语法如下：

```
<jsp:forward page="relative URL">
```

<jsp:forward>只有一个 page 属性，其值为相对的 URL。例如，执行如下代码将跳转到错误页面：

```
<jsp:forward page="/error.jsp"/>
```

下面来看一个<jsp:forward>的实例。如果随机数能被 2 整除，则跳转到 even.jsp，否则跳转到 odd.jsp。

【例 2.6】演示 jsp:forward 动作。

```
---------------------- forwardE.jsp--------------------------
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
        String url = "";
        int random = (int)(Math.random()*10);//产生 10 以内的随机数
        int m = random%2;
        switch(m){
                case 0:
                        url = "even.jsp";
                        break;
                case 1:
                        url = "odd.jsp";
                        break;
        }
 %>
<!DOCTYPE HTML>
<html>
   <head>
      <title>My JSP 'forwardE.jsp' starting page</title>
       <meta http-equiv="pragma" content="no-cache">
```

```
20          <meta http-equiv="cache-control" content="no-cache">
21          <meta http-equiv="expires" content="0">
22          <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
23          <meta http-equiv="description" content="This is my page">
24      </head>
25
26      <body>
27          <jsp:forward page="<%=url %>"/>
28      </body>
29  </html>
```

在上述代码中，第 2~14 行代码使用 Java 判断 url 的返回值，其中 even.jsp 中的内容为“能被 2 整除，跳转到偶数页界面”，odd.jsp 中的内容为“不能被 2 整除，跳转到奇数页界面”，效果如图 2.6 所示。

图 2.6　jsp:forward 的运行结果

从上述代码的运行结果可以看出，界面已经被重定向到 odd.jsp 页面，但是浏览器中的地址仍显示的是跳转前的地址，这也是<jsp:forward>动作的一个特点，即相对于请求者而言，所看到的响应仍然是原先请求的页面给出的，请求者并不会获得转发后的页面地址，因此相对来说，请求具有隐蔽性。

注　意

<jsp:forward>动作中的 url 页面只能是该 Web 应用中的文件，而不能是该 Web 应用之外的文件。

2.5.3　<jsp:param>动作

<jsp:param>用来传递参数信息，它经常与其他动作一起使用，例如与<jsp:forward>、<jsp:include>等结合使用，用于传递主页面的参数到目标页面。其基本语法如下：

```
<jsp:param name="参数名称" value="参数值">
```

例如：

```
<jsp:param name="username" value="李四">
```

【例 2.7】和【例 2.8】分别是<jsp:param>和<jsp:include>、<jsp:forward>结合使用的实例，由主页面和子页面构成。子页面从主页面中获得参数值显示。

【例 2.7】<jsp:param>和<jsp:include>结合使用。

```
---------------------- main.jsp--------------------------
01    <%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
```

```
02  <%request.setCharacterEncoding("utf-8"); //设定页面传递参数的编码格式%>
03  <!DOCTYPE HTML>
04  <html>
05     <head>
06       <title>主页面</title>
07     </head>
08
09     <body>
10
11                <table align="center">
12                     <tr>
13                          <th>主页面</th>
14                     </tr>
15                </table>
16
17           <jsp:include page="subPage.jsp" >
18                  <jsp:param value="John" name="userName"/>
19                  <jsp:param value="10086" name="passwd"/>
20                  <jsp:param value="北京丰台" name="address"/>
21           </jsp:include>
22     </body>
23  </html>
```

在上述代码中，第 17 行代码引用<jsp:include>动作，第 18~20 行代码使用<jsp:param>参数。

```
---------------------- subPage.jsp--------------------------
01  <%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
02  <%
03        String userName= request.getParameter("userName");
04        String passwd= request.getParameter("passwd");
05        String address= request.getParameter("address");
06        System.out.println(address);
07   %>
08  <!DOCTYPE HTML>
09  <html>
10     <head>
11        <title>子页面</title>
12     </head>
13     <body>
14
15                <table align="center">
16                     <tr>
17                          <th>子页面：人员信息</th>
18                     </tr>
19                </table>
20
21
22                <table align="center">
23                      <tr>
24                          <td>用户名：<%=userName%></td>
```

```
                    </tr>
                    <tr>
                        <td>密    码：<%=passwd%></td>
                    </tr>
                    <tr>
                        <td>用户地址：<%=address%></td>
                    </tr>
                </table>

    </body>
</html>
```

在上述代码中，第 22~32 行代码用于获取主页面传递的参数值。

注 意

所有的页面编码格式要统一，否则在传递中文时会出现乱码问题。request.setCharacterEncoding 方法是设定传递参数的编码格式，在传递中文时要特别注意。在 URL 中包含参数传递时，可能会有中文乱码或者特殊字符问题，需要对这些参数进行编码处理。在 JavaScript(简称 JS)中，通过调用函数 encodeURI 或者 encodeURIComponent 实现编码，在 JSP 脚本中，通过方法 URLEncoder.encode 实现转码。

页面效果如图 2.7 所示。

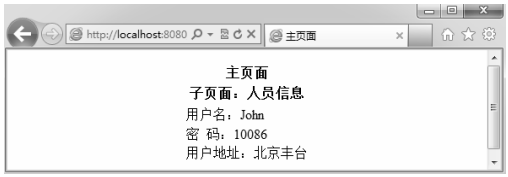

图 2.7　<jsp:param>和<jsp:include>结合使用的运行结果

【例 2.8】<jsp:param>和<jsp:forward>结合使用。

```
---------------------- main.jsp--------------------------
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%request.setCharacterEncoding("utf-8"); //设定页面传递参数的编码格式%>
<!DOCTYPE HTML>
<html>
    <head>
      <title>主页面</title>
    </head>

    <body>

                <table align="center">
                    <tr>
                        <th>主页面</th>
                    </tr>
                </table>
```

```
16
17          <jsp:forward page="subPage.jsp" >
18                  <jsp:param value="Smith" name="userName"/>
19                  <jsp:param value="10086" name="passwd"/>
20                  <jsp:param value="北京丰台" name="address"/>
21          </jsp:forward>
22      </body>
23  </html>
```

在上述代码中，第 2 行设置请求编码，第 17 行代码使用<jsp:forward>动作，第 18~20 行代码使用<jsp:param>参数。

```
---------------------- subPage.jsp---------------------------
01  <%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
02  <%
03      String userName= request.getParameter("userName");
04      String passwd= request.getParameter("passwd");
05      String address= request.getParameter("address");
06      System.out.println(address);
07   %>
08  <!DOCTYPE HTML>
09  <html>
10      <head>
11          <title>子页面</title>
12      </head>
13      <body>
14
15              <table align="center">
16                  <tr>
17                      <th>子页面：人员信息</th>
18                  </tr>
19              </table>
20
21
22              <table align="center">
23                  <tr>
24                      <td>用户名：<%=userName%></td>
25                  </tr>
26                  <tr>
27                      <td>密     码：<%=passwd%></td>
28                  </tr>
29                  <tr>
30                      <td>用户地址：<%=address%></td>
31                  </tr>
32              </table>
33
34      </body>
35  </html>
```

在上述代码中，第 2~7 行代码用于获取主页面传递的参数值，第 24~30 行代码用于显示获取的参数值，页面效果如图 2.8 所示。

图 2.8 <jsp:param>与<jsp:forward>结合使用的运行结果

从上述实例可以看出，<jsp:param>在<jsp:include>、<jsp:forward>中的用法基本一样，只要将<jsp:include>替换成<jsp:forward>动作就可以了。

2.6 小结

本章介绍了 JSP 注释、JSP 声明、JSP 表达式、JSP 指令以及 JSP 动作，这些都是编写 JSP 页面所必须了解和掌握的知识，其中掌握 JSP 指令是编写 JSP 页面的必要条件，JSP 动作能简化 JSP 页面编程。在 JSP 动作中<jsp:include>和<jsp:forward>经常用到，读者必须掌握其用法。

2.7 习题

1. 在 JSP 中应该如何声明一个变量？

2. JSP 的注释有哪几种？

3. JSP 的表达式如何表达？

4. JSP 的指令有哪几种？

5. JSP 的动作有哪些？include 指令和<jsp:include>动作有什么不同？

6. JSP 页面由哪三类元素组成？各有什么用？

7. 请编写一个 JSP 页面，网页有上部、中间、下部 3 部分，请通过由一个主 JSP 页面包含三个子页面的方式实现页面效果，要求利用两种方式实现，效果如图 2.9 所示。

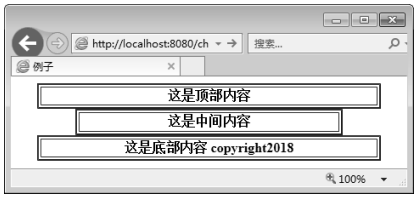

图 2.9 例图

第 3 章　JSP 内置对象

JSP 内置对象的含义是可以直接在 JSP 页面中使用的对象，使用前不需要声明它们。若能熟悉并了解 JSP 内置对象，可以方便读者更好地操作页面、开发页面、完成更复杂的业务流程。

本章的主要内容如下：

- 讲解 7 个内置对象 request、response、session、application、out、page、config 的作用和使用方法
- 了解 JSP 的 4 个作用域

3.1　request 对象

本节将要介绍 request 对象的作用范围及其常用的方法。用户每访问一个页面，就会产生一个 HTTP 请求。这些请求中一般都包含了请求所需的参数值或者信息，如果将 request 对象看作是客户请求的一个实例，那么这个实例就包含了客户请求的所有数据。因此，可以通过 request 来获取客户端和服务器端的信息，如 IP 地址、传递的参数名和参数值、应用系统名称、服务器主机名称等。

3.1.1　request 对象的常用方法

request 对象的常用方法参见表 3.1。

表 3.1　request 对象的常用方法

方法	方法说明
getParameter()	取得请求中指定的参数值，返回 String 类型，如果有必要，需要将取得的参数值转换为合适的类型
getParameterValues()	将同名称的参数一次性地读入 String 类型的数组中
getParameterNames()	获取参数名称，返回枚举类型
getMethod()	获取客户提交信息的方式，即 post 或 get
getServletPath()	获取 JSP 页面文件的目录
getHeader()	获取 HTTP 头文件中的指定值，例如 accept、user-agent、content-type、content-length 等
getRemoteAddr()	获取客户的 IP 地址
getServerName()	获取服务器的名称
getServerPort()	获取服务器的端口号
getContextPath()	获取项目名称，如果项目为根目录，则得到空的字符串
getHeaders()	获取表头信息，返回枚举类型

3.1.2 使用 request 对象接收请求参数

request对象有两种方法获得请求参数值：一个是getParameter()；另一个是getParameterValues()。下面将演示如何使用 getParameter()方法获取页面请求参数。

【例 3.1】获取网页请求参数。

getParameter.jsp 用于获取页面参数值，代码如下：

```
---------- getParameter.jsp ----------
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <!DOCTYPE HTML>
03  <html>
04    <head>
05      <title>' getParameter.jsp'</title>
06       <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
07    </head>
08    <body style="text-align: center;">
09
10                <%
11                    String name= request.getParameter("name");
12                    String city = request.getParameter("city");
13                    if(name!=null&&city!=null)
14                    {
15                 %>
16                    <p>Welcome    <%=name %>,您所在的城市是<%=city %></p>
17                  <%
18                     }else{
19                    %>
20                    <p>欢迎访问本页面！</p>
21                    <%
22                       }
23                     %>
24
25    </body>
26  </html>
```

在上述代码中，第 11、12 行代码分别用于获得 name、city 的参数值。

在访问地址上输入“http://localhost:8080/ch03/getParameter.jsp?name=John&city=Beijing”，页面效果如图 3.1 所示。

图 3.1 例 3.1 的运行结果

注 意

在 URL 参数传递时，页面地址后使用“？”连接请求参数；参数由“=”相连，表示参数名和参数值；一个请求携带多个参数时用“&”连接。

实际上，上述传递参数的方法在实际的开发中相比表单提交参数要较少使用，因为这种提交参数的方法完全将参数暴露出来，不利于隐私保护，所以我们一般采取表单提交的方式传递参数。

在表单 form 中，有两个非常重要的属性：action 和 method。属性 action 指明表单提交后的数据跳转到指定页面并处理参数。method 有两个值，分别是 get 和 post，通常指定为 post，因为指定为 get 时，在表单中设定的参数和参数值将附加到页面地址的末尾以参数的形式提交，这样会暴露参数值，不安全；而指定为 post 并提交表单时，表单中的参数将被作为请求头中的信息发送，不会在目标地址中添加任何参数，这样也就不会暴露出参数值，所以出于安全考虑，一般会选择 post 提交。

【例 3.2】指定为 post 并提交，以获取网页请求参数。

ex3_1.jsp 页面将用户填写的内容提交给 getParameter.jsp，页面关系如图 3.2 所示。

图 3.2　例 3.2 的页面关系图

ex3_1.jsp 是用户提交页面，源代码如下：

```
---------- ex3_1.jsp ----------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
    <head>
        <title>'ex3_1.jsp'</title>
         <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
    </head>

    <body >

            <form action="getParameter.jsp" method="post">
                <table align="center">
                        <tr>
                            <td>姓名</td>
                            <td><input type="text" name="name" value=""/></td>
                        </tr>
                        <tr>
                            <td>城市</td>
                            <td><input type="text" name="city" value=""/></td>
                        </tr>
                        <tr>
                            <td><input type="submit" value="提交"/></td>
```

```
23                                  <td><input type="reset" value="重置"/></td>
24                              </tr>
25                          </table>
26                      </form>
27
28      </body>
29  </html>
```

在上述代码中，第 11~26 行代码用于提交表单 form，其中 action 为 getParameter.jsp、method 方法为 post，其中第 12~25 行代码用于输入姓名和城市。此处的 getParameter.jsp 与例 3.1 中的 getParameter.jsp 代码相同。

在浏览器中访问 ex3_1.jsp 并在“姓名”和“城市”输入框中输入值“John”和“Beijing”，单击“提交”按钮后，可以看到显示的结果与例 3.1 的结果是一样的。

在测试时，读者可以设定 method 的两个值分别进行测试，并查看地址栏的变化情况。

3.1.3 请求中文乱码的处理

在上述实例中，请求信息中的参数值都是英文，但在实际开发中，请求信息通常是包含中文的。但在上述的例子中，若是使用中文，例如在表单的“姓名”输入框中输入中文字符，则在提交之后显示的值不是中文而是“？”或者其他字符，如图 3.3 所示。

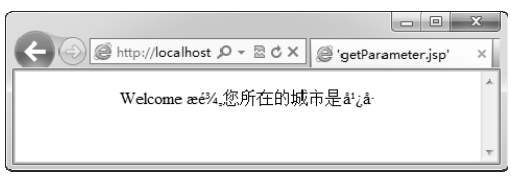

图 3.3 显示奇怪的字符

产生这种错误的原因是请求信息所使用的字符集与页面使用的字符集不同，有如下 3 种方法可以解决这个问题。

- 第 1 种方法是在接收请求的页面中规定请求字符编码的代码，例如在例 3.2 的 getParameter.jsp 页面中添加语句“request.setCharacterEncoding("utf-8")”即可。
- 第 2 种方法是在取得参数值后，通过转码的方式将值转为合适的字符集。例如，将前面例子中获得参数的代码修改为如下形式:

```
String name= new String(request.getParameter("name").getBytes("ISO-8859-1"),"utf-8");
String city= new String(request.getParameter("city").getBytes("ISO-8859-1"),"utf-8");
```

通过上述两种方法都可以解决中文乱码问题，但是在代码维护方面却有很大麻烦，因为它不但增加了代码量，而且移植性也差。

- 第 3 种方法，通过编写一个 Servlet 过滤器来解决中文乱码问题，并可以通过配置让过滤器解决所有请求处理字符集的问题，请求处理页面就不用关心字符集处理了。第 3 种方法可以有效防止上述问题且移植性强。

3.1.4 获取请求的头部信息

请求的头部信息在实际中也是有其重要作用的，这些信息有时候对服务器的响应特别有用，而且也能查看到服务器中的相关信息。

请求头部信息的相关方法除了第 3.1.1 小节提到的 getHeaders()外，还有以下方法。

- String getHeader(String name)：获取字符串型的表头信息。
- int getIntHeader(String name)：获取整型的表头信息。

【例 3.3】获取请求中所有的头名称和对应的值。

getHeaders.jsp 页面用于获取请求中的所有头名称和对应的值，其源代码如下：

```
---------- getHeaders.jsp ----------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>'getHeaders.jsp'</title>
     <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
  </head>

  <body >
            <%
                Enumeration<String> enumeration = request.getHeaderNames();
                while(enumeration.hasMoreElements()){
                    String name = enumeration.nextElement();
                    String value = request.getHeader(name);
                    if(value==null||"".equals(value)){
                        value="空字符串";
                    }
             %>
                  <p>表头名称：<%=name %>  对应的值：<%=value %></p>
              <%} %>
  </body>
</html>
```

在上述代码中，第 11~17 行代码获取请求的 Header 信息，并循环遍历输出。在浏览器中直接输入页面地址，效果如图 3.4 所示。对这些头信息的含义说明如下。

- accept：客户端能接收的 MIME 类型。
- accept-language：浏览器的首选语言。
- user-agent：客户端程序的相关信息，例如浏览器版本、操作系统类型等。

- host：表示服务器的主机名和端口号。
- ua-cpu：CPU 类型。
- connection：判断客户端是否可以持续性地连接 HTTP。
- accept-encoding：指明客户端能够处理的编码类型有哪些。
- cookie：会话的信息，每个会话都有一个会话 ID 或者其他信息。

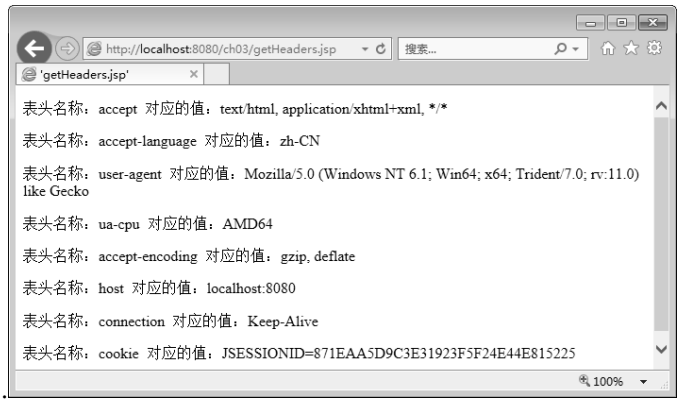

图 3.4　网页的请求头信息

以上是通过直接在地址栏中输入页面地址获得的头文件信息。如果通过 post 方式跳转到页面，那么头文件信息会稍微多些。例如，在一个表单中将 method 的属性指定为 post，然后将 action 指定为例 3.3 的跳转页面，则显示结果如图 3.5 所示。

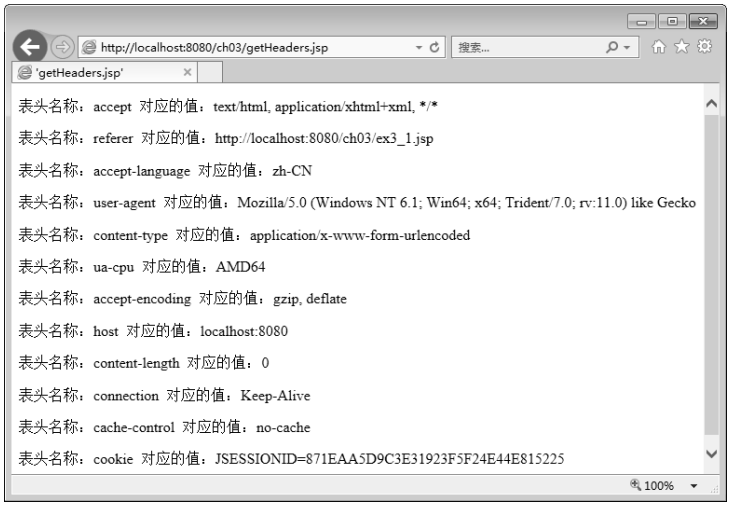

图 3.5　利用 post 方式获取的请求头信息

从结果对比来看，增加了以下几个信息。

- referer：向页面发起请求的地址。
- content-type：指明表单的 MIME 编码类型，一般值为 text/plain、multipart/form-data、application/x-www-form-urlencoded。默认为 application/x-www-form-urlencoded，如果表

单上传一个文件，则表单中的 enctype 设置为 multipart/form-data。

- content-length：用于设置提交请求中的数据长度，表单中提交的数据越多，长度就越大。
- cache-control：网页缓存控制。默认值为 no-cache，表明每次访问都从服务器中获取页面。

在一般的开发中，我们不需要关注请求的头信息，针对不同的需求，在网页中设置相关的属性值就可以了。但是，当我们了解它们之后，在解决某些问题上将起到关键的作用。

3.1.5 获取主机和客户机的信息

获取主机和客户机的信息，一般通过以下几个方法实现。

- getRemoteAddr()：获取客户主机的 IP。
- getRemoteHost()：获取客户主机的名称。
- getLocalAddr()：获取本地主机的 IP。
- getLocalHost()：获取本地主机的名称。
- getServerName ()：获取服务器主机的名称。
- getServerPort()：获取服务器端口。

【例 3.4】获取客户机和主机信息。

getHostInfo.jsp 是获取客户机和本地主机信息的页面，其源代码如下：

```
---------------- getHostInfo.jsp ----------------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>'getHostInfo.jsp'</title>
     <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
  </head>

  <body >
        <p>
              本地机器 IP：<%=request.getLocalAddr() %><br>
              本地机器名称：<%=request.getLocalName() %><br>
              本地机器端口：<%=request.getLocalPort() %><br>
          </p>
          <p>
              客户主机 IP：<%=request.getRemoteAddr() %><br>
              客户主机名称：<%=request.getRemoteHost() %><br>
              客户主机端口：<%=request.getRemotePort()%><br>
          </p>
          <p>
              服务器 IP：<%=request.getServerName() %><br>
              服务器端口：<%=request.getServerPort() %><br>
          </p>
    </body>
</html>
```

在上述代码中，第 11~13 行代码用于获取本地机器信息，第 16~18 行代码用于获取客户主机信息，第 21~22 行代码用于获取服务器信息。页面效果如图 3.6 所示。

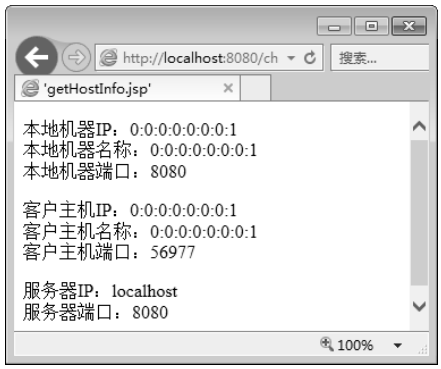

图 3.6 获取客户机和主机信息

一般而言，服务器主机名与本地机器 IP 应该是相同的，端口号也应该一样。

3.2 response 对象

本节将介绍 response 对象的常用方法。当用户访问一个页面时，就会产生一个 HTTP 请求，服务器做出响应时调用的是 response 响应包。response 响应包实现的是接口 javax.servlet.http.HttpServletResponse。

3.2.1 response 对象的常用方法

response 对象的常用方法参见表 3.2。

表 3.2 response 对象的常用方法

方法	方法说明
addHeader(String name,String value)	向页面中添加头和对应的值
addCookie(Cookie cookie)	添加 Cookie 信息
sendRedirect(String uri)	实现页面重定向
setStatus(int code)	设定页面的响应状态代码
setContentType(String type)	设定页面的 MIME 类型和字符集
setCharacterEncoding(String charset)	设定页面响应的编码类型
setHeader(String name,String value)	

3.2.2 设置头信息

设置头信息包括设置页面返回的 MIME 类型、返回的字符集、页面中的 meta 等信息。其中设

置 MIME 类型和返回的字符集尤为重要，因为它关系到页面的显示是否会出现乱码。有两种方法用于设置 MIME 类型和返回的字符集，分别如下：

- response.setContentType(String type)：其中 type 的值为“text/html;charset=utf-8”，当然也可以为其他的 MIME 类型和字符集。
- page 指令：基本固定格式为“<%@ page contentType="text/html;charset=utf-8" language="java" %>”。

【例 3.5】setContentType 的用法示例。

setContentType.jsp 用来设置 MIME 类型和字符集，MIME 设置为“text/html”，字符集设置为“UTF-8”，其源代码如下：

```
----------------setContentType.jsp----------------
<%
    response.setContentType("text/html;charset=UTF-8");        //设置字符集和 MIME 类型
    String str = new String("这是测试例子".getBytes("ISO-8859-1"),"utf-8");
%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>setContentType.jsp</title>
  </head>

  <body>
      <p>这里是一段中文字符</p>
      </br>
      <%=str %>
  </body>
</html>
```

页面效果如图 3.7 所示。

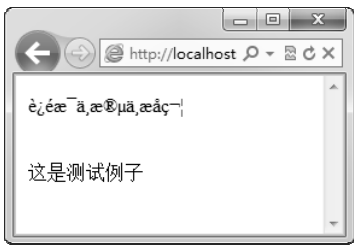

图 3.7　setContentType 运行示例

从运行结果可以看出，脚本中经过转码的文字显示正常，但在 HTML 中的字却是乱码。

使用 page 指令设定字符集时，在写法和处理上都相对简单些，HTML 中的中文字不需要转码，而且脚本中的中文字也不需要转码。

【例 3.6】利用 page 指令设置页面字符集。

```
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <%
03      String str = "这是测试例子";
04   %>
05  <!DOCTYPE HTML>
06  <html>
07    <head>
08      <title>page  指令 setContentType.jsp</title>
09    </head>
10
11    <body>
12        <p>这里是一段中文字符</p>
13        </br>
14        <%=str %>
15    </body>
16  </html>
```

在上述代码中，第 1 行代码利用 page 指令设置页面字符集，页面效果如图 3.8 所示。

图 3.8　利用 page 指令设置字符集

在 JSP 中的页面设置字符集会破坏 JSP 容器自身的页面编码处理，所以不建议设置，但可以在 Servlet 中设定。利用 page 指令设置字符集相对简单，所以在开发中一般都采用这种模式。

设置头 meta 信息是指在 HTML 页面中存在于<head></head>之间的信息。例如：

- <meta http-equiv="pragma" content="no-cache">: 设定禁止浏览器从本地缓存中调取页面内容，设定后离开网页就不能从 Cache 中再调用。
- <meta http-equiv="cache-control" content="no-cache">: 请求和响应遵循的缓存机制策略。
- <meta http-equiv="expires" content="0"> : 用于设定网页的到期时间，一旦过期就必须到服务器上重新调用。
- <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">: 向搜索引擎说明网页的关键字。
- <meta http-equiv="description" content="This is my page">: 向搜索引擎说明网页的主要内容。

【例 3.7】设定 meta 头信息。

页面 setMeta.jsp 用于设置 meta 中的信息，其源代码如下：

```
----------------setMeta.jsp----------------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>My JSP 'setMeta.jsp' starting page</title>
  </head>
  <body>
    <div class="aa" style="text-align: center;">
      <p class="bb">
        现在的时间为：<br/>
        <%
            out.print(""+ new Date());
            response.setHeader("refresh","1");
            response.setHeader("description","实时显示当前时间");
            response.setHeader("keywords","实时,显示,显示当前时间");
            response.setHeader("cache-control","no-cache");
         %>
        <br/><br/>
        copyright:2018
    </div>
  </body>
</html>
```

在上述代码中，第 12~16 行代码用于设置头信息，页面效果如图 3.9 所示。

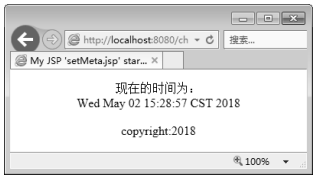

图 3.9　设置 meta 头信息

从运行结果可以看出在 response 中设置 meta 信息的效果，但在实际开发中是直接在 HTML 标记中写定这些值，而不是在响应中设定。

3.2.3　设置页面重定向

重定向是指一个页面在收到一个访问请求后，根据请求的 URL 重新跳转到其他的页面。设置重定向的方法是：

```
response.sendRedirect(String url);
```

其中 url 代表了跳转路径，路径可以是相对路径，也可以是绝对路径。

【例 3.8】设定页面重定向。

在 sendRedirect.jsp 页面中设定了跳转到一个不存在的页面，其源代码如下：

```
----------------sendRedirect.jsp----------------
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <%
03      response.sendRedirect("sendPageError.jsp");
04   %>
05  <!DOCTYPE HTML>
06  <html>
07    <head>
08      <title>My JSP 'sendRedirect.jsp' starting page</title>
09    </head>
10
11    <body>
12      This is my JSP page. <br>
13    </body>
14  </html>
```

运行结果如图 3.10 所示。

图 3.10　设置页面重定向

从运行结果可以看出跳转页面时，URL 地址改变了且不显示 sendRedirect.jsp 中的页面信息。重定向的运行过程如图 3.11 所示。

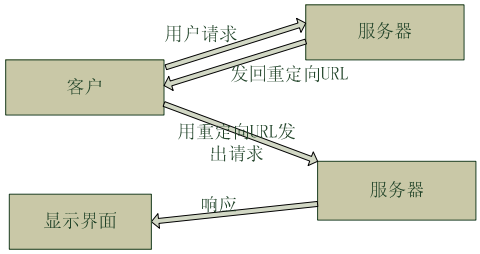

图 3.11　重定向的运行过程

3.3 session 对象

在 Web 开发中，session 对象同样占据着极其重要的位置，它是一个非常重要的对象，可以用来判断是否为同一用户，还可以用来记录客户的连接信息等。HTTP 协议是一种无状态的协议（不保存连接状态的协议），每次用户请求在接收到服务器的响应后，连接就关闭了，服务器端与客户端的连接被断开，因此，如果用户的浏览器还没关闭时又发起请求，那么网站就应该识别出该用户的情况。在这种情况下，session 对象就起到了关键作用。session 的相关概念见表 3.3，常用方法见表 3.4。

表 3.3 session 的相关概念

名称	说明
会话	从用户打开浏览器连接到一个 Web 应用或者是某个界面，直至关闭浏览器这个过程称为一个会话。其实打开一个浏览器就意味着打开了一个会话对象
session 对象生命周期	从用户访问某个页面到关闭浏览器这段时间称为 session 对象的生命周期，也可以说从会话开始到结束这段时间为 session 对象的生命周期
session 对象与 Cookie 对象	session 对象与 Cookie 对象是一一对应关系。JSP 引擎会将创建好的 session 对象存放在对应的 Cookie 中

表 3.4 session 的常用方法

方法	说明
void setAttribute(String name, Object value)	将参数名和参数值存放在 session 对象中
Object getAttribute(String name)	返回 session 中与指定参数绑定的对象，如果不存在就返回 null
Enumeration getAttributeName()	一个用户一个线程，从而保证多个用户单击同一页面时 session 对象的唯一性
String getId()	获取 session 对象的 ID 值
void removeAttribute(String name)	移除 session 中指定名称的参数
long getCreationTime()	获取对象创建的时间，返回结果是 long 型的毫秒数
int getMaxInactiveInterval()	获取 session 对象的有效时间
void setMaxInactiveInterval()	设置 session 对象的有效时间
boolean isNew()	用于判断是否为一个新的客户端
void invalidate()	使 session 对象不合法，即失效

3.3.1 获取 session ID

获取 session 对象 ID 可以判断会话是否为同一会话，用户可以通过会话中的信息来进行相关的操作。

【例 3.9】获取 session ID 值。

创建两个 Web 目录并部署应用，页面之间通过超链接联系起来。

假设我们有 3 个页面：ex3_2.jsp、ex3_3.jsp、ex3_4.jsp，ex3_2.jsp 和 ex3_3.jsp 部署在同一应用下，ex3_4.jsp 部署在另外一个应用中。ex3_2.jsp 通过表单提交到 ex3_3.jsp，ex3_3.jsp 通过超链接指向 ex3_4.jsp，ex3_4.jsp 通过表单提交指向 ex3_2.jsp，ex3_2.jsp 可以通过超链接指向 ex3_4.jsp，ex3_3.jsp 也可以通过超链接指向 ex3_2.jsp。它们之间的关系如图 3.12 所示。

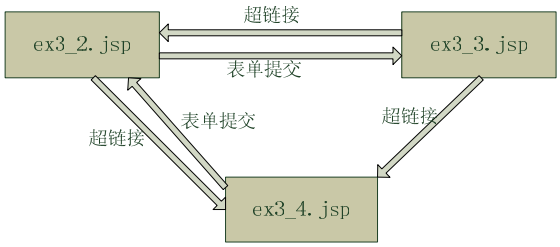

图 3.12　页面之间的关系图

ex3_2.jsp 发送请求和超链接到 ex3_3.jsp、ex3_4.jsp，其源代码如下：

```
----------------ex3_2.jsp----------------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>My JSP 'ex3_2.jsp' starting page</title>
     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="cache-control" content="no-cache">
  </head>
  <body>
  <%
          String sessionID = session.getId();
          session.setAttribute("name","John");                    //存储参数 name
          String author = (String)session.getAttribute("author");  //强制转换为 String 类型
          long time = session.getCreationTime();
          Date date = new Date(time);
   %>
   <div style="text-align: center;">
          <p>您访问的是 ex3_2.jsp 页面</br>
            <%=author %>, 您的 session 对象 ID 为：</br>
             <%=sessionID%></br>
             session 对象创建时间：<%=date %>
             </br>
          </p>
          <form action="ex3_3.jsp" method="post">
              <input type="submit" value="转向 ex3_3.jsp"/>
          </form>
          <a href="../ch03/ex3_4.jsp">欢迎到 ex3_4.jsp 页面</a>
   </div>
```

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

在上述代码中，第 10~16 行代码用于获取 session 中的 author 参数值，并设置当前日期，第 19~21 行代码用于显示获取的参数值，页面效果如图 3.13 所示。

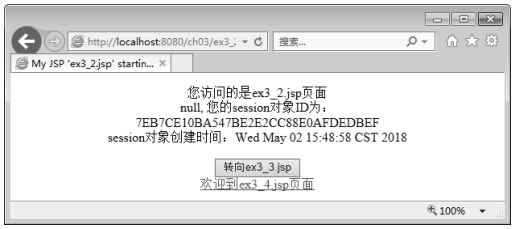

图 3.13　ex3_2.jsp 运行结果

从图 3.13 可以看出，页面输出了 session 对象的 ID 值和创建的时间，但是没有获得从页面 ex3_4.jsp 传来的参数值 author。

ex3_3.jsp 超链接到 ex3_2.jsp 和 ex3_4.jsp，其源代码如下：

```
----------------ex3_3.jsp----------------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>My JSP 'ex3_2.jsp' starting page</title>
     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="cache-control" content="no-cache">
  </head>
  <body>
  <%
          String sessionID = session.getId();
          String name = (String)session.getAttribute("name");//强制转换为 String 类型
          long time = session.getCreationTime();
          Date date = new Date(time);
   %>
   <div style="text-align: center;">
          <p>您访问的是 ex3_3.jsp 页面</br>
              <%=name %>,您的 session 对象 ID 为：</br>
              <%=sessionID%></br>
              session 对象创建时间：<%=date %>
              </br>
          </p>
          <a href="ex3_2.jsp"><%=name %>,欢迎到 ex3_2.jsp 页面</a></br>
          <a href="../ch03/ex3_4.jsp"><%=name %>,欢迎到 ex3_4.jsp 页面</a>
   </div>
  </body>
</html>
```

运行结果如图 3.14 所示。从中可以看出，页面 ex3_3.jsp 获得的 session 对象 ID 和创建时间与

ex3_2.jsp 中的 session 对象 ID 和创建时间是一样的，且获得了参数 name 的值。

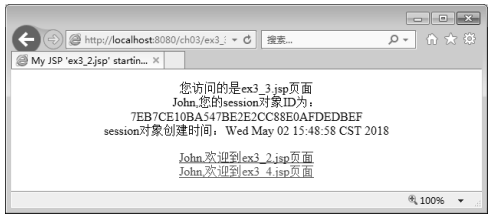

图 3.14　ex3_3.jsp 运行结果

ex3_4.jsp 发送请求到 ex3_2.jsp，其源代码如下：

```
----------------ex3_4.jsp----------------
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <!DOCTYPE HTML>
03  <html>
04    <head>
05      <title>My JSP 'ex3_4.jsp' starting page</title>
06       <meta http-equiv="pragma" content="no-cache">
07       <meta http-equiv="cache-control" content="no-cache">
08    </head>
09    <body>
10    <%
11            String sessionID = session.getId();
12            String name = (String)session.getAttribute("name");        //强制转换为 String 类型
13            session.setAttribute("author","Smith");                    //存参数 author
14            long time = session.getCreationTime();
15            Date date = new Date(time);
16     %>
17     <div style="text-align: center;">
18            <p>您访问的是 ex3_4.jsp 页面</br>
19              <%=name %>,您的 session 对象 ID 为：</br>
20              <%=sessionID%></br>
21              session 对象创建时间：<%=date %>
22              </br>
23            </p>
24            <form action="../ch03/ex3_2.jsp" method="post">
25                <input type="submit" value="转向 ex3_2.jsp"/>
26            </form>
27     </div>
28    </body>
29  </html>
```

在上述代码中，第 11~15 行代码用于获取 session 中的用户名参数，第 18~23 行代码用于显示参数值，页面效果如图 3.15 所示。

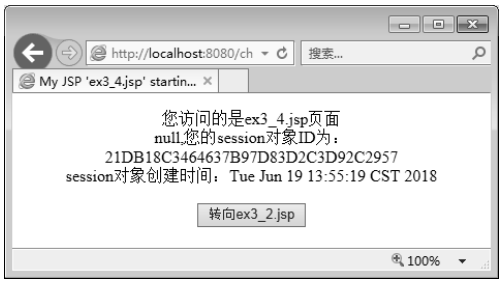

图 3.15　ex3_4.jsp 运行结果

从图 3.15 中可以看出，页面输出了 session 对象的 ID 值和创建时间，但是它和页面 ex3_2.jsp、ex3_3.jsp 的 session 对象 ID 值不同，并且页面没有获得从 ex3_2.jsp 传来的参数值 name。

从以上的结果可以看出：一个 Web 应用的 session 对象的 ID 值是唯一的，并且两个应用之间的参数利用 session 对象是获取不到值的。

注　意

请读者务必要理解 session 对象的生命周期。

3.3.2　用户登录信息的保存

【例 3.10】用户登录信息的保存。

login.jsp 是用户登录页面，validate.jsp 页面是验证用户合法性页面，class.jsp 是登录成功后显示班级管理的页面，logout.jsp 是退出登录页面。它们之间的关系如图 3.16 所示。

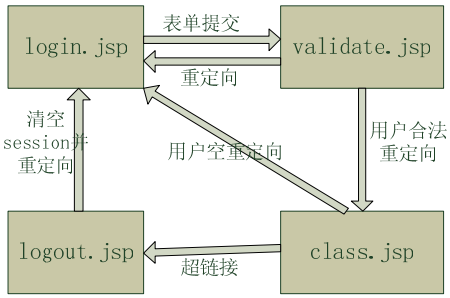

图 3.16　例 3.10 中页面之间的关系

login.jsp 是用户登录页面，其源代码如下：

```
----------------login.jsp----------------
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <%
03  String path = request.getContextPath();
04  String basePath =
05    request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
06  %>
07  <!DOCTYPE HTML>
```

```
08  <html>
09    <head>
10      <base href="<%=basePath%>">
11      <title>用户登录</title>
12       <meta http-equiv="pragma" content="no-cache">
13       <meta http-equiv="cache-control" content="no-cache">
14       <meta http-equiv="expires" content="0">
15       <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
16       <meta http-equiv="description" content="This is my page">
17    </head>
18
19    <body>
20          <div style="text-align: center;">
21                  <span style="font-size: 26px;">用户登录</span>
22                  <hr/>
23                  <form action="validate.jsp" method="post">
24                          用户名称：<input type="text" name="username"/>
25                  <br/>
26                          用户密码：<input type="password" name="password"/>
27                  <br/>
28                  <input type="submit" value="登录"/>
29                  </form>
30          </div>
31
32    </body>
33  </html>
```

在上述代码中，第 23~29 行代码利用 form 表单提交登录信息。

validate.jsp 是验证用户合法性页面，其源代码如下：

```
----------------validate.jsp----------------
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <!DOCTYPE HTML>
03  <html>
04    <head>
05      <title>success.jsp</title>
06       <meta http-equiv="pragma" content="no-cache">
07       <meta http-equiv="cache-control" content="no-cache">
08       <meta http-equiv="expires" content="0">
09       <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
10       <meta http-equiv="description" content="This is my page">
11    </head>
12     <%!
13          //声明一个用户集合，模拟从数据库中取出用户集
14          Map<String, String> map =new HashMap<String, String>();
15          //声明验证的标识
16          boolean flag = false;
17     %>
18       <%
```

```
19              //向集合添加数据
20              map.put("John","123456");
21              map.put("Smith","222222");
22              map.put("Bob","333333");
23              map.put("Bruth","666666");
24          %>
25          <%!
26              //声明验证方法
27              boolean validate(String username,String password){
28                      String passwd = map.get(username);
29                      if(passwd!=null&&passwd.equals(password)){
30                          return true;
31                      }else{
32                        return false;
33                      }
34              }
35          %>
36          <%
37                  //获得页面提交的用户名与密码
38                  String username = request.getParameter("username");
39                  String password = request.getParameter("password");
40                  if(username==null||username==""||password==null||password==""){
41                      response.sendRedirect("login.jsp");
42                  }
43                  flag = validate(username,password);
44                  if(flag){
45                      //保存在 session 对象中
46                      session.setAttribute("username",username);
47                      session.setAttribute("password",password);
48                      response.sendRedirect("class.jsp");
49                  }
50              %>
51          <body>
52              <div style="text-align: center;">
53                  <span style="font-size: 26px;">用户登录</span>
54              </div>
55              <br/>
56              <div style="text-align: center;">
57                  <%if(!flag){ %>
58                      <a href="login.jsp">重新登录系统</a>
59                  <%} %>
60              </div>
61      </body>
62  </html>
```

在上述代码中，第 18~24 行用于设置用户集合；第 25~35 行代码用于声明验证用户合法性的方法；第 38~49 行代码用于获取参数值，并判断其合法性，合法的保存在 session 中，否则跳转到 login.jsp 中。

class.jsp 是登录成功后显示班级管理的页面，其源代码如下：

```
----------------class.jsp---------------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
    String name = (String)session.getAttribute("username");
    if(name==null){
        response.sendRedirect("login.jsp");
    }
%>

<!DOCTYPE HTML>
<html>
  <head>
    <title>My JSP 'score.jsp' starting page</title>
     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="cache-control" content="no-cache">
     <meta http-equiv="expires" content="0">
     <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
     <meta http-equiv="description" content="This is my page">
  </head>

  <body>
        <div   style="text-align: center;">
            <span style="font-size: 24px;">班级管理</span>
        <hr/>
        <h3>学生：<%=name %></h3>
        <table>
            <tr>
                <td>
                    <a href="addClass.jsp">班级录入</a>
                </td>
                <td>
                    <a href="modifyClass.jsp">班级修改</a>
                </td>
                <td>
                    <a href="queryClass.jsp">班级查询</a>
                </td>
                <td>
                    <a href="delClass.jsp">班级删除</a>
                </td>
            </tr>
        </table>
        <a href="logout.jsp">退出登录</a>
        </div>
  </body>
</html>
```

在上述代码中，第 27~38 行代码用于列出班级的各项方法。

logout.jsp 是退出登录页面，其源代码如下：

```
----------------logout.jsp----------------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
    String username = (String)session.getAttribute("username");
    session.removeAttribute("John");
    session.invalidate();
    response.sendRedirect("login.jsp");
%>

<!DOCTYPE HTML>
<html>
  <head>
    <title>My JSP 'logout.jsp' starting page</title>
  </head>
  <body>
  </body>
</html>
```

在 login.jsp 页面中，第 23 行代码利用 form 表单的 post 方式提交，action 指向 validate.jsp 页面。在 validate.jsp 页面中，第 12~17 行代码用于声明 map 集合和成功标识 flag；第 18~24 行代码向 map 中添加数据；第 25~35 行是验证的方法；第 36~50 行用于获得页面传递的参数以及验证学生是否存在，如果存在，则存放在 session 中并重定向到 class.jsp；第 57~59 行代码用于验证是否成功，若失败则显示重新登录。在 logout.jsp 页面中，第 2~7 行代码用于获得 session 中的 name 参数值并移除该学生，使 session 失效。

运行结果如图 3.17 所示。

图 3.17　运行结果

该实例中的用户退出操作，只是利用简单的 session.invalidate 方法实现的，在开发中经常是结合 Struts 框架一起操作。

3.4　application 对象

application 对象实现的接口为 javax.servlet.ServletContext，它的生命周期是从 application 对象

创建到应用服务器关闭，也就是说当服务器关闭时 application 对象才消失。可以将它视为 Web 应用的全局变量，当服务器运行时有效，如果关闭服务器，其中保存的信息也就消失了。

3.4.1 application 对象的常用方法

application 对象的常用方法参见表 3.5。

表 3.5 application 对象的常用方法

方法	方法说明
getAttribute(String name)	获得存放在 application 中的含有关键字为 name 的对象
setAttribute(String name,Object obj)	将关键字 name 的指定对象 obj 放进 application 对象中
Enumeration getAttributeNames()	获取 application 中所有参数的名字，返回值是枚举类型
removeAttribute(String name)	移除 application 对象中 name 指定的参数值
getServletInfo()	获取 Servlet 的当前版本信息
getContext(String uripath)	获取 uripath 指定路径的 context 内容
getRealPath(String path)	获取指定文件的实际路径
getMimeType(String file)	获取指定的文件格式

3.4.2 获取指定页面的路径

【例 3.11】获取指定页面的实际路径、相对路径和当前应用程序路径。

application.jsp 用于指定页面输出其所在的实际路径和相对路径，其源代码如下：

```
----------------application.jsp----------------
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>My JSP 'application.jsp' starting page</title>
  </head>
  <body>
      <h3>指定页的实际路径、相对路径和当前应用程序路径</h3>
      <hr/>
      <table border="1" bordercolor="black">
          <tr>
              <td>当前服务器的名称和版本</td>
              <td><%=application.getServerInfo() %></td>
          </tr>
          <tr>
              <td>页面 application.jsp 的实际路径</td>
              <td><%=application.getRealPath("application.jsp") %></td>
          </tr>
          <tr>
              <td>页面 application.jsp 的 URL</td>
              <td><%=application.getResource("application.jsp") %></td>
```

```
22                  </tr>
23                  <tr>
24                      <td>当前 Web 程序的路径</td>
25                      <td><%=application.getContextPath() %></td>
26                  </tr>
27              </table>
28      </body>
29  </html>
```

在上述代码中，代码第 11~26 行分别输出 application 中指定页的实际路径、相对路径和当前应用程序路径等信息，页面效果如图 3.18 所示。

图 3.18　application.jsp 的运行结果

3.4.3　设计一个网站计数器

application 对象还可以用于保存访问网站的人数，也就是我们常说的网站计数器，下面通过一个例子来演示。

【例 3.12】网站计数器。

applicationCount.jsp 是网站计数器页面，其源代码如下：

```
----------------- applicationCount.jsp----------------
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <%
03      Integer count =(Integer) application.getAttribute("count");
04      if(count==null){
05       count=1;
06      }else{
07          count++;
08      }
09      application.setAttribute("count",count);
10   %>
11
12  <!DOCTYPE HTML>
13  <html>
14    <head>
15      <title>网站计数器</title>
```

```
16      </head>
17      <body>
18          欢迎访问本网站，您是第<%=count %>位访问客户！
19      </body>
20  </html>
```

第 2~10 行代码从页面中获得计数值，如果为空就设定初始值为 1，如果不为空就加 1。程序运行结果如图 3.19 所示。

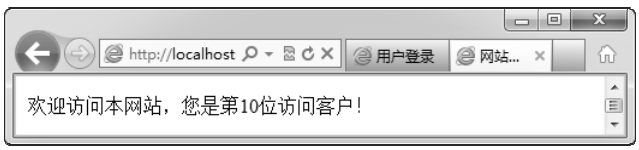

图 3.19 网站计数器

application 对象在 Web 应用运行时一直存在于服务器中，因此保存这种全局变量相对来说比较占用资源，因此不推荐使用。在实际开发中，一般都是让对象存在于必要的时间段中，否则当访问量加剧时，会造成内存不足等情况。

3.5 out 对象

out 对象是继承 javax.servlet.jsp.JspWriter 类的一个输出流对象。它包含很多 IO 流中的方法和特性，最常用的方法就是输出内容到 HTML 中。

3.5.1 out 对象的常用方法

out 对象的常用方法参见表 3.6。

表 3.6 out 对象的常用方法

方法	方法说明
append(char c)	将字符添加到输出流中
clear()	清空页面缓存中的内容
close()	关闭网页流的输出
flush()	网页流的刷新
println()	将内容直接打印在 HTML 标记中
write()	与 println()方法相似，区别在于 println()方法可以输出各种类型的数据，而 write 方法只能输出与字符相关的数据，例如字符、字符数组、字符串等

3.5.2 out 对象的使用示例

out 对象中的方法相对比较简单，而且也较少使用，下面就通过几个简单的例子来演示。

【例 3.13】演示 out 对象的 println 方法。

outprintln.jsp 是直接在 Java 脚本中描述 HTML 标记，其源代码如下：

```
---------- outprintln.jsp----------
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <!DOCTYPE HTML>
03  <html>
04    <head>
05      <title>演示 out 对象</title>
06    </head>
07
08    <body>
09      <div style="text-align: center;">
10          <hr>
11          <h4>以下就是一个表格</h4>
12          <%
13           out.println("<table border='2' align='center'>");
14           out.println("<tr>");
15           out.println("<td width='60'>"+"姓名"+"</td>");
16           out.println("<td width='40'>"+"性别"+"</td>");
17           out.println("<td width='80'>"+"出生年月"+"</td>");
18           out.println("<td width='60'>"+"城市"+"</td>");
19           out.println("</tr>");
20           out.println("<tr>");
21           out.println("<td width='60'>"+"Smith"+"</td>");
22           out.println("<td width='60'>"+"Male"+"</td>");
23           out.println("<td width='60'>"+"1984.8"+"</td>");
24           out.println("<td width='60'>"+"NewYork"+"</td>");
25           out.println("</tr>");
26           out.println("</table>");
27            %>
28      </div>
29    </body>
30  </html>
```

在上述代码中，第 13~26 行代码利用 out 对象输出 HTML 格式。页面效果如图 3.20 所示。

图 3.20　outprintln.jsp 页面的运行结果

注　意

现在在开发中很少使用这种方法输出 HTML 标记，因为比较烦琐而且容易出错。

【例 3.14】演示 out 对象的 clear 方法。

outclear.jsp 用于清空缓冲区中的内容，其源代码如下：

```
---------------- outclear.jsp----------------
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <!DOCTYPE HTML>
03  <html>
04    <head>
05      <title>演示 out 对象 clear 方法</title>
06    </head>
07
08    <body>
09      <h4>这是 HTML 中的内容</h4>
10      <%
11          out.print("<h4>这是 out 对象输出的信息</h4>");
12          out.clear();
13       %>
14       <h4 style="text-align: center;">这是 HTML 中的信息</h4>
15    </body>
16  </html>
```

在上述代码中，第 12 行代码用于清除缓冲区中的内容。页面效果如图 3.21 所示。

图 3.21 outclear.jsp 页面运行结果

从运行结果可以看出，若在 JSP 页面中调用 clear 方法，那么以前向客户输出流中写入的数据都将被清除。

注 意

在调用 clear 之前，不要调用 flush 方法，否则会抛出 IO 异常。

3.6 page 对象

page 对象的实质是 java.lang.Object 对象，它代表转译后的 Servlet。page 对象是指当前的 JSP 页面本身，在实际开发中并不常用。

3.6.1 page 对象的常用方法

page 对象的常用方法参见表 3.7。

表 3.7　page 对象的常用方法

方法	方法说明
getClass()	返回当时被转译的 Servlet 类
hashCode()	返回此时被转译的 Servlet 类的哈希代码
toString()	将此时被转译的 Servlet 类转换成字符串
equals(Object obj)	比较此时的对象是否与指定的对象相等
clone()	将此时的对象复制到指定的对象中
copy(Object obj)	对指定对象进行克隆

3.6.2　page 对象的使用示例

下面就通过简单的例子来演示 page 中的方法。

【例 3.15】演示输出 page 对象的 toString()方法和 hashCode()方法。

通过 page.jsp 页面演示 toString 和 hashCode 的用法，其源代码如下：

```
---------- page.jsp----------
01  <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
02  <!DOCTYPE HTML>
03  <html>
04    <head>
05      <title>演示 page 对象</title>
06    </head>
07
08    <body>
09      <%
10          int code = page.hashCode();//hashcode
11          String str = page.toString();
12          out.println("page 对象的 hash 码:"+code);
13          out.println("page 对象的值:"+str);
14        %>
15    </body>
16  </html>
```

在上述代码中，第 10~13 行代码用于获取 page 对象中的值，并输出在页面中，页面效果如图 3.22 所示。

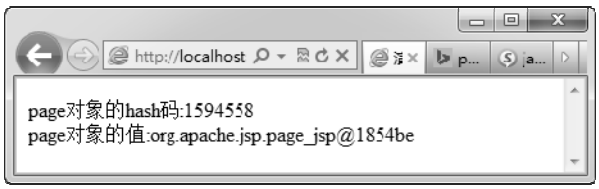

图 3.22　page.jsp 的运行结果

3.7 config 对象

config 对象实现了 javax.servlet.ServletConfig 接口，它一般用于在页面初始化时传递参数。

3.7.1 config 对象的常用方法

config 对象的常用方法参见表 3.8。

表 3.8 config 对象的常用方法

方法	方法说明
getInitParameter(String arg0)	获得指定的初始化值
getServletName()	获得 Servlet 名字
getServletContext()	获得 ServletContext 值
equals(Object obj)	比较此时的对象是否与指定的对象相等
getInitParameterNames()	获得初始化值的枚举值
toString()	获得此对象的值

3.7.2 config 对象的使用示例

下面就通过简单的实例来演示 config 中的方法。

【例 3.16】演示输出 config 对象的 getInitParameter()方法。

通过 config.jsp 页面演示 getInitParameter()方法的用法，在此假设 WEB-INF 文件夹下面存在 web.xml 文件，内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="4.0"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd">
    <welcome-file-list>
       <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
          <servlet-name>
                jspconfigdemo
           </servlet-name>
           <jsp-file>/config.jsp</jsp-file>
           <init-param>
                <param-name>url</param-name>
                <param-value>http://www.baidu.com</param-value>
           </init-param>
    </servlet>
```

```
    <servlet-mapping>
        <servlet-name>
            jspconfigdemo
        </servlet-name>
        <url-pattern>/config.jsp</url-pattern>
    </servlet-mapping>
</web-app>
```

在上述代码中，第 11~26 行代码用于在 web.xml 中配置 Servlet，包括其初始化参数等信息。config.jsp 页面用于显示配置内容，其源代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>演示 config 对象</title>
  </head>

  <body>
    <%
        String url = config.getInitParameter("url");
        String str = config.toString();
        out.config("config 对象的 initParameter 方法："+url+"</br>");
        out.config("config 对象的 toString 方法："+str);
     %>
  </body>
</html>
```

在上述代码中，第 10 行代码利用 config 对象获取初始化参数值的信息，页面效果如图 3.23 所示。

图 3.23　config.jsp 页面的运行结果

注 意

一般而言很少在页面中使用 config 对象，因为 JSP 页面的实质是 Servlet。

3.8　小结

本章主要介绍了 JSP 页面中的常用内置对象，在 Web 开发中要充分利用这些隐藏对象提供的方法来实现其 Web 开发中的强大功能。在开发过程中要充分理解各个对象的生命周期、作用范围，这样就可以方便操作页面中的属性和行为，实现页面与页面之间、页面与应用环境之间的通信等操

作。本书在以后的章节中，也会经常用到这些内置对象。

3.9 习题

1. JSP 的内置对象有几种作用域？分别是什么？

2. 表单向 JSP 提交数据的方式有哪些？

3. 请用 session 对象统计访问页面的客户数。

4. 请编写一个会员注册程序，要求用户在注册页面中填写信息后，提交请求，接收注册请求的页面可以显示出信息。