# 第 1 章　JSP 概述

JSP 是一种应用非常广泛的动态网站开发技术，本章将在介绍 Web 应用开发模式的基础上，重点介绍 JSP 的特点及其与其他动态 Web 开发技术的区别，并以 JDK＋Tomcat 的组合为例详细说明 JSP 开发环境的构建方法和 JSP 程序的简单调试过程。

## 1.1　开 发 模 式

技术的进步和网络环境的发展，使得 Web 应用程序开发技术也在不断地进步。在 Web 应用程序的开发中，存在着两种开发模式：一种是传统的 C/S 模式，另一种是近年来兴起的 B/S 模式。

### 1.1.1　C/S 模式

在传统的 Web 应用程序开发过程中，需要同时开发客户端和服务器端的程序，由服务器端的程序提供基本的服务，客户端是提供给用户的访问接口，用户可以通过客户端的软件获得服务器提供的服务，这种 Web 应用程序的开发模式就是传统的 C/S 模式，即客户端/服务器的开发形式，其示意图如图 1-1 所示。在这种模式中，由服务器端和客户端共同配合来完成复杂的业务逻辑。早期开发的网络软件一般都采用这种模式，现在的网络游戏中，一般也采用这种 Web 开发模式。在这些 Web 应用程序中，都需要用户安装客户端软件才可以使用。

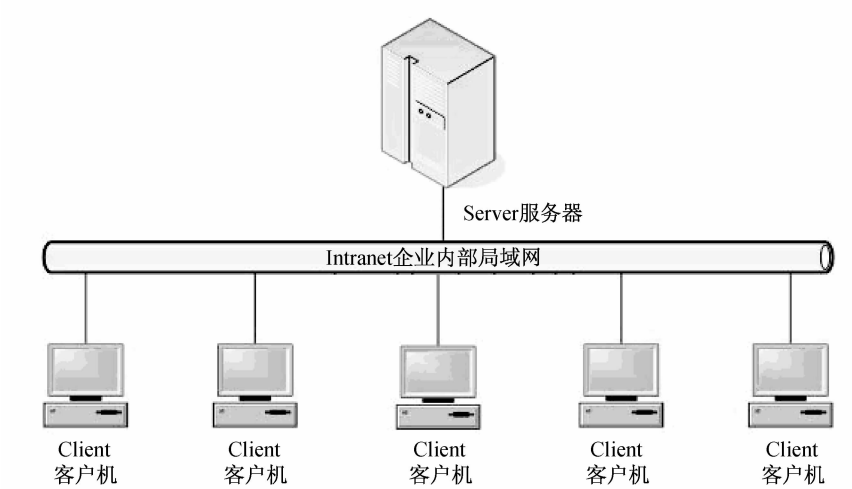


图 1-1　C/S 模式示意图

在 C/S 模式中，多个客户机围绕着一个或者多个服务器，这些客户机上安装有客户端软件，负责客户端业务逻辑的处理。在服务器端仅仅对重要的过程和数据库进行处理和存储，每个客户端都分担服务器的压力，这些客户端可以根据不同用户的需求进行定制。C/S 模式的出现大大提高了 Web 应用程序的效率，给软件开发带来革命性的飞跃。

但是，随着时间的推移，C/S 模式的弊端开始慢慢显现：系统部署的时候需要在每个客户机上安装客户端软件，工作量大；软件的升级很麻烦，哪怕是很小的一点改动，都要把所有客户端软件全部修改更新。因此，C/S 模式流行了一段时间以后，逐渐被另一种 Web 应用系统的开发模式所代替，这种新的模式就是 B/S 模式。

### 1.1.2 B/S 模式

B/S(浏览器/服务器)模式采取了基于浏览器的策略，是目前 Web 应用程序开发中比较常用的一种开发模式。在这种开发模式中，软件开发人员只需专注于开发服务器端的程序，不需要单独开发客户端软件，用户通过浏览器就可以访问服务器端提供的服务，其示意图如图 1-2 所示。使用 B/S 模式可加快 Web 应用程序开发的速度，提高开发效率，目前的各大门户网站、各种 Web 信息管理系统等大都采用这种模式。

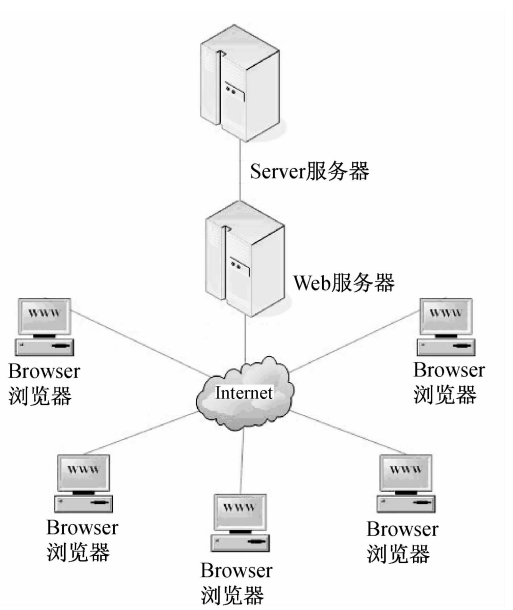


图 1-2 B/S 模式示意图

在 B/S 模式的浏览器端，也不用处理通信方面的问题，这些问题都由 Web 服务器解决，即由 Web 服务器处理用户的 HTTP 请求。

使用 B/S 模式，不仅减少了开发的任务量，而且使软件的部署和升级维护也变得非常简单，只需要把开发的 Web 应用程序部署在 Web 服务器中即可，而客户端更不需要进行任何改动，这是在 C/S 模式中无法实现的。但是 B/S 模式也有自身存在的一些缺点，例如，个性化特点明显降低，无法实现具有个性化的功能要求；页面动态刷新，响应速度明显降低。

### 1.1.3　C/S 与 B/S 的比较

C/S 是建立在局域网的基础上的，而 B/S 是建立在广域网的基础上的。虽然 B/S 模式在电子商务、电子政务等方面得到了广泛的应用，但并不是说 C/S 模式没有存在的必要，相反，在某些领域中 C/S 结构还将长期存在。下面从以下几个方面对 C/S 模式和 B/S 模式进行简单比较。

1）支撑环境

C/S 一般建立在专用的网络上，小范围内的网络、局域网之间再通过特定的服务器进行连接和数据交换；B/S 建立在广域网之上，不需要专门的网络硬件环境，它有比C/S更强的适应性，只要有操作系统和浏览器就能实现。

2）安全控制

C/S 一般面向相对固定的用户群，对信息安全的控制能力很强，一般高度机密的信息系统采用 C/S 结构比较合适；B/S 建立在广域网之上，对安全的控制能力相对较弱，面向的是不可知的用户群，可以通过 B/S 发布部分可公开的信息。

3）程序架构

C/S 模式更加注重流程，可以对权限进行多层次校验，对系统的运行速度可以较少考虑。B/S 模式在安全以及访问速度上有比 C/S 更高的要求，需要建立在软硬件更加优化的基础之上。Microsoft 公司的. NET 系列、Sun 和 IBM 推出的 JavaBean 构件技术都使得 B/S 更加成熟。

4）构件重用

C/S 模式侧重于整体性考虑，构件的重用性不是很好；B/S 模式一般采用多重结构，要求构件有相对独立的功能，能够相对较好地重用。

5）系统维护

由于 C/S 模式的程序的整体性，对它必须整体考察，因此，处理出现的问题以及进行系统升级都比较困难，一旦需要升级，可能要求开发一个全新的系统；B/S 模式的程序由构件组成，通过构件的个别更换即可以实现系统的无缝升级，系统维护开销小。

6）用户接口

C/S 模式的系统多数是建立在 Windows 平台上的，其表现方法有限，对程序员的要求普遍较高；B/S 的应用基于浏览器之上，有更加丰富和生动的表现方式，并且大部分开发难度较低，从而降低了成本。

7）信息流

在 C/S 模式中程序一般是典型的集中式机械处理，交互性相对低；在 B/S 模式中信息流向可变化，如在电子商务的 B2B、B2C 和 B2G 中信息流向的变化很多。

在 C/S 和 B/S 这两种模式之间，并没有严格的界限，两种模式之间没有好坏之分，使用它们都可以实现系统的功能，开发人员可以根据实际的需要进行选择。

## 1.2　动态 Web 开发技术

在互联网发展的最初阶段，所有网页都是静态的 HTML 页面，网站所能实现的功能仅

仅是静态的信息展示，而不能与客户产生互动，为解决这个问题，出现了动态页面。所谓的动态页面，就是指可以和用户产生交互，能根据用户的输入信息产生相应的响应，能满足这种需求的语言就可以称为动态语言。目前，广泛使用的动态页面技术种类繁多，本节简要介绍在国内应用比较广泛的 CGI、ASP、PHP、ASP. NET 和 JSP 技术。

### 1.2.1 CGI

CGI 的全称为 common gateway interface(公共网关接口)，它是最早被用来建立动态网站的后台技术。它可以使用各种语言来编写后台程序，如 C、C++ 、Java 等，但是目前应用最为广泛的是 Perl 语言。因此，狭义上所指的 CGI 程序一般都是指 Perl 程序，后缀为. pl 或者. cgi。

CGI 程序在运行的时候，首先由客户端向服务器上的 CGI 程序发送一个请求，服务器接收到客户的请求后，就会打开一个新的进程(process)来执行 CGI 程序，处理客户的请求，完成后将执行的结果(HTML 页面代码)传回给客户。

由于 CGI 程序每响应一个客户的请求就会打开一个新的进程，因此，当有多个用户同时进行 CGI 请求的时候，服务器就会打开多个进程，这样就加重了服务器的负担，使服务器的执行效率变得越来越低下。同时，CGI 程序的编写比较困难，修改维护也比较复杂，这也就是最近几年来随着各种新的后台技术的诞生，CGI 的应用越来越少的原因。

### 1.2.2 ASP

ASP 的全称为 active server pages，是 Microsoft 开发的一种后台脚本语言，它可以将用户的 HTTP 请求传入到 ASP 的解释器中，由解释器对这些 ASP 脚本进行分析和执行，然后通过服务器将处理的结果和文件中原来的 HTML 代码一同送往客户端，从而实现与用户交互的功能。

ASP 的语法和 Visual Basic 类似，比较简单，可以把后台脚本代码内嵌到 HTML 页面中。它对编程基础没有很高的要求，所以很容易上手，而且 Microsoft 提供的开发环境的功能十分强大，这就更加降低了 ASP 程序开发的难度。

虽然 ASP 简单易用，但是它自身存在着许多缺陷：首先，ASP 在本质上是一种脚本语言，除了使用大量的组件外，没有其他办法提高效率；其次，ASP 本身的安全性也是一大问题；再次，从某种角度来说，ASP 只能在 Microsoft 的 Windows+IIS 的服务器平台上良好地运行(虽然也有在 UNIX/Linux 上运行 ASP 的解决方案，但是目前 ASP 在 UNIX/Linux 上的应用几乎没有)，Windows 自身的一些限制制约了 ASP 的发展。这些都是 ASP 所无法回避的弊端。

### 1.2.3 PHP

PHP 的全称为 PHP：hypertext preprocessor，它是一种内嵌式的语言(就像 ASP 那样)。PHP 语言中混合了 C、Java、Perl 等多种语言的优秀部分，并具有其特殊的新语法，与 CGI/Perl、ASP 相比，可以更快速地执行动态页面。

PHP 是开源的，因此不断地有新的函数库加入和更新，这使得 PHP 无论在 UNIX/Linux 还是 Windows 的平台上都可以很好地发挥其功能。PHP 在 4.0 版后使用了全新的

Zend 引擎，其最佳化后的效率比传统的 CGI、ASP 等技术更高。

平台无关性是 PHP 的最大优点，但是在优点的背后，也有一些小小的缺点：在 PHP 中，提供了对常见数据库（如 MySQL、Oracle 等）的支持，当使用其自带的数据库函数（这样效率要比使用 ODBC 高）来连接不同的数据库时，所使用的 PHP 函数名不统一，从而使得程序的移植变得有些麻烦；另外，PHP 的开发运行环境的配置也比较复杂；同时，PHP 是开源的产品，缺乏正规的商业支持。这些因素在一定程度上限制了 PHP 的进一步发展。

### 1.2.4　ASP .NET

ASP .NET 是一个基于 .NET 的统一的 Web 开发平台，该 Web 开发平台使得 Web 开发人员可以使用任何 .NET 编程语言开发 Web 应用程序。ASP .NET 还是 .NET Framework 的一部分，所以它提供了对该框架的所有功能的访问。

从外观上看，ASP .NET 和 ASP 是相近的，但是从本质上两者是完全不同的。ASP .NET几乎完全基于组件和模块化，每一个页、对象和 HTML 元素都是一个运行的组件对象。在开发语言上，ASP .NET 抛弃了 VBScript 和 Java Script，而使用 .NET Framework 所支持的 VB .NET 和 C# .NET 等语言作为其开发语言，这些语言生成的网页在后台被转换成了类并编译成了一个 DLL。由于 ASP .NET 是编译执行的，所以与 ASP 相比，其效率更高。

与过去的 Web 开发模型相比，ASP .NET 的优势突出表现在以下几点上：

（1）增强的性能。ASP.NET 是在服务器上运行的编译好的公共语言运行库代码。与它可利用早期绑定、实时编译、本机优化和盒外缓存服务，这样在编写代码行之前便显著提高了性能。

（2）世界级的工具支持。ASP .NET 框架补充了 Visual Studio 集成开发环境中的大量工具箱和设计器（如 WYSIWYG 编辑、拖放服务器控件和自动部署控件等）。

（3）威力和灵活性。由于 ASP .NET 基于公共语言运行库，因此 Web 应用程序开发人员可以利用整个平台的威力和灵活性。.NET 框架的类库、消息处理和数据访问解决方案都可从 Web 无缝访问。ASP .NET 也与语言无关，所以可以选择最适合应用程序的语言，或跨多种语言分割应用程序。另外，公共语言运行库的交互性保证在迁移到 ASP .NET 时保留基于 COM 的开发中的现有投资。

（4）简易性。ASP .NET 使执行常见任务变得容易，从简单的窗体提交和客户端身份验证到部署和站点配置，都可以很方便地完成。

（5）可管理性。ASP .NET 采用基于文本的分层配置系统，简化了将设置应用于服务器环境和 Web 应用程序的过程。

（6）可缩放性和可用性。ASP .NET 在设计时考虑了可缩放性，增加了专门用于在聚集环境和多处理器环境中提高性能的功能。另外，ASP .NET 运行库可以密切监视和管理进程，以便当进程行为不正常（泄漏、死锁）时，可就地创建新进程，以帮助保持应用程序始终可用于处理请求。

（7）自定义性和扩展性。ASP .NET 随附了一个设计周到的结构，它使开发人员可以在适当的级别“插入”代码。实际上，用户可以用自己编写的自定义组件扩展或替换ASP .NET 运行库的任何子组件。

（8）安全性。借助内置的 Windows 身份验证和基于每个应用程序的配置，可以保证应

用程序是安全的。

### 1.2.5 JSP

JSP 的全称为 Java server pages，是由 Sun 公司倡导、多家公司参与，于 1999 年推出的一种 Web 服务设计标准。JSP 是基于 Java Servlet 以及整个 Java 体系的 Web 开发技术，利用这一技术可以建立安全的、跨平台的先进动态网站。

JSP 本质上就是把 Java 代码嵌套到 HTML 中，然后经过 JSP 容器的编译执行，根据这些动态代码的运行结果生成对应的 HTML 代码，从而可以在客户端的浏览器中正常显示。由于 JSP 中使用的是 Java 的语法，因此 Java 语言的所有优势都可以在 JSP 中体现出来，尤其是 J2EE 的强大功能，更是 JSP 语言发展的强大后盾。JSP 技术在多个方面加速了动态 Web 页面的开发，其优点主要有以下几方面：

**1. 将内容的产生和显示进行分离**

利用 JSP 技术，Web 页面开发人员可以使用 HTML 或者 XML 标识来设计和格式化最终页面，使用 JSP 标识或者小脚本来产生页面上的动态内容。产生内容的逻辑被封装在标识和 JavaBean 群组件中，并且捆绑在小脚本中，所有的脚本在服务器端执行。如果核心逻辑被封装在标识和 JavaBean 中，那么其他人（如 Web 管理人员和页面设计者）能够编辑和使用 JSP 页面，而不影响内容的产生。在服务器端，JSP 引擎解释 JSP 标识，产生所请求的内容，并且将结果以 HTML（或者 XML）页面的形式发送回浏览器。这样可以保护代码，同时保证任何基于 HTML 的 Web 浏览器的完全可用性。

**2. 强调可重用的群组件**

绝大多数 JSP 页面都依赖于可重用且跨平台的组件（如 JavaBean 或者 Enterprise JavaBeans）来执行应用程序所要求的更为复杂的处理。开发人员能够共享普通操作的组件，或者使得这些组件为更多的使用者或者用户团体所使用。基于组件的方法可以加速总体开发过程，并且使得各种群组织在它们现有的技术和优化结果中得到平衡。

**3. 采用标识简化页面开发**

JSP 技术封装了许多功能，这些功能是在易用的、与 JSP 相关的 XML 标识中进行动态内容生成所需要的。标准的 JSP 标识能够存取和实例化 JavaBean 组件，设定或者检索群组件属性，下载 Applet，以及执行用其他方法更难于编码和耗时的功能。

另外，通过开发定制标识库，JSP 技术是可以扩展的，今后，第三方开发人员和其他人员可以针对常用功能建立自己的标识库。

**4. 易于整合到多种应用体系结构中**

作为 Java 技术家族的一部分，以及 J2EE 的一个成员，JSP 技术能够支持高度复杂的基于 Web 的应用，如企业级的分布式应用等。

**5. 一次编写，各处执行**

由于 JSP 页面的内置脚本语言是基于 Java 程序设计语言的，作为 Java 平台的一部分，JSP 拥有 Java 程序设计语言“一次编写，各处执行”的特点。随着越来越多的供货商将 JSP 支持加入到他们的产品中，用户可以使用自己所选择的服务器和工具，修改工具或服务器并不影响目前的应用。

相比其他的开发技术而言，JSP 具有很好的跨平台性，开发及运行效率较高，安全性也优于其他开发技术，具有较好的市场前景。

## 1.3　JSP 的 Web 开发方式

JSP 既可以用于开发小型的 Web 站点，也可以用于开发大型的、企业级的应用程序，下面介绍对于不同规模的 Web 系统，使用 JSP 进行开发的不同方式。

### 1. 直接使用 JSP

对于小型的 Web 站点，可以直接使用 JSP 来构建动态网页。这种站点的要求一般比较简单，所需要的仅仅是简单的留言板、动态日期等基本的功能。在这种情况下，一般可以将所有的动态处理部分都放置在 JSP 的脚本语言中，这同使用 PHP 或 ASP 开发动态网页一样。

### 2. JSP＋JavaBean

中型站点面对的是数据库查询、用户管理和小量的商业业务逻辑。对于这种站点，不能将所有的东西全部交给 JSP 页面来处理，在单纯的 JSP 中加入 JavaBean 技术将有助于这种中型网站的开发。利用 JavaBean 可以很容易地完成如数据库连接、用户登录与注销、商业业务逻辑封装等任务。例如，将常用的数据库连接写为一个 JavaBean，既方便了使用，又可以使 JSP 文件简单而清晰，通过封装，还可以防止一般的开发人员直接获得数据库的控制权。

### 3. JSP＋JavaBean＋Servlet

无论是用 ASP 还是用 PHP 开发动态网站，长期以来都存在一个比较重要的问题，就是网站的逻辑关系和网站的显示页面不容易分开。常常可以看见一些夹杂着 if…then…、case select 和大量显示用的 HTML 代码的 ASP、PHP 页面，即使是有着良好的编程习惯的程序员，其作品也几乎无法阅读。另一方面，动态 Web 的开发人员也在抱怨，将网站美工设计的静态页面和动态程序合并的过程是一个异常痛苦的过程。

在 JSP 问世以后，Servlet 不再担负动态页面生成的任务，开始担负起决定整个网站逻辑流程的任务。在逻辑关系异常复杂的网站中，借助于 Servlet 和 JSP 良好的交互关系和 JavaBean 的协助，完全可以将网站的整个逻辑结构放在 Servlet 中，而将动态页面的输出放在 JSP 页面中来完成。在这种开发方式中，一个网站可以有一个或几个核心的 Servlet 来处理网站的逻辑，通过调用 JSP 页面来完成客户端(通常是 Web 浏览器)的请求。

这是一种目前应用较多的轻量级开发方式，本书将主要讲述这种开发方式。当然，利用 JSP 进行 Web 开发还有很多方式，如 JSP＋Struts、JSP＋Hiberate＋Spring 等，限于篇幅，不再赘述。

## 1.4　开发环境的构建

学习软件开发第一步要做的就是学会搭建开发环境，本节将以 JDK＋Tomcat 组合为例，在 Windows XP 平台下一步步搭建好 JSP 开发环境。

### 1.4.1 JDK 的安装与配置

JDK(Java™ SE development kit,Java 开发工具包)是从事 Java 开发的基础开发工具,Tomcat 需要 JDK 作为基础环境支持。在服务器上运行 JSP 动态网页时,需要用到 JDK 中的 Java 编译器和 Java 虚拟机,以便将服务器上的 Java 语言源程序(.java)编译为字节码文件(.class),并通过执行该字节码文件生成网页中的动态内容。

#### 1. JDK 的安装

JDK 可以在 Sun 公司网站上免费下载,下载过程中注意选择适合操作系统平台和语言环境的版本,并注意其安装方式。本书使用 JDK 6 Update 13 版本(JDK 1.6),它包含了 JRE(Java runtime environment,Java 运行时的环境)和用于开发 Java Applet 与 Java 应用程序的命令行开发工具。

双击运行下载得到的文件,在安装向导的提示下,完成 JDK 的安装。其中,“自定义安装”对话框用来选择安装路径和需要的组件,如图 1-3 所示。


图 1-3 JDK“自定义安装”对话框

“开发工具”为 JDK 6 Update 13,其中 JRE 6 Update 13 约需 300 MB 空间;“演示程序及样例”中包含一些小程序和应用程序的演示和样例;“源代码”是构成 Java 公共 API 的类的源代码;“Java DB”是 Sun 支持的纯 Java 技术实现的一种数据库。

#### 2. JDK 的环境设置

JDK 安装完成后,可以通过如下方法检验其环境配置是否正确:

在命令提示符窗口中输入“java -version”命令查看 JDK 的安装版本情况。若命令能够正确执行(结果如图 1-4 所示),则说明 JDK 环境配置正常。

若不能正确执行,则需要对 JDK 的环境变量进行基本的设置,方法如下:

(1)右击“我的电脑”图标,在弹出的快捷菜单中选择“属性”命令,在弹出的“系统属性”对话框中选择“高级”选项卡,单击“环境变量”按钮即可进入“环境变量”对话框,如图 1-5 所示。

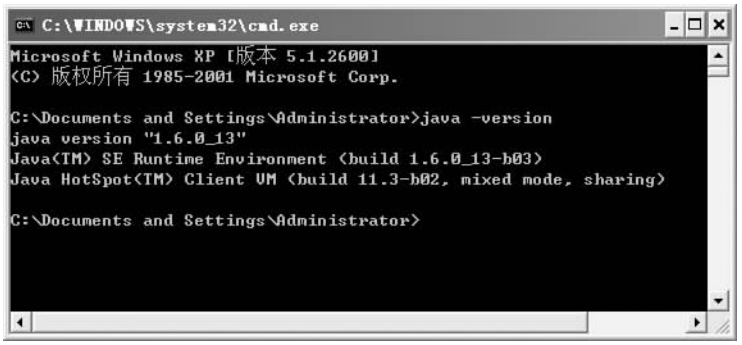


图 1-4　命令执行结果


图 1-5　“环境变量”对话框

(2)单击“系统变量”组合框内的“新建”按钮,打开“新建系统变量”对话框,在“变量名”文本框中填入“JAVA_HOME”,在“变量值”文本框中填入 JDK 的安装路径,这里填入“c:\jdk”,如图 1-6 所示。

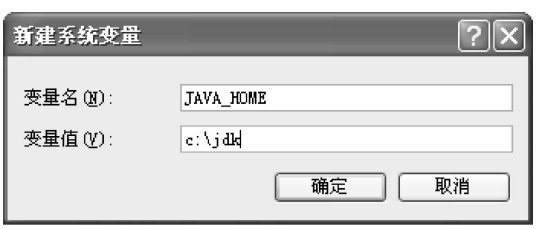


图 1-6　“新建系统变量”对话框

(3)按(2)所述步骤,新建一个名为“CLASSPATH”的系统变量,变量值为类和包文件的搜索路径,这里填写“.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar”。

(4)在“系统变量”列表框中双击“Path”系统变量,打开“编辑系统变量”窗口,在其中添加一个变量值“;%JAVA_HOME%\bin”,这是 Java 应用程序的位置。

**注意：**

- CLASSPATH 中的“.”表示当前目录，一定要输入，否则会出现错误。
- 在 Path 中添加变量时，该变量与前面的变量之间必须有分号相隔。

### 1.4.2 下载与安装 Tomcat

Tomcat 是 Apache 基金组织提供的一种 Web 服务器，它提供了对 JSP 和 Java Servlet 的支持，通过安装插件，同样可以提供对 PHP 语言的支持。Tomcat 所占资源少、费用低，且其功能正在不断完善，已成为 Java Web 程序员的首选开发工具。与此同时，也有大量的中小型网站或 B/S 模式的软件系统采用 Tomcat 作为 Web 服务器软件。但是 Tomcat 只是一个轻量级的 Java Web 容器，像 EJB 这样的服务在 Tomcat 中是不能运行的。

Tomcat 是一个实现了 Servlet 和 JSP 的容器，不同版本的 Tomcat 容器实现了不同的 Servlet/JSP 规范，比较典型的情况如表 1-1 所示。

**表 1-1 典型的 Tomcat 容器实现的 Servlet/JSP 规范**

| Tomcat 版本 | 实现的 Servlet/JSP 规范 |
|---|---|
| 6.0.18 | 2.5/2.1 |
| 5.5.26 | 2.4/2.0 |
| 4.1.37 | 2.3/1.2 |
| 3.3.2 | 2.2/1.1 |

#### 1. 下载 Tomcat

本书使用的 Tomcat 的版本是 Tomcat 6.0.18(后文简称为 Tomcat 6)，可以从 Apache 的官方网站免费下载得到。网站上提供了三种类型的 Tomcat 6 的下载：zip、tar.gz 和 Windows Service Installer。zip 类型文件下载后得到的是一个 zip 文件，无须安装，解压缩后即可使用；tar.gz 类型文件下载后得到的是一个 tar.gz 文件，是在 GNU 操作系统中用 tar 命令打包而成的，因此必须在与 GNU 相兼容的操作系统中解包，Solaris 和 Mac 操作系统中不能使用；Windows Service Installer 类型文件下载后得到的是一个 exe 文件，是在 Windows 操作系统下的安装程序，使用这种方式安装的 Tomcat 6 可以通过 Windows 的服务来控制其启动或停止。

#### 2. 安装 Tomcat

(1)双击刚下载的可执行文件 apache-tomcat-6.0.18.exe，弹出欢迎对话框，单击“Next”按钮。弹出软件许可对话框，单击“I Agree”按钮继续。

(2)打开如图 1-7 所示的安装类型选择对话框，选择安装类型后单击“Next”按钮。

(3)打开如图 1-8 所示的安装目录选择对话框，选择安装目录后单击“Next”按钮。

(4)打开如图 1-9 所示的基本配置对话框。在该窗口中，“HTTP/1.1 Connector Port”文本框用来设置 Tomcat 服务启动后所占用的端口号，系统默认分配为 8080，用户可自行更改，但注意一定不要与系统中其他启动的服务所使用的端口相冲突，否则 Tomcat 无法启动；“User Name”和“Password”文本框是通过 Web 方式对 Tomcat 进行管理时所使用的管理员账号和密码设置项。设置完成后单击“Next”按钮。

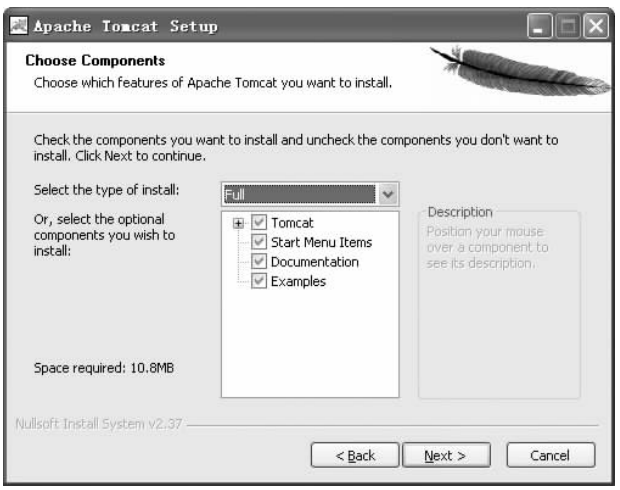


图 1-7　安装类型选择对话框

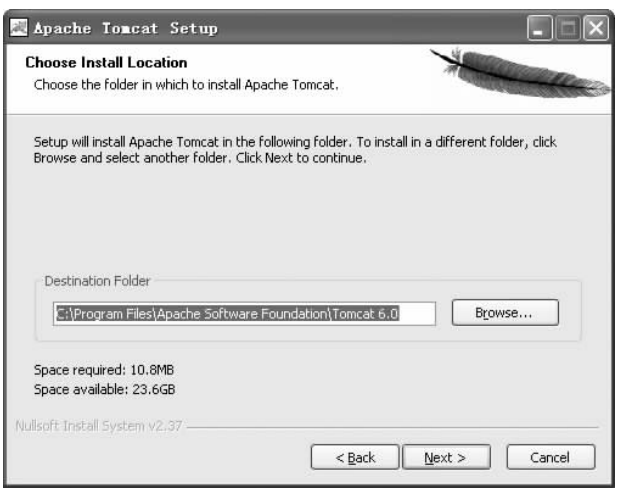


图 1-8　安装目录选择对话框

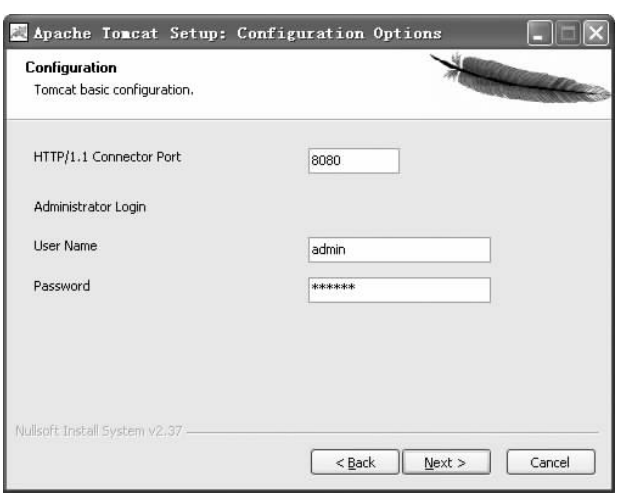


图 1-9　基本配置对话框

(5)打开如图 1-10 所示的 JRE 路径选择对话框,注意 JRE 路径要和在如图 1-3 所示的 JRE 安装过程中所设置的路径完全一致。设置完成后单击“Install”按钮进行安装即可。

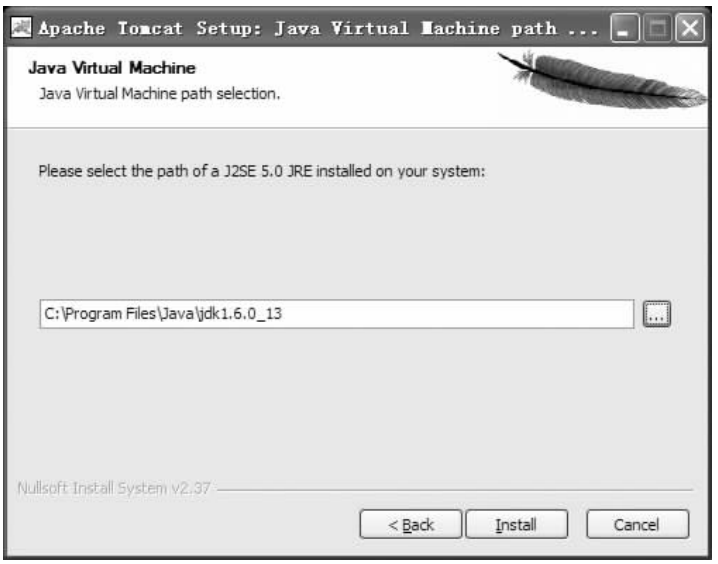


图 1-10 JRE 路径选择对话框

### 3. 启动 Tomcat

启动 Tomcat 的方法有很多:在 Tomcat 6 安装完成时会提示是否启动 Tomcat,此时即可启动 Tomcat,也可以通过“开始”菜单中的 Tomcat 项来启动,或通过“管理工具”中的服务来启动 Tomcat。Tomcat 启动后将会最小化为系统任务栏的通知区域中的图标。

在浏览器中输入地址 http://localhost:8080,若 Tomcat 6 启动成功,将出现如图 1-11 所示的界面。此处“localhost”也可以改为机器名或 IP 地址,如果是在 Internet 中,还可以使用域名,“localhost”与“127.0.0.1”均表示本机。

### 4. Tomcat 安装目录说明

Tomcat 6 安装后的目录如图 1-12 所示。

(1)bin 目录下是一些可执行文件,如启动、停止的批处理命令文件。在 Windows 操作系统中启动 Tomcat 6 的命令是 startup.bat,停止 Tomcat 6 的命令是 shutdown.bat,这些实际上是一些批处理文件,执行了一系列的命令。在 UNIX 下启动 Tomcat 6 的命令是 startup.sh,停止 Tomcat 6 的命令是 shutdown.sh。

(2)conf 目录下是一些有关 Tomcat 6 服务器的配置文件和参数文件,如 server.xml、logging.properties 等。

(3)lib 目录用于存放一些 Tomcat 6 中 Web 应用程序共用的类库(jar 包)和资源文件。

(4)work 目录是供 Web 应用程序使用的临时工作目录。

(5)temp 目录是供 JVM(Java virtual machine,Java 虚拟机)使用的存放临时文件的目录。

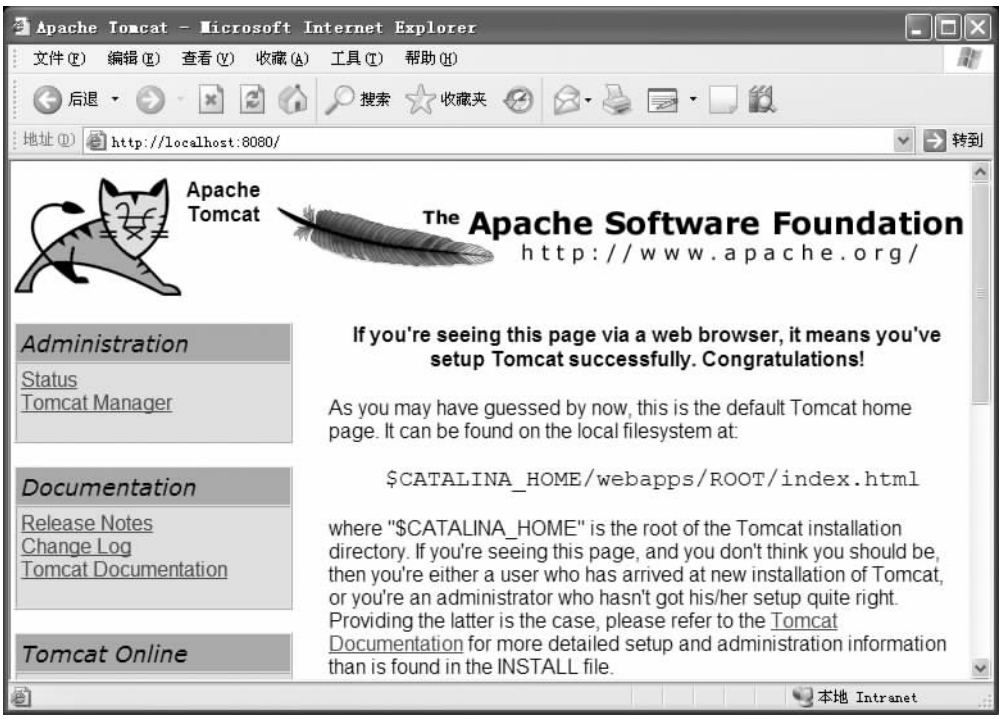


图 1-11　Tomcat 6 启动界面


图 1-12　Tomcat 安装后的目录

(6)webapps 目录用于存放一些 Tomcat 6 自动装载的 Web 应用,可以是 Web 应用程序的整个目录。这个目录中已自带了一些 Web 应用,其中 ROOT 应用是默认的根 Web 应用。在浏览器中输入地址 http://localhost:8080,访问的其实就是 ROOT 应用中的 index.jsp 页面。

### 1.4.3 第一个 JSP 程序

JSP 开发环境安装完成后，需要编写一个 JSP 程序来测试它们工作是否正常，是否能够正确解释 JSP 代码。编写 JSP 程序的工具有很多，如 Dreamweaver、Eclipse 等，其中最简便的莫过于 Windows 自带的“记事本”。

使用编辑工具创建第一个 JSP 程序 first.jsp，其内容如下：

```
<html>
  <head><title>My First JSP</title></head>
  <body>
  <h1><% out.println("Hello World!"); %></h1>
  <h2><% out.println("This is the first JSP program! Test OK!"); %></h2>
  </body>
</html>
```

在 Tomcat 服务器中调试 JSP 程序的方法如下：

(1)进入 Tomcat 的安装目录的 webapps 文件夹，可以看到 ROOT、examples、tomcat-docs 等 Tomcat 自带的目录。将刚才创建的 first.jsp 文件复制到 ROOT 文件夹中。

(2)在浏览器中输入 http://localhost:8080/first.jsp，若能出现如图 1-13 所示的运行结果，就说明 JSP 开发环境正常。


图 1-13 JSP 程序测试界面

## 1.5 JSP 开发工具

### 1.5.1 Dreamweaver

Dreamweaver 是用于网站设计和网页制作的软件，它提供了强大的可视化的布局工具和应用开发功能，使设计和开发人员能高效地设计、开发和维护基于标准的网站和应用程序。在 JSP 程序开发中，可以在 Dreamweaver 中选择建立 JSP 文件，并完成 JSP 程序的开

发。如图 1-14 所示即为在 Dreamweaver CS3 中选择新建一个 JSP 空白页的界面。


图 1-14　在 Dreamweaver CS3 中选择新建一个 JSP 空白页

### 1.5.2　Eclipse

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台，也是目前最著名的开源项目之一。它专注于为高度集成的工具开发提供全功能的、具有商业品质的工业平台。

Eclipse 主要由 Eclipse 项目、Eclipse 工具项目和 Eclipse 技术项目三个项目组成，具体包括 Eclipse Platform、JDT、CDT 和 PDE 这四个组成部分。JDT 支持 Java 开发，CDT 支持 C 开发，PDE 支持插件开发，Eclipse Platform 则是一个开放的可扩展 IDE，提供了一个通用的开发平台。Eclipse Platform 允许工具创建者独立开发与其他工具无缝集成的工具，从而无须分辨一个工具功能在哪里结束，而另一个工具功能在哪里开始。

1. 安装 Eclipse

Eclipse 是开放源代码的项目，可以到 http://www.eclipse.org 免费下载 Eclipse 的最新版本。一般 Eclipse 提供 Release、Stable Build、Integration Build 和 Nightly Build 等几个版本，建议下载 Release 或 Stable Build 版本。

Eclipse 本身虽然是 Java 语言编写的，但其本身并不包含 Java 运行环境，需要用户自己安装 JRE，并在环境变量中指明 JRE 的 bin 的路径。由于 JRE 在前面的已经安装过了，这里不再详细介绍。

实际上，Eclipse 并不需要进行安装，直接解压缩就可以使用。单击文件夹内的 eclipse.exe，就会显示如图 1-15 所示的对话框，要求选择工作空间，用户编写的文件将会存储在此工作空间中。

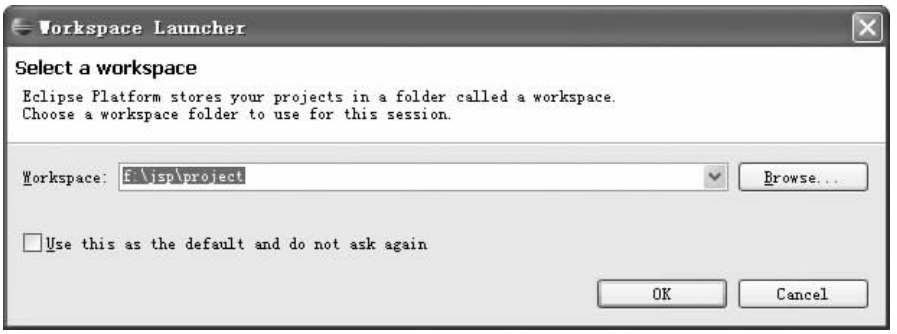


图 1-15　选择工作空间对话框

每当在 Eclipse 中新生成一个项目时，默认情况下都会在 workspace 中产生和项目同名的文件夹，以存放该项目所用到的全部文件。可以用 Windows 资源管理器直接访问或维护这些文件。

设置好后，单击“OK”按钮，显示如图 1-16 所示的 Eclipse 工具的欢迎窗口，至此，Eclipse 的安装完成。

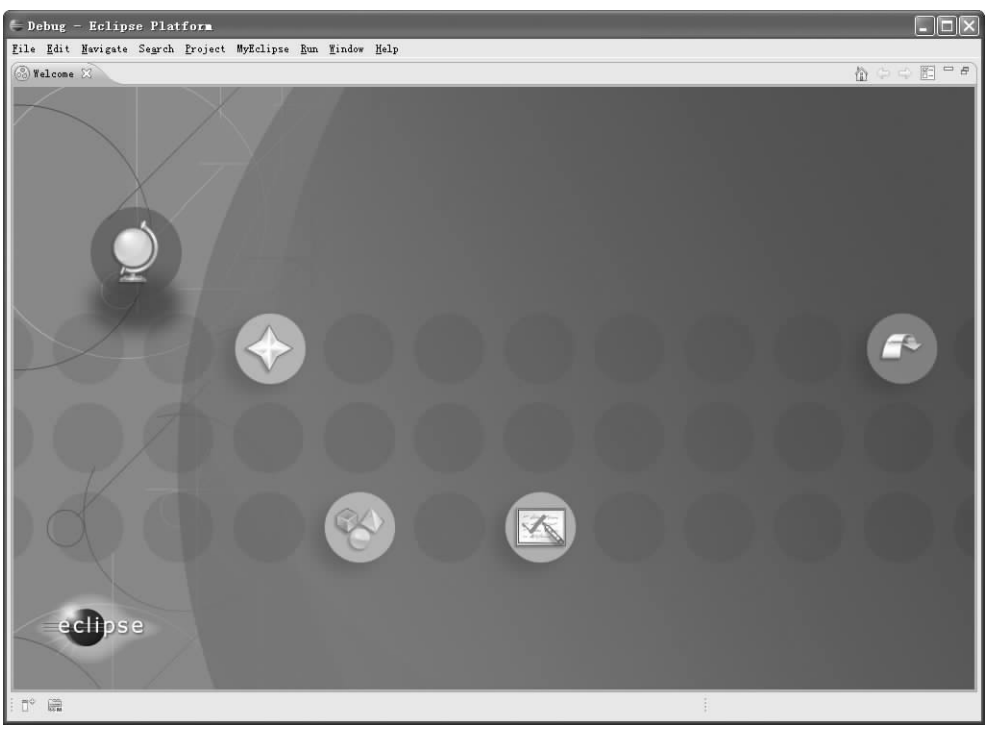


图 1-16　Eclipse 工具的欢迎窗口

**注意：**目前只能使用新建项目的方法建立或指定任何项目文件夹，即使是在默认存储文件项目的文件夹 workspace 路径下新建一个文件夹，在 Eclipse 环境中也无法将其变成一个

项目，也就是说，这个文件夹对 Eclipse 是不可视的。

### 2. Eclipse 开发环境

Eclipse 开发环境被称为 Workbench，主要由视图（perspective）、编辑窗口（editor）和观察窗口（view）三部分组成，如图 1-17 所示。

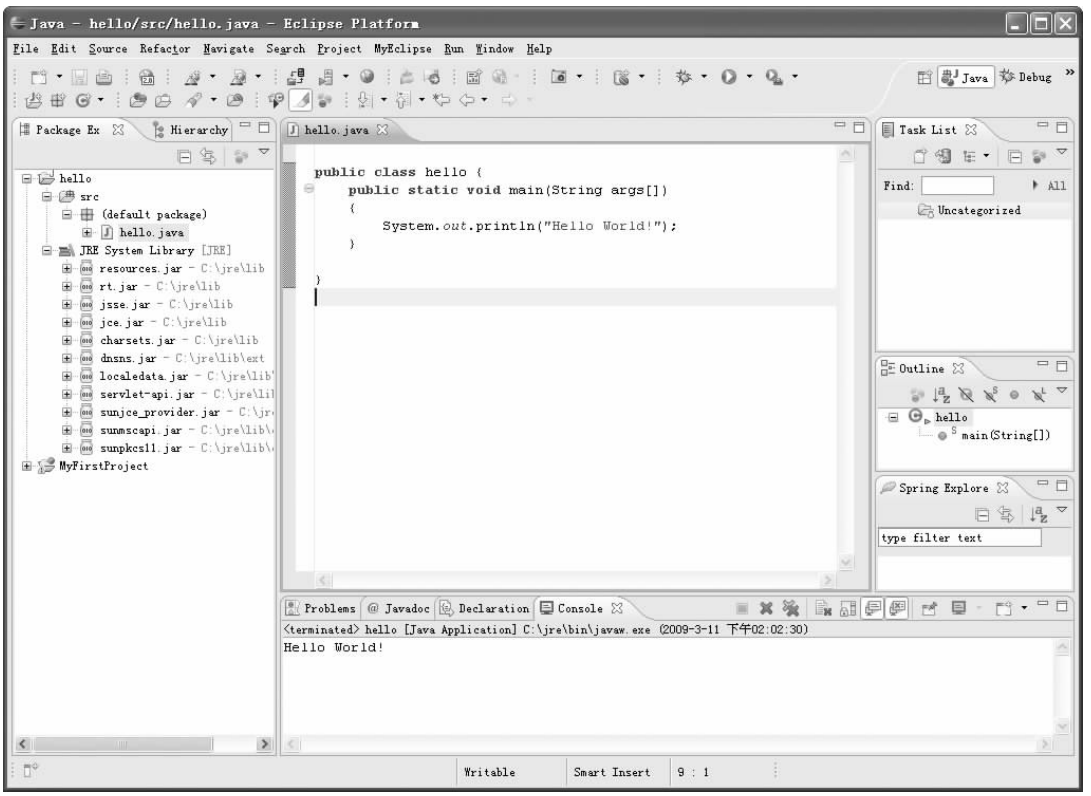


图 1-17　Eclipse 开发环境

（1）编辑窗口：所有文件的编辑和显示都包含在编辑窗口中，默认情况下打开的多个文件是以标签的形式在同一个窗口中排列的，可以用拖动的方式重新布局这些文件。可以在资源浏览窗口或 Java 包浏览窗口中双击文件名打开相应的文件。

（2）观察窗口：常用的观察窗口有资源浏览窗口（navigator）、Java 包浏览窗口（packages）、控制台（console）、任务栏（task）等，它主要配合编辑窗口提供多种相关信息和浏览方式。观察窗口是任何 IDE 开发环境的核心，用好观察窗口也就是用好 IDE 开发环境。

（3）视图：视图包括一个或多个编辑窗口和观察窗口。在开发环境最左边的快捷栏中的上部分显示的就是当前打开的视图图标。视图是 Eclipse 中最灵活的部分，可以自定义每个视图中包含的观察窗口的种类，也可以自定义一个新视图。Eclipse 的 Java 开发环境中提供了几种默认视图，如资源视图（resource perspective，它是 Eclipse 第一次启动时的默认视图）、Java 视图（Java perspective）、调试视图（debug perspective）、团队视图（team perspective）等。

### 3. 安装与配置 MyEclipse

在 Eclipse 中运行 JSP 需要安装插件。支持 Eclipse 运行的 JSP 插件主要有两个：一个是 MyEclipse，另一个 Lomboz 插件。这里选择 MyEclipse 插件，官方下载网站是 http://www.myeclipseide.com。

1）安装 MyEclipse

选择一个与已经安装的 Eclipse 对应的版本进行下载。

下载完成后，双击运行安装程序，会先收集信息。完成后进入正式的安装界面，如图 1-18 所示。


图 1-18　MyEclipse 安装界面

在窗口的提示信息中，说明了该 MyEclipse 的版本。确认无误后，单击 Next 按钮，在打开的窗口中选择“I acknowledge and accept ALL of the above licenses”即可，如图 1-19 所示。

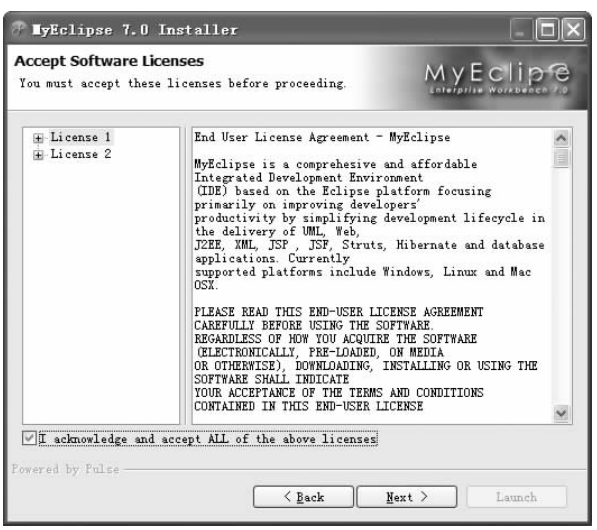


图 1-19　同意安装协议

单击“Next”按钮，会弹出如图 1-20 所示的窗口，在此可选择 MyEclipse 的安装路径。

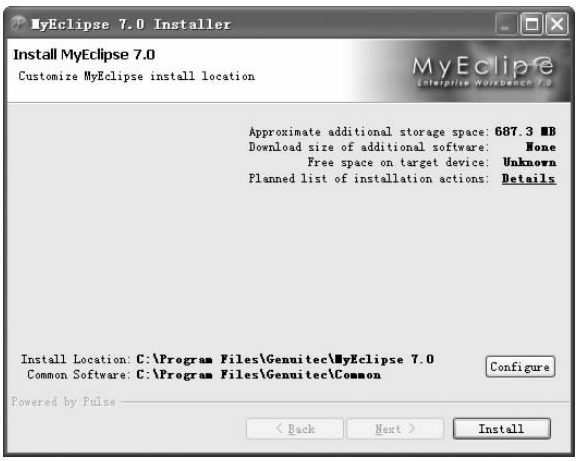


图 1-20　设置安装路径

在该窗口中，选择 MyEclipse 的安装路径，可以安装在除刚才选择的 Eclipse 的文件夹以外的系统中的任何一个位置。选择好后，单击“Install”按钮，进入如图 1-21 所示的对话框，表示开始安装 MyEclipse。


图 1-21　开始安装 MyEclipse

完成后，弹出对话框，要求选择工作空间。选择好后按“OK”键后即可打开 MyEclipse 7.0，如图 1-22 所示。

2）MyEclipse 的配置

MyEclipse 已经安装完成，但要使其在 Eclipse 中真正起作用还要进行必要的配置。把 MyEclipse 安装路径下的 features 文件夹和 plugins 文件夹下的内容分别复制到 Eclipse 下的 features 和 plugins 文件夹内，然后重启 Eclipse 就可以使用该插件了。

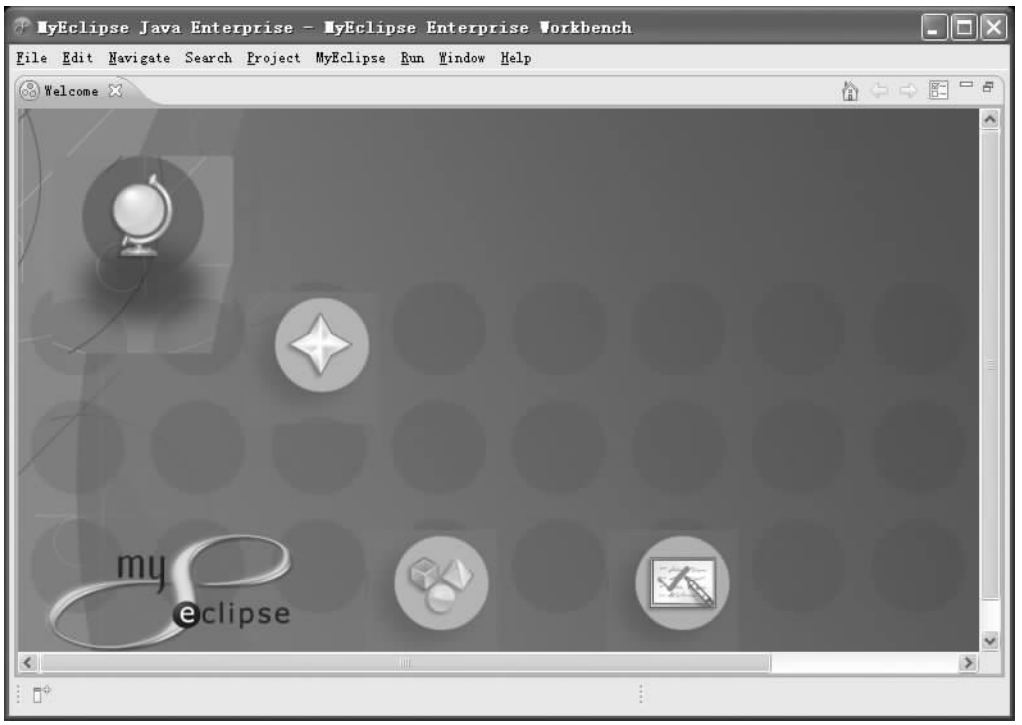


图 1-22 MyEclipse 界面

## 本 章 小 结

本章对 Web 开发中的一些基本知识进行了简要的介绍，阐述了 JSP 技术的特点并与其他开发技术进行了比较，并以 JDK 1.6＋Tomcat 6.0 组合为例详细说明了 JSP 程序的开发环境的安装配置过程及 JSP 程序的简单调试方法。

需要初学者注意的是，技术没有高低之分，只有应用场合的不同，没有最有用的技术，只有最合适的技术。在开发一些跨平台的 Web 应用时，JSP 是一种不错的选择。

## 习 题 1

**一、简答题**

1. 试列举一些利用 C/S 或 B/S 模式开发的网络应用的例子。

2. JSP 与 ASP、PHP 三者有什么共性？

3. 若 Tomcat 服务未启动，则书中 1.4.3 的 first.asp 程序的执行结果会是什么？试解释其原因。

**二、操作题**

1. 下载 JDK 1.6 并进行安装。

2. 下载 Tomcat 6.0 并进行安装，熟悉 Tomcat 服务器的启动、停止和退出等基本操作。
3. 编写一个简单的 JSP 程序，体验 JSP 程序的编写和调试方法。
4. 配置 Eclipse＋MyEclipse 开发环境。

# 第 2 章　JSP 编程基础

JSP 是对 HTML 语法的 Java 扩展，学习 Java 的语法是掌握 JSP 的必由之路。本章在介绍 JSP 文件的处理过程的基础上，介绍了 Java 的基础语法，为初学者提供帮助。表单是获取客户信息的有效途径，如何利用 JSP 实现对表单数据的获取和处理是本章的另一主要内容。

## 2.1　JSP 程序的初步体验

本节主要介绍 JSP 文件结构及其工作原理，并详细说明了 Tomcat 环境下 JSP 程序的配置。

### 2.1.1　JSP 文件结构

JSP 是对 HTML 语法的 Java 扩展，在 HTML 的基础上加入了新的标签(<%,%>)，其文件结构如下所示：

```
<%@ page contentType="text/html;charset=gb2312" %>
<%@ page import="java.util.* " %>
<HTML>
  <BODY>
    其他 HTML 语言
  <%
    符合 Java 语法的 Java 语句
  %>
    其他 HTML 语言
  </BODY>
</HTML>
```

详细示例可参阅第 1 章的 JSP 程序。

### 2.1.2　JSP 执行过程解析

JSP 文件是在一个普通的静态 HTML 文件中添加了一些 Java 代码得来的，文件的扩展名为.jsp。当 Web 服务器上的 JSP 页面第一次被请求执行时，它首先会被 JSP 容器翻译为一个 Java 源文件，即一个 Servlet(Java Servlet 是基于服务器端编程的 API，用 Java Servlet 编写的 Java 程序称为 Servlet，Servlet 通过 HTML 与客户交互，本书第 7 章有详细介绍)；

在转换时如果发现 JSP 文件有任何语法错误，转换过程将中断，并向服务端和客户端输出出错信息；如果转换成功，该 Java 源文件将被编译成相对应的字节码文件，后缀为. class，然后像其他 Servlet 一样，由 Servlet 容器来处理。Servlet 容器装载并执行这个字节码文件，然后把结果返回给客户端以响应客户的请求，过程如图 2-1 所示。

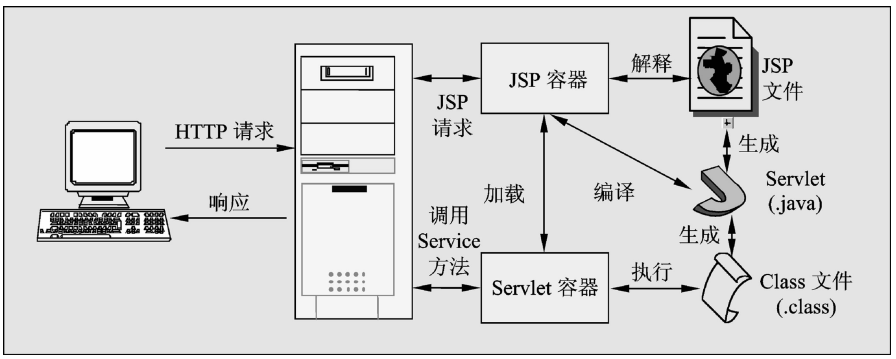


图 2-1　JSP 页面的执行过程

若这个 JSP 页面被再次请求，只要该 JSP 文件没有发生过更改，JSP 容器就直接调用已经装载的 Servlet。如果已经做过修改，那就会再次执行以上过程。因为首次访问的时候要执行一系列上面的过程，所以第一次访问某 JSP 页面时速度会较慢；但在以后运行时速度将非常快。

### 2.1.3　在 Tomcat 下配置 JSP 程序

Tomcat 提供了一系列的配置文件来帮助用户配置自己的 Tomcat，Tomcat 的配置文件主要是基于 XML 的，如 server. xml、web. xml 等，只有 workers. properties 和 uriworkermap. properties 等几个少数文件是传统的配置文件。本节将详细讨论 Tomcat 的主要配置文件。

#### 1. 主配置文件 server. xml

server. xml 文件描述了 Tomcat 的基本配置信息。server. xml 文件由元素组成，各元素及其属性的配置意义如表 2-1 所示。

表 2-1　server. xml 的元素及其属性说明

| 元 素 名 | 属　性 | 说　明 |
| --- | --- | --- |
| Server（该元素是 server. xml 文件最高级别的元素，它描述了一个 Tomcat 服务器） | port | 指定一个端口，这个端口负责监听关闭 Tomcat 的请求 |
| | shutdown | 指定向端口发送的命令字符串 |
| Service | name | 指定 Service 的名字 |

续表

| 元 素 名 | 属　性 | 说　明 |
| --- | --- | --- |
| Connector(表示客户端和 Service 之间的连接) | port | 指定服务器端要创建的端口号,并在这个端口监听来自客户端的请求 |
| | minThreads | 服务器启动时创建的用于处理请求的最小线程数 |
| | maxThreads | 最多可以创建的处理请求的线程数 |
| | enableLookups | 如果为 true,则可以通过调用 request. getRemoteHost()进行 DNS 查询来得到远程客户端的实际主机名,若为 false 则不进行 DNS 查询,而是返回其 IP 地址 |
| | redirectPort | 指定服务器正在处理 HTTP 请求时,如果收到了一个 SSL 传输请求后,重定向的端口号 |
| | acceptCount | 指定当所有可以使用的处理请求的线程数都被使用后,可以放到等待处理队列中的请求数,超过这个数的请求将不予处理 |
| | connectionTimeout | 以毫秒为单位,指定连接时的超时时间 |
| Engine(表示指定 Service 中的请求处理机,接收和处理来自 Connector 的请求) | defaultHost | 指定默认的处理请求的主机名,它至少应该与其中的一个 host 元素的 name 属性值是相同的 |
| Context(每一个 Context 都描述了一个 Tomcat 的 Web 应用程序的目录) | docBase | Context 的目录 |
| | path | 表示 Context 在 Web 服务器时的虚拟目录位置和目录名 |
| | reloadable | 如果属性为 true,则 Tomcat 会自动检测应用程序的/WEB-INF/lib 和/WEB-INF/classes 目录的变化,自动装载新的应用程序,可以在不重启 Tomcat 的情况下改变应用程序 |
| host(表示一个虚拟主机) | name | 指定主机名 |
| | appBase | 应用程序基本目录,即存放应用程序的目录 |
| | unpackWARs | 如果为 true,则 Tomcat 会自动将 WAR 文件解压,否则不解压,直接从 WAR 文件中运行应用程序 |
| Logger(表示日志、调试和错误信息) | className | 指定 Logger 使用的类名,此类必须实现 org. apache. catalina. Logger 接口 |
| | prefix | 指定 log 文件的前缀 |
| | suffix | 指定 log 文件的后缀 |
| | timestamp | 如果为 true,则 log 文件名中要加入时间,如下 localhost_log. 2001-10-04. txt |

续表

| 元 素 名 | 属　性 | 说　明 |
| --- | --- | --- |
| Realm(表示存放用户名、密码及 role 的数据库) | className | 指定 Realm 使用的类名,此类必须实现 org. apache. catalina. Realm 接口 |
| Valve(功能与 Logger 差不多,其 prefix 和 suffix 属性解释和 Logger 中的一样) | className | 指定 Valve 元素所使用的类名,如果使用 org. apache. catalina. valves. AccessLogValve 类可以记录应用程序的访问信息 |
| | directory | 指定 log 文件存放的位置 |
| | pattern | 有两个值,common 方式记录远程主机名或 IP 地址、用户名、日期、第一行请求的字符串、HTTP 响应代码、发送的字节数;combined 方式比 common 方式记录的值更多 |

下面使用 server. xml 文件设定一个新的 JSP 工作目录。设立新的 JSP 工作目录是比较简单的,只需要添加一个 Context 元素即可。例如,想要将“D:\jspWork”设定为一个新的 JSP 工作目录,并且使用户可以使用/myjsp 访问该目录,只需要在 server. xml 文件中添加如下的 Context 元素:

```
<Context path="/myjsp" docBase="D:\jspWork" reloadable="true">
</Context>
```

然后,就可在“D:\jspWork”目录下创建 JSP 文件,在浏览器的地址栏中输入 http://localhost:8080/myjsp/<jsp 文件名>,即可访问指定的 JSP 文件了。

**2. 部署描述符文件 web. xml**

除 server. xml 文件外,另一个重要的配置文件为部署描述符文件 web. xml。

1)定义头和根元素

部署描述符文件就像所有 XML 文件一样,必须以一个 XML 头开始。这个头声明可以使用的 XML 版本并给出文件的字符编码。DOCTYPE 声明必须立即出现在此头之后。这个声明告诉服务器适用的 servlet 规范的版本(如 2. 2 或 2. 3)并指定管理此文件其余部分内容的语法的 DTD(document type definition,文档类型定义)。

所有部署描述符文件的顶层(根)元素为 web-app。请注意,XML 元素是大小写敏感的,因此,web-App 和 WEB-APP 都是不合法的,web-app 必须用小写。

2)元素说明

XML 元素不仅是大小写敏感的,而且它们还对出现在其中的元素的次序是敏感的。例如,XML 头必须是文件中的第一项,DOCTYPE 声明必须是第二项,而 web-app 元素必须是第三项。在 web-app 元素内,元素的次序也很重要。服务器不一定强制要求这种次序,但它们允许(实际上有些服务器就是这样做的)完全拒绝执行含有次序不正确的元素的 Web 应用。这表示使用非标准元素次序的 web. xml 文件是不可移植的。

表 2-2 按出现顺序的先后给出了所有可直接出现在 web-app 元素内的合法元素及其说明。例如,此列表说明 servlet 元素必须出现在所有 servlet-mapping 元素之前。

**注意**:所有这些元素都是可选的,因此,可以省略掉某一元素,但不能把它放在不正确的

位置。

表 2-2 元素说明

| 元素名称 | 说 明 |
| --- | --- |
| icon | icon 元素指出 IDE 和 GUI 工具用来表示 Web 应用的一个或两个图像文件的位置 |
| display-name | display-name 元素提供 GUI 工具可能会用来标记这个特定的 Web 应用的一个名称 |
| description | description 元素给出与此有关的说明性文本 |
| context-param | context-param 元素声明应用范围内的初始化参数 |
| filter | 过滤器元素将一个名字与一个实现 Javax. servlet. Filter 接口的类相关联 |
| filter-mapping | 一旦命名了一个过滤器，就要利用 filter-mapping 元素把它与一个或多个 servlet 或 JSP 页面相关联 |
| listener | servlet API 的 2. 3 版本增加了对事件监听程序的支持，事件监听程序在建立、修改和删除会话或 servlet 环境时得到通知。listener 元素指出事件监听程序类 |
| servlet | 在向 servlet 或 JSP 页面指定初始化参数或定制 URL 时，必须首先命名 servlet 或 JSP 页面。servlet 元素就是用来完成此项任务的 |
| servlet-mapping | 服务器一般为servlet 提供一个缺省的 URL，即 http://host/webAppPrefix/servlet/ServletName。但是，常常需要更改这个 URL，以便 servlet 可以访问初始化参数或更容易地处理相对 URL。在更改缺省 URL 时，使用 servlet-mapping 元素 |
| session-config | 如果某个会话在一定时间内未被访问，服务器可以抛弃它以节省内存。可通过使用 HttpSession 的 setMaxInactiveInterval 方法明确设置单个会话对象的超时值，或者可利用 session-config 元素指定缺省超时值 |
| mime-mapping | 定义某一扩展名和某一 MIME Type 进行对应，包含 extension 和 mime-type 两个子元素 |
| welcom-file-list | welcome-file-list 元素指示服务器在收到引用一个目录名而不是文件名的 URL 时，使用哪个文件 |
| error-page | error-page 元素使得在返回特定 HTTP 状态代码时，或者特定类型的异常被抛出时，能够指定将要显示的页面 |
| taglib | taglib 元素为标记库描述符文件(tag library descriptor file，TLD 文件)指定别名。此功能使用户能够更改 TLD 文件的位置，而不用编辑使用这些文件的 JSP 页面 |
| resource-env-ref | 声明与资源相关的一个管理对象 |
| resource-ref | 定义利用 JNDI 取得站台可利用的资源 |
| security-constraint | 指定应该保护的 URL，它与 login-config 元素联合使用 |
| login-config | 用 login-config 元素来指定服务器应该怎样给试图访问受保护页面的用户授权。它与 sercurity-constraint 元素联合使用 |

续表

| 元素名称 | 说　明 |
| --- | --- |
| security-role | security-role 元素给出安全角色的一个列表，这些角色将出现在 servlet 元素内的 security-role-ref 元素的 role-name 子元素中。单独声明角色可使高级 IDE 处理安全信息更为容易 |
| env-entry | env-entry 元素声明 Web 应用的环境项 |
| ejb-ref | ejb-ref 元素声明一个 EJB 的主目录的引用 |
| ejb-local-ref | ejb-local-ref 元素声明一个 EJB 的本地主目录的应用 |

3)分配名称

为了提供初始化参数，对 servlet 或 JSP 页面定义一个定制 URL 或者分配一个安全角色，必须首先给 servlet 或 JSP 页面一个名称。可通过 servlet 元素分配一个名称。最常见的方法是指定 servlet-name 和 servlet-class 子元素(在 web-app 元素内)，如下所示：

```
<servlet>
    <servlet-name>Test</servlet-name>
    <servlet-class>moreservlets.TestServlet</servlet-class>
</servlet>
```

这表示位于 WEB-INF/classes/moreservlets/TestServlet 的 servlet 已经得到了注册名 Test。给 servlet 一个名称具有两个主要的含义：首先，初始化参数、定制的 URL 模式以及其他定制可以通过此注册名而不是类名引用此 servlet；其次，可在 URL 而不是类名中使用此名称。因此，利用刚才给出的定义，URL 地址 http://host/webAppPrefix/servlet/Test 与 http://host/webAppPrefix/servlet/moreservlets.TestServlet 是等价的。

4)示例

对于第 1 章中的 JSP 程序，可以建立内容如下所示的 web.xml 文件：

```
<? xml version="1.0" encoding="ISO-8859-1"? >
<! DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>My JSP Web Application</display-name>
    <description>The first JSP program for test.</description>
</web-app>
```

## 2.2　Java 语言基础

### 2.2.1　面向对象程序设计

面向对象程序设计是软件设计和实现的有效方法，这种方法可以提供软件的可扩充性和可重用性。客观世界中的一个事物就是一个对象，每个客观事物都有自己的特征和行为。

从程序设计的角度来看，事物的特性就是数据，行为就是方法。一个事物的特性和行为可以传给另一个事物，这样就可以重复使用已有的特性或行为。当某一个事物得到了其他事物传给它的特性和行为，再添加上自己的特性和行为，就实现了对已有的功能的扩充。面向对象的程序设计方法就是利用客观事物的这种特点，将客观事物抽象成为“类”，并通过类的“继承”实现软件的可扩充性和可重用性。

### 1. 类的基本概念

Java 语言与其他面向对象语言一样，引入了类和对象的概念，类是用来创建对象的模板，它包含被创建的对象的状态描述和方法的定义。因此，要学习 Java 编程就必须学会怎样去编写类，即怎样用 Java 的语法去描述一类事物共有的属性和行为。属性通过变量来刻画，行为通过方法来体现，即方法操作属性形成一定的算法来实现一个具体的功能。类把数据和对数据的操作封装成一个整体。类的声明方式如下所示：

```
[修饰符] class <类名> [extends 父类名] [implements 接口列表]
{
  类体
}
```

各部分说明如下：

- 修饰符：可选参数，用于指定类的访问权限，可选值为 public、abstract 和 final。
- 类名：必选参数，用于指定类的名称，类名必须是合法的 Java 标识符。一般情况下，要求首字母大写。
- extends 父类名：可选参数，用于指定要定义的类继承于哪个父类。当使用 extends 关键字时，父类名为必选参数。
- implements 接口列表：可选参数，用于指定该类实现的是哪些接口。当使用 implements 关键字时，接口列表为必选参数。
- 类体：在类声明部分的大括号中的内容为类体。类体主要由两部分构成，一部分是成员变量的定义，另一部分是成员方法的定义。

### 2. 定义成员方法

Java 中类的行为由类的成员方法来实现。类的成员方法由方法的声明和方法体两部分组成，其一般格式如下：

```
[修饰符] <方法返回值的类型> <方法名>([参数列表]) {
    [方法体]
}
```

各部分说明如下：

- 修饰符：可选参数，用于指定方法的被访问权限，可选值为 public、protected 和 private。
- 方法返回值的类型：必选参数，用于指定方法的返回值类型，如果该方法没有返回值，可以使用关键字 void 进行标识。方法返回值的类型可以是任何 Java 数据类型。
- 方法名：必选参数，用于指定成员方法的名称，方法名必须是合法的 Java 标识符。
- 参数列表：可选参数，用于指定方法中所需的参数。当存在多个参数时，各参数之间应使用逗号分隔。方法的参数的类型可以是任何 Java 数据类型。

- 方法体：可选参数，方法体是方法的实现部分，在方法体中可以定义局部变量。需要注意的是，当方法体省略时，其外面的大括号一定不能省略。

### 3. 成员变量与局部变量

在类体中的变量定义部分所声明的变量为类的成员变量，而在方法体中声明的变量和方法则称为局部变量。成员变量和局部变量的区别在于其有效范围不同。成员变量在整个类内都有效，而局部变量只在定义它的成员方法内才有效。

(1)声明成员变量：Java 用成员变量来表示类的状态和属性，声明成员变量的基本语法格式如下：

[修饰符] [static] [final] [transient] [volatile] ＜变量类型＞ ＜变量名＞；

各部分说明如下：

- 修饰符：可选参数，用于指定变量的被访问权限，可选值为 public、protected 和 private。
- static：可选参数，用于指定该成员变量为静态变量，可以直接通过类名访问。如果省略该关键字，则表示该成员变量为实例变量。
- final：编译器将定义为 final 的变量当成一个常量，即按只读变量来初始化和处理该变量。因为编译器知道常量是不变的，所以在程序的字节码中对其进行了内部优化。
- transient：定义为 transient 的字段值在对象串行化过程中将不被保存。
- volatile：如果字段声明为 volatile，则多线程将能访问此字段，而特定的编译器将防止最优化以使该字段能被适当地访问。
- 变量类型：必选参数，用于指定变量的数据类型，其值的类型为 Java 中的任何一种数据类型。
- 变量名：必选参数，用于指定局部变量的名称，变量名必须是合法的 Java 标识符。

(2)声明局部变量：声明局部变量的基本语法格式同定义成员变量类似，所不同的是不能使用 public、protected、private 和 static 关键字对局部变量进行修饰，但可以使用 final 关键字。语法格式如下：

[final] ＜变量类型＞ ＜变量名＞；

### 4. 构造方法的概念及用途

构造方法是一种特殊的方法，它的名字必须与它所在类的名字完全相同，并且没有返回值，也不需要使用关键字 void 进行标识。构造方法用于对对象中的所有成员变量进行初始化，在创建对象时被立即调用。需要注意的是，如果用户没有定义构造方法，Java 会自动提供一个默认的构造方法，用来实现成员变量的初始化。

### 5. 创建 Java 类对象

在 Java 中，创建对象的过程包括声明对象和为对象分配内存两部分，下面分别进行介绍。

1)声明对象

对象是类的实例，属于某个已经声明的类。因此，在对对象进行声明之前，一定要先定义该对象的类。声明对象的一般格式如下：

*类名　对象名*；

各部分说明如下：

- 类名：必选项，用于指定一个已经定义的类。

• 对象名：必选项，用于指定对象名称，对象名必须是合法的 Java 标识符。

例如，声明 Fruit 类的一个对象 fruit 的代码如下：

```
Fruit fruit;
```

在声明对象时，只是在内存中为其建立一个引用，并将其初值置为 NULL，表示不指向任何内存空间，因此，还需要为对象分配内存。

2)为对象分配内存

为对象分配内存也称为实例化对象。在 Java 中使用关键字 new 来实例化对象，具体语法格式如下：

```
对象名=new 构造方法名([参数列表]);
```

各部分说明如下：

• 对象名：必选，用于指定已经声明的对象名。

• 构造方法名：必选项，用于指定构造方法名，即类名，因为构造方法与类名相同。

• 参数列表：可选参数，用于指定构造方法的入口参数。如果构造方法无参数，则可以省略。

例如，在声明 Fruit 类的一个对象 fruit 后，可以通过以下代码为对象 fruit 分配内存：

```
fruit=new Fruit();
```

在上面的代码中，由于 Fruit 类的构造方法无入口参数，所以省略了参数列表。

在声明对象时，也可直接为其分配内存。例如，上面的声明对象和为对象分配内存的过程也可以通过以下代码一次性实现：

```
Fruit fruit=new Fruit();
```

### 6. 对象的使用

创建对象后，可以通过对象来引用其成员变量，并改变成员变量的值，并且还可以通过对象来调用其成员方法。通过使用运算符“.”实现对成员变量的访问和成员方法的调用。常见格式如下：

```
对象名.成员变量名
对象名.成员方法名([<参数列表>])
```

### 7. 对象的销毁

在许多程序设计语言中，需要手动释放对象所占用的内存，但是，在 Java 中则不需要手动完成这项工作。Java 提供的垃圾回收机制可以自动判断对象是否还在使用，并能够自动销毁不再使用的对象，收回对象所占用的资源。

Java 提供了一个名为 finalize()的析构方法，用于在对象被垃圾回收机制销毁之前，由垃圾回收系统调用。由于垃圾回收系统的运行是不可预测的。因此，在 Java 程序中，也可以使用析构方法 finalize()随时来销毁一个对象。析构方法 finalize()没有任何参数和返回值，每个类有且只有一个析构方法。

### 8. 接口

在一些面向对象编程语言中，多继承的机制允许一个类从不同的类继承方法和属性。Java 语言不支持多继承，但是通过引入接口概念实现了从多方面继承方法的功能。

接口是一个完全抽象的类，在其中能定义数据成员，但这些数据成员都必须是常量。抽象类中可以包含抽象方法和非抽象方法，而在接口中就只能定义抽象方法。

Java 中接口的定义形式如下：

```
[修饰词] interface <接口名>
{
    常量声明;
    方法声明;
}
```

接口中方法的修饰词只能是 public，默认也是 public。含义与类修饰符相同。接口名的命名的规则也与类名相同。接口中变量的修饰词只能是 public、final、static，所以不显式地使用修饰词在接口中声明的都是常量。接口中的方法都没有方法体，除了定义的常量以外也没有变量。

接口只是声明了提供的功能和服务，而功能和具体服务的实现还是要在实现接口的类中定义。实现接口的类时，必须实现接口声明的所有方法。

实现一个接口的类时，注意事项如下：

(1)在类的声明部分，用 implements 关键字声明该类将要实现哪个接口。

(2)当实现某接口的类是 abstract 的抽象类时，它可以不实现该接口所有的方法。但是对于这个抽象类的任何一个非抽象子类而言，它们父类所实现的接口中的所有抽象方法都必须有实在的方法体。这些方法体可以来自子类自身，也可以来自抽象的父类，但是不允许存在未被实现的接口方法，这是非抽象类中不能存在抽象方法的原则体现。

(3)当实现某接口的类不是 abstract 抽象类时，在类的定义部分必须实现指定接口的所有抽象方法，即为所有抽象方法定义方法体，而且方法头部分应该与接口中的定义完全一致，即所有返回值和参数列表应一致。

(4)当一个类实现某接口的抽象方法时，必须使方法名完全一样。如果所实现的方法与抽象方法方法名相同，但参数列表不同，此时不是实现已有的抽象方法，而是在重载一个新方法。

(5)类在实现方法时，必须显式地使用 public 修饰符，否则将被系统警告为缩小了接口中定义的方法的访问控制范围。这是因为接口抽象方法的访问限制符都已指定为public。

**9. 包的使用**

公共资源可以重用是面向对象程序设计的另一个特点。在 Java 语言中，当应用软件比较大时，如果将这些 Java 文件放在一个文件夹中，管理起来就比较麻烦，以后的软件资源重用也很不方便。通过使用包，Java 可以解决此类问题。

包(package)是 Java 提供的一种区别类的名字空间的机制，是类的组织方式，是一组相关类和接口的集合，它提供了访问权限和命名的管理机制。Java 中提供的包主要有以下三种用途：

(1)将功能相近的类放在同一个包中，可以方便查找与使用。

(2)由于在不同包中可以存在同名类，所以使用包在一定程度上可以避免命名冲突。

(3)在 Java 中，某些访问权限是以包为单位的。

1)创建包

可以通过在类或接口的源文件中使用 package 语句创建包，package 语句的语法格式如下：

```
package 包名;
```

包的名称为合法的 Java 标识符。当包中还有包时，可以使用“包 1. 包 2. …. 包 $n$”进行指定，其中，包 1 为最外层的包，而包 $n$ 则为最内层的包。

package 语句通常位于类或接口源文件的第一行，而且其前面不能有注释和空格。例如，定义一个类 SimpleH，将其放入 com. wgh 包中的代码如下：

```
package com.wgh;
public class SimpleH{
    ……  //此处省略了类体的代码
}
```

编译后，生成的 SimpleH. class 文件将导入到包 com. wgh 包中。

2)使用包中的类

类可以访问其所在包中的所有类，还可以使用其他包中的所有 public 类。访问其他包中的 public 类可以有以下两种方法：

(1)使用长名引用包中的类。使用长名引用包中的类比较简单，只需要在每个类名前面简单地加上完整的包名即可。例如，创建 Circ 类(保存在 com. wgh 包中)的对象并实例化该对象的代码如下：

```
com.wgh.Circ circ=new com.wgh.Circ();
```

(2)使用 import 语句引入包中的类。由于采用使用长名引用包中的类的方法比较繁琐，所以 Java 提供了 import 语句来引入包中的类。import 语句的基本语法格式如下：

```
import 包名 1[.包名 2.…].类名|*;
```

当存在多个包名时，各个包名之间使用“.”分隔，同时包名与类名之间也使用“.”分隔。

使用“*”则表示引用包中所有的类。

例如，引入 com. wgh 包中的 Circ 类的代码如下：

```
import com.wgh.Circ;
```

如果 com. wgh 包中包含多个类，也可以使用以下语句引入该包下的全部类：

```
import com.wgh.*;
```

### 2.2.2 基本数据类型

#### 1. 基本数据类型概述

Java 基本数据类型主要包括整型、浮点型、字符型和布尔型。其中整型又分为字节型(byte)、短整型(short)、整型(int)和长整型(long)，它们都可以用来定义一个整数，唯一的区别就是它们所定义的整数所占用内存的空间不同，因此整数的取值范围也不同。Java 中的浮点型又包括单精度型(float)和双精度型(double)，在程序中使用这两种类型来存储小数。

Java 中的各种基本数据类型及它们的取值范围和占用的内存大小如表 2-3 所示。

**表 2-3 基本数据类型**

| 数据类型 | | 占用内存 | 取值范围整数 |
|---|---|---|---|
| 整数型 | 字节型(byte) | 8 位 | −128～127 |
| | 短整型(short) | 16 位 | −32768～32767 |
| | 整型(int) | 32 位 | −2147483648～2147483647 |
| | 长整型(long) | 64 位 | −9223372036854775808～9223372036854775807 |

续表

| 数据类型 | | 占用内存 | 取值范围整数 |
|---|---|---|---|
| 浮点型 | 单精度型(float) | 32 位 | 1.4E－45～3.4028235E38 |
| | 双精度型(double) | 64 位 | 4.9E－324～1.7976931348623157E308 |
| 字符型 | 字符型(char) | 16 位 | 16 位的 Unicode 字符，可容纳各国的字符集；若以 Unicode 来看，就是'\u0000'到'\uufff'；若以整数来看，范围在 0～65535，例如，65 代表 A |
| 布尔型 | 布尔型(boolean) | 8 位 | true 和 false |

**2.基本数据类型间的转换**

在 Java 语言中，当多个不同基本数据类型的数据进行混合运算时，如整型、浮点型和字符型进行混合运算，需要先将它们转换为统一的数据类型，然后再进行计算。在 Java 中，基本数据类型之间的转换可分为自动类型转换和强制类型转换两种。

1)自动类型转换

在程序运行时，每种数据类型所占的空间不一样，这就使得每种数据类型所容纳的信息量也不一样。当一个容量较小的类型转化为一个容量较大的类型时，数据本身的信息不会丢失，所以它是安全的，这时编译器会自动完成类型转换的工作，这种转换称为自动类型转换。这种转换将由系统按照各数据类型的级别从低到高自动完成，Java 编程人员无须进行任何操作。各数据类型按级别由低到高的顺序如下所示：

```
byte,short,char→int→long→float→double
```

例如：

```
byte b=3;
short s=4;
int i=b+s;
```

在对 byte 和 short 类型的数据组成的表达式求值时，Java 会自动将其转化为 int 整型数，再进行计算。

一般情况下，将一种类型的数据赋给另一种类型的变量时，如果下列两个条件都能满足，那么将执行自动类型转换：

(1)目的类型数的范围比来源类型的大。

(2)这两种类型是兼容的。

2)强制类型转换

如果把高级数据类型数据赋值给低级类型变量，就必须进行强制类型转换，否则编译会出错。强制类型转换格式如下：

```
(欲转换成的数据类型)值
```

其中“值”可以是常量或者变量，例如：

```
int  s1=65,s2;
char  c1='a',c2;
s2=(int)c1;  //将 char 型强制转换为 int 型,s2 值为 97
```

```
c2=(char)s1;  //将 int 型强制转换为 char 型,c2 值为:A
```

强制类型转换实际上是一种显式的类型转换。要完成两种不兼容的类型之间的转换，必须进行强制类型转换。

### 2.2.3 常量

常量是指在程序运行过程中不能被修改的量。Java 中常用的常量有数值常量、字符常量和布尔常量。

**1. 数值常量**

在数值类型中，常量可以是整型、字节型和浮点型等，它们都可以采用十进制、八进制和十六进制表示。

**2. 字符常量**

字符常量代表一个字符，Java 用 Unicode 字符集来表示字符。

**3. 布尔常量**

布尔常量只有两种值：true 和 false。

在 Java 中，也可以用 final 关键字来定义常量。通常情况下，在通过 final 关键字定义常量时，常量名全部为大写字母。需要说明的是，由于常量在程序执行过程中保持不变，所以在常量定义后，如果再次对该常量进行赋值，程序将会出错。

### 2.2.4 变量

Java 中用变量(variable)来存放经常变化的数据，也就是说，变量存储的是在程序中可以修改的值。变量是 Java 程序中的基本存储单元，它的定义包括变量名、变量类型和作用域几个部分。

**1. 变量的声明**

使用变量之前必须进行声明，声明变量的作用有两个：一是给该变量分配内存空间，二是为了防止因错误输入而对不存在的变量进行操作。变量声明过程如下：

```
type identifier [,identifier];
```

其含义是以 identifier 为名建立一个 type 类型的变量，方括号表示可选，即一条语句中可以定义多个同类型的变量，中间用逗号隔开，分号表示声明语句结束。

Java 中的变量命名时，需要注意以下问题：

(1)必须不是一个布尔型字符(true 或者 false)、关键字或者保留字 NULL。

(2)必须在作用域中是唯一的，但在不同的作用域允许存在相同名字的变量。

(3)必须是一个合法的标识符。一个标识符中间不能包含空格，必须是以下划线、字母或 $ 符号开头的一串 Unicode 字符。

(4)Java 对变量名区分大小写。例如，Name 和 NAME 是两个不同的变量。

合法的变量名如：value_1、Name、dollar $ 等，非法的变量名如：2tool、test # 、class(保留字)等。

变量名最好有一定的含义，以增加程序的可读性。

表 2-4 中的标识符被称为关键字，它们被 Java 系统保留使用。其中，加“*”标记的是被

Java 保留但当前却未使用的。需要注意的是，这些关键字不能作为变量名或其他用途。

表 2-4　Java 关键字

| | | | | |
|---|---|---|---|---|
| abstract | continue | for | new | switch |
| boolean | default | goto* | null | synchronized |
| break | do | if | package | this |
| byte | double | implements | private | threadsafe |
| byvalve* | else | import | protected | throw |
| case | extends | instanceof | public | transient |
| catch | false | int | return | true |
| char | final | interface | short | try |
| class | finally | long | static | void |
| const* | float | native | super | while |

**2. 变量的作用域**

变量作用域取决于变量何时创建或取消，可以在声明特定变量时建立相应的作用域。变量作用域是一段特定的代码，只有在这段代码中才能访问该变量。变量作用域的主要类型如下：

(1)成员变量作用域。

(2)本地变量作用域。

(3)方法参数作用域。

(4)异常处理参数作用域。

作用域可以嵌套。内部作用域的变量对外部是不可见的，但外部作用域的变量对于内部作用域是可见的。

本地变量是在一个方法或一个方法的某个代码块内声明的变量。一般来说，一个本地变量从它的声明部分开始到代码块结束都可以被访问。它们是在主方法中声明的，所以只能在主方法中应用。

成员变量不能在某一个类的方法里声明成员变量，它是在类中声明的类或对象的成员。在一个类的某个方法里声明一个变量时，只能在该方法中访问这个变量。也就是说，该变量被看成是这个方法的本地变量。类中的静态方法只能访问静态变量，而不能访问非静态变量。

### 2.2.5　运算符

在 Java 语言中表达各种运算的符号叫做运算符。Java 运算符主要包括赋值运算符、算术运算符、关系运算符、逻辑运算符、位运算符、条件运算符，下面分别进行介绍。

**1. 算术运算符**

Java 中的算术运算符包括：+(加号)、-减号、*(乘号)、/(除号)、%(求余)以及递增递减运算符。算术运算符支持整型和浮点型数据的运算，当整型与浮点型数据进行算术运算

时，会进行自动类型转换，结果为浮点型。

Java 中的算术运算符如表 2-5 所示。

表 2-5 Java 中的算术运算符

| 运 算 符 | 说 明 | 举 例 | 结果及类型 |
|---|---|---|---|
| + | 加法 | 1.23f+10 | 结果为 11.23，类型为 float |
| - | 减法 | 4.56-0.5d | 结果为 4.06，类型为 double |
| * | 乘法 | 3 * 9L | 结果为 27，类型为 long |
| / | 除法 | 9/4 | 结果为 2，类型为 int |
| % | 求余数 | 10%3 | 结果为 1，类型为 int |
| ++ | 递增 | a++ | 假设 a 的值为 3，则运行结果为 4 |
| -- | 递减 | a-- | 假设 a 的值为 3，则运行结果为 2 |

### 2. 赋值运算符

Java 中的赋值运算符可以分为简单赋值运算和复合赋值运算。简单赋值运算是将赋值运算符(=)右边的表达式的值保存到赋值运算符左边的变量中，复合赋值运算是混合了其他操作(算术运算操作、位操作等)和赋值操作，例如：

```
sum+=i;     //等同于 sum=sum+i;
```

Java 中的赋值运算符如表 2-6 所示。

表 2-6 Java 中的赋值运算符

| 运 算 符 | 说 明 | 运 算 符 | 说 明 |
|---|---|---|---|
| = | 简单赋值 | &= | 进行与运算后赋值 |
| += | 相加后赋值 | \|= | 进行或运算后赋值 |
| -= | 相减后赋值 | ^= | 进行异或运算后赋值 |
| *= | 相乘后赋值 | >>= | 带符号右移后赋值 |
| /= | 相除后赋值 | <<= | 左移之后赋值 |
| %= | 求余后赋值 | >>>= | 填充零右移后赋值 |

### 3. 关系运算符

通过关系运算符计算的结果是一个 boolean 类型值。对于应用关系运算符的表达式，计算机将判断运算对象之间通过关系运算符指定的关系是否成立，若成立则表达式的返回值为 true，否则为 false。

Java 中的关系运算符如表 2-7 所示。

表 2-7 Java 中的关系运算符

| 运 算 符 | 说 明 | 运 算 符 | 说 明 |
|---|---|---|---|
| > | 大于 | <= | 小于等于 |
| < | 小于 | == | 等于 |
| >= | 大于等于 | != | 不等于 |

其中等于和不等于运算符适用于引用类型和所有的基本数据类型，而其他的关系运算符只适用于除 boolean 类型外的所有基本数据类型。

### 4. 逻辑运算符

逻辑运算符经常用来连接关系表达式，对关系表达式的值进行逻辑运算，因此逻辑运算符的运算对象必须是逻辑型数据，其逻辑表达式的运行结果也是逻辑型数据。Java 中的逻辑运算符如表 2-8 所示。

**表 2-8　Java 中的逻辑运算符**

| 运 算 符 | 意　　义 | 运算结果 |
|---|---|---|
| & | 逻辑与 | true&true 结果为 true；false&false 结果为 false；true&false 结果为 false |
| \| | 逻辑或 | true\|true 结果为 true；false\|false 结果为 false；true\|false 结果为 true |
| ^ | 异或 | true^true 结果为 false；false^false 结果为 false；true^false 结果为 true |
| \|\| | 短路或 | true\|\|true 结果为 true；false\|\|false 结果为 false；true\|\|false 结果为 true |
| && | 短路与 | true&&true 结果为 true；false&&false 结果为 false；true&&false 结果为 false |
| ! | 逻辑反 | !true 结果为 false；!false 结果为 true |

### 5. 位运算符

位运算符用于对数值的位进行操作，参与运算的操作数只能是 int 或 long 类型。在不产生溢出的情况下，左移一位相当于乘以 2，用左移实现乘法运算的速度比普通的乘法运算速度快。Java 中的位运算符如表 2-9 所示。

**表 2-9　Java 中的位运算符**

| 运 算 符 | 说　　明 | 实　　例 |
|---|---|---|
| ～ | 进行数值的相反数减 1 运算 | ～50＝－50－1＝ －51 |
| >> | 向右移位 | 15 >> 1 ＝ 7 |
| << | 向左移位 | 15 << 1 ＝ 30 |
| >>> | 无符号向右移位 | 15 >>> 1 ＝ 7 |

### 6. 条件运算符

条件运算符是三元运算符，其语法格式如下：

布尔表达式？表达式 1：表达式 2

如果布尔表达式的结果为 true，则计算表达式 1，而且它的结果将成为最终运算符产生的值，否则计算表达式 2。

### 7. 运算符优先级

对一个表达式进行运算时，要按照运算符的优先级顺序从高到低进行运算，运算符的优先级是指一个表达式中多个运算符被执行的顺序。

Java 语言中的运算符的优先级与 C++ 语言基本上相同，圆括号内的优先级最高，赋值

运算符的优先级最低。

表达式在运算时完成一个或多个操作，最终得到一个运算结果，结果的数据类型由参加运算的数据的类型来决定。

对复杂表达式进行运算时，要按照运算符的优先级顺序从高到低进行，同级的运算符则按从左到右的方向进行。表 2-10 列出了 Java 中运算符的优先级顺序。

**表 2-10 Java 中运算符的优先级顺序**

| 优先级顺序 | 运 算 符 | 优先级顺序 | 运 算 符 |
|---|---|---|---|
| 1 | [],() | 10 | & |
| 2 | . | 11 | ^ |
| 3 | ++,--,!,~ | 12 | \| |
| 4 | New (type) | 13 | && |
| 5 | *,/,% | 14 | \|\| |
| 6 | +,- | 15 | ?: |
| 7 | >>,<<,>>> | 16 | =,+=,-=,*=,/=,%=,^= |
| 8 | >,<,>=,<=,instanceof | 17 | &=,\|=,<<=,>>=,>>>= |
| 9 | ==,!= | | |

### 2.2.6 流程控制语句

Java 语言中，流程控制语句主要有分支语句、循环语句和跳转语句三大类种。

**1. 分支语句**

所谓分支语句，就是对语句中不同条件的值进行判断，进而根据不同的条件执行不同的语句。在分支语句中主要有 if 条件语句和 switch 多分支语句两种。

1)if 语句

if 语句是条件语句最常用的一种形式，它针对某种条件有选择地做出处理。通常表现为“如果满足某种条件，就进行某种处理，否则就进行另一种处理”。其语法格式如下：

```
if(条件表达式){
    语句序列 1
}
else{
    语句序列 2
}
```

条件表达式是必要参数，它可以由多个表达式组成，但是其最后结果一定是 boolean 类型的值，也就是其结果只能是 true 或 false。

语句序列 1 为可选参数，它可以是一条或多条语句，当表达式的值为 true 时执行这些语句。

语句序列 2 也为可选参数，它也可以是一条或多条语句，当表达式的值为 false 时执行这些语句。

2)switch 语句

switch 语句是多分支选择语句,常用来根据表达式的值选择要执行的语句。switch 语句的基本语法格式如下:

```
switch(表达式){
    case 常量表达式 1: 语句序列 1
      [break;]
    case 常量表达式 2: 语句序列 2
      [break;]
    ……
    case 常量表达式 n: 语句序列 n
      [break;]
    default: 语句序列 n+1
      [break;]
}
```

其中,case 分支的常量表达式的类型应与表达式的值的类型一致。将表达式的值计算出来后,先与第一个 case 分支的常量表达式相比较,若相等,则执行语句序列 1,若不相同,则转入第二个 case 分支中,与常量表达式 2 相比较,以此类推。如果表达式的值与任何一个 case 分支的值都不相等,则会执行最后的 default 分支,若 default 分支不存在,则跳出整个 switch 语句。

**2.循环语句**

所谓循环语句,主要就是在满足条件的情况下反复执行某一个操作。在 Java 中,提供了三种常用的循环语句,分别是 for 循环语句、while 循环语句和 do…while 循环语句。

1)for 循环语句

for 循环语句也称为计次循环语句,一般用于循环次数已知的情况。for 循环语句的基本语法格式如下:

```
for(初始化语句;循环条件;迭代语句){
    语句序列
}
```

初始化语句是为循环变量赋初始值的语句,该语句在整个循环语句中只执行一次。循环条件是决定是否进行循环的表达式,其结果为 boolean 类型,也就是其结果只能是 true 或 false。迭代语句是用于改变循环变量的值的语句。语句序列即循环体,在循环条件的结果为 true 时,重复执行。

for 循环语句执行的过程是:先执行为循环变量赋初始值的初始化语句,然后判断循环条件,如果循环条件的结果为 true,则执行一次循环体,否则直接退出循环,最后执行迭代语句,改变循环变量的值,至此完成一次循环,接下来将进行下一次循环,直到循环条件的结果为 false,才结束循环。

2)while 循环语句

while 循环语句也称为前测试循环语句,它的循环重复执行方式是利用一个条件来控制是否要继续重复执行这个语句。while 循环语句与 for 循环语句相比,无论是语法还是执行的流程,都更为简明易懂。while 循环语句的基本语法格式如下:

```
while(条件表达式){
    语句序列
}
```

条件表达式是决定是否进行循环的表达式，其结果为 boolean 类型，也就是其结果只能是 true 或 false。语句序列即循环体，在条件表达式的结果为 true 时，重复执行。

while 循环语句执行的过程是：先判断条件表达式，如果条件表达式的值为 true，则执行循环体，并且在循环体执行完毕后，进入下一次循环，否则退出循环。

3）do…while 循环语句

do…while 循环语句也称为后测试循环语句，它的循环重复执行方式也是利用一个条件来控制是否要继续重复执行这个语句。与 while 循环所不同的是，它先执行一次循环语句，然后再去判断是否继续执行。do…while 循环语句的基本语法格式如下：

```
do{
    语句序列
} while(条件表达式);
```

语句序列在循环开始时首先被执行一次，然后再判断条件表达式的值是否为 true。当条件表达式的结果为 true 时，重复执行。条件表达式决定是否进行循环的表达式，其结果为 boolean 类型，也就是其结果只能是 true 或 false。

**3. 跳转语句**

Java 语言中提供了三种跳转语句，分别是 break 语句、continue 语句和 return 语句。

1）break 跳转语句

break 语句可以应用在 for、while 和 do…while 循环语句中，用于强行退出循环，也就是忽略循环体中任何其他语句和循环条件的限制。

2）continue 跳转语句

continue 语句只能应用在 for、while 和 do…while 循环语句中，用于让程序直接跳过其后面的语句，进行下一次循环。

3）return 跳转语句

return 语句用于从当前方法中返回到调用该方法的语句处，并从该语句的下一条语句继续程序的执行。其语法格式如下：

```
return [表达式];
```

表达式表示要返回的值，它的数据类型必须同方法声明中的返回值类型一致，这可以通过强制类型转换实现。

return 语句通常被放在被调用方法的最后，用于退出当前方法并返回一个值。当把单独的 return 语句放在一个方法的中间时，会产生“Unreachable code”的编译错误。但是可以通过把 return 语句用 if 语句括起来的方法，将 return 语句放在一个方法中间，用来实现在程序未执行完方法中的全部语句时退出。

## 2.2.7 数组

数组是由多个元素组成的，每个单独的数组元素就相当于一个变量，可用来保存数据，因此可以将数组视为一连串变量的组合。根据数组存放元素的复杂程度，可将数组依次分

为一维数组、二维数组及多维(三维以上)数组。

### 1. 一维数组

Java 中的数组必须先声明,然后才能使用。声明一维数组有以下两种格式:

数据类型 数组名[] = new 数据类型[个数];

数据类型[] 数组名 = new 数据类型[个数];

当按照上述格式声明数组后,系统会分配一块连续的内存空间供该数组使用,例如,下面的两行代码都是正确的:

```
String any[] = new String[10];
String[] any = new String[10];
```

这两个语句实现的功能都是创建了一个新的字符串数组,它有 10 个元素,可以用来容纳 String 对象。需要注意,当用关键字 new 来创建一个数组对象时,必须指定这个数组能容纳多少个元素。

一维数组的赋值语法格式如下:

数据类型 数组名[] = {数值 1,数值 2,…,数值 $n$};

数据类型[] 数组名 = {数值 1,数值 2,…,数值 $n$};

括号内的数值将依次赋值给数组中的第 1 到 $n$ 个元素。另外,在赋值声明时,不需要给出数组的长度,编译器会按所给的数值个数来决定数组的长度,例如下面的代码:

```
String type[] = {"乒乓球","篮球","羽毛球","排球","网球"};
```

在上面的语句中,声明了一个数组 type,虽然没有特别指名 type 的长度,但由于括号里的数值有五个,编译器会分别依次为各元素指定存放位置,例如,type[0] ="乒乓球",type[1]="篮球"。

### 2. 二维数组

在 Java 语言中,实际上并不存在称为“二维数组”的明确结构,而二维数组实际上是指数组元素为一维数组的一维数组。声明二维数组的语法格式如下:

数据类型 数组名[][] = new 数据类型[个数][个数];

如下面的代码:

```
int arry[][] = new int [5][6];
```

上述语句声明了一个二维数组,其中 5 和 6 表示该数组有 5(0～4)行,每行有 6(0～5)个元素,因此该数组有 30 个元素。

对于二维数组元素的赋值,同样可以在声明时进行,例如:

```
int number[][] = {{20,25,26,22},{22,23,25,28}};
```

在上面的语句中,声明了一个整型的 2 行 4 列的数组,并在声明的同时进行了赋值,结果如下:

```
number[0][0] = 20; number[0][1] = 25;
number[0][2] = 26; number[0][3] = 22;
number[1][0] = 22; number[1][1] = 23;
number[1][2] = 25; number[1][3] = 28;
```

### 2.2.8 异常处理

Java 的异常处理机制是 Java 语言的一个很重要的特色。通过异常处理机制，可以预防程序代码错误或系统错误所造成的不可预期的情况的发生，使系统更健壮、更安全。Java 的异常处理机制也是面向对象的，当有一个异常发生时，程序生成一个异常对象，同时抛出一个异常。其核心思想是捕获预先判断可能发生错误的代码段可能发生的异常。

在 Java 语言中，处理异常的语句有四种：try…catch 语句、finally 语句、throw 语句及 throws 语句。

#### 1. try…catch 语句

在 Java 语言中，用 try…catch 语句来捕获异常，代码格式如下：

```
try{
  /*可能出现异常状况的代码*/
}catch(IOException e){
  /*处理输入输出出现的异常*/
}catch(SQLException e){
  /*处理操作数据库出现的异常*/
}
```

在上述代码中，try 语句块用来监视这段代码运行过程中是否发生异常，若发生则产生异常对象并抛出；catch 用于捕获异常并根据不同情况进行处理。

#### 2. finally 语句

由于异常程序将中断执行，这会使得某些不管在任何情况下都必须执行的步骤被忽略，从而影响程序的健壮性。使用 finally 语句的目的就在于不管捕获的异常是否出现，都强制执行 finally 代码块。

#### 3. throw 语句

当程序发生错误而无法处理时，会抛出相应的异常对象。除此之外，在某些代码中，可能会自行抛出异常。例如，在捕捉异常并处理结束后，再将异常抛出，让下一层异常处理区块来抛出，或者是重新包装异常，将捕捉到的异常以自己定义的异常对象加以包装抛出。如果要自行抛出异常，可以使用 throw 关键字，并生成指定的异常对象。其格式如下面的代码所示：

```
throw new ArithmeticException();
```

#### 4. throws 语句

一个方法可能会出现多种异常，throws 语句允许声明抛出多个异常。throws 语句代码格式如下：

```
返回类型方法名(参数表)throws 异常类型表{
    方法体
}
```

例如，下面的代码：

```
public void methodServlet ( int number ) throws NumberFormatException,
IOException{
…….
}
```

异常声明是接口（这里的接口是指概念上的程序接口）的一部分。根据异常声明，方法调用者了解到被调用方法可能抛出的异常，从而采取相应的措施：捕获异常并处理异常或者声明继续抛出异常。

如果不确定这个方法会抛出哪种异常，那么可以直接抛出 Exception 异常，例如，下面的代码：

```
public void methodServlet(int number) throws Exception{
…….
}
```

**注意**：throw 和 throws 关键字尽管只有一个字母之差，却有着不同的用途，注意不要将两者混淆。

## 2.3　JSP 与 Web 页面的交互

动态页面的一大特点就是它的交互性，因此，在进行设计开发时，实现客户端和服务器端的信息交互是其中不可缺少的一项内容，本节将在介绍表单相关知识的基础上讲解在 JSP 中如何实现对表单数据的获取和处理。

### 2.3.1　表单及表单的创建

在日常的网络信息浏览和网络应用开发中，经常会遇到信息交互的情况，如注册个人信息、在线调查等，图 2-2 就是一个邮箱注册页面的示例。

服务器端要实现与客户端的交互，首先就需要收集客户端的相关信息，表单是最常用的一种技术。表单从用户收集信息，然后将这些信息提交给服务器进行处理。表单可以包含允许与用户进行交互的各种控件，如文本框、列表框、复选框和单选按钮等。交互过程如下：

(1)用户填写表单。

(2)用户在填好表单之后提交表单，送出所输入的数据。

(3)表单中的信息被发送到服务器。

(4)利用表单处理程序对表单数据进行处理。

(5)服务器返回处理结果，完成整个交互过程。

#### 1. HTML 表单的运行原理

客户端的浏览器使用 HTTP（hypertext transfer protocol）协议从服务器获取 HTML 页面，HTTP 协议规定了信息在 Internet 上的传输方法，特别规定了浏览器与服务器的交互方法。


图 2-2 邮箱注册页面

在浏览器从服务器获取页面的过程中，浏览器在网站上打开了一个对 Internet 服务器的连接并发出请求，服务器收到请求后给予回应，因此，HTTP 协议又被称为请求和响应协议。

所有的浏览器和服务器间的通信都发生在离散的请求—响应对中，浏览器必须发出请求才可能开始通信，服务器角色完全是被动的，它在被请求后才能起作用。

一个典型请求通常包含许多头，头提供了关于消息内容的附加信息以及请求的来源，其中有些头是标准的，其他的是有关特定浏览器的。

一个请求还包含信息体，例如，若请求用 POST 方法而不是 GET 方法发送，信息体可以包含 HTML 表单的内容，在 HTML 表单上单击 submit 键时，该表单使用 ACTION = "POST"特征，输入表单的内容都被发送到服务器上，该表单的内容就由 POST 方法在请求的信息体中发送。

服务器收到请求时返回 HTTP 响应，每个响应都由状态行开始，可以包含几个头及可能的信息体。例如，对页面的请求如下：

```
GET/index.html HTTP/1.1
HOST:www.edu.cn
```

上例中使用的请求方法是 GET 方法，此方法可以获取特定的资源，此处，GET 方法用来获取名为 index.html 的网页。其他请求方法包括 POST、HEAD、DELETE、TRACE 及 PUT 方法等。例如，可以使用 POST 方法用来提交 HTML 表单的内容。

第二行是头(header)，HOST 头规定了在网站上 index.html 文件的 Internet 地址，此例

中主机是 www. edu. cn。

### 2. 创建表单

在 HTML 语言中，表单通过 FORM 标记来定义，其基本语法格式如下：

```
<FORM name = "字符串" method = "GET|POST" action="字符串"onSubmit=""
onReset="" TARGET="" enctype="">
    ……
</FORM>
```

FORM 标记具有以下属性：

(1)name：指定表单的名称，以标识表单。表单命名后，可以使用脚本语言(如 VBScript 或 Java Script)来引用或控制该表单。

(2)method：指定将表单数据传输到服务器的方法，其取值可以是 POST 或 GET。

(3)action：指定将要接收表单数据的服务器端程序或动态页面的网址。

(4)onSubmit：指定提交表单时调用的事件处理程序。

(5)onReset：指定重置表单时调用的事件处理程序。

(6)target：指定一个目标窗口，其取值可以为如下四个值的一个：

- _blank：在未命名的新窗口中打开目标文档。
- _parent：在显示当前文档的窗口的父窗口中打开目标文档。
- _self：在提交表单所使用的窗口中打开目标文档。
- _top：在当前窗口内打开目标文档，确保目标文档占用整个窗口。

(7)enctype：编码方式。

### 3. 表单控件及其添加方法

为了让用户通过表单输入数据，可以使用 input 标记创建各种输入型表单控件。通过将 input 标记的 TYPE 属性设置为不同的值，可以创建不同类型的输入型表单控件，包括单行文本框、密码域、复选框、单选按钮、文件域以及按钮等。

1)在表单中添加单行文本框

如果要获取站点访问者提供的一行信息，可以在表单中添加单行文本框。为此，可以在<FORM>和</FORM>之间添加一个 input 标记，并将其 type 属性指定为“text”。创建单行文本框的基本语法格式如下：

```
<input type= "text" name= "字符串" value= "字符串" size= "整数" maxlength
= "整数">
```

其中，“name”属性指定文本框的名称，通过它可以在脚本中引用该文本框控件；“value”属性指定文本框的初始值；“size”属性指定文本框的宽度；“maxlength”属性指定允许在文本框内输入的最大字符数。当提交表单时，该文本框的名称和内容都会包含在表单结果中。

2)在表单中添加密码域

如果要求站点访问者输入密码后才能进入站点，则应在表单中添加密码域。密码域其实也是一个单行的文本框。当站点访问者在密码域中键入数据时，大部分的 Web 浏览器都会以星号或圆点显示密码，从而保证安全。若要创建一个密码域，可以在<FORM>和</FORM>之间添加一个 input 标记，并将其 type 属性指定为“password”。创建密码域的基本语法格式如下：

＜input type＝ "password" name＝ "字符串" value＝ "字符串" size＝ "整数" maxlength＝ "整数"＞

其中，“name”属性用于指定密码域的名称，通过这个名称可以在脚本中引用该控件；“value”属性用于指定密码域的初始值；“size”属性用于指定密码域的宽度；“maxlength”属性用于指定允许在密码域内输入的最大字符数。当提交表单时，该域的名称和内容都会被包含在表单结果中。

3)在表单中添加按钮

使用＜input＞标记可以在表单中添加三种类型的按钮：提交按钮、重置按钮和自定义按钮。创建按钮的基本语法格式为：

＜input type ＝ "submit|reset|button" name＝ "字符串" value＝ "字符串" OnClick ＝ "过程"＞

其中，“type”属性指定按钮的类型，取值可以是 submit(创建一个提交按钮)、reset(创建一个重置按钮)、button(创建一个自定义按钮)；“name”属性指定按钮的名称；“value”属性指定显示在按钮上的标题文本。

4)在表单中添加复选框

如果想让站点访问者去选择一个或多个选项或不选取任何选项时，可以在表单中添加复选框。若要创建复选框，可以在＜FORM＞和＜/FORM＞之间使用 input 标记，并将 type 属性设置为“checkbox”，基本语法格式如下：

＜input type ＝ "checkbox" name＝ "字符串" value＝ "字符串" [CHECKED]＞选项文本

其中，“name”属性指定复选框的名称；“value”属性指定提交时的值；“CHECKED”属性是可选的，若使用该属性，则当第一次打开表单时该复选框处于选中状态。

当提交表单时，假如复选框被选中，它的内部名称和值都会包含在表单结果中。否则，只有名称会被纳入表单结果中，但值为空白。

5)在表单中添加单选按钮

如果想让站点访问者从一组选项中选择其中一个，可以在表单中添加单选按钮。若要创建单选按钮，可以在＜FORM＞和＜/FORM＞之间使用 input 标记，并将 type 属性设置为“radio”，基本语法格式如下：

＜input type ＝ "radio" name＝ "字符串" value＝ "字符串" [CHECKED]＞选项文本

其中，“name”属性指定单选按钮的名称，若干个名称相同的单选按钮构成一个控件组，在该组中只能选中一个选项；“value”属性指定提交时的值；“CHECKED”属性是可选的，若使用该属性，则当第一次打开表单时该单选按钮处于选中状态。

当提交表单时，该单选按钮组名称和所选取的单选按钮指定值都会包含在表单结果中。如果没有任何单选按钮被选取，组名称会被纳入表单结果中，但值为空白。

6)在表单中添加文件域

文件域由一个文本框和一个“浏览”按钮组成，用户既可以在文本框中输入文件的路径和文件名，也可以通过单击“浏览”按钮从磁盘上查找和选择所需文件。

其创建方法是在＜FORM＞和＜/FORM＞之间添加 input 标记，并将其 type 属性设置为“file”，其语法格式如下：

＜input type ＝ "file" name＝ "字符串" size ＝ "整数" value＝ "字符串"＞

其中，“name”属性指定文件域的名称；“value”属性给出初始文件名；“size”属性指定文

件名输入框的宽度。

7)在表单中添加隐藏域

若要在表单结果中包含不希望让站点访问者看见的信息,可以在表单中添加隐藏域。每一个隐藏域都有自己的名称和值。当提交表单时,隐藏域的名称和值就会与可见表单域的名称和值一起包含在表单结果中。例如,如果有许多使用相同自定义表单处理程序的表单时,就可以利用独特的名称或数字作为隐藏域的名称来区分各个表单。

其创建方法是在<FORM>和</FORM>之间添加 input 标记,并将其 type 属性设置为“hidden”,其语法格式如下:

```
<input type = "hidden" name= "字符串" value= "字符串">
```

其中,“name”属性指定隐藏域的名称;“value”属性给出隐藏域的默认值。

8)在表单中添加滚动文本框

若要接受站点访问者输入的多于一行的文本,可以在表单中添加滚动文本框。在<FORM>和</FORM>之间添加<TEXTAREA>标记,即可创建滚动文本框,其基本语法格式如下:

```
<TEXTAREA name= "字符串" rows= "整数" cols= "整数" [READONLY]></TEXTAREA>
```

其中,“name”属性指定滚动文本框控件的名称;“pows”属性指定该控件的高度(以行为单位);“cols”属性指定该控件的宽度(以字符为单位);“READONLY”属性为可选项,用于指定滚动文本框的内容不被用户修改。

创建滚动文本框时,在<TEXTAREA>和</TEXTAREA>标记之间输入的文本将作为该控件的初始值。当提交表单时,该域名称和内容都会包含在表单结果中。

9)在表单中添加选项菜单

若要创建选项菜单,应在<FORM>和</FORM>之间添加 select 标记,并使用<option>标记将每个选项列出来。基本语法格式如下:

```
<select name= "字符串" size= "整数" [MULTIPLE]>
    <option[SELECTED] value= "字符串">选项 1</option>
    <option[SELECTED] value= "字符串">选项 2</optton>
    ……
</select>
```

其中,“name”属性指定选项菜单控件的名称;“size”属性指定在列表中一次可以看到的选项数目;布尔属性“MULTIPLE”指定是否允许进行多项选择;“SELECTED”属性指定该选项的初始状态为选中。当提交表单时,菜单的名称会被包含至表单结果中,其后有一份所有选项值的列表。

**4.表单提交和处理**

当用户填写完表单数据后,单击提交按钮即可将表单数据提交给 Web 服务器上的表单处理程序。提交信息表单处理程序的方法由<FORM>标记的 method 属性来确定。提交表单的方法有两种:GET 方法和 POST 方法。

1)GET 方法

GET 方法把参数作为统一资源指示器(uniform resource indicator)查询字符串的一部分,从一个页面传递到另一个页面。当用于表单处理时,GET 方法用问号作为分隔符把变量名和值附加给在 ACTION 属性中指定的 URL,并把所有内容提交给处理器。

表单处理中的 GET 方法有一个 POST 方法不具备的很大的优点：它创建了一个真正新的、完全不同的 URL 查询字符串，从而使用户可以把这一页作为书签进行保存。而使用 POST 方法的表单得到的结果是不能被记作书签的。

GET 方法的缺点有如下几方面：

(1)GET 方法不适合用于登录界面，因为在把用户名和密码作为访问过的页面潜在地存储在客户浏览器内存中时，在屏幕上是完全可见的。

(2)每个以 GET 方法进行的提交被记录在 Web 服务器日志中，数据集也被包括在内。

(3)因为 GET 给服务器环境变量分配数据，所以 URL 的长度受到限制。

**建议**：使用 GET 方法处理表单的最合适的用途是搜索框。除非迫不得已的原因将 GET 方法用于非搜索性质的表单处理程序，否则使用 POST。

2)POST 方法

POST 方法尤其适合需要得到长期的副作用效果的情况，如给数据库添加信息等。当表单数据被送到处理程序时被包括在表单体内。提交的数据不同时，在 URL 中看不出什么变化。

POST 方法有以下优点：

(1)它比 GET 更安全，因为在 URL 查询字符串、服务器日志中，或者(如果采用了预防措施)在屏幕上从来看不到用户输入的信息。

(2)对能被传递的数据的数量限制放宽了(2 000 B，而不是 200 多个字符)。

但是 POST 也有一些缺点：

(1)结果不能被记作书签。

(2)该方法和某些防火墙设置不兼容，为了安全，防火墙要去掉表单数据。

表单处理程序或 URL 地址由<FORM>标记的 action 属性来确定。如果要处理表单数据，需要在服务器端编写脚本(CGI、ASP 或 JSP 等)作为表单处理程序。

表单的提交方式可通过普通按钮的 JavaScript 函数实现，也可通过 submit 类型按钮提交，例如：

```
<input type="submit" value="提交">
```

表单内容的清除可通过 reset 按钮来实现，从而将表单内容重置为初始值，其格式为：

```
<input type="reset" value="重置">
```

### 5. 表单示例

下面是一个表单实例，内容如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
  <head><title>表单</title></head>
  <body>
  <p><strong>表单元素综合演示</strong></p>
  <form id="form1" name="form1" method="get" action="">
   <table width="100%" border="0" cellspacing="0" cellpadding="5">
    <tr>
     <td width="20%">用户名:</td>
     <td width="80%"><input name="name" type="text" id="txt" /></td>
```

```
</tr>
<tr>
  <td>所在地区:</td>
  <td><select name="dq" id="da">
   <option value="0">请选择</option>
   <option value="1">河南</option>
   <option value="2">北京</option>
   <option value="3">上海</option>
   </select>
   </td>
</tr>
<tr>
   <td>性别:</td>
   <td>男<input type="radio" name="sex" value="m"/>
   女<input type="radio" name="sex" value="f"/>
  </td>
  </tr>
  <tr>
  <td align="center"><input type="submit" value="提交" /></td>
</tr>
</table>
</form>
</body>
</html>
```

本例中用到了多个表单控件，如单行文本框、单选按钮、提交按钮、选项菜单等，其运行效果如图 2-3 所示。在“用户名”文本框中输入名称，在“所在地区”下拉列表框中选择地区，然后选中“男”或“女”单选按钮，单击“提交”按钮即可。


图 2-3　表单效果

### 2.3.2 JSP 对表单的处理

#### 1. 表单的处理方式

提交表单后，对表单数据的处理有以下两种情况：

(1)客户端处理。在客户端对表单数据进行处理，如采用 JavaScript 实现。这种处理方式的好处是可以减少服务器的负载，缩短用户等待时间。但也有很大的缺点，即由于客户端情况的不同，实现起来兼容性较差。

(2)服务器端处理。表单数据提交至后台服务器进行处理。这样做的好处是统一确认、兼容性强，但同时也会加重服务器端的负载，延长用户等待时间。

JSP 是种典型的服务器端技术，它对表单数据的处理采用服务器端处理方式，它可以采用 Java 类对表单数据进行复杂的检查，具有很好的客户端兼容性。

#### 2. 从表单中获得参数

JSP 通过 request 内置对象获取表单信息，使用不同的方法可以获取不同种类的信息。获取参数的方法主要有如下三种：

(1)getParameter()：获取参数的值。

(2)getParameterNames()：获取参数的名称。

(3)getParameterValues()：获取多个值。

其中，使用 getParameter()方法可以获得文本框、文本区域、单选按钮、复选框等的值，表单中各元素的 name 属性的值可以唯一标识该元素，getParameter()方法用它获取参数。例如，表单中用<input type="text" name="User" >定义了一个文本框，在 JSP 中获取该文本框数值的语句如下：

```
request.getParameter("User");
```

#### 3. 向客户端输出

在 JSP 中将信息或数据处理结果向客户端输出的方法主要有两种：

(1)使用内置对象 out，其语法格式如下：

```
out.println(content)
```

例如：

```
out.println("用户名:" +User+"<br>")。
```

(2)使用“=”，其语法格式如下：

```
<%= content% >
```

例如：

```
您是第<%=application.getAttribute("counter")%>位访问者!
```

#### 4. 实例

下面是一个 JSP 页面程序，用来实现简单的用户登录功能。代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
  <head><title>确认 JSP</title></head>
  <body>
```

```
        <form method="POST" name="frm1" action="2-3.jsp">
          <p align="center">用户登录
          <p align="center">用户名:<input type="text" name="name" size="20" value="">
          <p align="center">密码:<input type="password" name="pwd" size="20" value="">
          <br><br>
          <input type="submit" value=" 提交">
          <input type="reset" value="全部重写"></p>
        </form>
      </body>
    </html>
```

该程序运行效果如图 2-4 所示。


图 2-4　用户登录界面程序运行效果

在上述代码的表单定义中有“action="2-3.jsp""语句,它表明当用户单击“提交”按钮后,对表单元素的处理操作将由 2-3.jsp 文件来完成,其代码如下:

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
  <head><title>JSP 确认</title></head>
  <body>
<%
  String name=request.getParameter("name");
  String pwd=request.getParameter("pwd");
  if(name! =null&&pwd! =null)
  {
    out.println("用户名:" +name+"<br>");
    out.println("密码:"+pwd+"<br>");
  }
```

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

在上述代码中，JSP 利用 request. getParameter()方法分别来获取表单中名称为 name 和 pwd 的元素的值，然后利用 out. println()方法分别将其值进行输出。因此，当用户在图 2-4 的表单中输入完整的信息后单击“提交”按钮，将会弹出如图 2-5 所示的界面，显示用户所输入的相关信息，即用户名和密码。


图 2-5 显示用户输入的信息

**5. 中文乱码问题**

在图 2-4 所示的界面中，如果用户输入的用户名为中文，如“张三”，则单击“提交”按钮后将会出现乱码，如图 2-6 所示。


图 2-6 中文乱码现象示例

在 JSP 编程中经常会遇到这样的情况，即从一个页面获得的中文信息，在另一个页面中显示是乱码。对于中文乱码问题，需要根据不同情况进行不同的处理。

(1)纯 HTML 页面中的乱码。在纯 HTML 页面中一般不出现乱码，但在浏览器默认语言不是中文时可能出现。如果出现乱码，可能是在头部信息中少了字符集的说明。需要在<head>与</head>之间插入如下语句：

```
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
```

(2)JSP 页面中的乱码。对于 JSP 页面“<%”和“%>”对中输出内容的乱码，可以通过将下面的语句插入到 JSP 文件的前几行来解决：

```
<%@ page contentType="text/html;charset=gb2312" %>
```

或

```
<%@ page pageEncoding="gb2312" %>
```

或

```
<%@ page pageEncoding="gbk" %>
```

(3)request 请求中的乱码。对于这种乱码,可以在获取 request 参数之前添加下面的语句解决:

```
<% request.setCharacterEncoding("gb2312"); %>
```

(4)response 响应中输出的乱码。对于这种乱码,可以在 response 输出之前添加下面的语句解决:

```
<% response.setContentType("text/html;charset=gb2312"); %>
```

针对图 2-6 所示的情况,用户就可以按第三种方案进行处理,从而避免中文信息出现乱码的问题。同样,当需要把获得的数据存入数据库的时候,如果有中文内容,也要进行这样的转换以保证存入数据库中的数据是中文而不是乱码。

## 2.4　综 合 实 例

以下实例展示了 JSP 提取并处理客户端表单中数据的过程。本实例涉及 2-4.jsp 和 2-5.jsp 两个文件,前者用于实现表单界面,并提示用户填写相应的注册信息;后者是对表单提交后的处理,呈现用户填写完成后的注册信息。

**1.实现表单界面**

表单界面用于呈现用户信息注册界面,需要用户提交的资料分别有:姓名、密码、性别、爱好和行业,因此,界面上有文本框、密码框、单选按钮、复选框、列表框、提交按钮和重置按钮等表单元素。

实现表单界面的文件 2-4.jsp 的代码如下:

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
  <head><title>用户注册</title></head>
  <body>
    <form method="post" action="2-5.jsp ">
      <p align="center">用户信息注册</p>
      <table align="center">
        <tr><td align="left">您的姓名:<input type="text" name="name"
size="18"></td></tr>
        <tr><td align="left">您的密码:<input type="password" name=
"pwd" size="18"></td></tr>
        <tr><td align="left">您的性别:
            <input type="radio" name="sex" value="男">男
            <input type="radio" name="sex" value="女">女</td></tr>
        <tr><td align="left">您的爱好:<br>
          <input type= "checkbox" name="like" value="音乐">音乐
```

```
        <input type= "checkbox" name="like" value="电影">电影
        <input type = "checkbox" name="like" Value="阅读">阅读
        <input type = "checkbox" name="like" Value="旅游">旅游
        <input type = "checkbox" name="like" Value="运动">运动
        </td></tr> <tr><td align="left">您的行业:
            <select name = "job" size ="1">
            <option value="">----</option>
            <option value="教育">教育</option>
            <option value="行政">行政</option>
            <option value="私企">私企</option>
            <option value="事业">事业</option>
            <option value="IT">IT</option>
            </select></td></tr>
        <tr><td align="center"><br>
<input type="submit" value=" 填好了!">
<input type="reset" value="重置"></td></tr>
      </table>
    </form>
  </body>
</html>
```

运行效果如图 2-7 所示。


图 2-7 表单界面效果

**2. 实现表单提交后的处理**

用户在图 2-7 所示的界面中进行信息的输入,单击“重置”按钮可撤销用户所做的所有输入,单击“填好了!”按钮可将用户所输入的信息提交至 2-5.jsp 进行处理,其对表单元素数据进行处理的代码如下:

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
    <title>用户注册信息提取</title>
</head>
<body>
<%
  request.setCharacterEncoding("gb2312");
  String  username=request.getParameter("name");
  String  pwdinfo=request.getParameter("pwd");
  String  sexinfo=request.getParameter("sex");
  String[]  s=request.getParameterValues("like");
  String  jobinfo=request.getParameter("job");
  out.println("<br>");
  if(username=="") out.println("您没有输入姓名! <br>");
  else
    {
        out.println("您的姓名:");
        out.println(username);
        out.println("<br>");
    }
  if(pwdinfo=="") out.println("您没有输入密码! <br>");
  else
    {
        out.println("您的密码:");
        out.println(pwdinfo);
        out.println("<br>");
    }
  if(sexinfo==null) out.println("您没有选择性别! <br>");
  else
    {
        out.println("您的性别:");
        out.println(sexinfo);
        out.println("<br>");
    }
  if(s==null) out.println("您没有选择爱好! <br>");
   else
    {
      out.println("您的爱好:");
      for(int i=0;i<s.length;i++)
```

```
        {
          out.println(s[i]+"  ");
        }
        out.println("<br>");
       }
    if(jobinfo==="") out.println("您没有选择行业! <br>");
    else
      {
          out.println("您的行业:");
          out.println(jobinfo);
          out.println("<br>");
      }
 %>
</body>
</html>
```

在该程序中,首先利用 request. setCharacterEncoding("gb2312")方法进行字符编码格式设定,防止当用户输入信息为中文时出现乱码,然后利用 request. getParameter()方法调用表单元素的 name 以获取其数据,即用户输入的相关信息,并赋值给相应的变量,最后利用 if 语句对这些变量进行分析:若变量值为空即用户未输入相应信息,则输出"您没有选择…"这样的提示信息;反之,则输出用户的信息。

没有进行个人设置设置和进行了设置的效果分别如图 2-8 和图 2-9 所示。

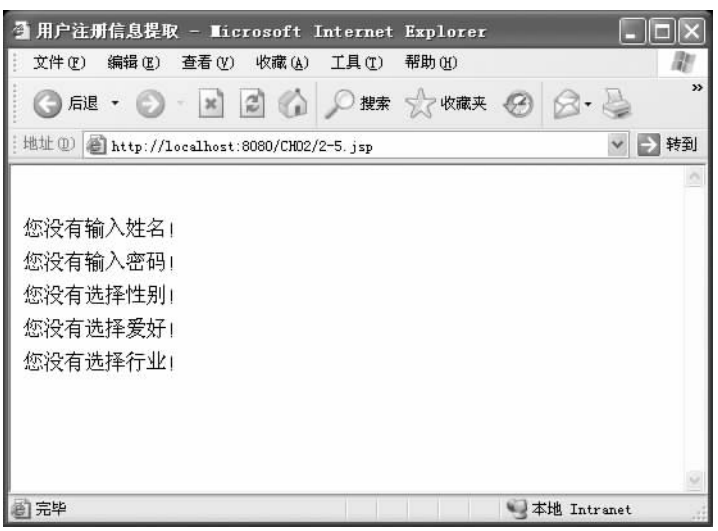


图 2-8　没有进行设置而提交后的结果

从这个程序中不难发现,JSP 主要是通过 request. getParameter()方法来提取表单中的数据,但需要注意的是,在编写表单时,对于表单中任一元素的 name 必须赋值,因为 JSP 调用 request 对象的方法 getParameter()时,正是将 name 值作为该方法的形参来提取表单中相应元素中的输入数据。除此之外,还可以通过 JavaBean 技术(第 5 章介绍)来获取表单中

的信息，当然在这个 Bean 对应的类体中必须定义一个区域，并且该区域跟表单中的每一个区域相对应。


图 2-9　进行了设置而提交后的结果

## 本 章 小 结

本章主要介绍了 JSP 程序的执行过程、JSP 编程的基础——Java 基础语法规则、JSP 与 Web 页面交互等主要内容。通过本章的学习，要求牢记 JSP 的工作原理，通过编程实践熟悉基本的语法规则，掌握 JSP 对表单数据的处理方法，能够进行简单的交互式网页设计。

## 习　题　2

**一、简答题**

1. 简述 JSP 的工作原理。

2. 表单处理的两种方式各有什么优缺点？

**二、操作题**

1. 调试本章的 JSP 程序代码，查看其对应的. java 和. class 文件内容，加深对 JSP 工作原理的认识。

2. 编写一个简单的交互式网页，实现对学生学籍信息的输入、获取和显示。

# 第 3 章　JSP 语法

掌握一门编程语言要从它的语法开始。本章在介绍 JSP 的通用语法和脚本元素的基础上，详细介绍 JSP 的内置对象、指令和动作。

## 3.1　JSP 通用语法

JSP 是由 Sun 公司在 Java 语言上开发出来的一种动态网页制作技术，可以将网页中的动态部分和静态的 HTML 相分离。用户可以使用平常惯用的工具（如 Dreamweaver）并按照平常的方式来书写 HTML 语句。然后，将动态部分用特殊的标记嵌入即可，这些标记常常以“<%”开始并以“%>”结束。例如，下面是一个 JSP 页面的代码：

```
<html>
<head><title>jsp 页面</title></head>
<body>
<I><% out.println("hello world"); %></I>
</body></html>
```

它将输出“hello world”。

通常，JSP 页面文件以“.jsp”为扩展名，可以存放在任何可以放置普通 Web 页面的路径下。尽管 JSP 文件看起来更像是 HTML 文件而不是 Servlet 文件，但事实上，它恰恰将转换为 Servlet 文件，其中的静态 HTML 仅仅用来输出 Servlet 服务方法返回的信息。如果 JSP 页面已经被转换为 Servlet，且 Servlet 被编译进而被装载（在第一次被请求时），当用户再次请求此 JSP 页面时，将察觉不到一瞬的延迟。

**注意：**一些 Web Servers 允许用户为它定义别名，从而使得一个 URL 从表面上看是指向一个 HTML，但事实上它指向的是一个 Servlet 或 JSP 页面。

构造一个 JSP 页面，除了可内嵌的 HTML 文本，还有三类主要的 JSP 元素：Scripting elements，Directives 和 Actions。使用 Scripting elements 用户可以定义最终转换为 Servlet 的部分，Directives 使用户可以控制这个 Servlet 的整体结构，而 Actions 使用户可以指定可重用的已有组件，另外，还可控制 JSP 引擎的运行。为了简化 Scripting elements，用户可以在某一段上利用一些预定义的变量，如 request。

### 3.1.1　JSP 基本语法

下面以 JSP 1.1 版本为例，其语法概述如表 3-1 所示，它的详细使用将在随后的课程中详细讲解。

表 3-1　JSP 语法概述

| JSP 元素 | 语　　法 | 解　　释 |
| --- | --- | --- |
| 表达式 | <%= 表达式 %> | 表达式用于计算并输出 |
| 程序片段 | <% 代码 %> | 插入程序代码 |
| 声明 | <%! 代码 %> | 属于 Servlet 部分的代码但并不是服务方法 |
| 编译器指令 | <%@ page att="val" %> | 以下是其合法的属性：<br>import="package. class"<br>contentType="MIME－Type"<br>isThreadSafe="true|false"<br>session="true|false"<br>buffer="sizekb|none"<br>autoflush="true|false"<br>extends="package. class"<br>info="message"<br>errorPage="url"<br>isErrorPage="true|false"<br>language="java" |
| include 指令 | <%@ include file ="URL" %> | 当 JSP 页面被翻译成 Servlet 时将被包含进本地系统上的文件内。其中，“URL”必须是相对的，当页面被请求时才用<jsp:include>动作调入 |
| JSP 注释 | <%-- 注释--%> | 当 JSP 页面转换为 Servlet 时将被忽略 |
| <jsp:include>动作 | <jsp:include page="relative URL" flush="true"/> | 在页面被请求(requested)时调入文件 |
| <jsp:useBean>动作 | <jsp:useBean att=val */><br>或<jsp:useBean att=val *><br>…<br></jsp:useBean> | 寻找或生成一个 JavaBean<br>可能的属性如下：<br>id="name"<br>scope="page|request|session|application"<br>class="package. class"<br>type="package. class"<br>BeanName="package. class" |
| <jsp:setProperty>动作 | <jsp:setProperty att=val */> | 设置 Bean 的属性<br>合法的属性如下：<br>name="beanName"<br>property="propertyName|*"<br>param="parameterName"<br>value="val" |

续表

| JSP 元素 | 语　法 | 解　释 |
| --- | --- | --- |
| ＜jsp：getProperty＞动作 | ＜jsp:getProperty name＝"propertyName" value＝"val"/＞ | 检索并输出 bean 的属性 |
| ＜jsp:forward＞动作 | ＜jsp:forward page＝"relative URL"/＞ | 向前请求(request)另一个页面 |
| ＜jsp:plugin＞动作 | ＜jsp:plugin attribute＝"value"＊＞ ＜/jsp:plugin＞ | 生成特定的浏览器的 OBJECT 或 EMBED 标签,用来明确运行 Applet 所使用的 Java 插件(plugin) |

### 3.1.2　注释

JSP 的注释可分为三种:HTML 注释、隐藏注释和标准 Java 注释。

#### 1. HTML 注释

HTML 注释在客户端显示,其语法格式如下:

＜! --comment [＜％＝ expression ％＞]--＞

JSP 中的 HTML 注释与其他 HTML 中的注释类似,它可以在客户端浏览器的“查看源代码”中看到,唯一有些不同的就是,用户可以在这个注释中使用各种表达式,只要表达式是合法的就行,JSP 容器会对表达式求值并将结果作为注释内容发往客户端。例如:

```
＜! -- This page was loaded on ＜％＝ (new java.util.Date()).toLocaleString()
％＞ --＞
```

上述代码在客户端的 HTML 源代码中显示为:

```
＜! -- This page was loaded on December 10, 2008 --＞
```

其中,“December 10,2008”为当前时间。

#### 2. 隐藏注释

HTML 注释会被客户端利用“查看源代码”看到,如果用户不想被客户端看到注释的内容,就应该将其嵌入到“＜％--”和“--％＞”标记中。JSP 编译器会将隐藏注释标记中的字符忽略掉,而不对该语句进行编译。例如:

```
＜％@ page language＝"java" ％＞
＜html＞
＜head＞＜title＞Hidden Comment Test＜/title＞＜/head＞
＜body＞
＜h2＞A Test of Hidden Comments＜/h2＞
＜％-- 这部分注释在"查看源代码"中是不可见的 --％＞
＜/body＞
＜/html＞
```

运行效果如图 3-1 所示。


图 3-1　隐藏注释效果图

#### 3. 标准 Java 注释

标准 Java 注释只能包含在脚本代码中。JSP 容器不对该部分内容编译，也不会发送到客户端浏览器。标准 Java 注释的 JSP 语法如下：

```
//comment
```

或

```
/* comment */
```

## 3.2　JSP 脚本元素

JSP 脚本由三个元素组成，即 JSP 表达式、JSP 脚本程序和 JSP 声明。JSP 表达式包含在“<%=”和“%>”标签内，JSP 脚本程序包含在“<%”和“%>”内，JSP 声明包含在“<%!”和“%>”内。

### 3.2.1　表达式

表达式是对数据的表示，系统将其作为一个值进行计算和显示。

#### 1. 语法格式

表达式的语法格式如下：

```
<%=expression %>
```

JSP 表达式是一个值，包含在“<%=”和“%>”中，在运行后被自动转化为字符串，然后插入到这个表达式所在的位置中。因为这个表达式的值已经被转化为字符串，所以用户能在一行文本中插入这个表达式。

#### 2. 注意事项

(1)用户不能用分号(;)作为表达式的结束符。

(2)一个表达式元素可以是任何一个符合 Java 语言规范的表达式，也可以由多个表达式组成。

(3)表达式也能作为其他 JSP 元素的属性值。

**3. 示例**

下面的程序演示了表达式的使用,运行效果如图 3-2 所示。

```
<html>
<head>
<title>表达式的使用</title>
<body>
  <h1>JSP 表达式 </h1>
  <b>PI 的值:</b><%=Math.PI %><br/>
  <b>100 至 99 中最大的值:</b><%=Math.max(100,99) %><br/>
  <b>100 至 99 中最小的值:</b><%=Math.min(100,99) %><br/>
  <b>3+2-5 的值:</b><%=3+2-5 %><br/>
  <b>(3+2)==5 的值:</b><%=(3+2)==5 %><br/>
  <b>(3+2)! =5 的值:</b><%=(3+2)! =5 %><br/>
</body>
</html>
```


图 3-2 表达式的使用

## 3.2.2 脚本程序

JSP 脚本程序就是在 JSP 页面里嵌入的一段 Java 代码。

**1. 语法格式**

JSP 脚本程序的语法格式如下:

```
<% 代码段 %>
```

其中的代码段可以包含多个 Java 语句、方法变量和表达式,在 Web 服务器响应请求时就会运行。在 JSP 容器将 JSP 页面转换为 Servlet 类时,页面中的脚本代码会按照出现的次序,依次被转换为_jspService()方法中的代码。

### 2. 注意事项

(1)在脚本程序中也可以像表达式那样随意定义变量。

(2)在脚本程序中可以使用任何隐含的对象和任何用<jsp:useBean>声明过的对象,可以编写 JSP 语句。

(3)任何 HTML 文本标记必须写在脚本程序,即"<%"和"%>"对之外。

(4)当 JSP 编译器收到客户端的请求时,脚本程序就会被执行。如果脚本程序有显示内容,这些显示的内容被保存在 out 对象中。

### 3. 示例

下面的程序演示了脚本程序的使用,运行效果如图 3-3 所示。

```
<html>
<head>
<title>scriptlet 的使用</title>
</head>
<body>
  <h1>以直角三角形的形式显示数字</h1>
  <%
    for(int i=1;i<10;i++){
        for(int j=1;j<=i;j++){
          out.println(j);
        }
    out.println("<br/>");
    }
  %>
</body>
</html>
```

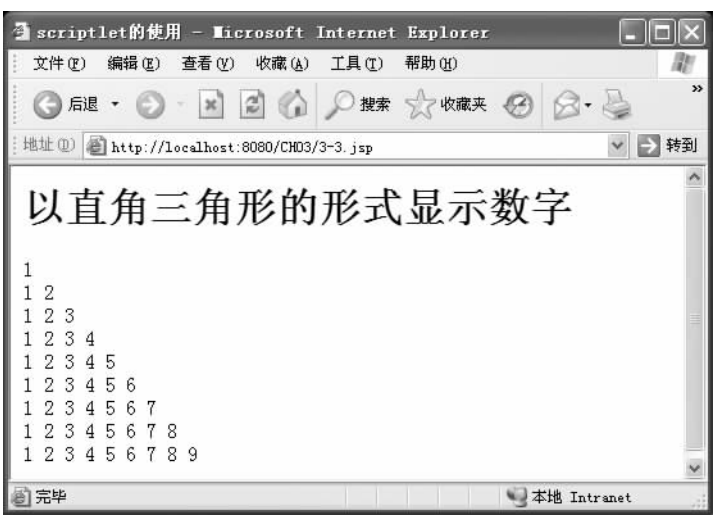


图 3-3　脚本程序使用示例

### 3.2.3 声明

JSP 声明就是在 JSP 页面中声明 Java 方法或变量等。

**1. 语法格式**

JSP 声明的语法格式如下：

```
<%! declaration; [ declaration; ]… %>
```

例如：

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

利用 JSP 声明，用户可以对 JSP 程序中的变量、类、方法的类型进行合法的定义，如定义整型、字符串型等。如果用户发现代码太多，也可以把它们写成一个独立的 Java 类进行声明。声明的变量可以在声明处到本 JSP 页面结尾处的范围内使用，JSP 页面中类和方法的声明必须放在声明块内。

**2. 注意事项**

(1)用户可以一次性声明多个变量和方法，但一定要以分号(;)结束变量声明，因为任何内容都必须是有效的 Java 语句。

(2)用户可以直接使用在“<% @ page”和“%>”对中已经声明的变量和方法，不需要对它们重新进行声明。

(3)一个声明仅在一个页面中有效。如果要使一些声明在每个页面中都可用，最好把它们写成一个单独的文件，然后用<%@ include %>或<jsp:include >元素包含进来。

(4)声明不能产生任何输出，它通常用作 JSP 表达式和脚本代码之间的连接。

**3. 示例**

在 JSP 页面中将不带小数的金额转换为带两位小数的金额。

可以利用 JSP 声明声明一个方法，将整数转换为两位小数，并利用 JSP 脚本代码调用声明的方法转换金额，然后在 JSP 页面利用表达式将转换后的金额显示出来。

具体代码如下：

```
<%!
//声明一个常量
final String SEPARATOR ="." ;
//声明一个方法
public String covertAmountWithSeparator(String money)
{
    int index=money.indexOf(SEPARATOR);
    String str=money;
    if(index==-1)
    str=money+".00";
    return str;
}
%>
```

## 3.3　JSP 指令

JSP 指令是为 JSP 引擎设计的，JSP 指令并不直接产生任何可见输出，而只是告诉引擎如何处理 JSP 页面的相关信息。JSP 指令共有三种类型，即 page 指令、include 指令和 taglib 指令，包含在“<%@”和“%>”内。page 指令用于设置 JSP 页面的属性；include 指令用于在 JSP 页面嵌入其他文件；taglib 指令用于在 JSP 页面中创建和使用自定义标签。JSP 指令的语法格式如下：

<%@ 指令名称 属性 1="值 1" 属性 2="值 2" …… 属性 n="值 n" %>

### 3.3.1　include 指令

include 指令用于在运行时将 HTML 文件或 JSP 页面嵌入到另一个 JSP 页面。其语法格式如下：

<%@ include file ="relativeURL" %>

其中，“relativeURL”表示要嵌入文件的路径，它可以是文档相对路径或站点相对路径。

使用 include 指令将会在 JSP 编译时插入一个包含文本或代码的文件，当用户使用 <%@ include%> 指令时，这个被包含的过程就是静态的。包含的文件可以是 JSP 文件、HTML 文件、文本文件或只是一段 Java 代码。如果包含的是 JSP 文件，这个包含的 JSP 的文件中的代码将会被执行。

如果用户仅仅只是用 include 来包含一个静态文件，那么这个包含的文件所执行的结果将会插入到 JSP 文件中 include 指令所在的地方。一旦被包含文件执行完成，那么主 JSP 文件的过程将会被恢复，继续执行下一行。

**注意：**在这个被包含文件中不能使用<html>、<body>标记对。因为这将会影响主 JSP 文件中同样的标记，有时会导致错误。

有一些 include 指令的行为是以特殊的 JSP 编译条件为基础的，例如，这个被包含的文件必须对所有客户都开放且必须有效，或者它有安全限制，如果它被改变，包含此文件的 JSP 文件将被重新编译。

例如，下面的示例中利用 include 指令将 head.jsp 嵌入到本 JSP 页面中：

```
<%@page language="java" %>
<%@include file="head.jsp" %>
<html>
<head>
<title>include 指令测试</title>
</head>
<body>
<br><br>
<h4 align="center">这是 JSP 的 include 指令测试页面！ </h4>
</body>
</html>
```

**注意：**

(1)file="relativeURL"这个被包含文件的路径名一般来说是指相对路径，不需要端口协议和域名。如果这个路径以"/"开头，那么它主要是参照 JSP 应用的上下关系路径；如果路径是以文件名或目录名开头，那么它就是正在使用的 JSP 文件的当前路径。

(2)include 文件是不能嵌套使用的。

### 3.3.2 page 指令

page 指令用于设置 JSP 页面的全局属性，其语法格式如下：

```
<%@page language= "java"
extends= "类名"
        import= "Java 包列表"
        session= "true|false"
        buffer= "none|8KB|自定义缓冲区大小"
        autoFlush="true|false"
        inThreadSafe= "true|false"
        info= "页面信息"
        errorPage= "页面出错时错误处理页面的 URL"
        contentType= "内容类型信息"
        isErrorPage="true|false"
%>
```

各属性含义如下：

(1)language：声明该页面使用的脚本语言，默认值是 Java。设置 language 属性的示例如下：

```
<%@page language= "java" %>
```

(2)extends：用于指定 JSP 页面转换后的 Servlet 类的父类，属性的取值是包含类名和所在包名的完整类名。通常情况下不需要使用这个属性，JSP 容器会提供转换后的 Servlet 类的父类。例如，下面的 page 指令使当前 JSP 页面转换后的 Servlet 类继承自定义类 My_Package. Example：

```
<%@page extends="My_Package. Example" %>
```

(3)import：表示需要导入的 Java 包的列表，其作用与 Java 语言中的 import 语句相同。这些包以及它们的类作用于程序段、表达式、声明以及 JSP 文件内的标签，它必须放在调用它的类的标签之前。

下面的包在 JSP 编译时已经导入了，因此不需要再特别指明：

```
java. lang. *
javax. servlet. *
javax. servlet. jsp. *
javax. servlet. http. *
```

(4)session：决定用户是否需要管理用户的会话信息，这些信息可能来自多个网页。其值的设定取决于客户是否需要 HTTP session。如果它为 true，那么 session 对象是有用的；如果它为 false，那么就不能使用 session 了，也不能在<jsp：useBean>动作中定义 scope= session，否则会导致错误。session 的缺省值是 true。

(5)buffer:设置用来存储客户端请求的缓冲区的大小,buffer 的大小以 KB 为单位,缺省值是 8 KB。若用户自定义其值,则必须不小于 8 KB。

(6)autoFlush:设置 buffer 溢出时是否需要强制输出,默认值为 true,此时输出正常。如果其值被定义为 false,一旦 buffer 溢出就会导致一个意外错误的发生。

**注意**:如果 buffer 设置为 none,那么 autoFlush 就不能设置为 false。

(7)isThreadSafe:设置 JSP 文件是否能多线程使用。如果设置为 true,有多个请求访问实例变量的时候,多重请求都可以被单个 servlet 实例同时执行。如果设置为 false,表明 servlet 为单线程模式,每个请求分配给一个独立的 servlet 实例,即一次只能处理一个请求。isThreadSafe 的缺省值是 true。

(8)info:指定一个插入到 JSP 页面中的文本,被合并编译后,用户能够使用 Servlet.getServletInfo()方法重新得到这个文本。

(9)errorPage:设置用于处理异常事件的 JSP 文件。例如,要使页面执行错误时转向 errorPage.jsp 文件,语句如下:

```
<%@ page errorPage="errorPage.jsp" %>
```

(10)isErrorPage:设置此页是否是另一个 JSP 页面的异常处理页面。如果被设置为 true,用户就能使用 exception 异常处理对象。如果设置为 false,意味着在用户的 JSP 文件中不能使用异常处理对象。该属性通常与 errorPage 属性页面配合使用。

(11)contentType:定义 JSP 页面的应答的 MIME(多用途的网际邮件扩充协议)类型和字符。用户能够使用任何对 servlet 有效的 MIME 类型和字符集。缺省 MIME 类型是 text/html,缺省字符集为 ISO-8859-1。例如:

```
<%@ page contentType = "text/html; charset = gb2312" %>
```

**注意**:

(1)page 指令作用于整个 JSP 页面,同样包括静态的包含文件。但是 page 指令不能作用于动态的包含文件。

(2)用户可以在一个页面中使用多个 page 指令,但是除 import 属性外,相同的属性只能用一次。

(3)无论用户把 page 指令放在 JSP 文件的哪个地方,它的作用范围都是整个 JSP 页面。不过最好把它放在 JSP 文件的顶部,以方便阅读。

下面是一个使用 page 指令的例子:

```
<%@ page language="java" import="java.util.*" buffer="8KB" errorPage
="error.jsp" isErrorPage="false" %>
<html>
    <head>
        <title> page 指令测试页面</title>
    </head>
    <body>
        <br><br>
        <h1> 这是一个 page 指令测试页面! </h1>
    </body>
</html>
```

### 3.3.3 taglib 指令

taglib 指令的作用是将标签库描述符文件引入到该页面中，并设置前缀。其语法格式如下：

```
<%@ taglib uri="URIToTaglibary" prefix="prefix" %>
```

属性说明如下：

(1)“URIToTaglibary”是指标签库表述符文件。“uri”是 uniform resource identifier 的缩写，即统一资源标记符，根据标签的前缀对自定义的标签进行唯一的命名。uri 可以是 URL(uniform resource locator)、URN(uniform resource name)以及一个相对或绝对路径。

(2)“prefix”表示自定义标签前的前缀。

**注意**：不要用 jsp、jspx、Java、Javax、Servlet、Sun 和 Sunw 作为用户的前缀。

下面是一个使用 taglib 指令的例子。JSP 页面在根目录中搜索 mytaglib 标签库描述符文件，而 mytags 前缀将 mytaglib 中的标签嵌入 JSP 页面。

```
<html>
    <body>
      <%@ taglib uri ="/mytaglib.tld" prefix="mytags" %>
    </body>
</html>
```

## 3.4 JSP 内置对象

为了简化页面开发的复杂度，JSP 提供了一些可在脚本中使用的内置对象，在使用这些内置对象之前不需要对它们进行声明。通过使用这些对象，可以使用户更容易收集客户端发送的请求信息，并响应客户端的请求以及存储客户信息。内置对象的名称是 JSP 的保留字，JSP 使用内置对象来访问网页的动态内容，它一般分为输入/输出对象、作用域通信对象、Servlet 对象和错误对象四类。

### 3.4.1 输入/输出对象

输入/输出对象用于控制页面的输入和输出，访问同请求和响应有关的数据，它包括 request、response 和 out 对象。

#### 1. request 对象

request 对象主要用于接收客户端通过 HTTP 协议传输到服务器端的数据。客户端的请求信息被封装在 request 对象中，通过它才能了解到客户的需求，然后做出响应。它是 HttpServletRequest 类的实例，其主要方法如下：

(1)getCookies()：使用该方法可以返回客户端 Cookie 对象，结果是一个 Cookie 数组。

(2)getHeader(String name)：使用该方法可以获得 HTTP 协议定义的传送文件头信息，例如，Request. getHeader("User-Agent")将返回客户端浏览器的版本号、类型。

(3)getAttribute(String name)：使用该方法可以返回 name 指定的属性值，如果不存在

指定的属性则返回空值 NULL。

(4)getAttributeNames():使用该方法可以返回 request 对象所有属性的名字,结果集是一个 Enumeration 类(枚举类)的实例。

(5) getHeaderNames():使用该方法可以返回所有请求头的名字,结果集是一个 Enumeration 类(枚举类)的实例。

(6)getHeaders(String name):使用该方法可以返回指定名字的请求头的所有值,结果集是一个 Enumeration 类(枚举类)的实例。

(7)getMethod():使用该方法可以获得客户端向服务器端传送数据的类型,有 GET、POST 或 PUT 等类型。

(8)getParameter(String name):获得客户端传送给服务器端的参数值,参数由 name 指定。

(9)getParameterNames():使用该方法可以获得客户端传送给服务端的所有的参数名字,结果集是一个 Enumeration 类的实例。

(10)getParameterValues(String name):以字符串的形式返回指定参数的所有值。

(11)getQueryString():使用该方法可以返回查询字符串,该字符串由客户端以 GET 方法向服务器端传送。

(12)RequestURI():使用该方法可以获得发出请求字符串的客户端地址。

(13)getRequestURI():返回响应请求的服务器地址。

(14)setAttribute(String name, java.lang.Object o):使用该方法可以设定名字为 name 的 request 参数的值,该值由 Object 类型的 o 指定。

(15)getServerName()、getServerPort()、getRemoteAddr()、getRemoteHost():使用以上四个方法可以分别获得服务器的名字、端口号、客户端的 IP 地址和客户端主机的名字,其中 getRemoteHost()方法如果失败,将会返回客户端主机的 IP 地址。

(16)getProtocol():使用该方法可以获取客户端向服务器端传送数据所依据的协议名称。

例如,以下代码通过调用 request 对象的不同方法来获取系统的相关信息,如 IP 地址、主机名、表单元素的值等:

```
<%@ page contentType="text/html;charset=gb2312" %>
<% request.setCharacterEncoding("gb2312"); %>
<html>
  <head>
    <title>request 对象测试 1</title>
  </head>
<body bgcolor="#FFFFF0">
  <form action="" method="post">
    <input type="text" name="user">
    <input type="submit" value="提交">
  </form>
请求方式:<%=request.getMethod() %><br>
请求的资源:<%=request.getRequestURI() %><br>
```

```
请求用的协议:<%=request.getProtocol()%><br>
请求的文件名:<%=request.getServletPath()%><br>
请求的服务器名/IP:<%=request.getServerName()%><br>
请求服务器的端口:<%=request.getServerPort()%><br>
客户端 IP 地址:<%=request.getRemoteAddr()%><br>
客户端主机名:<%=request.getRemoteHost()%><br>
表单提交来的值:<%=request.getParameter("user")%><br>
</body>
</html>
```

运行程序,在文本框中输入“JSP 学习”,单击“提交”按钮,效果如图 3-4 所示。

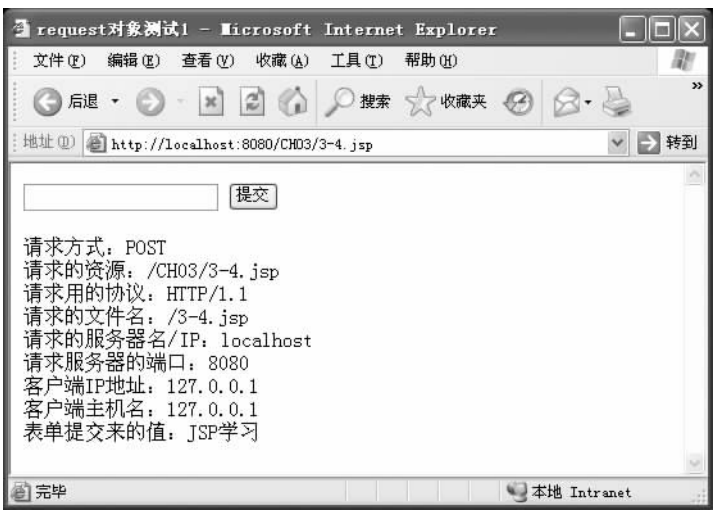


图 3-4 request 对象应用实例

### 2. response 对象

response 对象主要用来向客户端发送数据,如 Cookie HTTP 文件头信息等。response 对象包含了响应客户请求的有关信息,但在 JSP 中很少直接用到它。它是 HttpServletResponse 类的实例,其主要方法如下:

(1)addCookie(Cookie cookie):使用该方法可以添加一个 Cookie 对象,用来保存客户端的用户信息,用 getCookies()方法可以获得这个 Cookie。

(2)addHeader(String name, String value):使用该方法可以添加 HTTP 文件头,该 HTTP 文件头将会传送到客户端,如果存在同名的 HTTP 文件头,那么原来的 HTTP 文件头会被覆盖。

(3)containsHeader(String name):使用该方法可以判断指定名字的 HTTP 文件头是否存在,然后返回 true 或 false。

(4)sendError(int sc):使用该方法可以向客户端发送错误信息,例如,服务器内部错误 505, 网页找不到错误 404。

(5)setHeader(String name, String value):使用该方法将指定的值加入到响应标题,如果这个标题已经设置了值,那么它将被新的值代替。

(6)setContentType(String s):使用该方法动态设置响应的 MIME 类型,参数 s 可取 text/html、text/plain、application/msword 等。

(7)sendRedirect(String URL):使用该方法重定位页面到指定的 URL。

以下代码调用了 response 对象的两种方法,其中 setContentType()用来设置字符集类型,sendRedirect()用来设置重定向的目标文件:

```
<%@ page contentType="text/html; charset=GBK" %>
<html>
  <head>
    <title>response 对象</title>
    </head>
  <body>
    <%
        response.setContentType("text/html;charset=GBK");
        response.sendRedirect("response2.jsp");
     %>
  </body>
</html>
```

### 3. out 对象

out 对象创建输出流,用来向客户端输出显示信息,它是 javax.servlet.jsp.JspWriter 类的实例,其主要方法如下:

(1)print():使用该方法可以输出各种格式的数据,这些数据类型包括 boolean、Char、double、float、int、long、String 和 Object。out.print()方法在输出完毕后不换行。

(2)println():使用该方法可以输出的数据类型与 out.print 相同,不同的是 out.println()方法在输出完毕后进行换行。

(3)newLine():使用该方法将输出一个换行符号。

(4)flush():使用该方法将输出缓冲区里的数据。

(5)close():使用该方法将关闭输出流,并清除所有内容。

(6)ClearBuffer():使用该方法可以把数据写到客户端,并且清除缓冲区里的数据。

(7)Clear():使用该方法可以清除缓冲区里的数据,但不把数据写到客户端去。

(8)getBufferSize():使用该方法可以获得缓冲区的大小,缓冲区的大小可以用<%@ page buffer="size" %>来设置。

(9)getRemaining():使用该方法可以获得缓冲区没有使用空间的大小。

(10)isAutoFlush():使用该方法可以返回用<%@page isAutoFlush="true|false" %>设置的值。

下面的代码分别使用 out 对象的 print()和 write()方法实现字符串内容的输出,程序如下:

```
<html>
  <body>
    <%
    out.print("欢迎学习 JSP 程序设计知识!");
```

```
        out.write("欢迎使用 JSP 的 out 对象!");
        %>
    </body>
</html>
```

运行效果如图 3-5 所示。

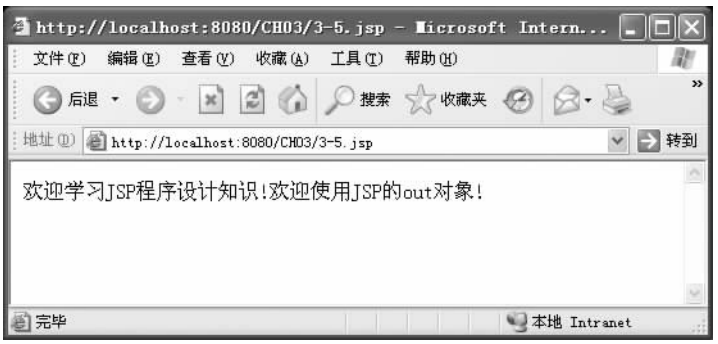


图 3-5 out 对象应用实例

### 3.4.2 作用域通信对象

作用域通信对象向 JSP 页面提供页面环境的访问权，包括 session、application 和 pageContext 对象。

#### 1. session 对象

Web 服务器为每个用户发送的多个请求创建会话，会话状态的维持是 Web 应用开发者必须面对的问题。session 对象用来在每一个用户之间分别保存用户信息，对于那些希望通过多个页面完成一个事务的应用是非常有用的。它是 javax. servlet. http. HttpSession 接口的实例，其主要方法如下：

(1)getAttribute(String name)：使用该方法可以获得指定名字的属性，如果该属性不存在，将会返回 NULL。

(2)getAttributeNames()：使用该方法可以返回 session 对象中存储的每一个属性对象，结果集是一个 Enumeration 类的实例。

(3)getCreationTime()：使用该方法可以返回 session 对象创建的时间，单位为毫秒。

(4)getId()：使用该方法返回 session 对象在服务器端的编号。每生成一个 session 对象，服务器就会给它一个编号，该编号不会重复。

(5)getLastAccessedTime()：使用该方法可以返回当前 session 对象最后一次被操作的时间，单位为毫秒。

(6)getMaxInactiveInterval()：使用该方法可以获取 session 对象的生存时间，单位为秒。

(7)removeAttribute(String name)：使用该方法可以删除指定的属性。

(8)setAttribute(String name, java. lang. Object. Value)：使用该方法可以设定指定名字的属性值，并且把它存储在 session 对象中。

下面通过示例说明 session 的使用方法，该示例用到两个文件：3-6.jsp 和 session.jsp。

3-6.jsp 程序代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<%
    session.setAttribute("username","MYT"); //把用户名保存到 session 中
    session.setAttribute("password","12345"); //把密码保存到 session 中
%>
<a href="session.jsp">转向 session.jsp</a>
```

session.jsp 程序代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<body>
你好！<br>
你的用户名为：<%=session.getAttribute("username").toString() %><br>
你的密码为：<%=session.getAttribute("password").toString() %>
</body>
```

运行 3-6.jsp，效果如图 3-6 所示，单击超链接，网页转向 session.jsp，如图 3-7 所示。


图 3-6　3-6.jsp 运行效果


图 3-7　session.jsp 运行效果

### 2. application 对象

application 对象用来存储有关文档运行环境的信息。与 session 对象相比，application 对象所保存的对象可以被所有用户共享，而 session 对象则是每个用户专用的。application

对象在 _jspService( ) 开始执行时自动被创建，直到服务关闭。它是 javax. servlet. ServletContext 类的一个实例，其主要方法如下：

(1)getAttribute (String name)：使用该方法可以返回指定名字的 application 对象属性值，如果指定属性不存在，则返回 NULL。

(2)getAttributeNames()：使用该方法可以返回所有 application 对象属性的名字，结果是一个 Enumeration 类的实例。

(3)getInitParameter(String name)：使用该方法可以返回指定名字的 application 对象属性的初始值，如果没有参数，则返回 NULL。

(4)getServerInfo()：使用该方法可以获得 Servlet 编译器当前版本的信息。

(5)setAttribute (String name,Object object)：将参数 Object 指定的对象 object 添加到 application 对象中，并为添加的对象指定一个属性。

下面通过一个程序来演示 application 对象的使用方法。该程序实现计数功能，代码如下：

```
<%@page language="java" contentType="text/html;charset=gb2312" %>
  <%
    int count=0;
    String number=null;
    number=(String)application.getAttribute("number");
    if (number!=null)
    {
      count=Integer.parseInt(number);
    }
  count++;
  application.setAttribute("number",String.valueOf(count));
  out.println("总访问次数:"+count);
 %>
```

运行效果如图 3-8 所示。


图 3-8 application 对象的应用

3. PageContext 对象

PageContext 对象的主要功能是让 JSP 容器控制其他隐含对象，如对象的生成与初始化、释放对象本身等。PageContext 对象为 JSP 默认的隐含对象，以及其他可用的对象提供了基本的处理方法，通过这些方法可以让各个对象的该对象是 javax. servlet. jsp. PageContext 类的实例，其常用的方法如下：

(1)void setAttribute(String name，Object value)：以键/值的方式，将一个对象的值存放到 PageContext 中。

(2)void getAttribute(String name)：根据名称去获取 PageContext 中存放对象的值。

### 3.4.3　Servlet 对象

JSP 引擎为每个 JSP 生成一个 Servlet，Servlet 对象提供了访问 Servlet 信息的方法和变量。Servlet 对象包括 page 对象和 config 对象。

1. page 对象

page 对象是 java. lang. Object 类的一个实例，使用 page 对象可以访问 Servlet 类的所有变量和方法。下面举例说明，程序代码如下：

```
<% @ page info="我的信息" contentType="text/html;charset=GBK" %>
<html>
  <body>
    <%=((javax.servlet.jsp.HttpJspPage)page).getServletInfo() %>
  </body>
</html>
```

2. config 对象

config 对象用于存储在编译 JSP 页面的过程中创建的 Servlet 的信息，它被封装为 javax. servlet. ServletConfig 接口，当初始化一个 Servlet 时，容器把某些信息通过 config 对象传递到应用该 Servlet 的对象中。例如：

```
String propertyFile=(String)config.getInitParameter("PropertyFile");
```

### 3.4.4　错误对象

错误对象用于处理 JSP 页面中产生的错误，常用的错误对象是 exception 对象。exception 对象只能用于由 page 指令的 isErrorPage 属性指定的错误处理页。它是 java. lang. Throwable 类的实例，其主要方法如下：

(1)getMessage()：使用该方法可以返回错误信息。

(2)printStackTrace()：使用该方法可以以标准错误的形式输出一个错误和错误的堆栈。

(3)toString()：使用该方法可以以字符串的形式返回一个对异常的描述。

例如，下面的代码利用 exception 对象的 printStackTrace()方法输出错误：

```
<%@ page isErrorPage="true" %>
  <html>
```

```
<head>
  <title>错误对象使用示例</title>
</head>
<body>
  <h1>内置对象:异常</h1>
  已检测到以下错误:<br>
  <b><% = exception %></b><br>
  <% exception.printStackTrace(out); %>
</body>
</html>
```

## 3.5 JSP 动作

JSP 动作利用 XML 语法格式的标记来控制 Servlet 引擎的行为,利用 JSP 动作可以动态地插入文件。JSP 中的动作使用“jsp”作为前缀,且动作中的属性区分大小写。

### 3.5.1 <jsp:forward>动作

#### 1.语法格式及说明

<jsp:forward>动作将当前页面的执行过程终止,忽略它的输出,并重定向到一个静态资源、JSP 页面,或者一个程序段。其语法格式如下:

```
<jsp:forward page={"relativeURL"|"<% = expression %>"} />
```

如果添加参数,则其语法格式如下:

```
<jsp:forward page={"relativeURL"|"<% = expression %>"} >
<jsp:param name="parameterName" value="{parameterValue|<% = expression %>}"/>
</jsp:forward>
```

具体说明如下:

(1)<jsp:forward>动作会终止当前页面的执行。

(2)<jsp:param name="parameterName" value="{parameterValue|<% = expression %>}" />:通过此标签向一个动态文件传送参数。如果需要传递多个参数,可在 JSP 文件中使用多个<jsp:param>标签。

(3)如果用户使用缓冲区输出,即在 page 描述中定义了 buffer 的大小或使用了缺省值,则请求被重定向之前,buffer 被清空;如果使用了非缓冲输出,即 buffer=none,而某些输出已经发送到客户端,那么会抛出 IllegalStateExcetion 异常。

#### 2.属性及用法

(1)page 属性用于说明用户将要重定向的文件的文件名或 URL,这个文件可以是 JSP 程序或者其他能够处理 request 对象的文件(如 asp、cgi、php 文件等)。这个文件名可以是一个表达式或是一个字符串,它们描述了重定向文件的相对 URL。

(2)<jsp:param>标签能够向目标文件传送参数名和值,“name”指定参数名,“value”指定参数值。如果用户使用了<jsp:param>标签,参数被发送到一个动态文件,参数可以是一个或多个值,而这个文件却必须是动态文件。要传递多个参数,则可以在一个 JSP 文件中使用多个<jsp:param>将多个参数发送到一个动态文件中。

**3. 示例**

以下示例演示了利用<jsp:forward>动作实现程序跳转的方法,本例使用了两个文件:3-8.jsp和 requestedpage.jsp。

3-8.jsp 程序代码如下:

```
<html>
  <head>
    <title> 转发此页面 </title>
  </head>
  <body>
  <jsp:forward page="requestedpage.jsp"/>将用户的请求转发给 requestedpage 页面,page 属性指定了页面的地址
  </body>
</html>
```

requestedpage.jsp 程序代码如下:

```
<html>
  <head>
    <title>将请求转发到此处</title>
  </head>
  <body>
    此页面从<b>3-8.jsp</b>中收到一个转发的请求,<br>此页面是<b>
  requestedpage.jsp</b>中的输出结果,<br>但 URL 仍是<b>3-8.jsp</b>
  </body>
</html>
```

运行 3-8.jsp,<jsp:forward page="requestedpage.jsp"/>使网页转向requestedpage.jsp,运行结果如图 3-9 所示。注意网页地址栏中显示的仍是 3-8.jsp,而不是 requestedpage.jsp。


图 3-9　forward 动作使用示例

### 3.5.2 ＜jsp:include＞动作

#### 1. 语法格式及说明

＜jsp:include＞动作可以将外部的静态或动态资源(包括在当前页面中的资源)插入到JSP 页面的特定位置。其语法格式如下：

```
＜jsp:include page="{relativeURL|＜% = expression %＞}" flush="true" /＞
```

还可以使用＜jsp:param＞动作元素设置参数，语法如下：

```
＜jsp:include page="{relativeURL|＜% = expression %＞}" flush="true" ＞
＜jsp:param name="parameterName" value="{parameterValue|＜% = expression %＞}" /＞
＜/jsp:include＞
```

具体说明如下：

(1)＜jsp:include＞动作允许用户在当前 JSP 页面中插入动态文件和静态文件。这两种包含文件的结果是不同的。如果文件是静态文件，那么这种包含仅仅是把包含文件的内容加到 JSP 文件中。如果文件是动态文件，那么这个包含文件也会被 JSP 编译器执行，如请求和传送回来一个结果给这个被包含的 JSP 文件。如果文件是动态文件，那么用户还可以用＜jsp:param＞子句传递参数名和参数值。用户还能在一个页面中多次使用＜jsp:param＞子句，传递一个或多个参数给动态文件。＜jsp:include＞动作能够识别这两种文件，因此就能很方便地使用，而不需要判断此文件是动态的还是静态的。

(2)必须将 flush 设置为 true，它默认值是 false。

(3)＜jsp:param＞子句允许用户传递一个或多个参数给动态文件，也可在一个页面中使用多个＜jsp:param＞来传递多个参数给动态文件。

(4)＜jsp:include＞动作与 include 指令的区别是＜jsp:include＞动作用于在当前 JSP 页面中加入静态和动态的文件资源；include 指令用来指定怎样把另一个文件包含到当前的 JSP 页面中。include 指令应用更加灵活一些，可以实现 JSP 页面的模块化，使 JSP 的开发和维护变得简单。

(5)当这个包含文件执行完毕后，JSP 编辑器继续执行余下的 JSP 文件。

#### 2. 属性及用法

(1)page：指定被包含资源的相对路径，该路径相对于当前 JSP 页面的 URL，它不能包含协议名、端口号和域名。

(2)flush：设置是否刷新当前页面的缓冲区。

(3)param：指定附加的 request 参数。

#### 3. 示例

以下程序演示了＜jsp:include＞动作的使用方法。该程序包含两个文件：welcome.html 和 3-9.jsp，在 3-9.jsp 中包含了 welcome.html。

welcome.html 程序代码如下：

```
＜html＞
  ＜head＞
    ＜title＞欢迎！＜/title＞
```

```
    </head>
    <body>
      <h1 align="center">欢迎您来到 JSP 大家庭! </h1>
    </body>
  </html>
```

3-9.jsp 程序代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<%@ page import="java.util.*" %>
<html> <body>
<center>
<jsp:include page="welcome.html" />
此文本在显示完<b>welcome.html</b>的内容后显示
</body> </html>
```

运行 3-9.jsp，效果如图 3-10 所示。可以看到，welcome.html 的运行结果会作为 3-9.jsp 网页的一部分显示出来。


图 3-10　include 动作使用示例

### 3.5.3　<jsp:plugin>动作

#### 1. 语法格式及说明

<jsp:plugin>动作用于在 JSP 页面中嵌入在客户端运行的 Java 程序(如 Applet 和 JavaBean 等)，其语法格式如下：

```
<jsp:plugin
      type="bean|applet"
      code="classFileName"
      codebase="classFileDirectoryName"
      [ name="instanceName" ]
      [ archive="URLToArchive, …" ]
      [ align="bottom|top|middle|left|right" ]
      [ height="displayPixels" ]
      [ width="displayPixels" ]
```

```
[ hspace="leftRightPixels" ]
[ vspace="topBottomPixels" ]
[ jreversion="JREVersionNumber|1.1" ]
[ nspluginurl="URLToPlugin" ]
[ iepluginurl="URLToPlugin" ] >
[ <jsp:params>
[ <jsp:param name="parameterName" value="{parameterValue|<% =
expression %>}"/> ] </jsp:params> ]
[ <jsp:fallback> text message for user </jsp:fallback> ]
</jsp:plugin>
```

具体说明如下：

(1)当 JSP 文件被编译送往浏览器时，<jsp:plugin>动作会根据客户 Web 浏览器的情况替换最合适的<object>或者<embed>标签。

**注意**：<object>用于 HTML 4.0，<embed>用于 HTML 3.2。

(2)<jsp:plugin>动作的许多属性来自于 HTML 的<applet>和<object>标签(即 HTML 3.2 的<applet>及 HTML 4.0 的<object>)，如果用户想深入了解的话，可参考 HTML 说明书中的相关部分。

(3)一般来说，<jsp:plugin>标签会指定对象是 applet 或 bean，同样也会指定 class 的名字和位置，另外还会指定从哪里下载这个 Java 插件。

**2.属性及用法**

(1)type="bean|applet":plugin 将要执行的对象的类型，用户必须指定 bean 或者 applet，因为这个属性没有缺省值。

(2)code="classFileName":plugin 将要被 Java 插件执行的 Java 类的名字，它的名字必须包含.class 扩展名，并且这个 code 定义的 Java 插件名必须保存在 codebase 属性指定的目录中。

(3)codebase="classFileDirectoryName":指定将要被执行的 Java Class 文件的目录(或者是路径)，如果用户没有提供此属性，那么<jsp:plugin>将使用 JSP 文件的目录。

(4)name="instanceName":这个属性给出了 bean 或 applet 实例的名字，以便 JSP 文件调用。

(5)archive="URLToArchive,…":archive 的值是一些由逗号分开的文件名，这些文件名都是一些将要使用的 class，将它们预装在 codebase 目录下，可提高 applet 程序的执行效率。

(6)align="bottom|top|middle|left|right":定义了由 applet 或 bean 产生的图形在结果页面上的显示位置，具体含义如下：

- bottom:图形的底部与文本的基线对齐。
- top:图形的顶部与文本的顶部对齐。
- middle:图形的水平中心线与文本的基线对齐。
- left:图形放置在左边的空白页，文本沿着图形的右侧排列。
- right:图形放置在右边的空白页，文本沿着图形的左侧排列。

(7)height="displayPixels" 和 width="displayPixels":指定 applet 或 bean 所要显示

图形大小的初始值，displayPixels 为数字，单位为像素。

(8)hspace="leftRightPixels"积 vspace="topBottomPixels"：指定 applet 或 bean 所显示的图形在屏幕上下左右所需的余量，取值必须是一个很小的非零数字，单位为像素。

(9)jreversion="JREVersionNumber|1.1"：指定 applet 或 bean 运行所需的 Java Runtime Environment(JRE)的版本号，缺省值是 1.1。

(10)nspluginurl="URLToPlugin"：指定 Netscape Navigator 用户能够使用的 JRE 的下载地址，此值为一个标准的 URL，可以带有协议名、端口号或域名。

(11)iepluginurl="URLToPlugin"：IE 用户能够使用的 JRE 的下载地址，此值为一个标准的 URL，可以带有协议名、端口号或域名。

(12)<jsp:params>[<jsp:param name="parameterName" value="{parameterValue|<%=expression %>}" />]</jsp:params>：此属性向 applet 或 bean 传送参数名或参数值，特别是传送一个以上的参数名字和值时，可使用多个带有<jsp:params>元素的<jsp:param>标签。

(13)<jsp:fallback> text message for user </jsp:fallback>：如果 Java 插件不能启动时，提示给用户的信息。如果 Java 插件能够启动，而 applet 或 bean 不能启动，那么浏览器会弹出一个出错信息。

**3. 示例**

以下代码是 Tomcat 自带的实例，用来演示利用 plugin 动作执行一个 applet 的方法，程序如下：

```
<jsp:plugin type=applet code="Molecule.class" codebase="/html">
<jsp:params>
<jsp:param name="molecule" value="molecules/benzene.mol" />
</jsp:params>
<jsp:fallback>
<p>Unable to load applet</p>
</jsp:fallback>
</jsp:plugin>
```

### 3.5.4　<jsp:useBean>动作

**1. 语法格式及说明**

JavaBean 是可以在多个应用程序中重复使用的组件(后续章节有详细介绍)，JSP 中对 JavaBean 的使用正是通过<jsp:useBean>来完成的。<jsp:useBean>寻找或者实例化一个 JavaBean，可在 JSP 页面中提供 JavaBean 组件。其语法格式如下：

```
<jsp:useBean
        id="beanInstanceName"
        scope="page|request|session|application"
        {
        class="package.class" |
        type="package.class" |
```

```
        class="package.class" type="package.class" |
        beanName="{package.class|<% = expression %>}" type="package.
        class"
    } />
```

具体说明如下：

<jsp:useBean>首先会试图定位一个 Bean 实例，如果这个 Bean 不存在，那么<jsp:useBean>标签就会以一个 class 或相关的模版为基础，创建一个 Bean 实例。

<jsp:useBean>会按照以下步聚来定位或示例一个 Bean：

(1)试图通过用户给定的名字(name)和范围(scope)来定位一个 Bean。

(2)用用户指定的名字，来命名这个 Bean 对象的引用变量。

(3)如果这个 Bean 存在，将会在它的变量中储存这个引用。如果用户还同时指定了类型的话，那么这个 Bean 也设置为相应的类型。

(4)如果这个 Bean 不存在，将会对用户指定的 class 进行示例，并将在一个新的变量中储存这个引用。如果这个 class 的名字代表的是一个模版，那么这个 Bean 将被 java.beans.Beans.instantiate 示例。

(5)如果<jsp:useBean> 已经示例(不是定位)这个 Bean，同时<jsp:useBean> 和</jsp:useBean>中有元素，那么将会执行其中的代码。

在<jsp:useBean>元素的主体内，通常包含有<jsp:setProperty>元素，用于设置 bean 的属性值。正如上面所说的，<jsp:useBean>的主体仅仅只有在<jsp:useBean>示例 bean 时才会被执行，如果这个 bean 已经存在，<jsp:useBean>能够定位它，那么主体中的内容将不会起作用。

**2.属性及用法**

(1)id="beanInstanceName"：id 属性定义了在用户所指定范围内，用户所要提取的 Bean 的名字，用户能在后面的同一个 JSP 文件的表达式或脚本程序中看到这个变量名，它区别于不同的 Bean。这个变量名对大小写敏感，一定要注意符合用户所使用的脚本语言的规定。如果用户用的是 Java 编程语言，那么就应该遵从 Java 的语言规范。如果这个 Bean 已经在别的<jsp:useBean>中创建，那么这个 id 的值必须与那个 id 的值一致。

(2)scope="page|request|session|application"：scope 定义了 Bean 存在的范围以及 id 变量名的有效范围。缺省值是 page，详细说明如下：

- page：意味着用户能在含有<jsp:useBean>元素的 JSP 文件以及它的所有静态包含文件中使用 Bean，直到页面执行完毕向客户端发回响应或转到另一个程序为止。
- request：意味着用户能在任何执行相同请求的 JSP 文件中使用 Bean，直到页面执行完毕向客户端发回响应或转到另一个程序为止。用户能够使用 request 对象访问 Bean，如 request.getAttribute(beanInstanceName)。
- session：从创建 Bean 开始，就能在任何使用相同 session 的 JSP 文件中使用 Bean。这个 Bean 存在于整个 session 生存周期内，任何分享此 session 的 JSP 文件都能使用同一个 Bean。

**注意**：如果此处指定为 session，那么在用户创建 Bean 的 JSP 文件的<% @ page %>指令中，必须指定 session=true。

- application：从创建 Bean 开始，就能在任何使用相同 application 的 JSP 文件中使用

Bean。这个 Bean 存在于整个 application 生存周期内，任何分享此 application 的 JSP 文件都能使用同一个 Bean。

(3)class="package. class"：使用 new 关键字以及 class 构造器从一个 class 中示例一个 Bean。这个 class 不能是抽象的，必须有一个公用的、没有参数的构造器。这个 package 的名字区别大小写。

(4)type="package. class"：如果这个 Bean 已经在指定的范围中存在，那么给这个 Bean 指定一个新的数据库类型。如果用户没有使用 class 或 beanName 指定 type，Bean 将不会被示例。package 和 class 的名字区分大小写。

(5)beanName="{package. class|<%= expression %>}" type="package. class"：使用 java. beans. Beans. instantiate 方法从一个 class 或连续模版中示例一个 Bean，同时指定 Bean 的类型。beanName 可以是 package 和 class 也可以是表达式，它的值会传给 Beans. instantiate. type，其值可以和 Bean 相同。package 和 class 名字区分大小写。

### 3.5.5　<jsp:setProperty>动作

#### 1. 语法格式及说明

<jsp:setProperty>用来设置 JavaBean 在 JSP 页面中的属性，其语法格式如下：

```
<jsp:setProperty name="beanInstanceName"
      {
      property= "*" |
      property="propertyName" [ param="parameterName" ] |
      property="propertyName" value="{string|<%= expression %>}"
      }
/>
```

具体说明如下：

(1)<jsp:setProperty>动作使用 Bean 给定的 set 方法，在 JavaBean 中设置一个或多个属性值。

(2)使用<jsp:setProperty>动作之前必须首先使用<jsp:useBean>动作声明这个 Bean。因为<jsp:useBean>和<jsp:setProperty>是相互联系的，它们使用的 Bean 实例的名字也应当相匹配，即在<jsp:setProperty>中的名字要与<jsp:useBean>中的 id 定义的名字相同，并且<jsp:useBean>标签在 JSP 程序中的位置一定要排在<jsp:setProperty>标签之前。

(3)用户能用<jsp:setProperty>标签的多种方法设定属性值：通过用户输入的所有值(作为参数储存在 request 对象中)来匹配 Bean 中的属性；通过 request 对象的特定值来匹配 Bean 中的属性或不同名的属性；用一个表达式的值或特定的字符串来匹配特别设置的 Bean 的属性。

#### 2. 属性及用法

(1)name：描述了 Bean 实例的名字，而这个 Bean 实例已经存在或已经由<jsp:useBean>标签创建完成。

(2)property="*"：储存用户输入的所有数据(被称为请求参数)，用于匹配 Bean 中的

属性。Bean 中的属性的名字必须和 request 对象中的参数名一致，这个参数名通常来自于 HTML 元素和用户的输入。

从客户端传到服务器上的参数值一般都是字符类型，这些字符串为了能够与 Bean 属性相匹配，就必须转换成其他的类型，表 3-2 中列出了 Bean 属性的类型以及它们的转换方法。

**表 3-2 把字符串转化为其他类型的方法**

| Bean 属性类型 | 转换方法 |
|---|---|
| boolean | java. lang. Boolean. valueOf(String) |
| byte | java. lang. Byte. valueOf(String) |
| char | java. lang. Character. valueOf(String) |
| double | java. lang. Double. valueOf(String) |
| integer | java. lang. Integer. valueOf(String) |
| float | java. lang. Float. valueOf(String) |
| long | java. lang. Long. valueOf(String) |

如果 request 对象的参数值中有空值，那么对应的 Bean 属性将不会设定任何值。同样，如果 Bean 中有一个属性没有与之对应的 request 参数值，那么这个属性同样也不会设定值。

(3)property="propertyName" [ param="parameterName" ]：使用 request 中的一个参数值来指定 Bean 中的一个属性值。其中，property 指定 Bean 的属性名，param 指定 request 中的参数名。如果 Bean 属性和 request 参数的名字不同，那么就必须指定 property 和 param；如果相同那么就只需要指明 property 即可。如果参数值为空(或未初始化)，那么对应的 Bean 属性也不会被设定。

(4)property="propertyName" value="{string|<%= expression %>}"：使用指定的值来设定 Bean 属性，这个值可以是字符串，也可以是表达式。如果值是字符串，那么它就会被转换成 Bean 属性的类型，如果它是一个表达式，那么这个表达式的数据类型也必须和它将要设定的属性值的类型一致。如果参数值为空，那么对应的属性值也不会被设定。

另外，用户不能在一个<jsp:setProperty>中同时使用 param 和 value 这两个属性。

**提示：**如果用户使用 property="*"，那么 Bean 的属性没有必要按 HTML 表单中的顺序排列。

### 3. 示例

以下代码利用<jsp:useBean>动作定位一个 Bean 实例 MyBean，然后利用<jsp:setProperty>动作设置其属性信息，程序如下：

```
……
<head>
    <jsp:useBean id="BeanID" class="MyBean" scope="page"/>
    <jsp:setProperty name="BeanID" property="name" value="示例"/>
</head>
……
```

### 3.5.6　＜jsp:getProperty＞动作

**1. 语法格式及说明**

＜jsp:getProperty＞动作用来获取 Bean 的属性的值并将之转化为一个字符串，然后将其插入到输出的页面中，与用来设置属性的＜jsp:setProperty＞动作相对应。其语法格式如下：

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

具体说明如下：

(1)使用＜jsp:getProperty＞动作将获得 Bean 的属性值，并可以将其显示在 JSP 结果页面中。但是如果在使用＜jsp:getProperty＞之前，不存在可利用的＜jsp:useBean＞实例的话，用户必须首先使用＜jsp:useBean＞创建实例。

(2)＜jsp:getProperty＞动作的使用在 JSP 1.0 版中是有限制的，即用户不能用＜jsp:getProperty＞动作获取一个已索引的属性的值。用户能够和 JavaBean 组件结合使用＜jsp:getProperty＞，但是不能与 Enterprise Bean 结合使用＜jsp:getProperty＞。

**2. 属性及说明**

(1) name＝"beanInstanceName"：name 为必选属性，其值为 Bean 的名称，由＜jsp:useBean＞动作标记中的 id 指出。

(2)property＝"propertyName"：property 也为必选属性，其值为 Bean 属性的名字，可以显示。

**3. 示例**

以下代码利用＜jsp:useBean＞动作定位一个 Bean 实例 MyBean，然后利用＜jsp:setProperty＞动作设置其属性信息，最后利用＜jsp:getProperty＞动作获取它的属性值，程序如下：

```
……
＜head＞
    ＜jsp:useBean id="BeanID" class="MyBean" scope="page"/＞
    ＜jsp:setProperty name="BeanID" property="name" value="示例"/＞
＜/head＞
＜body＞
    ＜jsp:getProperty name="BeanID" property="name"/＞
＜/body＞
……
```

## 本 章 小 结

本章在介绍 JSP 的通用语法规则和脚本元素的基础上，详细介绍了 JSP 的内置对象、指令和动作。通过本章的学习，要求对 JSP 的基本语法有一个较为全面的认识，能够利用相应的指令、内置对象和动作等进行基础的编程。

# 习 题 3

## 一、简答题

1. JSP 中的注释有哪几种?
2. 简述 JSP 中的常用内置对象。
3. 简述 JSP 的常用指令。
4. 简述 JSP 的常用动作。

## 二、操作题

1. 将本章中的代码补充完整并进行调试。
2. 设计程序利用 JSP 的内置对象获取表单上的数据。
3. 设计程序实现多页面间的跳转操作。