第 3 章

Web 应用开发基础

学习目标

- 理解开发环境与运行环境
- 掌握 Java Web 应用开发环境的搭建
- 理解 JSP 的运行机制和基础语法
- 掌握结构化网页开发的方法

3.1 开发环境和运行环境

在以往的程序设计学习中，读者可能使用过不同的开发工具，如学习 C 语言时使用的 VC++，学习 Java 语言时使用的 IDEA 等。一般的开发过程是在开发工具中创建源程序并编写代码，然后再使用开发工具所提供的编译功能将源程序编译为目标程序，最后使用开发工具的运行功能来运行目标程序，得到需要的结果。因为通常会使用集成开发工具，所开发的程序也大多运行在操作系统上，所以对开发程序和运行程序两种情况下的环境需求不做明确地区分。但对于 Java Web 应用程序来说，开发环境和运行环境是两个不同的概念，需要区分理解。

3.1.1 开发环境

开发环境是指程序员为了编写程序所需要的软件集合。利用这些软件，程序员可以使用某种程序设计语言编写应用程序，实现各种功能。

完整的程序开发包括编码、编译、调试运行等阶段。最简单的开发环境是在不同的开发阶段使用不同的工具软件。例如，先用文本编辑软件编写源程序，然后编译工具进行编译，最后用运行工具进行调试运行。

对于 Java 程序开发，可以用任意的文本编辑软件编写.java 源文件，然后使用 javac 把源文件编译成.class 字节码文件，最后使用 Java 运行字节码文件；也可以使用集成开发环境(Integrated Development Environment，IDE)来进行程序开发。IDE 中通常包含了代码编辑器、程序编译器、调试器以及图形界面工具，将代码编写功能、编译功能、调试功能、运行功能等集成为一体。常用的 Java 集成开发工具有 IntelliJ IDEA、Eclipse、JBuilder 以及微软公司的 Visual Studio 等。

集成开发工具可以有效地提高开发效率，简化开发过程，在实际项目开发中一般都会使用某种 IDE。但对于初学者来说，过早地使用集成开发工具会影响对程序底层运行机

制的理解，导致在程序出现运行错误时，需要分析原因却无从下手。因此本书没有使用集成开发工具，而是使用了最简单的开发工具。编写源代码使用文本编辑器软件，推荐使用EditPlus或UltraEdit，编译Java代码使用JDK。另外，在第10章也对IDEA的使用方法做了简单介绍。

3.1.2 运行环境

运行环境是指能够使一个程序顺利运行的所有外部条件之和。运行环境包括符合基本需要的硬件配置、特定版本的操作系统以及其他系统软件及运行库。Java Web应用程序的运行环境涉及三个方面的需要。首先要能够运行Java代码，由于Java语言的跨平台特性，所以对硬件和操作系统都没有特定的要求，但需要提供JRE(Java Runtime Environment，Java运行环境)。其次要能够提供Web服务，需要有Web服务器和Web浏览器。最后要能够提供应用程序功能，需要有运行相关程序的基础平台，这个平台称为JSP引擎。另外，如果程序需要数据库的支持，运行环境还需要包括一种数据库管理系统。

本书使用的运行环境由JDK 8内置的JRE、Tomcat 9、通用Web浏览器和MySQL 8构成。其中，JRE用来运行Java代码，Tomcat提供Web服务和JSP引擎服务，MySQL提供数据库功能。

3.1.3 安装和配置JDK

1. JDK简介

1996年，SUN公司发布了Java的第一个开发工具包：JDK(Java Development Kit) 1.0。JDK是整个Java的核心，包括Java运行环境、Java工具和Java基础类库。

1998年，SUN公司发布了第二代Java平台JDK 1.2(简称Java 2)，包括三个版本：J2ME(Java 2 Micro Edition，Java 2平台的微型版)，应用于移动、无线及有限资源的环境；J2SE(Java 2 Standard Edition，Java 2平台的标准版)，应用于桌面环境；J2EE(Java 2 Enterprise Edition，Java 2平台的企业版)，应用于基于Java的应用服务器。

2004年，JDK 1.5的发布是Java语言发展史上的里程碑事件。为了表示这个版本的重要性，JDK 1.5更名为JDK 5.0(内部版本号为1.5.0)。

2006年，SUN公司发布了JDK 6.0。这次，Java的各种版本已经更名，取消了其中的数字2，J2ME更名为Java ME(微型版)，J2SE更名为Java SE(标准版)，J2EE更名为Java EE(企业版)。

因为JDK从6.0版本开始改名为Java SE，所以我们通常使用的J2SE也就是标准版的JDK，与大家经常看到的Java SE 6.0以及JDK1.6其实都是一回事。

2009年，Oracle(甲骨文)公司收购SUN公司。从此以后，JDK的获取就变成了从Oracle的网站上免费下载，SUN公司就此退出了历史舞台。

Java目前的最新版本已经更新到JDK 17。但2014年发布的JDK 8仍然是使用最为广泛的版本之一，本书也以此版本为主。

2. 下载 JDK

以下载 JDK 8 的最新版本 jdk-8u311-windows-x64.exe 为例，打开浏览器，在地址栏输入 https://www.oracle.com/java/technologies/downloads/#java8 后，在打开的页面中，列出了不同版本的 JDK。

单击“Windows”选项卡，选择列表中的 jdk-8u311-windows-x64.exe，进入用户登录页面，输入用户名和密码进行登录后，开始下载。如果没有用户名，则需要创建一个新用户，创建新用户的过程在此省略。

JDK 支持不同的操作系统及其位数，如 Linux、Mac OS、Solaris 和 Windows，版本列表中的 i586 指 32 位操作系统，x64 指 64 位操作系统，32 位操作系统的 JDK 版本可以在 64 位操作系统上使用，反之则不行。所以 jdk-8u311-windows-x64.exe 实际是 Windows 64 位操作系统的 JDK 安装包，不能在 Windows 32 位操作系统上使用。在实际开发中使用的 JDK 版本必须与操作系统完全匹配才能将 Java 的性能发挥到最佳，在学习时则无须拘泥于此。

3. 安装 JDK

双击下载的 jdk-8u311-windows-x64.exe 文件，在打开的安装界面中单击“下一步”按钮，进入定制安装界面，如图 3.1 所示。

图 3.1　JDK 定制安装

在定制安装界面中可以选择要安装的程序功能，其中“公共 JRE”为独立的 Java 运行环境，由于 JDK 中已经包含了 JRE，因此可以不安装此功能（单击左侧按钮，在弹出的下拉列表中选择×）。安装路径采用默认的“C:\Program Files\Java\jdk1.8.0_311\”。

单击“下一步”按钮，开始安装所选功能，安装完毕后单击“关闭”按钮退出安装程序。此时在安装路径“C:\Program Files\Java\jdk1.8.0_311\”下可以看到如图 3.2 所示的目录结构。

- bin 目录包含了编译、调试、打包、运行 Java 程序的工具程序。例如 javac.exe 为编译 Java 源代码的工具，java.exe 为运行编译得到的字节码文件的工具。

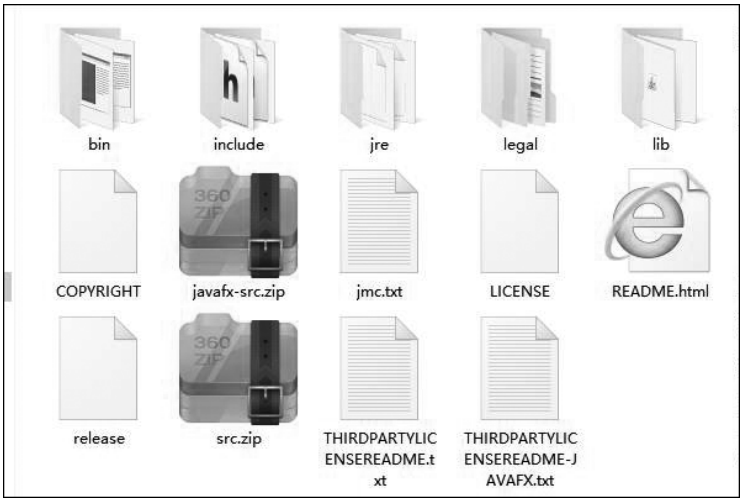

图 3.2　安装完成后的 JDK 目录结构

- jre 目录为前面提到的 Java 运行环境。
- lib 目录为 Java 类编译打包后的基础类库。bin 目录下 java.exe 等工具的运行依赖于 lib 目录下的基础类库。
- src.zip 文件为 Java 类的源代码。

4. 配置 JDK

以 Windows 10 系统为例，右击桌面上的“此电脑”图标，选择“属性”，在打开的控制面板主页左侧选择“高级系统设置”，弹出“系统属性”对话框。然后在“高级”选项卡中单击“环境变量”按钮，弹出“环境变量”对话框。在该对话框上方的“用户变量”列表中选择“Path”行，然后单击“编辑”按钮，弹出“编辑环境变量”对话框。单击“新建”按钮，在文本框中输入“C:\Program Files\Java\jdk1.8.0_311\bin”后，依次单击“确定”按钮，返回到控制面板主页，最后关闭该窗口。

把 JDK 的 bin 路径添加到 Path 环境变量中后，就可以在执行 Java 的工具程序时自动在指定的路径下进行搜索并执行，无须每次在命令行输入 Java 的工具程序全路径，只输入工具名即可。

5. 测试 JDK

单击任务栏左下角的图标，在搜索框中输入“cmd”后按回车键打开命令行窗口。在命令行输入“java -version”后按回车键，显示如图 3.3 所示的 JDK 版本信息，即表示 JDK 安装配置成功。

3.1.4　安装和配置 Tomcat

1. Tomcat 简介

Tomcat 是一个开源的轻量级 Web 应用服务器，是 Apache 软件基金会（Apache Software Foundation）的 Jakarta 项目中的一个核心项目，由 Apache、SUN 和其他一些公

图 3.3　JDK 版本信息

司及个人共同开发而成。因为 Tomcat 技术先进、性能稳定，而且免费，因而深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，成为了目前比较流行的 Web 应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是学习 Java Web 开发的首选。Tomcat 官方网络如图 3.4 所示。

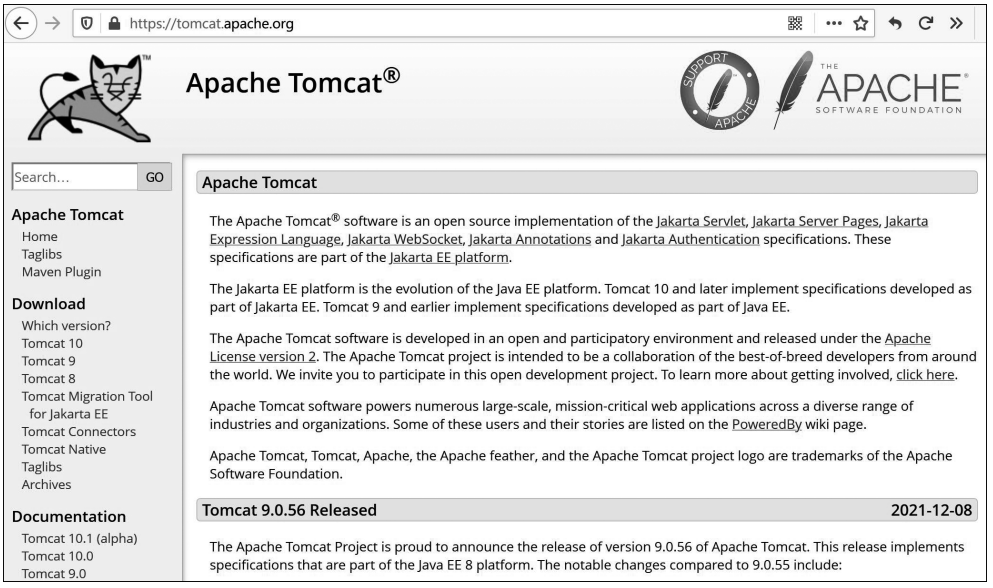

图 3.4　Tomcat 官方网站

2. 下载 Tomcat

Tomcat 下载包分为安装版和绿色版两种。安装版和常规的软件安装包类似，在安装向导的提示下进行操作，并会以服务的形式注册到系统，可以很方便地控制 Tomcat 的启动、关闭和重启。绿色版则是在解压后需要通过人工方式在配置文件中进行参数的配置，启动和停止 Tomcat 都需要通过执行脚本来实现。本书使用绿色版，读者有兴趣也可以下载安装版进行尝试。本书使用的版本为 tomcat-9.0.54，其下载地址为 https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.54/bin/apache-tomcat-9.0.54-windows-x64.zip。

3. 配置 Tomcat

下载完成后，将其解压缩到某个路径，如 C 盘根目录。进入解压后的目录“C:\apache-tomcat-9.0.54”，可以看到如图 3.5 所示的目录结构。

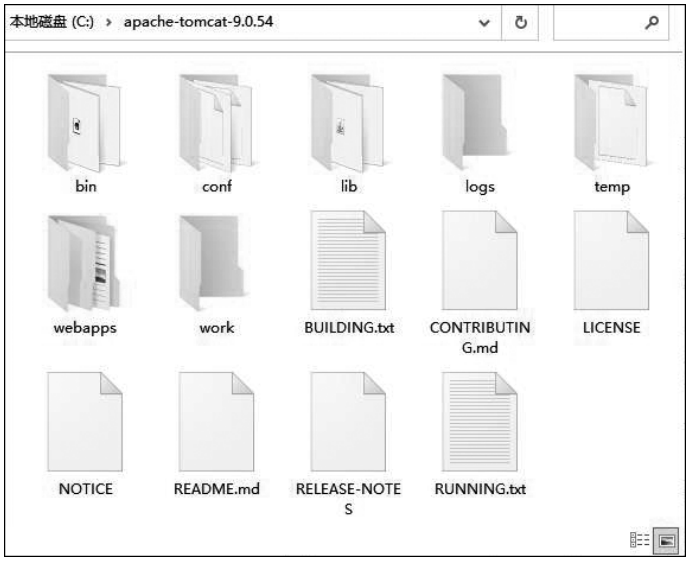

图 3.5　Tomcat 目录结构

- bin 目录包含了启动、停止等操作 Tomcat 的程序或者脚本。由于本书采用的是绿色版，因此在该文件夹下有后缀为.sh 和.bat 的 Tomcat 控制程序，其中.sh 为 Linux 系统下运行的脚本，.bat 则为 Windows 系统下运行的脚本。安装版的控制程序后缀为.exe。
- conf 目录为 Tomcat 的配置文件夹，最重要的配置文件是 server.xml，此文件中配置了 Tomcat 的各个端口、应用部署的路径等。
- logs 目录为 Tomcat 运行的日志，在开发和运维时经常需要通过查看此文件夹下的日志来定位系统出现的问题。
- webapps 目录为 Tomcat 默认部署应用的路径。本书中所有的示例应用都需要部署到此目录中才能运行。

Tomcat 运行需要 JDK 的支持，因此需要为 Tomcat 配置 JDK 的安装路径。右击 bin 目录下的 startup.bat，在弹出的快捷菜单中选择“编辑”后，打开该文件，在第一行添加以下内容即可。

```
SET JAVA_HOME=C:\Program Files\Java\jdk1.8.0_311
```

4. 测试 Tomcat

双击 bin 目录下的 startup.bat，启动 Tomcat。当出现如图 3.6 所示的提示信息时，表示启动成功。

如果界面显示汉字乱码，可以修改 conf 目录下的 logging.properties 文件，添加以下代码，将字符集设置为 GBK，重新启动即可。

```
java.util.logging.ConsoleHandler.encoding = GBK
```

```
01-Oct-2021 10:51:07.221 信息 [main] org.apache.catalina.startup.Catalina.start
[5205]毫秒后服务器启动
```

图 3.6　Tomcat 启动成功提示信息

打开浏览器，在地址栏输入 http://localhost:8080，出现 Tomcat 信息显示页面，如图 3.7 所示，说明安装成功。

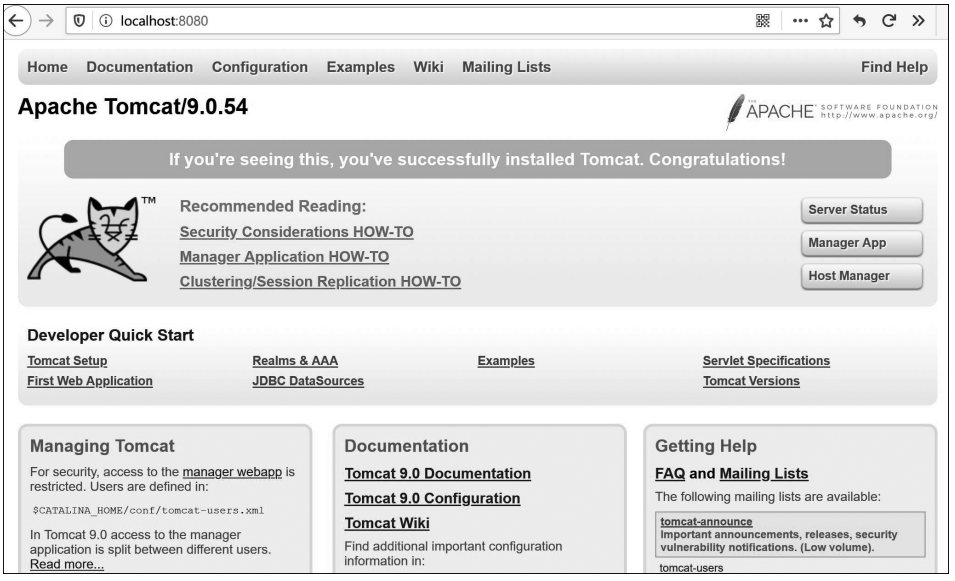

图 3.7　Tomcat 信息显示页面

3.2　JSP 基本概念

Java Web 应用开发的基本形式是编写 JSP 页面。JSP(Java Server Pages，Java 服务器页面)是一种创建和管理动态网页的技术标准。通过在传统的网页 HTML 文件中插入 Java 脚本代码和 JSP 标记，就可以得到 JSP 页面文件。JSP 程序本质上是在服务器端执行的 Java Servlet，执行完毕通常会返回给客户端一个 HTML 文件作为响应，客户端浏览器即可查看该响应文件的内容。

3.2.1　JSP 开发方法

这里用一个简单的示例来说明 JSP 的开发方法。

【程序 3.1】 在“Tomcat 安装目录\webapps”下创建 demo03 文件夹，在其中编写示例页面文件 serverDemo.jsp。

```
<%@page contentType="text/html;charset=utf-8" %>
<%@page import="java.util.Date"%>
```

```
<%@page import="java.text.SimpleDateFormat"%>
<html>
    <head>
        <title>serverDemo</title>
    </head>
    <body>
        <h3>欢迎光临中国北京</h3>
        <%
            SimpleDateFormat df = new SimpleDateFormat("yyyy-M-d HH:mm:
            ss");
        %>
        <span id="myspan"><%=df.format(new Date())%></span>
        <hr>
    </body>
</html>
```

serverDemo.jsp 文件的功能是输出一行文字和当前的系统时间。

【程序 3.2】 在 demo03 文件夹中编写示例网页文件 clientDemo.html。

```
<!DOCTYPE html>
<html>
    <head>
<meta charset="utf-8">
     <title>JavaScript Date 对象</title>
     <script language="javaScript">
         function showTime(){
             var Timer=new Date();
             var h=Timer.getHours();
             var m=Timer.getMinutes();
             var s=Timer.getSeconds();
             var d=Timer.getDate();
             var m1=Timer.getMonth()+1;
             var y=Timer.getFullYear();
             var strShow=""+y+"-"+m1+"-"+d+" "+h+":"+m+":"+s;
             myspan.innerText=strShow;
         }
     </script>
    </head>
    <body>
         <h3>欢迎光临中国北京</h3>
         <span id="myspan">时间内容</span>
```

```
            <script>
                showTime();
            </script>
            <hr>
        </body>
    </html>
```

clientDemo.html 文件的功能也是输出一行文字和当前的系统时间。

启动 Tomcat，打开浏览器，在地址栏输入 http://localhost:8080/demo03/serverDemo.jsp，可以看到程序 3.1 的运行效果，如图 3.8 所示，而输入 http://localhost:8080/demo03/clientDemo.html，可以看到程序 3.2 的运行效果，如图 3.9 所示。

图 3.8　serverDemo.jsp 页面的运行效果

图 3.9　clientDemo.html 页面的运行效果

上面两个程序的运行效果几乎是完全一样的。但它们的源代码完全不同。下面通过介绍 JSP 的运行机制和 JSP 页面构成来分析这两个页面的运行过程。

3.2.2　JSP 运行机制

JSP 的运行机制如图 3.10 所示。当浏览器向服务器发出请求来访问一个 JSP 页面时，所请求的 JSP 文件会被服务器端的 JSP 引擎转换为一个 Java 类，并被编译成一个字节码文件，再装载到 Java 虚拟机中运行，最后把运行产生的输出作为对本次请求的响应返回给浏览器。

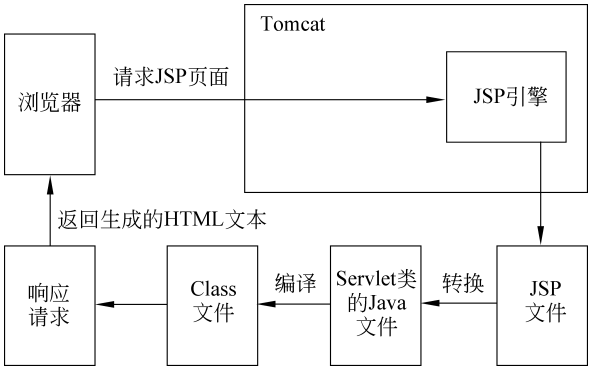

图 3.10　JSP 页面执行的处理过程

实际上，服务器并非在每次收到 JSP 请求后都严格按照上述过程进行处理，而是会

先检查是否为针对该 JSP 文件创建后的第一次请求，或者所请求的 JSP 文件在上次被请求之后是否被更新过。如果是第一次请求，或者被更新过，就会按照上述的处理过程来运行。如果该 JSP 文件之前已经被请求过，而且之后没有更新过，就会直接找到该文件对应的字节码文件来执行。因此，JSP 页面在第一次被请求时响应速度会比较慢，之后再次访问就会快很多。

可以在“Tomcat 安装目录\work\Catalina\localhost\demo03\org\apache\jsp”下找到程序 3.1 被转译成的 Java 类文件 serverDemo_jsp.java，代码如下：

```
package org.apache.jsp;
import javax.servlet.* ;
import javax.servlet.http.* ;
import javax.servlet.jsp.* ;
import java.util.Date;
import java.text.SimpleDateFormat;

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

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

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

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

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

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

    public void _jspDestroy() {
    }
```

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

final javax.servlet.jsp.PageContext pageContext;
javax.servlet.http.HttpSession session =null;
final javax.servlet.ServletContext application;
final javax.servlet.ServletConfig config;
javax.servlet.jsp.JspWriter out =null;
final java.lang.Object page =this;
javax.servlet.jsp.JspWriter _jspx_out =null;
javax.servlet.jsp.PageContext _jspx_page_context =null;

try {
  response.setContentType("text/html;charset=utf-8");
  pageContext =_jspxFactory.getPageContext(this, request, response,
              null, true, 8192, true);
  _jspx_page_context =pageContext;
  application =pageContext.getServletContext();
  config =pageContext.getServletConfig();
  session =pageContext.getSession();
  out =pageContext.getOut();
  _jspx_out =out;

  out.write("\r\n");
  out.write("\r\n");
  out.write("\r\n");
  out.write("<html>\r\n");
  out.write("  <head>\r\n");
  out.write("    <title>serverDemo</title>\r\n");
  out.write("  </head>\r\n");
  out.write("  <body>\r\n");
  out.write("    <h3>欢迎光临中国北京</h3>\r\n");
  out.write("\t ");

  SimpleDateFormat df =new SimpleDateFormat("yyyy-M-d HH:mm:ss");

  out.write("\r\n");
  out.write("\t <span id=\"myspan\">");
  out.print(df.format(new Date()));
  out.write("</span>\r\n");
  out.write("  <hr>\r\n");
  out.write("  </body>\r\n");
```

```
            out.write("</html>");
          } catch (java.lang.Throwable t) {
            if (!(t instanceof javax.servlet.jsp.SkipPageException)){
              out = _jspx_out;
              if (out !=null && out.getBufferSize() !=0)
                try { out.clearBuffer(); } catch (java.io.IOException e) {}
              if (_jspx_page_context !=null)
                  _jspx_page_context.handlePageException(t);
            }
          } finally {
                _jspxFactory.releasePageContext(_jspx_page_context);
          }
        }
      }
```

serverDemo_jsp.java 是一个完整的 Java 类。其中只有下面这段代码是来自 serverDemo.jsp，其余的语句都是 Tomcat 在转译过程中自动生成的。

```
            out.write("\r\n");
            out.write("\r\n");
            out.write("\r\n");
            out.write("<html>\r\n");
            out.write("  <head>\r\n");
            out.write("    <title>JSP Date 对象</title>\r\n");
            out.write("\r\n");
            out.write("  </head>\r\n");
            out.write("  <body>\r\n");
            out.write("    <h1>欢迎光临中国北京</h1>\r\n");
            out.write("  \r\n");
            out.write("\t ");

            SimpleDateFormat df =new SimpleDateFormat("yyyy-M-d HH:mm:ss");

            out.write("\r\n");
            out.write("\t <span id=\"myspan\">");
            out.print(df.format(new Date()));
            out.write("</span>\r\n");
            out.write("<hr>\r\n");
            out.write("  </body>\r\n");
            out.write("</html>");
```

正是通过这种运行机制，原本十分晦涩的 Java Web 服务器端开发变成了简单的 JSP 开发。

在图 3.8 所示的运行效果页面中使用浏览器的“查看页面源代码”功能，可以看到浏览器所收到的响应信息是如下内容。

```
<html>
    <head>
        <title>serverDemo</title>
    </head>
    <body>
        <h3>欢迎光临中国北京</h3>

        <span id="myspan">2017-5-7 22:18:04</span>
        <hr>
    </body>
</html>
```

不难看出，上面这段代码正是 serverDemo_jsp 类的运行产生的输出。其中的大部分内容与程序 3.1 完全相同。

JSP 页面中可以包含 JSP 脚本、JSP 指令、JSP 标记、HTML 标记、JavaScript 语句以及 CSS 样式定义等内容。这些内容可以在一个 JSP 文件中组合使用。

根据这些代码的运行位置，可以把 JSP 页面的内容分为服务器端代码和客户端代码。服务器端代码是指运行在服务器端的 JSP 脚本、JSP 指令和 JSP 标记等内容。客户端代码是指需要在客户端的浏览器上运行和处理的 HTML 标记、JavaScript 语句和 CSS 样式定义等内容。

JSP 文件存储在服务器端。当浏览器向服务器发出 JSP 请求后，从上面分析的处理过程可以看到，JSP 页面中的客户端代码在服务器端不做任何处理，服务器端代码则是在服务器环境下被运行之后，将其所产生的输出与客户端代码组合成一个整体返回给浏览器，最后由浏览器对接收到的内容进行解析和显示。

由此可知，在程序 3.1 的运行效果图 3.8 中所显示的时间是服务器端的当前系统时间。浏览器只负责把接收到的时间字符串显示在页面中。

而程序 3.2 是一个 HTML 文件。Tomcat 对这个文件的处理不像 JSP 文件那么复杂，只是将文件内容原样返回给浏览器，然后由浏览器对文件内容进行解析和显示。所以，在运行效果图 3.9 中所显示的时间是客户端的当前系统时间。

3.2.3 Web 应用目录结构

虽然，Java Web 应用开发的基本形式是编写 JSP 页面，但一个典型的 Java Web 应用通常会包含一组 JSP 文件、Servlet、其他 Java 类以及 HTML 文档和各种资源文件。这些文件在 Web 应用中都有固定的存入目录。

按照 Java EE 规范的规定，一个典型的 Java Web 应用包含如下 4 个部分。

- 公开文件夹，存放能够被用户访问的资源，包括.jsp、.htm、.js、.css、.jpg 等文件。
- WEB-INF/web.xml 文件，为应用的部署描述文件。

- WEB-INF/classes 文件夹，存放编译好的 Java 类文件(.class)。
- WEB-INF/lib 文件夹，存放 Java 类库文件(.jar)。

公共文件夹中存放所有可以被用户访问的资源文件。也可以把这些文件根据类别放在公共文件夹下的不同子文件夹中。这个公开文件夹的名字就是 Web 应用程序的名称。

WEB-INF 文件夹是一个专用区域，其中的文件用户不能直接访问，只能被 Web 应用本身和 JSP 引擎使用。

运行 Java Web 应用需要先把它部署到运行环境中。最简单的部署方式是把应用文件夹存放到 Tomcat 安装目录下的 webapps 中。例如，对于本书中使用的示例应用“新闻发布系统”，在 webapps 目录下创建 newsPub 文件夹，相应的目录结构如图 3.11 所示。

图 3.11　newsPub 应用的目录结构

每个文件夹中放置不同类型的代码文件，具体内容如下。

- newsPub 文件夹：是 Java Web 应用的公开文件夹，所有的 JSP 文件和 HTML 文件都直接放在这个文件夹下。
- common 文件夹：存放应用中的公共文件，比如通用的 JSP 文件。此文件夹的名字不一定必须是 common，可以根据需要自行设定。
- css 文件夹：存入页面中使用的 CSS 样式定义文件。
- image 文件夹：存放页面文件中使用的图片文件。
- js 文件夹：存放页面中使用的 JavaScript 代码文件。
- WEB-INF 文件夹：是 Java Web 应用的安全目录。
- classes 文件夹：用于存放 Java Web 应用中用到的 Java 类字节码文件。
- lib 文件夹：用于存放 Java Web 应用中用到的外部 jar 文件。
- src 文件夹：用于存放 Java Web 应用中用到的 Java 类源文件。这个文件夹及其中的 Java 类源文件对于 Java Web 应用的运行不是必需的，只是为了代码管理的方便，习惯把源文件放置在此处。

把第 2 章编写的 index.html 放到 newsPub 文件夹中，main.css 放到 css 文件夹中，所用到的图片文件放到 image 文件夹中，fun.js 放到 js 文件夹中。然后启动 Tomcat，打开浏览器，在地址栏输入 http://localhost:8080/newsPub/index.html，即可打开新闻发布系统首页。

这里直接在运行环境中创建 newsPub 应用的目录结构，接下来将会在 newsPub 文件夹中依次完成各个功能的开发，逐步完善新闻发布系统。

3.3　JSP 基础语法

JSP 页面中的服务器端代码包括 JSP 脚本、JSP 指令、JSP 标记等构成元素。这些代码需要符合 JSP 语法的要求。

3.3.1 JSP 脚本元素

所有的 JSP 脚本元素都以“<%”标记开始，以“%>”标记结束。JSP 脚本元素包括 3 类，分别是脚本代码、声明和表达式。JSP 脚本代码就是一些 Java 代码片段；JSP 声明包括变量、方法和类的声明，分别用于定义变量、方法和类；JSP 表达式用于输出计算结果。另外，还可以在 JSP 页面文件中添加不同形式的注释。

1. JSP 脚本代码

JSP 脚本代码就是一些 Java 代码片段，可以实现业务逻辑处理，也可以产生输出。其语法格式如下：

```
<%  脚本代码  %>
```

一个 JSP 页面可以有多个脚本代码，这些脚本代码将被 JSP 引擎按顺序执行。

例如在程序 3.1 中，就使用了如下代码实现获取系统当前时间的功能。

```
<%
    SimpleDateFormat df =new SimpleDateFormat("yyyy-M-d HH:mm:ss");
%>
```

脚本代码中声明的变量在当前页面内的所有脚本代码和 JSP 表达式中有效，这样的变量被称为 JSP 页面的局部变量。因为这里的变量定义语句在发生 JSP 转译时，都被转换成了_service()方法中的语句，所以这些变量实际也相应地转变成了方法中的变量。这种变量的生存周期仅限于方法的执行过程中。当多个用户请求同一个 JSP 页面时，一个用户对 JSP 页面局部变量的操作，不会影响到其他用户的这个局部变量。

2. JSP 声明

前面已经说过，JSP 页面在运行时会被首先转译为一个 Java 类。在 JSP 声明中所定义的变量和方法都会成为转译后的 Java 类的成员变量及类成员方法，声明的类则成为内部类。所声明的变量、方法和类可以被同一 JSP 页面中的其他代码访问。

JSP 声明的语法格式如下：

```
<%! 变量或方法、类的声明%>
```

JSP 声明中定义的变量也称为 JSP 页面的全局变量，所有访问同一个 JSP 页面的客户操作的都是同一个全局变量。

【程序 3.3】 在 demo03 文件夹下编写示例页面文件 declareDemo1.jsp。

```
<%@page contentType="text/html;charset=utf-8" %>
<html>
    <head>
```

```
        <title>declareDemo1</title>
    </head>
    <body>
        <H3>
        <%! int number=0; %>
        <%
            int localNumber =0;
            localNumber++;
            synchronized(this){
                number++;
            }
        %>
        您是第
        <%out.println(number); %>
        个访问本页面的客户。局部变量值是
        <%out.println(localNumber);%>
        </H3>
    </body>
</html>
```

declareDemo1.jsp 中的声明部分定义了一个变量 number。这个 number 是 JSP 页面的全局变量。接下来又在脚本代码中定义了一个局部变量 localNumber。

然后分别对两个变量进行加 1 运算，再进行输出。其中对于全局变量 number 的操作，需要进行同步处理。

启动 Tomcat，打开浏览器，在地址栏输入 http://localhost:8080/demo03/declareDemo1.jsp，查看程序 3.3 的运行结果，如图 3.12 所示。在同一个浏览器窗口和不同的浏览器窗口多次访问该页面，可以看到 number 变量的值会依次递增，而 localNumber 变量的值一直保持为 1。这是因为，局部变量只在处理当前页面请求时有效，处理完毕则被销毁，而全局变量作为 JSP 页面对应类的成员变量，会一直存在于内存中。只有当前应用被停止运行，例如关闭 Tomcat 时，全局变量才会被销毁。

图 3.12　declareDemo1.jsp 的运行结果

【程序 3.4】 在 demo03 文件夹下编写示例页面文件 declareDemo2.jsp。

```
<%@page contentType="text/html;charset=utf-8" %>
<html>
    <head>
```

```
        <title>declareDemo2</title>
    </head>
    <body>
        <H3>
        <%! int number=0;
            synchronized void countPeople(){
                number++;
            }
        %>
        <%
            int localNumber =0;
            localNumber++;
            countPeople();
        %>
        您是第
        <%out.println(number); %>
        个访问本页面的客户。局部变量值是
        <%out.println(localNumber);%>
        </H3>
    </body>
</html>
```

declareDemo2.jsp 中的声明部分除了定义全局变量 number 之外，还定义了 countPeople()方法。在 countPeople()方法中实现对 number 的加 1 处理，然后在脚本代码中调用此方法，最后输出结果。

启动 Tomcat，打开浏览器，在地址栏输入 http://localhost：8080/demo03/declareDemo2.jsp，查看程序 3.4 的运行结果，可以看到与程序 3.3 完全相同。

【程序 3.5】 在 demo03 文件下编写示例页面文件 declareDemo3.jsp。

```
<%@page contentType="text/html;charset=utf-8" %>
<html>
    <head>
        <title>declareDemo3</title>
    </head>
    <body>
        <H3>
         <%!
            class Counter {
                int number=0;
                void countPeople(){
                    number++;
                }
```

```
            }
            Counter c1 =new Counter();
        %>
        <%
            Counter c2 =new Counter();
            c1.countPeople();
            c2.countPeople();
        %>
        您是第
        <%out.println(c1.number); %>
        个访问本页面的客户。局部变量值是
        <%out.println(c2.number);%>
        </H3>
    </body>
</html>
```

declareDemo3.jsp 的声明部分定义了一个 Counter 类，该类含有成员变量 number 和 countPeople()方法，并定义了一个全局变量 c1。然后在脚本代码中定义了局部变量 c2，并分别调用 c1 和 c2 的 countPeople()方法。最后输出结果。

启动 Tomcat，打开浏览器，在地址栏输入 http://localhost：8080/demo03/declareDemo3.jsp，查看程序 3.5 的运行结果，也可以看到与程序 3.3 完全相同。

3. JSP 表达式

可以将 JSP 表达式理解为一种简单的输出形式。JSP 表达式的语法格式如下：

```
<%=表达式%>
```

其中的表达式可以是任意合法的 Java 表达式。该表达式会被计算并将得到的结果以字符串的形式显示到页面中。需要注意的是，由于 JSP 表达式不是程序代码，所以末尾不能出现分号“;”。

【程序 3.6】 在 demo03 文件夹下编写示例页面文件 expressionDemo.jsp，输出九九乘法表。

```
<%@page language="java" contentType="text/html;charset=utf-8" %>
<html>
    <head>
        <title>expressionDemo</title>
    </head>
    <body>
    <h1>九九乘法表</h1>
    <table>
```

```
            <%for (int i=1;i<=9;i++){%>
                <tr>
                <%for (int j=1;j<=i;j++){%>
                    <td><%=i%>* <%=j%>=<%=i* j%></td>
                <%}%>
                </tr>
            <%}%>
            </table>
        </body>
    </html>
```

expressionDemo.jsp 中使用多段脚本代码实现了循环处理，并用 JSP 表达式进行输出。

启动 Tomcat，打开浏览器，在地址栏输入 http://localhost:8080/demo03/expressionDemo.jsp，程序运行结果如图 3.13 所示。

图 3.13　expressionDemo.jsp 运行结果

4. 注释

除了上面所说的 3 种组成内容之外，JSP 页面中也可以编写必要的注释内容。JSP 注释的语法如下：

```
<%--JSP 注释信息--%>
```

这些注释信息在 JSP 转译为 Java 类时会被忽略。

另外，在 JSP 脚本代码中，也可以使用 Java 的单行注释和多行注释方式添加注释内容。这些 Java 注释与其他脚本代码一样，被转换到转译之后的 Java 类中，然后在编译 Java 类时被忽略。

在 JSP 页面中还可以使用 HTML 的注释语法编写注释内容，语法如下：

```
<!--HTML 注释信息-->
```

需要注意的是，HTML 注释与普通的 HTML 标记一样，会作为客户端代码被原样发送给浏览器。虽然在浏览器窗口中不会显示这些注释内容，但是利用浏览器的“查看页面源代码”功能就可以看到这些 HTML 注释。所以，如果不希望被用户看到注释内容，就需要使用前面两种注释方式。

为了提高程序的可读性，程序员应该合理地使用注释，将代码所实现的算法、功能等通过注释描述清晰，以提高代码的可读性和可维护性。

【程序 3.7】 在 demo03 文件夹下编写示例页面文件 commentDemo.jsp，使用不同的注释方式。

```
<%@page language="java" contentType="text/html;charset=utf-8" %>
<html>
    <head>
        <title>JSPComment</title>
    </head>
    <body>
    <%--JSP comment --%>
    <h1>hello world</h1>
    <%
        //Java comment1;
        out.println("<h2>hello world</h2>");
        /*Java comment2*/
    %>
    <!--html comment-->
    </body>
</html>
```

commentDemo.jsp 中分别使用了 JSP 注释、Java 注释和 HTML 注释。

启动 Tomcat，打开浏览器，在地址栏输入 http://localhost:8080/demo03/commentDemo.jsp，程序运行结果如图 3.14 所示。

可以通过查看和对比转译后的 Java 类、浏览器接收到的源代码以及浏览器窗口中的显示，来理解 3 种注释的处理时机。

图 3.14　commentDemo.jsp 运行结果

3.3.2　JSP 指令元素

JSP 指令主要用来通知 JSP 引擎如何处理 JSP 页面，所有指令都是在 JSP 的整个页面有效。JSP 指令是在 JSP 文件转译时处理，用于