



第 1 章 背景知识简介

本书的重点内容是讲解 Java Web 开发的知识，在本章中，首先简单介绍了 Java 语言的历史和现状，然后对网站运行的基本知识进行了简单的介绍，在本章的最后，对比了各种动态开发语言之间，介绍了各种动态 Web 开发语言的优劣，通过本章知识的学习，读者可以掌握 Java Web 开发所需的基本知识。

1.1 Java 语言简介

Java 是一种跨平台的面向对象语言，是由 Sun 公司于 1995 年推出，自从 Java 语言问世以来，受到越来越多开发者的喜爱，在 Java 语言出现以前，很难想象在 Window 环境下编写的程序可以不加修改就在 Linux 系统中运行，因为计算机硬件只识别机器指令，而不同操作系统中的机器指令是有所不同的，所以，要把一种平台下的程序迁移到另一个平台，必须要针对目标平台进行修改，如果想要程序运行在不同的操作系统，就要求程序设计语言能够跨平台，可以跨越不同的硬件、软件环境，而 Java 语言就能够满足这种要求。

Java 语言的目标就是为了满足在复杂的网络环境中开发软件，在这种复杂的网络环境中，充满这各种各样的硬件平台和不同的软件环境，而 Java 语言就是针对这种复杂的平台环境设计，使用 Java 语言，可以开发出适应这种复杂网络环境的应用系统。

1.1.1 Java 语言简介

Java 是一种优秀的面向对象语言，在 Java 语言中，有着健壮的安全设计，它的结构是中立的，可以一直到不同的系统平台，优秀的多线程设计也是 Java 语言的一大特色，但是 Java 语言的最大优势还是在于其对多种操作系统平台的支持，这种特性是其他编程语言所无法比拟的。

Java 最初是 Sun 公司为消费电子产品设计的一种通用环境，最初的目的是为了开发一种与平台无关的编程语言，这种技术在开始的时候并没有太大用武之地，在网络出现以后，由于网络软硬件环境的复杂性，常见的编程语言逐渐不能适应这种环境的要求，而 Java 语言平台无关性的特性正好适应网络这个潮流。所以 Java 语言在网络出现以后得到突飞猛进的发展。

目前，Java 语言最大的用途就是 Web 应用的开发，使用 Java 语言可以不用考虑系统平台的差异，在一种系统下开发的应用系统，可以不加任何修改就能运行在另一种不同的系统中，例如在目前的 Web 应用开发中，很多用户会选择使用 Linux 或者是 Unix 作为服务器环境，而开发人员一般情况下会选择在 Windows 平台下开发，因为在 Windows 平台中的开发环境的效率要相对较高点，在这种情况下，就可以使用 Java 语言，由于 Java 语言是跨平台的，所以在 Windows 中开发出的系统，可以直接部署在 Linux 或者是 Unix 的服务器系统中，这就是使用 Java 语言的便利之处。

1.1.2 Java 语言的特性和优势

在目前的软件开发中，尤其是应用系统的开发中，Java 语言成为大部分开发人员的选择，经常会有

用户自己提出要使用 Java 语言进行开发，可见 Java 语言的发展已经是深入人心，Java 语言之所以如此受欢迎，是由其自身的优点决定的，以下简单介绍 Java 语言的特性：

（1）平台无关性

平台无关性是 Java 语言最大的优势，在 Java 中，并不是直接把源文件编译成硬件可以识别的机器指令，Java 的编译器把 Java 源代码编译为字节码文件，这种字节码文件就是编译 Java 源程序时得到的 class 类文件，Java 语言的跨平台主要是指字节码文件可以在任何软硬件平台上运行，而执行这种类文件的就是 Java 虚拟机，Java 虚拟机是软件模拟出的计算机，可以执行编译 Java 源文件得到的中间码文件，而各种平台的差异就是由 Java 虚拟机来处理的，由 Java 虚拟机把中间码文件解释成目标平台可以识别的机器指令，从而实现了在各种平台中运行 Java 程序的目的，在 Java 语言中针对不同的平台环境提供了不同的 Java 虚拟机，例如在 Sun 的官方网站中就提供了 Windows、Linux 和 Solaris 等各种版本 Java 虚拟机的下载。

（2）安全性

在 C/C++ 中，指针的使用是一个高级话题，如果熟练掌握指针可以给程序的开发带来很大的方便，但是对于如果指针使用不当，就有可能带来系统资源泄漏，更严重的是错误的指针操作有可能非法访问的系统文件的地址空间，从而给系统带来灾难性的破坏，所以在 C/C++ 中，在使用指针的时候，需要非常的小心。

Java 语言放弃了指针操作，在 Java 中，没有显式提供指针的操作，不提供对存储器空间直接访问的方法，所有的存取过程都有 Java 语言自身来处理，这样就可以保证系统的地址空间不会被有意或者无意的破坏。而且经过这样的处理，也可以避免系统资源的泄漏，例如在 C/C++ 中，如果指针不及时释放，就会占用系统内存空间，大量的指针不及时释放就有可能耗尽可用的内存空间。在 Java 中就不用担心这样的问题，Java 提供了一套有效的资源回收策略，会自动回收不再使用的系统资源。从而保证了系统的安全性和稳定性。

另外，Java 虚拟机在运行字节码文件的时候，会把 Java 程序的代码和数据限制在具体的内存空间内，不允许 Java 程序范围指定内存地址之外的空间，这样就可以保证 Java 程序不会破坏系统的内存空间，从而保证系统的安全性。

（3）面向对象

面向对象是现在软件开发中的主流技术，在 Java 中同样吸取了各种面向对象语言的优点，从而更加彻底的实现了面向对象的技术，在 Java 程序中，基本所有的操作都是在对象的基础上实现的，为了实现模块化和信息的隐藏，Java 语言采用了功能代码封装的处理，Java 语言对继承性的实现使功能代码可以重复利用，用户可以把具体的功能代码封装成自定义的类，从而实现对代码的重用。

C++ 是一种经典的面向对象的语言，Java 语言继承了 C++ 中面向对象的理论，但是在 Java 中简化了这种面向对象的技术，去掉了一些复杂的技术，例如多继承、运算符的重载等功能。经过这样的处理，Java 中的面向对象技术变得简单容易掌握，同时保留这面向对象中核心的技术，可以是用户方便的享受面向对象技术带来的便利。

（4）异常处理

在 Java 中，提供异常处理的策略，在 Java 程序的开发中，可以对各种异常和错误进行处理。这些错误包括程序在编译和运行阶段的错误和异常，例如空指针异常、数组越界异常、类型错误等。Java 中的异常处理可以帮助用户定位处理各种错误，从而大大提高了 Java 应用程序的开发周期。而且，这种异常策略，可以捕捉到程序中的所有异常，针对不同的异常用户可以采取具体的处理方法，从而保证了应用程序在用户的控制中运行，从而保证了程序的稳定和健壮。

1.1.3 Java 语言的发展现状

Java 语言并不是为网络环境设计的，用户可以使用 Java 语言来编写独立的桌面应用程序，在桌面应用程序这个领域，Java 已经被各大厂商接受，例如 Oracle 数据库、Borland 的 JBuilder 开发环境，Eclipse 开发环境等工具都是使用 Java 语言编写的，这些软件产品的性能都是非常优秀的，可见使用 Java 同样可以编写出功能强大的应用软件。而且，如果用户需要开发跨平台运行的软件的时候，Java 就成了唯一的选择，跨平台的需要也是各大厂商选择使用 Java 开发桌面应用程序的原因之一。

虽然说 Java 语言并不是为网络环境设计的，但是 Java 语言目前还是主要被用于网络环境中，尤其是在服务器端的程序设计中，Java 语言的地位是其他动态语言所无法替代的。尤其是在 B/S 开发结构盛行的今天，Java 语言的地位更是举足轻重，例如，目前，各种信息管理系统都采用 B/S 进行开发，在 J2EE 中，提供了优秀的 B/S 应用程序的解决方案。再加上 Java 语言跨平台、简单易用等特性，用户自然会选择 Java 语言进行开发。事实上，在服务器端的程序开发中，Java 所占的比例份额是占这绝对优势的。

1.1.4 Java 语言的发展前景

随着网络技术的急速发展，Java 语言必然会取得更大的发展，在这个复杂的网络环境中，Java 语言有着广阔的前景。例如在如下几种开发需求中，Java 语言都有着很大的发展前景：

（1）跨平台的应用软件开发

随着 Linux、Unix 等操作系统逐渐被用户接受，Windows 的地位正面临着巨大的挑战，同时各大软件厂商也必须应对这样的变化，在这种情况下，需要兼顾各种操作系统用户的需要，当然可以选择正对不同操作系统开发出不同的软件版本，但是如果软件产品的规模超大的时候，这样的做法就不太合适，这时候就需要用到 Java，虽然桌面应用软件的开发不是 Java 的强项，但是 Java 语言跨平台的特性弥补了在这方面的不足，软件厂商采用 Java 语言进行开发，只需要开发一个版本就可以运行在不同的操作系统环境中，这就大大降低了重复开发的成本和时间。

所以，Java 语言在跨平台应用软件开发领域的前景还是非常广阔的。

（2）企业信息化解决方案

企业信息化是目前的一大潮流，而且现在的信息化解决方案中，基本上采用的都是 B/S 架构，这样的架构方便应用程序的部署，而且节省了界面程序开发的成本，在客户端需要一个浏览器即可，所有的功能代码都在服务器实现。

J2EE 是 Java 的企业版本，是 Sun 公司针对企业信息化提出的一套技术解决方案，使用这些技术，可以非常方便的实现企业信息化的需求，而且在近几年中，J2EE 正以飞快的速度向前发展，相信在未来几年中，Java 在企业信息化建设中会占到更大的比重。

（3）嵌入设备

J2ME 是 Java 针对嵌入设备，例如手机等设备设计的，在 J2ME 出现之前，在嵌入设置中编程只能选择使用 C/C++，这样底层的编程是相当复杂的，当 J2ME 技术问世以后，就可以使 Java 语言十分方便的开发嵌入设备中的应用软件，目前，J2ME 在手机中使用的比较多，各大手机厂商推出的手机产品中基本都会内置支持 Java 的功能。所以，J2ME 的使用也会越来越多。

Java 语言不断的在发展和完善，现在各大厂商都在努力推动 Java 技术的发展，在这些厂商中间，Sun、IBM、Sybase 等做的都相当出色，而且在 Java 领域中，还有开源力量的支持，例如 Apache、JBoss 等，这些开发力量给 Java 的发展带来巨大的推动作用，很多优秀的 Java 框架都是有这些开源力量开发维护的。

在各方面力量的努力下,Java 语言会越来越趋于完美,使用 Java 开发会给用户带来更大的乐趣和更高的效率。

1.2 Web 应用程序开发基本知识

Java Web 开发也就是基于 B/S 结构的 Java 应用程序开发,在接下来的章节中,将介绍 Java Web 开发最基本的知识,在这里不涉及具体的技术实现,只对 Java Web 开发的基本原理进行介绍。

1.2.1 Web 应用程序的运行原理

在传统的 Web 应用程序开发中,需要同时开发客户端和服务器的程序,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访问服务器提供的服务,这种 Web 应用程序的开发模式就是传统的 C/S 开发模式,在这种模式中,由服务器端和客户端的共同配合来完成复杂的业务逻辑。例如以前的网络软件中,一般都会采用这种模式,而且现在的网络游戏中,一般还会采用这种 Web 开发模式,在这些 Web 应用程序中,都是需要用户安装客户端才可以使用的。

在目前的 Web 应用程序开发中,一般情况下会采用另一种开发模式,在这种开发模式中,不在单独开发客户端软件,客户端只需要一个浏览器即可,这个浏览器在每个操作系统中都是自带的,软件开发人员只需专注开发服务器端的功能,用户通过浏览器就可以访问服务器端提供的服务,这种开发模式就是当前流行的 B/S 架构,在这种架构中,只需要开发服务器端的程序功能,而无需考虑客户端软件的开发,客户通过一个浏览器就可以访问应用系统提供的功能。这种架构是目前 Web 应用程序的主要开发模式,例如各大门户网站、各种 Web 信息管理系统等,使用 B/S 的架构加快 Web 应用程序开发的速度,提高了开发效率。

1.2.2 Web 服务器汇总

在 C/S 架构的开发模式中,服务器端完全是有开发人员自己提供,开发人员自己制定客户端的访问规则,这时候的服务器就是不仅要提供逻辑功能的服务,还要提供一点的协议支持,通过这样的协议,客户端程序才可以与服务器端进行通信,从而享受服务器端提供的服务。

在 B/S 架构的开发模式中,客户端就是简单的浏览器程序,可以通过 HTTP 协议访问服务器端的应用,在服务器端,与通信相关的处理都是由服务器软件负责,这些服务器软件都是有第三方的软件厂商提供,开发人员只需要把功能代码部署在 Web 服务器中,客户端就可以通过浏览器访问到这些功能代码,从而实现向客户提供的服务,下面简单介绍 B/S 结构中常用的服务器。

- ❑ IIS 是微软提供的一种 Web 服务器,提供对 ASP 语言的良好支持,通过插件的安装,也可以提供对 PHP 语言的支持。
- ❑ Apache 服务器是由 Apache 基金组织提供的一种 Web 服务器,其特长是处理静态页面,对于静态页面的处理效率非常高。
- ❑ Tomcat 也是 Apache 基金组织提供的一种 Web 服务器,提供对 JSP 和 Servlet 的支持,通过插件的安装,同样可以提供对 PHP 语言的支持,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。
- ❑ JBoss 是一个开源的重量级的 Java Web 服务器,在 JBoss 中,提供对 J2EE 各种规范的良好支持,

而且 JBoss 通过了 Sun 公司的 J2EE 认证，是 Sun 公司认可的 J2EE 容器。

- 另外 J2EE 的服务器还有 BEA 的 Weblogic 和 IBM 的 WebSphere 等，这些产品的性能都是非常优秀的，可以提供对 J2EE 的良好支持。用户可以根据自己的需要选择合适的服务器产品。

1.2.3 开发一个 Web 应用程序的简单流程

在传统 Web 应用程序的开发过程中，开发一个应用系统一般情况下需要以下几个步骤：客户端/服务器端软件的开发、服务器端程序的部署、客户端软件的安装，只有完成这几个步骤，用户才可以通过客户端访问服务器提供的服务。

而在基于 B/S 架构的 Web 程序大开发过程中，只需要开发服务器端的功能代码，然后把服务器端的程序部署在 Web 服务器软件中即可，在部署结束之后，启动 Web 服务器，用户就可以通过浏览器访问 Web 应用程序提供的服务。

1.3 Web 应用程序开发

由于技术的进步和网络环境的进化，Web 应用程序开发的技术也在不断的进步，在 Web 应用程序开发的过程中，存在着不少争议，当然，这些争议都是开发人员对各种技术的看法不同造成的，在接下来的内容中，简单介绍这方面的内容，是读者对技术进化过程中的一些问题有所了解。

1.3.1 C/S 与 B/S 之争

在前面的章节中已经介绍过，在 Web 应用程序的开发中，存在这两种开发模式，一种是传统的 C/S 架构，另一种是近些兴起的 B/S 架构。

由于硬件成本的降低，再加上应用系统复杂程度的提高，Web 应用程序的开发逐渐转向到 C/S 架构，所谓的 C/S 架构就是客户端/服务器端的架构形式，在这种架构方式中，多个客户端围绕这一个或者多个服务器，这些客户端是安装在客户机上，负责用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，每个服务器端都分担这服务器的压力，这些客户端可以根据不同的用户的需求进行定制。C/S 这种架构方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 架构的弊端开始慢慢显现，在 C/S 架构中，系统部署的时候需要在每个用户的机器上安装客户端，这样的处理方式带来很大的工作量，而且在 C/S 架构中，软件的升级也是很麻烦的一件事情，哪怕是再小的一点改动，都得把所有的客户端全部修改更新，这些致命的弱点决定了 C/S 结构的命运。在 C/S 架构模式流行一段时间以后，逐渐被另一种 Web 应用系统的架构方式所代替。这种新的 Web 软件架构的模式就是 B/S。

B/S 架构就是浏览器/服务器的架构形式，在这种架构方式中，采取了基于浏览器的策略，简化了客户端的开发工作，在 B/S 架构的客户机中，不用安装客户端软件，只要有通用的浏览器工具，就可以访问服务器端提供的服务。在各种操作系统中，都提供了浏览器中工具，这些浏览器工具都是遵循这相同的协议规范，所以 B/S 的结构客户端在各种系统环境中都已经实现。而且，在浏览器访问服务器的过程中，使用的 HTTP 协议，所以这种方式非常容易就可以穿过防火墙的限制。

而且在 B/S 结构的服务器端，也不用处理通信相关的问题，这些问题都由 Web 服务器提供，Web

服务器处理用户的 HTTP 请求，开发人员只需要专注开发业务逻辑功能即可，总之，Web 服务器完成了底层的操作，给应用软件的开发提供了最基础的通信服务，从而减轻了开发人员重复开发通信相关的功能，从而提高了开发的效率，降低了 B/S 结构应用程序开发的难度。

使用 B/S 架构，不仅开发减轻了开发的任务，而且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，而客户端根部不需要做任何改动，这是在 C/S 架构中无法实现的。

但是在 B/S 架构中也有自身存在的一些缺点，例如界面元素单调，在 B/S 结构的程序中，失去了桌面应用程序丰富的用户界面，程序在交互性上没有 C/S 架构的人性化。

在 C/S 和 B/S 两种架构之间，并没有严格的界限，两种架构之间没有好坏之分，使用这两种架构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，例如需要丰富的用户体验，那就选择 C/S 架构，例如在目前的网络游戏中，基本都是选择 C/S 架构；如果更偏重的是功能服务方面的实现，就需要选择 B/S 架构，这也正是目前绝大部分管理应用系统采用的软件架构方法。

1.3.2 动态页面语言对比

在互联网发展的最初阶段，所有的网页内容都是静态的 HTML 网页，在这种情况下，网站所能实现的任务仅仅是静态的信息展示，而不能与客户产生互动，当然这样的网站是不能满足用户不同的需要。在现实的生活中，用户的需要总是各种各样的，这就需要网站或者是 Web 应用程序具有收集并处理响应用户需要的功能，而静态的 HTML 是不能满足这种需要的，为了满足这种特殊的需要，就有了后来一系列的动态页面语言的出现。

所谓的动态页面是指可以和用户产生交互，能根据用户的输入信息产生对应的响应，能满足这种需求的语言就可以称之为动态语言。

在最早的时候，动态网页技术主要使用 CGI，现在常用的动态网页技术有 ASP、JSP、PHP 等，下面分别介绍这几种动态语言：

(1) CGI

在互联网发展的早期，动态网页技术主要使用 CGI（共用网关接口），CGI 程序被用来解释处理表单中的输入信息，并在服务器中产生对应的操作处理，或者是把处理结果返回给客户端的浏览器，从而可以给静态的 HTML 网页添加上动态的功能。但是由于 CGI 程序的编程比较困难、效率低下，而且修改维护也比较复杂，所以在一段时间以后，CGI 逐渐被其他新的动态网页技术所替代。

(2) ASP

ASP 是微软公司推出的一种动态网页语言，它可以将用户的 HTTP 请求传入到 ASP 的解释器中，这个解释器对这些 ASP 脚本进行分析和执行，然后从服务器中返回处理的结果，从而实现了与用户交互的功能，ASP 的语法比较简单，对编程基础没有很高的要求，所以很容易上手，而且微软提供的开发环境的功能十分强大，这更是降低了 ASP 程序开发的难度。但是 ASP 也有其自身的缺点。ASP 在本质上还是一种脚本语言，除了使用大量的组件，没有其他办法提高效率，而且 ASP 还只能运行在 Windows 环境中，这样 Windows 自身的一些限制就制约了 ASP 的发挥，这些都是使用 ASP 无法回避的弊端。

(3) JSP

JSP（Java Server Page）是 SUN 公司开发的一种服务器端的脚本语言，自从 1999 年推出以来，逐步发展为开发 Web 应用一项重要技术。JSP 可以嵌套在 HTML 中，而且支持多个操作系统平台，一个用 JSP 开发的 Web 应用系统，不用做什么改动就可以在不同的操作系统中运行。

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，可以根据这些动态

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，所以 Java 语言的所有优势都可以在 JSP 中体现出来，尤其是 J2EE 中的强大功能，更是成为 JSP 语言发展的强大后盾。

(4) PHP

PHP 是和 JSP 类似，都是可以嵌套到 HTML 中的语言，不同之处在于，PHP 的语法比较独特，在中混合了 C、Java 等多种语法中的优秀部分，而且 PHP 网页的执行速度要被 CGI 和 ASP 等语言要快很多。在 PHP 中，提供了对常见数据库的支持，例如 SQL Server2000、MySQL、Oracle、Sybase 等，这种内置的方法使 PHP 中的数据库操作变得异常简单。而且 PHP 程序可以在 IIS 和 Apache 中运行，提供对多种操作系统平台的支持。

但是 PHP 也存在一些劣势，PHP 的开发运行环境的配置比较复杂，而且 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。

总之，各种动态语言都有着自身的优势和劣势，只有根据客户的需求来选择具体的语言。只要能够保证系统的性能和功能，选择什么语言是无关紧要的。

1.3.3 .NET 与 J2EE 之争

自从 .NET 和 J2EE 推出以来，对 J2EE 和 .NET 的比较已经不是一天两天的事了，钟情于 Windows 的用户会选择 .NET，而选择 Unix/Linux 的用户会更钟情于 J2EE，其实这两种技术都有各自的优势和不足，下面简单分析下这两种技术自身的优劣。

1. .NET 的优点

在 Windows 平台的应用程序中，对用户界面的要求比较高，所以 .NET 提供了便捷的开发环境和工具，在 Visual Studio 中，用户的界面都可以通过简单的拖拽来完成，这可视化的编程方式在 Java 中还不是成熟，.NET 的可视化编程环境是得到一些程序员支持的原因之一。

.NET 运行在 Windows 操作平台中，而且和 Windows 一样，都是微软开发的产品，所以，在 .NET 中可以访问到操作系统中的各个细节，因此可以调用系统中的各种功能，对于 J2EE 的程序来说，这样的操作就很难实现了。在 Java 中无法访问到操作系统底层的细节的。

.NET 的优点还有很多，在这里不再一一列举，现在介绍使用 .NET 的局限性。

2. .NET 的局限

.NET 只能运行在 Windows 平台中，不能跨平台，这是 .NET 最大点一个劣势。其次 .NET 是微软一个公司的产品，所有的开发设计仅仅局限在一个公司之内，而 Java 则虽然是由 Sun 公司开发，但是在发展的过程中得到了类似 IBM、BEA 这样知名公司的支持，而且还有很多开源力量的支持。这些都是 .NET 中不可能拥有的。

3. J2EE 可以弥补 .NET 的局限

而在 J2EE 中，可供使用的类库是非常广泛的，这些类库都是非常成熟的，在 Java 发展的十多年中，这些类库的功能经过了大量的检验和测试，已经十分成熟。而且在 Java 语言的跨平台的特性在这十几年的发展中经受了考验，在 J2EE 领域中，有很多的开源的资源可供使用，例如 Tomcat、JBoss 这样的 Web 服务器，还有 Spring、Hibernate、Struts 这样的开源框架，这些资源都是 Java 社区中开源力量的贡献。这些资源同样是在 .NET 中无法享受的。

.NET 和 J2EE 都是企业级应用系统的解决方案，这两种解决方案都可以很好的实现应用系统的功能，这两种解决方案之间并没有非常明显的优劣区别，关于 .NET 和 J2EE 谁好谁坏类似问题的讨论是没有意

义的，在实际的开发过程中，应该根据具体的需要来选择使用哪种技术，例如用户只要求在 Windows 系统中使用，并没有要求跨平台，那选择.NET 和 J2EE 都是可以的，如果用户要求一定在 Linux 平台中部署应用系统，那 J2EE 就是一种很好的选择，所有选择.NET 还是 J2EE 是由需求而定的，两种技术没有优劣之分。

1.4 小结

在本章内容中，对 Java Web 开发中的一些基本知识进行简单的介绍，读者通过本章的学习可以了解开发 Java Web 应用程序的一些基本的概念，而且对于 Java Web 开发中的一些存在争议的问题也有所了解，尤其是一些有争议的问题，读者可以稍加注意，在初学者中，很容易犯这些错误，例如会过多关注具体技术的优劣，期望学到一种最有用的技术，这些想法都是不可取的。技术没有高低分，只有应用场合的不同。所以不要花费太多的精力来考虑这种没有意义的问题。

第 2 章 Java 开发环境及开发工具

本章主要介绍 Java 开发环境的搭建，首先介绍 JDK 的下载安装和环境变量的设置，并通过一个简单的示例程序展示 JDK 的简单使用方法，对于 Java 开发工具方面，简单介绍集成开发环境 Eclipse 的基本使用方法，通过本章的学习，读者可以迅速掌握 Java 开发环境的搭建，并对 Eclipse 开发工具的基本用法有所了解。

2.1 下载安装 JDK

JDK 中包含了 Java 开发中必需的工具和 Java 程序的运行环境（即 JRE）。JDK 的安装文件可以从 <http://java.sun.com/javase/downloads/index.jsp> 下载，目前的 JDK 版本为 6.0，JDK 的安装文件有三个不同操作系统的版本，在这里我们使用 Windows 版本的 JDK，下载下来的文件为 jdk-6u2-windows-i586-p.exe。如果需要 6.0 之前版本的 JDK，也可以在 Sun 的官方网站进行下载，具体的链接地址为。<http://java.sun.com/javase/downloads/previous.jsp>，在这个地址中提供了 JDK1.3 以后的各个版本 JDK 安装文件下载服务。

在下面的内容中，对 JDK 的安装过程进行详细的说明。

(1) 双击运行下载下来的安装文件 jdk-6u2-windows-i586-p.exe，就可以进入如图 2.1 所示的 JDK 安装界面。

(2) 图 2.1 的欢迎界面在几秒中之后会自动切换到如图 2.2 所示的安装协议界面。



图 2.1 安装 JDK 的欢迎界面



图 2.2 JDK 安装协议选择界面

(3) 在图 2.2 中，提供的是 JDK 的安装许可证协议信息，在这里只有选择接受，如果选择拒绝就会退出 JDK 的安装，在这里直接单击“接受”按钮，就会进入如图 2.3 所示的安装功能选择界面。

(4) 在图 2.3 所示的界面中，可以选择需要安装的功能，在这里接受默认的安装即可，在默认的安装中已经提供了基本的 Java 开发和运行环境。单击“下一步”按钮就可以进入如图 2.4 所示的安装界面。



图 2.3 JDK 安装功能选择界面



图 2.4 JDK 安装进度界面

(5) 在如图 2.4 的 JDK 安装进度结束以后，会自动进入如图 2.5 所示的 JRE 的安装界面，在 JDK 的默认安装中，会自动安装 JRE 的功能模块。

(6) 在如图 2.5 的安装界面中，可以选择 JRE 的功能模块和安装位置，在这里我们选择默认的设置即可，单击“下一步”按钮，可以进入如图 2.6 所示的 JRE 的安装进度界面。

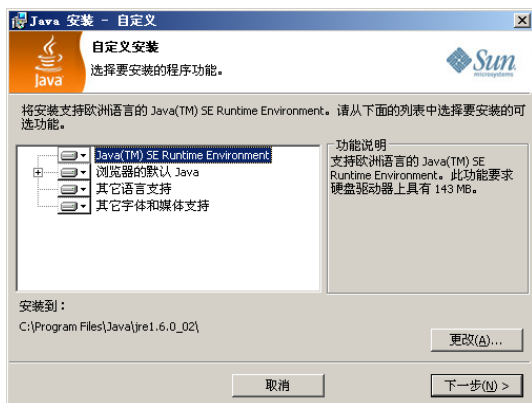


图 2.5 JRE 安装功能选择界面



图 2.6 JRE 安装进度界面

在如图 2.6 中的安装进度结束以后，就会自动进入如图 2.7 所示的结束信息提示界面。



图 2.7 JDK 安装结束界面

在图 2.6 所示的界面中单击“完成”按钮，就可以完成 JDK 的安装，到这一步为止，JDK 和 JRE 的安装工作已经全部结束，但是现在还不能马上使用 JDK 中提供的开发工具，JDK 安装结束之后，必需设置必要的环境变量，然后才能正常使用。在接下来的章节中将介绍 JDK 环境变量的设置方法。

2.2 环境变量设置

在上面的章节中，介绍了 JDK 的安装方法，但是在 JDK 安装结束之后，必需进行环境变量的设置，然后才可以使用 JDK 提供的开发工具。下面对环境变量的设置步骤进行详细的介绍。

(1) 右键单击“我的电脑”，在弹出的菜单中选择“属性”可以打开系统属性窗口，在系统属性选项卡中选择“高级”|“环境变量”，可以得到如图 2.8 所示的环境变量界面。

(2) 在如图 2.8 所示的界面中，可以对系统的环境变量进行设置，图中上半部分是用户变量，下半部分是系统变量，在下部的系统变量中，选择“Path”这个变量，然后单击“编辑”按钮，就可以进入如图 2.9 所示“Path”环境变量的编辑界面。

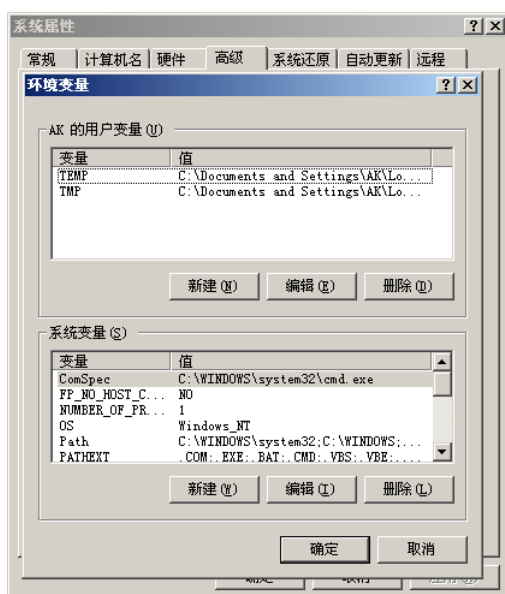


图 2.8 环境变量设置界面

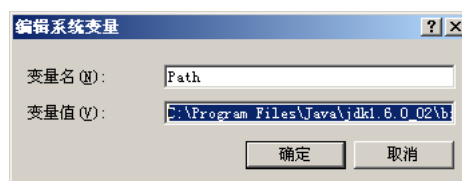


图 2.9 Path 变量编辑界面

(3) 在如图 2.8 所示的界面中，可以修改 Path 变量的值，在这里需要在 Path 变量中添加 JDK 的路径信息，例如在安装 JDK 的时候，选择的安装路径为“C:\Program Files\Java\jdk1.6.0_02”，所以需要在把 Path 变量值的最前面添加下面引号中的内容“C:\Program Files\Java\jdk1.6.0_02\bin;”，需要注意的是，在引号的内容中，最后有一个逗号，这个一定不能缺少。

经过上面的处理，JDK 的环境变量设置已经完成，下面需要测试 JDK 的安装配置是否成功。可以在 DOS 命令行中测试 JDK 是否安装成功。选择“开始”|“运行”，会弹出运行程序的选择界面，输入 cmd 回车，就可以进入 DOS 界面，在这个界面中输入 java - version，如果可以得到如图 2.10 所示的界面，说明 JDK 的安装配置已经成功。



图 2.10 JDK 安装配置测试界面

2.3 小试身手--HelloWorld

在上面两个小节的介绍中，已经成功安装配置 JDK，在 JDK 中提供了编译执行 Java 的基本工具，使用这些工具已经可以进行基本的 Java 程序的编写工作，虽然在使用继承的开发环境进行开发的效率会更高，但是，为了是读者对 JDK 的基本使用方法有基本的了解，在接下来的内容中，将不使用集成开发环境，而是通过 DOS 命令行对简单的 Java 示例程序进行编译和运行。

2.3.1 编辑 Java 源文件

在编辑 Java 源文件的时候，有很多工具可供选择，只要是能够进行简单文本编辑的工具都可以用来编辑 Java 源文件。在这里我们选择使用 Windows 中自带的记事本工具。在记事本中输入下面的代码。

```
//-----文件名: HelloWorld.java-----  
public class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("Hello world!");  
    }  
}
```

上面这段 Java 代码中，仅仅在控制台打印“Hello World!”这个字符串。在记事本中输入这些代码以后，保存为 HelloWorld.java 文件，需要注意的是，Java 的源文件名必需和类名是相同的。否则源代码的编译就不能通过。完成这些操作，一个简单的 Java 文件的编辑工作就完成了。在接下来的内容中将对这个源文件进行编译。

2.3.2 编译 Java 源文件

完成对 Java 源文件的编辑工作以后，就可以对源代码进行编译，在 JDK 中提供了编译 Java 源文件的工具，可以在 DOS 命令行中调用 JDK 中的 javac 命令，这个命令可以对 Java 源文件进行编译。

例如上面的 HelloWorld.java 保存的路径为 C:\HelloWorld.java，那么需要把 DOS 命令中的当前路径切换到 HelloWorld.java 这个 Java 源文件所在的目录，然后调用 javac 对这个源文件进行编译，具体的处理过程如图 2.11 所示。

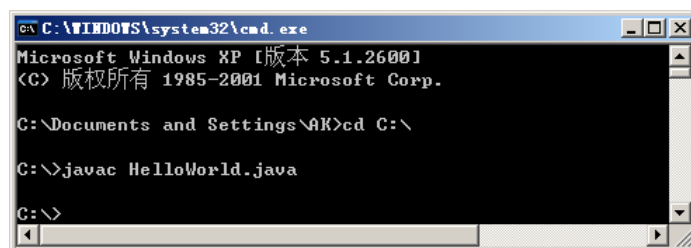


图 2.12 HelloWorld 示例程序编译过程

在图 2.12 所示的界面中，调用 javac 来编译 HelloWorld.java，在调用 javac 的时候需要注意的是，必需提供完整的 Java 源文件名，包括后缀名.java。在 javac 命令成功执行以后，会在 HelloWorld.java 的同一个目录中生成一个名为 HelloWorld.class 的文件，这是由 HelloWorld.java 源文件编译得到的 Java 类文件，在执行 Java 程序的时候，实际执行的就是编译得到的类文件。

2.3.3 执行类文件

在编译工作成功通过以后，可以得到对应的 Java 类文件，在 JDK 中同样提供了执行 Java 类文件的工具，可以在 DOS 命令行中调用 `java` 命令执行 Java 的类文件。在上面的操作中，成功编译了 `HelloWorld.java` 这个 Java 源文件，并在 C 盘的根目录下生成了 `HelloWorld.class` 文件，在 DOS 命令行中需要把当前的路径切换到 Java 类文件的目录，然后调用 JDK 中的 `java` 命令就可以执行这个类文件，具体的处理过程和运行结果如图 2.13 所示。



图 2.13 HelloWorld 示例程序运行过程

在如图 2.13 所示的界面中，调用 JDK 中的 `java` 命令执行 `HelloWorld.class` 这个类文件，在执行 `java` 命令的时候需要注意，只需要提供类文件的名称即可，不需要带 `.class` 的后缀名，JDK 中的 `java` 工具可以根据名称自动搜索所需要的类文件。

在这个简单的示例程序中，仅仅在控制台打印一个字符串，在图 2.13 中 `java` 命令执行完成就可以得到运行的结果。

在以上的内容中，简单介绍一个简单的 Java 程序的编写执行过程，通过这个程序的编译执行，读者可以对 Java 程序的开发有一个初步的认识，对于 JDK 中的 `javac` 工具和 `java` 工具也有基本的了解。其实在 JDK 中还提供了其他一系列的 Java 工具，在这里不再一一举例说明，在后面的章节中，会对涉及到的 JDK 工具的用法进行详细的说明。

2.4 开发工具 Eclipse 简介

在前面章节的内容中，介绍了直接使用 JDK 提供的工具开发一个简单的示例程序，在这个示例程序的开发过程中，没有使用任何集成的开发工具，这只是为了使读者对 JDK 的功能有一个大体的了解，在实际的开发过程中，是不可能脱离集成开发工具的帮助的，使用集成开发工具可以大大提高开发效率，从而保证项目的进度。

在本节的内容中，将简单介绍几种常用的 Java 开发工具，其中，对 Eclipse 开发平台会做比较详细的介绍。

2.4.1 Java 开发工具简介

在开发工具这方面，IBM、Borland、Sun 公司等也都推出了自己的 Java 开发工具，例如 `WSAD`、`JBuilder`、`NetBeans` 等，在开发 Java 程序的过程中，有很多开发工具可供选择，用户可以根据项目性质和用途的不同，选择适合项目需要的开发工具，目前常用的 Java 开发工具基本上可以分为两大类。

一种是简单小巧的开发工具。例如 `TextPad`、`JCreator` 等，这些开发工具的特点就是简单易用，只提

供最基本的编辑、编译、运行 Java 程序的功能，而没有提供太多的附加功能，例如在 TextPad 中，只能提供编辑、编译、运行的功能，无法创建工程，不能引入第三方类库，而且还没有提供 Debug 的功能。这类工具只适合初学者学习使用，或者开发简单的示例程序时使用，不能用来开发大型的工程。

另一种是具有强大功能的集成开发环境，例如 Eclipse、JBuilder 等，这些开发工具不仅仅提供最基本的编辑、编译功能，而且还提供了强大的附加功能，以 Eclipse 为例，在 Eclipse 中，可以创建不同类型的工程，Eclipse 可以根据工程类型的不同进行不同的配置，为用户准备好这种类型工程所需要的基本文件和基本的配置，这样用户就可以专注于项目自身业务逻辑的处理，而不用花费精力来准备配置项目的基本信息，而且在 Eclipse 中还提供了强大的 Debug 功能，用户可以通过这种功能迅速定位程序的错误，从而提高来开发的效率。

在实际的开发过程种，Eclipse、JBuilder 等集成开发工具都是很好的选择，这些集成开发工具提供的功能基本类似，读者可以根据自己的兴趣从中选择。

2.4.2 Eclipse 安装

在 Java 项目的开发过程中，越来越多的开发人员选择使用 Eclipse，在这里就介绍 Eclipse 开发环境的安装和使用。Eclipse 的安装文件可以从 <http://www.eclipse.org/downloads/> 下载，目前 Eclipse 最新的版本为 europa，下载下来的文件为 eclipse-java-europa-win32.zip，Eclipse 的安装非常简单，只需要把下载下来的文件解压缩到某路径下即可，解压后的目录结构如图 2.14 所示。

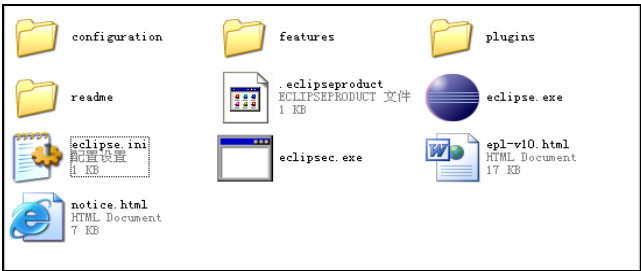


图 2.14 Eclipse 安装目录结构

在如图 2.14 所示的目录结构中，运行 eclipse.exe 就可以启动 Eclipse，Eclipse 也是用 Java 语言编写的，在使用 Eclipse 之前需要提前安装 JDK，Eclipse 可以自动从系统环境中寻找 JRE 路径，也正是这点特性，使 Eclipse 不用安装即可运行。

如果在运行 Eclipse.exe 的时候，出现如图 2.15 所示的界面，就说明 JDK 没有安装，或者是环境变量没有配置成功，这时候 Eclipse 找补到 JDK 的路径，所以就不能运行。



图 2.15 Eclipse 找不到 JDK 路径错误界面

上面这种错误情况需要重新检查 JDK 的安装配置是否成功，成功安装配置 JDK 以后，就可以正常运行 Eclipse，如图 2.16 就是 Eclipse 的开发界面。

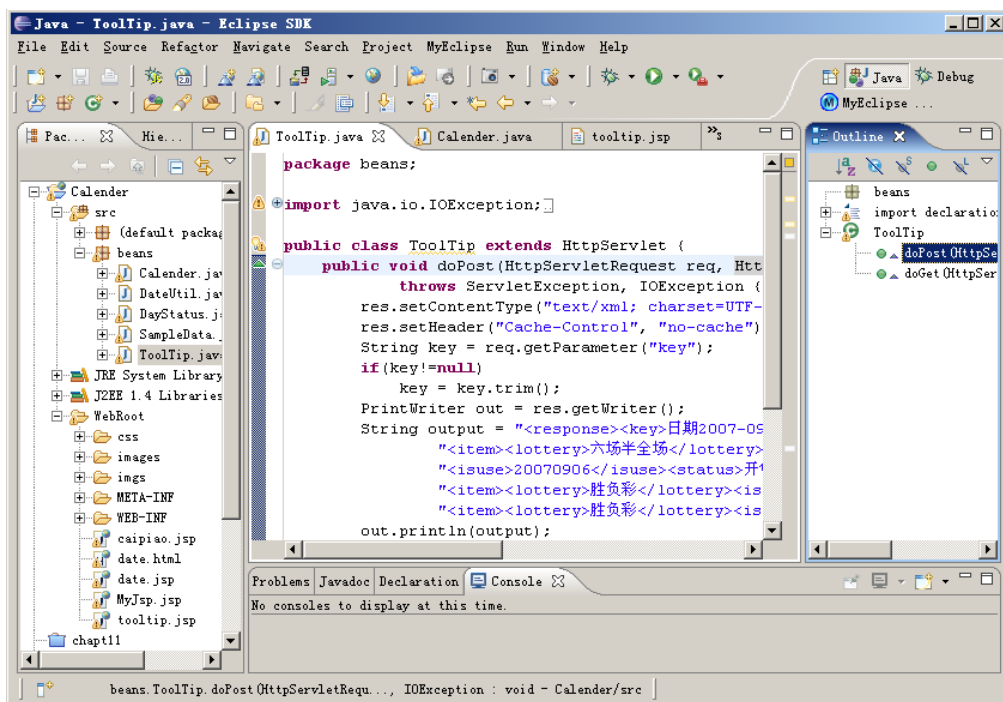


图 2.16 Eclipse 的开发界面

注意：在安装 Oracle 等软件的时候，往往会自动安装这些软件自带的 JDK，而且会修改 Path 系统变量值，这样的 JDK 版本就有可能不能支持 Eclipse 的运行，所以，在 Eclipse 不能运行，报出 JDK 错误的时候，就要考虑是否是这些软件修改了 Path 中的 JDK 路径。

2.4.3 Eclipse 使用简单例程

在 Eclipse 中，编译运行 Java 程序的方法和其他 Java 开发工具稍微有些不同，在本节的内容中，将对 Eclipse 的基本使用方法进行简单的介绍。在下面的内容中，将使用 Eclipse 创建并运行一个简单的工程。

(1) 在使用 Eclipse 进行 Java 开发的时候，首先要做的工作就是建立一个工程。在 Eclipse 的工具栏中选择“File”|“New”|“Project”就可以进入如图 2.17 所示的新建工程界面。

(2) 在如图 2.17 所示的界面中，选择“Java Project”，然后单击“Next”按钮就可以进入如图 2.18 所示的工程详细信息设置界面。

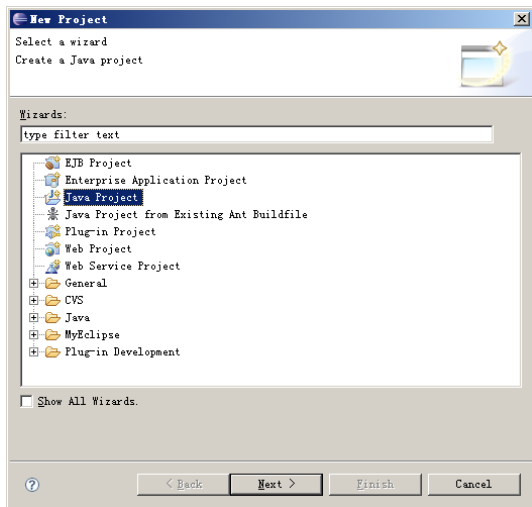


图 2.17 Eclipse 新建工程界面

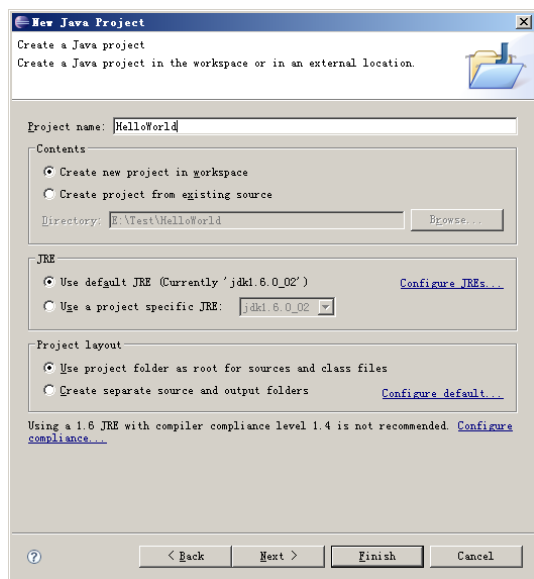


图 2.18 Eclipse 新建工程详细设置界面

因为在如图 2.18 所示的界面中，可以设置新建工程的具体信息，包括工程的名称、工程存放位置、所使用的 JRE 版本等信息。设置完这些信息以后，单击“Finish”按钮就可以结束工程的设置，这时候 Eclipse 会自动进入的工程开发视图。

(3) 现在开始创建 Java 文件，在 Eclipse 的工具栏中选择“File”|“New”|“Class”就可以进入如图 2.19 所示的新建 Java 类界面。

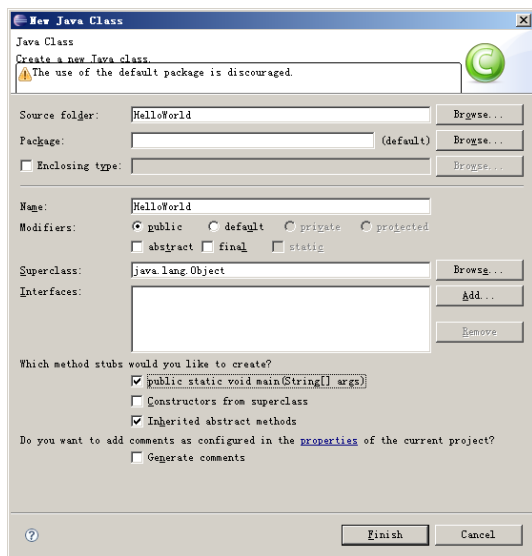


图 2.19 新建 Java 类界面

(4) 在图 2.19 所示的界面中，可以设置所要创建 Java 类的信息，例如类名、包路径、所需创建的方法等信息。在这里创建一个名为 HelloWorld 的 Java 类，选择使用默认的包路径，即把 HelloWorld 这个类放在工程的根目录下。然后在生成方法中选择自动创建 Main 方法。完成这些信息设置以后，单击“Finish”完成新建类的设置，就可以进入如图 2.20 所示的文件编辑界面。

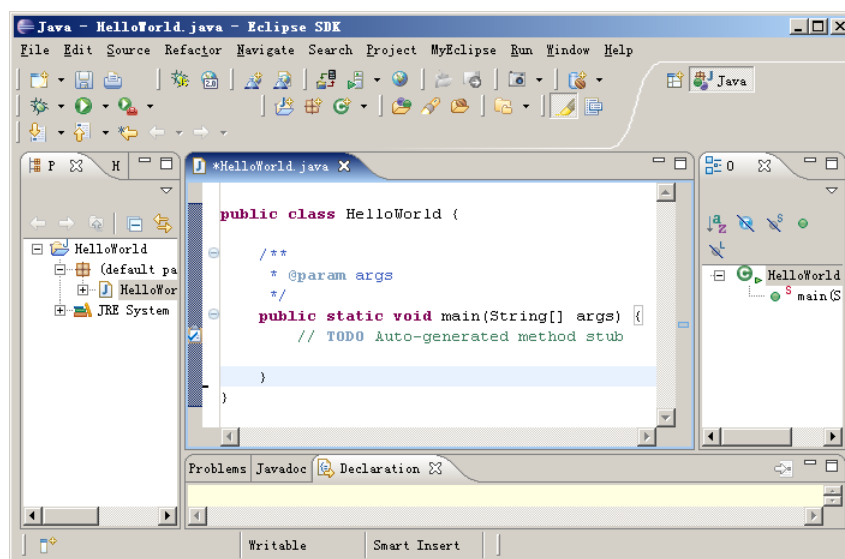


图 2.20 Eclipse 代码编辑界面

在如图 2.20 所示的界面中，已经生成了 HelloWorld 这个 Java 类的基本代码，用户只需要在 Main 方法中加入要运行的内容即可。在这里我们在 Main 方法中加入 `System.out.println("Hello World!");` 这行代码，然后保存，在保存 Java 文件的同时，Eclipse 已经自动对这个 Java 文件进行编译。

(5) 接下来要做的工作是运行 HelloWorld 这个 Java 类。在 Eclipse 开发界面左侧的目录树中右键单击 HelloWorld.java 这个文件，在弹出的菜单中选择“Run As”|“Java Application”，就可以运行 HelloWorld.java 这个文件，在如图 2.21 所示的控制台界面可以得到程序运行的结果。

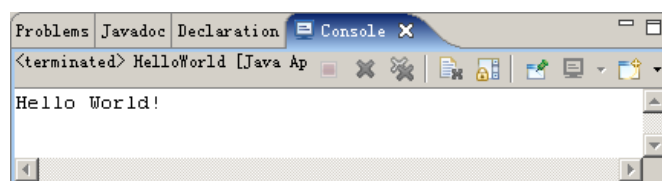


图 2.21 在 Eclipse 中 HelloWorld.java 的运行效果

2.4.4 在 Eclipse 中调试程序

在 Eclipse 中不仅可以方便的编辑执行 Java 程序，而且还提供了功能强大的调试功能，在调试 Java 程序的过程中，可以给程序设置断点，程序在运行到断点以后会暂停执行，通过设置断点，可以跟踪程序中的变量，从而对程序中的错误进行定位。

要调试程序，首先需要在 Java 源文件中添加断点，在如图 2.22 所示界面中左侧的阴影部分双击鼠标左键即可添加断点，图中的小圆点就是断点，断点和源代码中的对应行是一一对应的。如果要取消某个断点，只需要用鼠标右键单击断点，在弹出的菜单中选择“Toggle Breakpoint”就可以取消这个断点。一般情况下，只需要在怀疑程序出错的地方设置断点，用来跟踪错误。

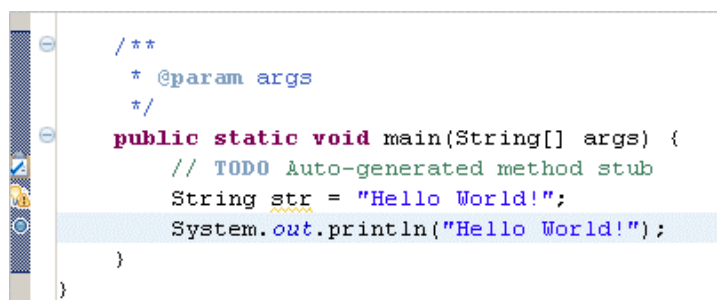


图 2.22 在 HelloWorld.java 中添加断点

在断点设置结束以后，就可以对程序进行调试，在 Eclipse 开发界面左侧的目录树中右键单击 HelloWorld.java 这个文件，在弹出的菜单中选择“Debug As”|“Java Application”，就可以进入如图 2.23 所示的调试界面。在图 2.23 所示的调试界面中，左上角的窗口中可以查看变量的值、断点和表达式，在中间的源代码窗口可以查看程序执行到的位置，在调试的过程中，默认的快捷键有 F8（恢复）、F5（进入方法）、F6（跳过语句）、F7（跳出方法），通过这些快捷键可以方便的控制调试的过程。

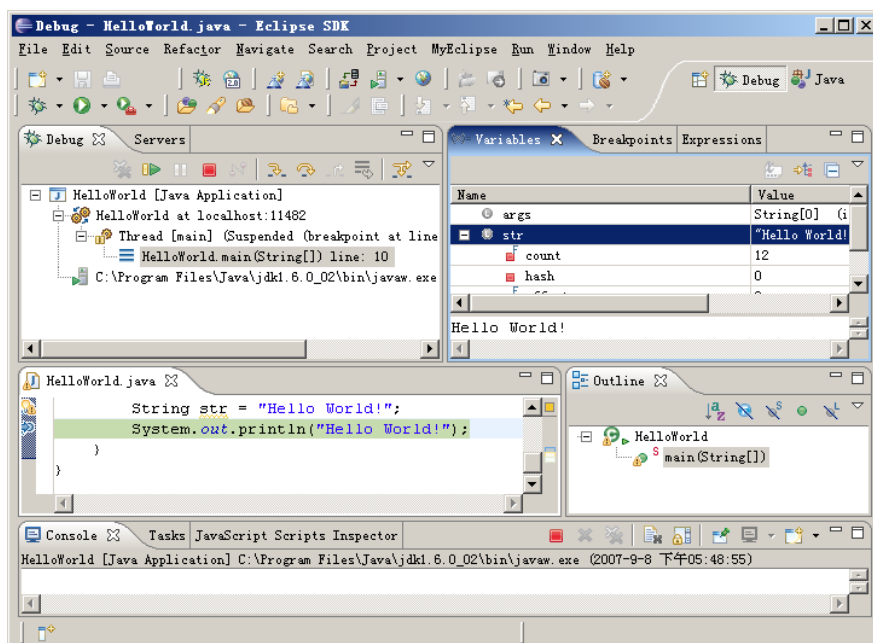


图 2.23 Eclipse 调试界面

2.4.5 Eclipse 常用快捷键

Eclipse 提供了丰富的辅助开发功能，而且很多常用的功能都提供了快捷键，在本节内容中，整理出一些相对比较常用的快捷键。编辑功能的快捷键如表 2.1 所示。

表 2.1 Eclipse 编辑快捷键

快捷键	功能	作用范围
Ctrl+F	查找、替换	全局
Ctrl+Shift+K	查找上一个	文本编辑器
Ctrl+K	查找下一个	文本编辑器
Ctrl+Z	撤销	全局

Ctrl+C	复制	全局
Alt+Shift+↓	恢复上一个选择	全局
Ctrl+X	剪切	全局
Ctrl+I	快速修正	全局
Alt+/	内容辅助	全局
Ctrl+A	全部选中	全局
Delete	删除	全局
Alt+?	上下文信息	全局
F2	显示工具提示描述	Java编辑器
Alt+Shift+↑	选择封装元素	Java编辑器
Alt+Shift+←	选择上一个元素	Java编辑器
Alt+Shift+→	选择下一个元素	Java编辑器
Ctrl+J	增量查找	文本编辑器
Ctrl+Shift+J	增量逆向查找	文本编辑器
Ctrl+V	粘贴	全局
Ctrl+Y	重做	全局

Eclipse 是一个多窗口的编辑器，在操作每个窗口的时候也提供了对应的快捷键，Eclipse 的窗口快捷键如表 2.2 所示。

表 2.2 Eclipse中的窗口快捷键

快捷键	功能	作用范围
F12	激活编辑器	全局
Ctrl+Shift+W	切换编辑器	全局
Ctrl+Shift+F6	上一个编辑器	全局
Ctrl+Shift+F7	上一个视图	全局
Ctrl+Shift+F8	上一个透视图	全局
Ctrl+F6	下一个编辑器	全局
Ctrl+F7	下一个视图	全局
Ctrl+F8	下一个透视图	全局
Ctrl+W	显示标尺上下文菜单	文本编辑器
Ctrl+F10	显示视图菜单	全局
Alt+-	显示系统菜单	全局

在 Eclipse 提供了导航的快捷键，具体设置如表 2.3 所示。

表 2.3 Eclipse中的导航快捷键

快捷键	功能	作用范围
Ctrl+F3	打开结构	Java编辑器
Ctrl+Shift+T	打开类型	全局
F4	打开类型层次结构	全局
F3	打开声明	全局
Shift+F2	打开外部javadoc	全局
Ctrl+Shift+R	打开资源	全局
Alt+←	后退历史记录	全局
Alt+→	前进历史记录	全局
Ctrl+,	上一个	全局

Ctrl+.	下一个	全局
Ctrl+O	显示大纲	Java编辑器
Ctrl+Shift+H	在层次结构中打开类型	全局
Ctrl+Shift+P	转至匹配的括号	全局
Ctrl+Q	转至上一个编辑位置	全局
Ctrl+Shift+ ↑	转至上一个成员	Java编辑器
Ctrl+Shift+ ↓	转至下一个成员	Java编辑器
Ctrl+L	转至行	文本编辑器

在 Eclipse 中提供了强大的搜索功能，这些搜索功能的快捷键如表 2.4 所示。

表 2.4 Eclipse中的搜索快捷键

快捷键	功能	作用范围
Ctrl+Shift+U	出现在文件中	全局
Ctrl+H	打开搜索对话框	全局
Ctrl+G	工作区中的声明	全局
Ctrl+ Shift+G	工作区中的引用	全局

Eclipse 中对文件的操作也提供了一系列的快捷键，具体设置如表 2.5 所示。

表 2.5 Eclipse中的文件快捷键

快捷键	功能	作用范围
Ctrl+S	保存	全局
Ctrl+P	打印	全局
Ctrl+F4	关闭	全局
Ctrl+Shift+S	全部保存	全局
Ctrl+Shift+F4	全部关闭	全局
Alt+Enter	属性	全局
Ctrl+N	新建	全局

Eclipse 中对源代码的操作也提供了一系列的快捷键，具体设置如表 2.6 所示。

表 2.6 Eclipse中的源代码快捷键

快捷键	功能	作用范围
Ctrl+Shift+F	格式化	Java编辑器
Ctrl+\	取消注释	Java编辑器
Ctrl+/	注释	Java编辑器
Ctrl+Shift+M	添加导入	Java编辑器
Ctrl+Shift+O	组织导入	Java编辑器

Eclipse 中对运行程序的操作也提供了一系列的快捷键，具体设置如表 2.7 所示。

表 2.7 Eclipse中的运行快捷键

快捷键	功能	作用范围
F7	单步返回	全局
F6	单步跳过	全局
F5	单步跳入	全局
Ctrl+F5	单步跳入选择	全局
F11	调试上次启动	全局
F8	继续	全局
Shift+F5	使用过滤器单步执行	全局
Ctrl+Shift+B	添加/去除断点	全局

Ctrl+D	显示	全局
Ctrl+F11	运行上次启动	全局
Ctrl+R	运行至行	全局
Ctrl+U	执行	全局

2.4.6 Java Web 开发工具 MyEclipse 简介

在开发 Java 桌面应用程序的时候,使用 Eclipse 是非常方便的,但是在进行 Web 开发的时候,Eclipse 的一些功能就不能够满足用户的需求了,例如在使用 Struts 或者是 Hibernate 这些开源框架的时候,在 Eclipse 中就没有很好的辅助工具,在这种情况下,自由自己安装 Eclipse 的对应插件,然而还有很多 Web 开发中的需求是 Eclipse 插件所不能满足的。

针对上面这种情况,在开发 Web 应用的时候,我们选择使用 MyEclipse 集成开发工具,MyEclipse 是依赖于 Eclipse 的一个开发工具,对 Eclipse 的功能进行了扩展,主要是给 Eclipse 增加了一系列的 Web 开发工具,从而是 Web 开发的效率大大提高。正是因为 MyEclipse 强大的 Web 开发功能,在目前的 Java Web 软件开发中,很大部分开发人员会选择是使用 Eclipse+MyEclipse 进行开发。

MyEclipse 的安装源文件可以从 MyEclipse 的官方网站 <http://www.myeclipseide.com> 下载,具体的下载地址为 <http://www.myeclipseide.com/module-htm/pages-display-pid-4.html>,在这里可以下载 MyEclipse 各种版本的安装文件,MyEclipse 的版本为 6.0,可以和 Eclipse3.3 配合使用,其他版本的 Eclipse 不能使用这个版本的 MyEclipse,在这里可以下载 All-in-One 的版本,在这个版本中,已经集成了 Eclipse,不需要另外安装 Eclipse,直接安装 MyEclipse 即可使用。MyEclipse 的安装过程比较简单,按照安装向导的指示就可以进行安装,安装以后无需设置就可以使用。

在 MyEclipse 安装成功以后,就可以创建 Web 工程,Web 工程的所有的基础配置信息都由 MyEclipse 来完成,MyEclipse 提供了一系列的 Web 开发工具,大大简化了 Java Web 开发的难度,提高了开发的效率,这样是 MyEclipse 受到开发人员喜欢的最大原因。

2.5 使用版本控制工具 CVS

在团队开发中,需要团队各个成员之间进行分工配合,这就给源代码的版本控制带来很大困难,不可能手工来完成源代码版本的迭代,在这里选择使用版本控制工具,通过版本控制工具完成对源代码的控制,各个团队成员只需要把修改过的版本提交给版本控制工具,有版本控制工具来把每个成员提交的版本整合成一个最新的版本。

在本节内容中,将介绍如何在 Eclipse 中使用版本控制工具,通过本节内容的学习,读者可以了解团队合作开发中版本控制的基本知识。

2.5.1 CVSNT 的安装与配置

在源代码版本控制方面,有很多工具可供选择,在这里选择使用 CVS 来对版本进行控制,其中 CVS 需要客户端和服务端配合使用,在使用 CVS 的时候,首先需要建立一个 CVS 服务器,然后团队中的每个成员此可以把自己的版本通过客户端提交给 CVS 服务器,从而由 CVS 服务器完成版本的整合更新任务。

在这里选择使用 CVSNT 作为 CVS 的服务器,CVSNT 的安装文件可以从 <http://www.cvsnt.org/archive/>

下载，在这里有各种版本的 CVSNT 安装文件可供下载，用户可以选择下载自己需要的版本，在本书中使用 cvsnt-2.5.03.2382，所以选择下载 cvsnt-2.5.03.2382.msi 这个文件即可。

CVSNT 的安装过程比较简单，在安装向导的指引下就可以顺行安装。安装完成以后，需要重启计算机，然后才能正常使用 CVSNT 的功能。

完成 CVSNT 安装以后，就可以配置 CVSNT 服务器，下面简单介绍 CVSNT 服务器的配置。

(1) 选择“开始”|“CVSNT”|“cvsnt control panel”，就可以进入如图 2.24 所示的 CVSNT 控制面板。在这个控制面板中，可以管理 CVSNT 的服务、添见 CVS 资源、设置 CVSNT 服务器的参数。

(2) 在如图 2.24 所示的界面中，选择“repository config”选项卡，鼠标单击“Add”按钮，就可以进入如图 2.25 所示添加资源的界面。

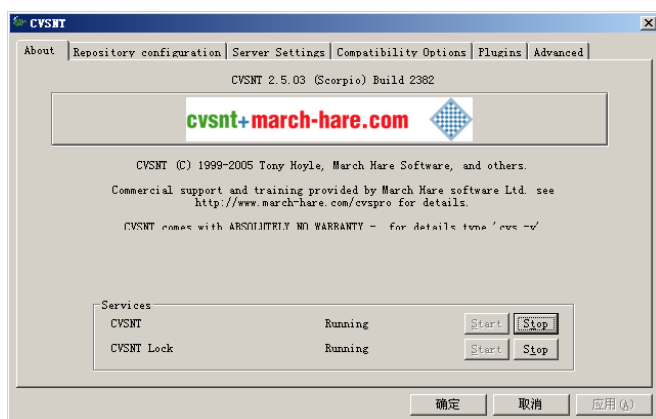


图 2.24 CVSNT 控制面板界面

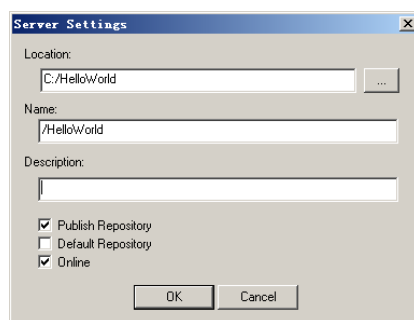


图 2.25 添加 CVS 资源界面

(3) 在如图 2.25 所示的界面中，可以把资源添加到 CVS 服务器中，其中，Location 是要添加到 CVS 服务器中的资源路径，Name 是 CVS 客户端访问这个资源需要的名称。Description 是这个 CVS 资源的描述信息。输入这些信息以后，单击“OK”按钮就完成了资源的添加。

(4) 资源添加完成以后，在如图 2.24 所示的界面中，重新启动 CVSNT 的两个服务，使刚刚添加的 CVS 资源设置生效，重启这两个服务以后，就可以通过 CVS 客户端访问这个 CVS 资源。

注意：当使用 CVS 客户端访问 CVSNT 服务器的时候，需要用户名和密码，这时候使用 CVSNT 服务器所在计算机的系统用户名和密码即可。

2.5.2 使用 Eclipse 集成的 CVS 客户端

当 CVS 服务器安装配置结束以后，就可以通过 CVS 客户端访问 CVS 服务器中的资源。其中 CVS 客户端有很多中选择，例如 WinCVS、TortoiseCVS 等，使用这些工具都可以非常方便的访问 CVS 资源，在 Eclipse 中同样也内置了 CVS 客户端的功能。在下面的内容中就简单介绍如何使用 Eclipse 内置的 CVS 客户端来访问 CVS 资源。

(1) 在 Eclipse 的菜单栏中选择“Window”|“Show View”|“Other”就可以打开如图 2.26 所示的视图选择界面。

(2) 在如图 2.26 所示的视图选择界面中，选择“CVS Repositories”，然后单击“OK”按钮，就可以打开如图 2.27 所示的 CVS 视图。

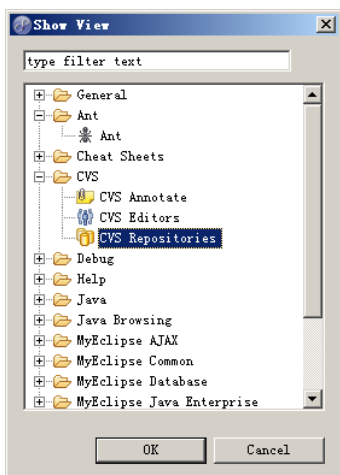


图 2.26 Eclipse 中的视图选择界面

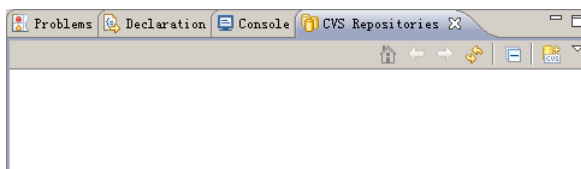


图 2.27 Eclipse 中的 CVS 视图

(3) 在如图 2.27 所示的界面中,单击鼠标右键,在弹出的菜单中选择“New”“Repository Location...”就可以进入如图 2.28 所示的连接 CVS 服务器的信息配置界面。

(4) 在如图 2.28 所示的界面中,Host 是 CVS 服务器所在的 IP 地址,在这里 CVSNT 的服务器就安装在本机,所以输入 localhost 即可,Repository Path 即 CVS 资源的访问名称,也就是在 CVSNT 中设置的资源名称,用户名和密码使用 CVSNT 服务器所在计算机的用户名和密码即可。其他设置使用默认值即可。设置完这些信息以后,单击“Finish”完成设置,就可以把这个 CVS 地址添加在如图 2.27 所示的 CVS 视图中。

(5) 在 CVS 视图中右键单击上面添加成功的 CVS 服务器路径,在弹出的菜单中选择“Refresh Branches”即可连接到 CVS 服务器,进入如图 2.29 所示的 CVS 资源选择界面。

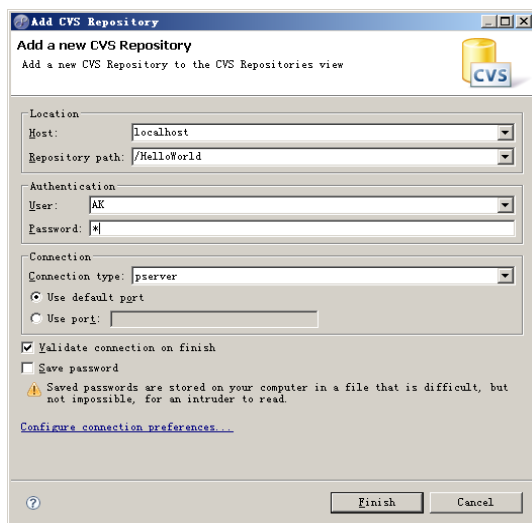


图 2.28 添加 CVS 服务器信息界面

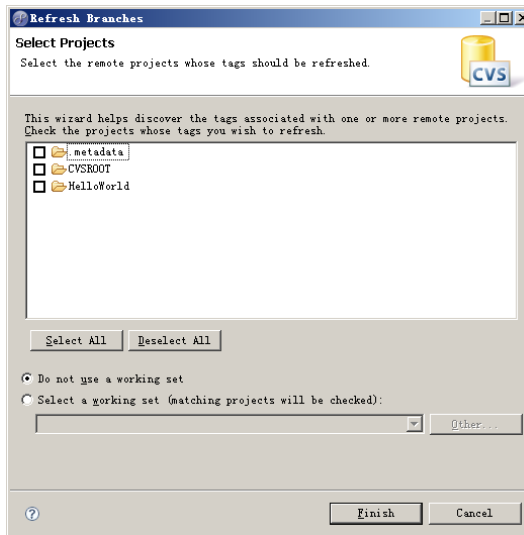


图 2.29 CVS 资源选择界面

(6) 在如图 2.29 所示的界面中,选择需要的资源,然后单击“Finish”按钮,就可以把 CVS 服务器上的这个资源导入到 Eclipse 中。

上面这些操作完成了把 CVS 服务器中的资源导入到 Eclipse 中的工作,然后既可以通过 Eclipse 来进行各种 CVS 的操作,例如更新到最新版本、提及自己的版本、恢复到历史版本等操作。这些工作都可以在 Eclipse 中完成。

2.6 小结

在本章内容中，对 Java 开发环境的搭建进行了大体的介绍，其中重点讲述了 JDK 的安装设置和 Eclipse 的基本使用方法，而且还提供了大量 Eclipse 中的快捷键，在本章最后的内容中，介绍了团队写作中源代码的版本控制问题，介绍了如何架设 CVS 服务器，如何使用 Eclipse 中内置的 CVS 客户端访问 CVS 服务器，通过本章内容的学习，读者可以对基本了解 Java 开发环境的基本知识，并且学会自己搭建设置这样的环境，为后面章节中的开发打下坚实的基础。这些技能都是在实际开发过程中必备的基础技能。读者需要熟练掌握。

第 3 章 HTML 相关技术基础知识

纵观各种动态页面开发技术，无论是 JSP、ASP 还是 PHP 都无法摆脱 HTML 的影子。这些动态的页面开发技术无非是在静态 HTML 页面的基础上添加了动态的可以交互的内容。HTML 是所有动态页面开发技术的基础。在接下来的章节将要详细介绍的就是 HTML 相关的一系列技术，包括 HTML、JavaScript 和 CSS。其中 HTML 是一组标签，负责网页的基本表现形式；JavaScript 是在客户端浏览器运行的语言，负责在客户端与用户的互动；CSS 是一个样式表，起到美化整个页面的功能。

本书不是详细介绍 HTML 的专著，在本章也只是讲解 Web 开发中最常见的 HTML 知识，目的在于使读者能尽快进入 Web 开发的状态。如果读者有更深层次的需求可以参考专门讲解 HTML 的书籍。

3.1 HTML 基础知识

在各种 Web 开发技术中，HTML 无疑是最为基础的。任何动态语言都离不开 HTML 的支持。所以在开始 Web 开发的学习之前，读者还是需要静下心来打好这个基础。在这个章节中将会概述 HTML 的框架知识。

3.1.1 什么是 HTML

HTML（Hyper Text Markup Language）即超文本标记语言，用来描述 Web 文档数据。用户可以通过 URL 链接来访问这种 Web 文档，从而达到信息展示、信息共享的目的。下面就是一个简单的 HTML 文档的例子。

```
//-----文件名: First.html-----  
<html>  
  <head>  
    <title>这是第一个 HTML 例子</title>  
  </head>  
  <body>  
    欢迎光临！这是我的第一个 HTML 文档。 <br/>  
  </body>  
</html>
```

在这个 HTML 文档中，可以看出 HTML 的简单结构，每个 HTML 文档都包括一对<html></html>标签，这是所有 HTML 文档所必需的。在这个标签中间还包括着其他两对<head></head>、<body></body>，其中在<head></head>中是 HTML 文档的头信息，包括标题、关键字、页面编码格式、引入的 CSS 或者是 JavaScript 文件的路径等基本信息。在<body></body>中间放置的是文档要表述展示的内容，在上面这个例子中我们要展示的仅仅是“欢迎光临！这是我的第一个 HTML 文档。”这句话。所以在<body></body>这对标签中间没有其他内容。

注意：一般情况下，可以包括其他内容的 HTML 标签都是成对出现的，例如上面例子中的 `<title></title>` 这对标签，它包含了一个文字的标题信息，所以成对出现。而 `
` 这样的标签仅仅是一个回车换行的作用，它不包含其他内容，所以不成对出现。

3.1.2 HTML 运行原理

在上面的例子中我们看到的仅仅是几行代码，而在大家的印象中 HTML 文档就是网页，究竟如何把 HTML 文档转化成我们常见的网页，这就是接下来要介绍的内容。

我们平时所看的丰富多彩的网页都是通过浏览器看到的，当你在浏览器中敲入网址，然后就可以看到对应的一个网页，上面有各种各样的内容。其实在本质上你敲入的网址就是互联网上一个 HTML 文档的访问路径，浏览器可以根据这个路径取回这个 HTML 文档，取回的文档格式和上面例子中的类似。只不过把这些代码翻译成了一个形象的网页。

前面介绍 HTML 定义的时候就说过，HTML 是一种标记语言，每一种 HTML 标签都是有一定表现含义的。浏览器就是按照 HTML 标签的语义规则把 HTML 代码翻译成漂亮的网页。就像上面的 HTML 文档示例会被翻译成如图 3.1 所示的效果。



图 3.1 First.html 在浏览器中的运行效果

在图 3.1 中可以明显看到，上面文档中 “`<title>这是第一个 HTML 例子</title>`” 这行代码被翻译成了网页的标题。而 `<body></body>` 标签中包含的内容被翻译成了网页只能够显示的内容，在这里就只有一句话。等接下来常用的 HTML 标签介绍结束之后，就可以构造出丰富表现形式的网页了。

3.1.3 HTML 常用标签

在本节要介绍的是常用标签的基本用法。

1. `<table>`

在 HTML 的布局标签中，`<table>` 标签是使用频率最高的一个。它可以把一组信息用表格的形式表示出来，具体使用示例参考下面这个示例。

```
//-----文件名: Table.html-----  
<html>  
  <head>  
    <title>表格标签示例</title>  
  </head>  
  <body>  
    <table border="1">  
      <tr>  
        <td>第一行第一列</td>  
        <td>第一行第二列</td>  
      </tr>  
    </table>  
  </body>  
</html>
```

```

        <td>第二行第一列</td>
        <td>第二行第二列</td>
    </tr>
</table>
</body>
</html>

```

在这个示例程序中展示了表格的基本用法，其中<table></table>标签的含义是表格，这个表格还有一个属性 border='1'，含义是这个表格的边框是 1 像素。在表格中间有<tr></tr>标签，这个标签的含义是表格的行，在行中间有<td></td>，这个标签的含义是表格的列。在<td></td>中间有文字，这些是这个单元格要显示的内容。其中<td></td>标签必需用在<tr></tr>之间，而<tr></tr>标签只可以用在<table></table>之间。如果这三对标签的位置或者是顺序用错，整个表格就不能正常显示。

按照上面介绍的标签的语义，上面这段程序表示的就是一个两行两列的表格。在浏览器中的运行效果如图 3.2 所示。



图 3.2 表格示例程序运行效果

上面这个示例程序仅仅展示了一个简单的布局，而常见的复杂布局都可以是利用表格的嵌套来实现，这也是在很长一段时间内开发人员采取最多的一种布局手段。

2. DIV

在以往的 Web 页面开发中，表格是首选的布局元素，但是近来 DIV 元素越来越受开发者的欢迎，究其原因就是定位比较方便，可以采取相对定位，也可借助于 CSS 绝对定位，同时采用 DIV 作为布局元素可以避免像表格那样的层层嵌套。

其实层也就是一个容器，在里面可以放任何需要展示的内容。和表格的使用方法类似。在章节 3.3.3 中我们将结合 CSS 详细解释 DIV 的使用方法。

3. <a>

在浏览一个网站的时候，我们经常会遇到一些链接，单击这些链接就会导航的其他的页面。这个链接使用的就是<a>超链接标签，这个标签可以像下面这个程序中这样使用。

```

//-----文件名: Link.html-----
<html>
    <head>
        <title>超链接示例</title>
    </head>
    <body>
        <a href="http://www.sohu.com" target="_blank">搜狐</a>
    </body>
</html>

```

在这个程序中定义了一个超链接。

```
<a href="http://www.sohu.com" target="_blank">搜狐</a>
```

`<a>`是超链接的标签，其中 `href` 是链接的地址，在这里我们指向 <http://www.sohu.com>，`target="_blank"`指的是在一个新的浏览器窗口打开这个链接，这个属性的值还可以选择 `_self`、`_parent`、`_top`。“搜狐”是我们显示在这个链接标签中的内容，单击这两个字的时候就跳转到链接地址。上面这个例子中 `href` 属性是必不可少的。`target` 是可选的属性。这个程序的运行效果如图 3.3 所示。

单击“搜狐”这个链接就会在新窗口打开搜狐网站主页面。

注意：在 HTML 中对字母的大小写不敏感，同样一个标签大些小写都不影响显示的效果。

4. ``

在目前的网站开发中，对图片的依赖是其他元素所不能替代的，一个漂亮的网页往往是由一系列图片组合而成。在这里介绍 HTML 中图像标签 `` 的使用方法，具体方法参考下面例子。

```
//-----文件名: Image.html-----  
<html>  
  <head>  
    <title> 图片使用示例</title>  
  </head>  
  <body>  
      
  </body>  
</html>
```

``是图像标签，`src` 属性指明了图像文件的位置，上面这个程序运行效果如图 3.4 所示。



图 3.3 超链接示例程序运行效果



图 3.4 图片使用示例程序运行效果

注意：HTML 文档的源文件都是以 `.html` 或者 `.htm` 作为后缀名。在上面这个程序中，图片和 HTML 文档在放在同一个文件夹中，所以只用给 `src` 属性提供文件名即可，如果图片和 HTML 文档不在同一个文件夹中就需要提供相对的访问路径。

3.1.4 HTML 表单标签

前面讲述的都是 HTML 向用户展示信息的标签，在本节要介绍的内容就是 HTML 用来收集用户输入的标签。`<form></form>`是表单标签，只有在这个标签中的用户输入才会被提交给服务器。表单标签的使用方法类似下面这种格式。

```
<form action="" method="get"></form>
```

其中 `action` 属性指明了处理这个表单数据的对象。`method` 属性指明传送这个表单中的数据所使用的方法，`method` 属性有两个值可选，`get` 提交表单的时候，表单的所有的输入都会显示在浏览器的地址栏中，而且 `get` 对所能提交数据的大小也比较小，因为所有提交的内容都要显示在地址栏，而 URL 的大小是有限制的。`method` 还可以选择 `post` 方法，这个方法对提交数据的时候不在浏览器的地址栏中显示，而且所能支持数据量比较大。

在表单中可以包含用输入标签收集用户输入的数据，其中 `<input></input>`是输入标签，使用方法类

似下面这种格式。

```
<input type="text"></input>
```

其中 type 属性指明了用户的输入方式，type 可以选择的值如图 3.5 所示。



图 3.5 输入标签 type 属性取值范围

其中 button 就是一个简单的按钮；checkbox 是复选框，file 是文件选择对话框；hidden 是隐藏域，当要提交一个值又不想在页面显示的时候用到这个选项；image 是构造图片按钮；password 是密码输入框，当输入的时候输入的值自动变为小黑点；radio 是单选按钮；reset 是重置按钮，这个按钮实现的功能是把当前表单中所有已经输入的值清空；submit 是提交按钮，单击这个按钮就会把表单中所有的输入数据提交给 action 对应的处理对象，对于一个表单来说这个按钮是必不可缺的，因为表单的目的就是提交用户输入到服务器，当然少不了提交按钮；text 是文本输入框，在表单中也是用的比较多的。

除了<input></input>输入标签外，在表单中还有<select></select>标签，这个标签是一个下拉列表，其中每一个下拉选项都是由<option></option>组成的。下面给出一个简单的表单示例，仅仅是展示如何使用上面介绍的各种常用的输入标签，具体代码如下。

```
//-----文件名：Form.html-----
<html>
  <head>
    <title>表单使用示例</title>
  </head>
  <body>
    <form action="" method="post">
      用户名： <input type="text" name="userName"></input><br>
      密码： <input type="password" name="password"></input><br>
      重新输入密码:<input type="password" name="rePassword"></input><br>
      性别： <input type="radio" name="sex" value="男">男
            <input type="radio" name="sex" value="女">女<br>
      出生日期： <select name="birth">
                    <option value="0">— 请选择 —</option>
                    <option value="1981">1981</option>
                    <option value="1982">1982</option>
                    <option value="1983">1983</option>
                    <option value="1984">1984</option>
                    <option value="1985">1985</option>
                    <option value="1986">1986</option>
                </select>年<br>
      兴趣： <input name="habit" type="checkbox" value="1">音乐</input>
            <input name="habit" type="checkbox" value="2">动漫</input>
            <input name="habit" type="checkbox" value="3">电影</input><br>
      <input type="submit" value="提交"/>
    </form>
  </body>
</html>
```

```
<input type="reset" value="取消" />
</form>
</body>
</html>
```

这个程序展示了一个简单的表单，展示了最长用的几种输入标签的用法。其中。

```
性别: <input type="radio" name="sex" value="男">男
<input type="radio" name="sex" value="女">女<br>
```

单选按钮的使用需要注意，属于一组单选按钮名称必需一样，例如在这个示例中男、女选项只能选择一个，那么只有它们的名字相同才可以实现。否则两个按钮都可以选择。

程序中的提交按钮如果不提供 value 值的话会在按钮上显示默认的“提交查询内容”；同样重置按钮如果不提供 value 值，也将显示默认值“重置”。上面这个表单的运行效果如图 3.6 所示。

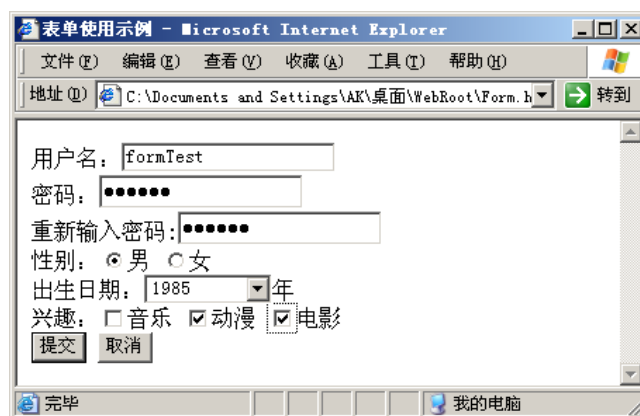


图 3.6 表单使用示例程序运行效果

3.1.5 HTML 其他标签

在本章只是介绍了 HTML 最基本最常用的几个标签，HTML 还有很多其他标签，例如应用程序标签中可以加入不同动态程序代码，多媒体标签中可以加入多媒体文件，Flash 标签中可以加入 Flash 动画，文本标签可以用各种方式组织文本内容的显示。

读者如果要深入全面的研究 HTML 标签，可以参考这方面的参考手册。在本书中不再对这些内容进行详细的介绍。

3.2 JavaScript 基础知识

JavaScript 的出现给静态的 HTML 网页带来很大的变化。JavaScript 增加了 HTML 网页的互动性，使以前单调的静态页面变得富有交互性，它可以在浏览器端实现一系列动态的功能，仅仅依靠浏览器就可以完成一些与用户的互动。在下面的章节中将要简单介绍这种技术的基础知识。

3.2.1 什么是 JavaScript

JavaScript 是一种简单的脚本语言，可以在浏览器中直接运行，无需服务器端的支持。这种脚本语言可以直接嵌套在 HTML 代码中，它响应一系列的事件，当一个 JavaScript 函数响应的动作发生时，浏

浏览器就会执行对应的 JavaScript 代码，从而在浏览器端实现与客户的交互。

3.2.2 JavaScript 中的事件

在 HTML 的标签中，绝大部分都可以触发一些事件，例如鼠标单击、双击、鼠标经过、鼠标离开等一系。JavaScript 最主要的功能就是与用户的交互，而用户只能在表单中提交输入内容，所以表单的所有输入标签都可以出发鼠标事件、键盘事件等 JavaScript 所能响应的常见事件。

在这里我们不在一一列举每一个 HTML 标签所能触发的事件，在下面仅仅以一个普通按钮所能触发的事件为例介绍 JavaScript 的事件，其他方法的处理过程都是相同的。

如图 3.7 就是一个简单的按钮所能触发的事件的列表。

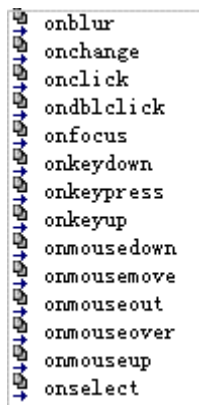


图 3.7 按钮标签所能触发的事件列表

下面介绍如何通过这样一个事件调用一个 JavaScript 函数，具体代码如下所示。

```
//-----文件名: Event.html-----
<html>
  <head>
    <title>事件触发示例</title>
    <script language="javascript">
      function test()
      {
        alert("事件触发测试! ");
      }
    </script>
  </head>
  <body>
    <form action="" method="post">
      <input type="button" value="单击事件测试" onclick="test()">
    </form>
  </body>
</html>
```

在上面这个程序中可以看出，单选按钮触发一个鼠标单击事件，当鼠标单击这个按钮的时候，浏览器就会调用 JavaScript 中的 test()这个方法，test()方法的功能就是弹出一个如图 3.8 所示的对话框，其中对话框中显示的内容就是 test()方法中 alert 中间的参数内容。



图 3.8 JavaScript 鼠标单击测试弹出的对话框

```
<script language="javascript">
  function test()
  {
    alert("事件触发测试! ");
  }
</script>
```

上面这段代码就是 JavaScript 代码的典型格式，所有的 JavaScript 代码都是以函数（function）的方式存在的，HTML 触发的动作对应的就是这里面的方法。

注意：当 JavaScript 的代码量比较大的时候，或者在多个页面都会用到同样功能的 JavaScript 代码的时候，可以把这些 JavaScript 代码放在一个以 .js 为后缀名的文件中。然后在 HTML 页面中按照 `<script src=" " "></script>` 这种格式进行引用，其中 src 是 .js 文件放置的相对路径。

3.2.3 JavaScript 中的对象简介

JavaScript 所实现的动态功能，基本上都是对 HTML 文档或者是 HTML 文档运行的环境进行的操作。那么要实现这些动态功能就必需找到相应的对象。JavaScript 中有已经定义过的对象供开发者调用，在了解这些对象之前先看图 3.9 所示的内容。

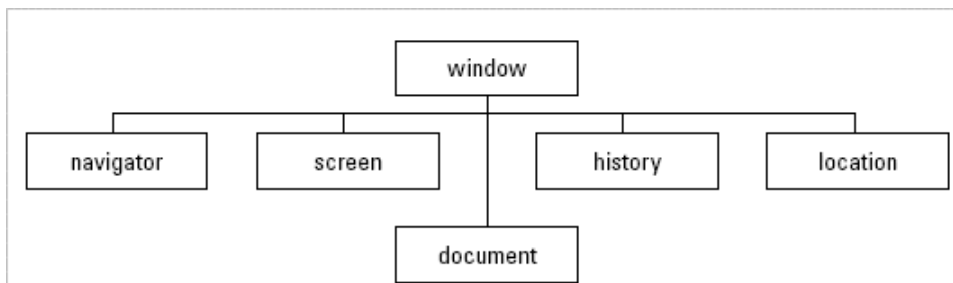


图 3.9 在浏览器窗口中的文档对象模型

图 3.9 中的内容是一个简单的 HTML 文档在浏览器窗口中的文档对象模型，其中 window、navigator、screen、history、location 都是 HTML 文档运行所需的环境对象，document 对象才是前面讲述的 HTML 文档，当然这个 document 对象还可以划分出 html、head、body 等分支。

- ❑ window 对象是所有对象中最顶层的对象，HTML 文档在 window 对象中显示。
- ❑ navigator 对象可以读取浏览器相关的信息。
- ❑ screen 对象可以读取浏览器运行的物理环境，例如屏幕的宽和高，此处的单位为像素。
- ❑ document 对象是整个网页 HTML 内容，每个 HTML 文档被浏览器加载以后都会在内存中初始化一个 document 对象。
- ❑ history 对象可以控制浏览器的前进和后退。
- ❑ location 对象可以控制页面的跳转。

在接下来的章节中将对最常用的 window 对象、location 对象、document 对象进行详细的介绍。

3.2.4 window 对象简介

如图 3.9 所示，window 对象是所有 JavaScript 对象中最顶层的对象，整个 HTML 文档就是在一个浏览器的一个窗口，即 window 对象中显示。当打开一个浏览器窗口以后，甚至是一个空白窗口，这时候在内存中间已经定义了一个 window 对象。window 对象所提供的方法很多，在下面的内容中将进行对最常用的几种方法进行介绍。

1. 窗体的创建和关闭

利用 window 对象可以新建浏览器窗口，也可以关闭浏览器窗口，下面来看具体的操作代码。

```
//-----文件名: Window.html-----
<html>
<head>
<title>窗体的创建和关闭示例</title>
<script type="text/javascript">
    var win;
    function createWin() {
        win = window.open("", "", "width=300,height=200");
    }
    function closeWin() {
        if (win) {
            win.close();
        }
    }
</script>
</head>
<body>
<form>
<input type="button" value="创建新窗口" onclick="createWin()">
<input type="button" value="关闭新窗口" onclick="closeWin()">
</form>
</body>
</html>
```

这个程序在浏览器中运行以后，界面上会有两个按钮，鼠标单击“创建新窗口”按钮会弹出一个新的浏览器窗口，这个窗口的宽为 300 像素，高为 200 像素。鼠标单击“关闭新窗口”，这个弹出窗口就会被关闭。

上面这个程序中用到的就是 window 对象的 open 和 close 两个方法，open 方法新建一个窗口，close 方法关闭指定窗口。

2. 三种常用的对话框

在 window 对象中，有三种常用的对话框，第一种是警告对话框，第二种是确认对话框，第三种是输入对话框。在面这个示例中展示了这三个对话框的用法。

```
//-----文件名: Dialog.html-----
<html>
<head>
<title>三种常用的对话框</title>
<script type="text/javascript">
    function alertDialog() {
```

```

        alert("您成功执行了这个操作。");
    }
    function confirmDialog() {
        if(window.confirm("您确认要进行这个操作吗? "))
            alert("您选择了确定! ");
        else
            alert("您选择了取消");
    }
    function promptDialog() {
        var input = window.prompt("请输入内容: ");
        if(input !=null)
        {
            window.alert("您输入的内容为"+input);
        }
    }
}
</script>
</head>
<body>
<form>
<input type="button" value="警告对话框" onclick="alertDialog()">
<input type="button" value="确认对话框" onclick="confirmDialog()">
<input type="button" value="输入对话框" onclick="promptDialog()">
</form>
</body>
</html>

```

上面这个程序在浏览器中运行以后，鼠标单击“警告对话框”按钮，会弹出如图 3.10 所示的对话框。鼠标单击“确认对话框”按钮，会弹出如图 3.11 所示的对话框。

鼠标单击“输入对话框”按钮，会弹出如图 3.12 所示的对话框。



图 3.10 警告对话框



3.11 确认对话框



图 3.12 输入对话框

在上面这个程序中，对后两种对话框的返回值也进行了示例处理，读者可以参照上面的格式稍加修改就可以用到自己的程序中去。

3.2.5 document 对象简介

document 对象是在具体的开发过程中用的最频繁的对象，利用 document 对象可以访问页面上任何的元素。通过控制这些元素可以完成与用户的互动。

1. 利用 document 定位 HTML 页面元素

所有的 HTML 页面元素都可以用 document.getElementById()这个方法访问，还有一部分 HTML 页面元素可以使用数组来访问，例如表单元素就可以使用 document.forms[“formName”]或者是 document.forms[“formIndex”]来访问，其中 formName 是表单的名称，formIndex 是表单的序号。

当 HTML 页面中使用了 iframe 时，使用 document 对象定位 iframe 中的元素时，首先要取得 iframe

的 document 对象，然后在这个对象上继续操作。这个 document 对象可以这样获得：document.frames[“framesName”].document，这里的 frameName 是 iframe 的名称，取得的这个 iframe 的 document 对象使用方法和其他 document 的使用方法是一样的，在这个 document 基础上可以继续定位 iframe 中的元素。

同样道理，如果在页面中使用了框架集 frameset 的时候，也可以采用像上面 iframe 一样的处理方法。

2. 利用 document 对象动态生成 HTML 页面

用 document 对象不仅仅可以取出或者设置 HTML 页面元素的值，而且可以动态的生成整个新的 HTML 文档。下面的例子就是利用 document 对象生成一个新的 HTML 文档。

```
//-----文件名: CreateHtml.html-----
<html>
<head>
<title>动态生成 HTML 页面</title>
<script type="text/javascript">
function create() {
    var content = "<html><head><title>动态生成的 HTML 文档</title></head>";
    content += "<body><font size='2'><b>这个文档的内容是利用 document 对象动态生成的</b></font></h1>";
    content += "</body></html>";

    var newWindow = window.open();
    newWindow.document.write(content);
    newWindow.document.close();
}
</script>
</head>
<body>
<form>
<input type="button" value="创建 HTML 文档" onclick="create()">
</form>
</body>
</html>
```

在上面这个示例程序中，利用 JavaScript 动态生成一个 HTML 代码串，并且利用 document 对象把这段代码串写入新建窗口的 document 对象中，这样就完成动态生成 HTML 页面的功能，此处如果是在原窗体显示，只需要把新建的窗体对象替换成当前窗体的 window 对象即可。

上面这个程序在浏览器中打开以后，鼠标单击“创建 HTML 文档”按钮，就会弹出一个如图 3.13 所示的新窗体，窗体内容是利用 document 对象动态创建的。

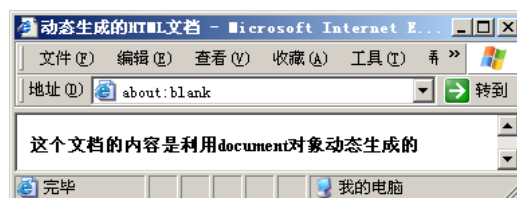


图 3.13 利用 document 对象动态生成的 HTML 页面

注意：在 JavaScript 的字符串操作中，不允许在单引号中嵌套单引号或者在双引号中嵌套双引号，但是这两种引号可以交叉使用，可以在双引号中嵌套单引号，也可以在单引号中嵌套双引号。

3.2.6 location 对象简介

在 HTML 标签中可以用[`<a>`](#)超链接标签来控制网页中的跳转，在 JavaScript 中如果要想实现类似的网页跳转功能只能选择 location 对象，这个对象的使用方法非常简单，只需要在 JavaScript 代码中添加下面这行代码即可。

```
window.location.href = "http://www.sohu.com" ;
```

window 对象就是要控制的目标窗体，赋值的内容就是窗体将要跳转到的页面，这行代码可以实现类似超链接标签的效果。

3.2.7 JavaScript 输入验证

在前面章节 3.1.4 中我们简单介绍了 HTML 表单的基本知识，在本章将介绍在浏览器端对用户输入的简单验证，这种验证仅仅局限于输入格式等方面。

在这里我们仍然使用 3.1.4 中的表单内容，只是添加了输入验证的内容，假设验证规则为：用户名、密码、重新输入密码这三项不能为空，用户名长度不能小于 6 位，两次密码输入必需相同。

下面是添加按照上面定义的这也规则验证的表单代码。

```
//-----文件名: ValidateForm.html-----  
<html>  
  <head>  
    <title>表单输入验证示例</title>  
    <script type="text/javascript">  
      function validate()  
      {  
        var userName=document.forms[0].userName.value;  
        var password=document.forms[0].password.value;  
        var rePassword=document.forms[0].rePassword.value;  
  
        if(userName.length<=0)  
          alert("用户名不能为空！");  
        else if(password<=0)  
          alert("密码不能为空！");  
        else if(rePassword.length<=0)  
          alert("重新输入密码不能为空！");  
        else if(userName.length<6)  
          alert("用户名不能小于 6 位！");  
        else if(password!=rePassword)  
          alert("两次输入密码不一致！");  
        else  
        {  
          alert("验证通过，表单可以提交！");  
          document.forms[0].submit();  
        }  
      }  
    </script>  
  </head>  
  <body>  
    <form action="" method="post">
```

```

用户名: <input type="text" name="userName"></input><br>
密码: <input type="password" name="password"></input><br>
重新输入密码:<input type="password" name="rePassword"></input><br>
性别: <input type="radio" name="sex" value="男">男
      <input type="radio" name="sex" value="女">女<br>
出生日期: <select name="birth">
            <option value="0">— 请选择 —</option>
            <option value="1981">1981</option>
            <option value="1982">1982</option>
            <option value="1983">1983</option>
            <option value="1984">1984</option>
            <option value="1985">1985</option>
            <option value="1986">1986</option>
          </select>年<br>
兴趣: <input name="habit" type="checkbox" value="1">音乐</input>
      <input name="habit" type="checkbox" value="2">动漫</input>
      <input name="habit" type="checkbox" value="3">电影</input><br>
<input type="button" value="提交" onClick="validate()"/>
<input type="reset" value="取消" />

</form>
</body>
</html>

```

这个程序针对上面的验证规则，对输入的各项进行检查，如果有一条不满足就不提交表单，例如用户没有输入密码就提交表单，就会弹出如图 3.14 所示的对话框，其他各种错误提示跟这个密码提示类似。



图 3.14 没有输入密码的提示信息

注意：在进行表单输入验证的时候，必需把<input type="submit" value="提交"/>中间的 type 换为 button，同时给这个 button 添加一个 JavaScript 事件，这时候在输入验证中使用 JavaScript 提交窗体。如果不把输入的类型改为 button，则无论输入是否合法窗体都会被提交。

3.2.8 JavaScript 高级应用探讨

上面介绍的示例中，JavaScript 都没有和服务端进行互动，都是在浏览器中独立执行，这样所能实现的与客户互动的功能就比较有限了，例如现在用户注册的时候需要验证这个用户名是否已经被占用，这个功能便需要到服务器中进行查询。然而在我们上面的验证中，只有当表单提交的时候服务器才会相应请求，其他情况下，如果没有刷新整个页面是不能实现与服务端之间的通信的。

现在有这样一种解决方案，那就是使用 Ajax，利用 Ajax 就可以实现页面的局部刷新，当输入用户名的时候可以同时进行与服务端进行通信，在数据库中进行查询，这时候只需要刷新局部的页面即可，这种技术的核心语言就是 JavaScript，通过 JavaScript 操作 XMLHttpRequest 对象来实现与服务端之间的局部通信。这种技术在第二十章中将会详细介绍。

3.3 CSS 基础知识

在前面的内容中讲解了 HTML 和 JavaScript，现在我们已经基本可以编出具有简单互动的网页，但是仅仅这样还是不够的，一个专业的网页需要在字体、颜色、布局等方面进行各种设置，需要给用户带来视觉的冲击。接下来的内容将要介绍这种美化页面的技术。

3.3.1 什么是 CSS

CSS (Cascading Style Sheets) 即层叠样式表, 也就是通常所说样式表。CSS 是一种美化网页的技术。通过使用 CSS, 可以方便、灵活地设置网页中不同元素的外观属性, 通过这些设置可以使网页在外观上达到一个更高的级别。

同时，CSS 与 JavaScript 结合可以在给用户带来更具变化的外观体验，例如下拉菜单等功能的实现都是通过 CSS 与 JavaScript 来实现的。

3.3.2 CSS 属性设置

CSS 美化网页就是通过设置页面元素的属性来实现的,在下面的内容中将介绍 CSS 属性设置的基本方法。

1. 字体、颜色、背景等属性设置

字体属性是 CSS 最基本的属性，其最常用的属性包括 `font-family`、`font-style`、`font-weight`、`font-size` 等，其中 `font-family` 定义使用哪种字体，`font-style` 定义是否使用斜体，`font-weight` 定义字体的粗细，`font-size` 定义字体的大小。

color 定义前景颜色，background-color 定义背景颜色，background-image 定义背景图片，background-repeat 定义背景图片重复方式，background-attachment 定义滚动，background-position 定义背景图片的初始位置。下面通过一个示例程序展示这些属性的设置方法。

FontColor.html

```
<html>  
    <head>  
        <title>字体、颜色、背景属性设置示例</title>  
    </head>  
    <body>  
        <div align="left">字体属性设置： </div>  
        <span style="font-family:幼圆;font-style:italic;font-weight:bold;font-size:10pt;">幼圆、斜体、  
        黑体、10pt</span><br>  
        <span style="font-family:隶书;font-size:16pt;">隶书、黑体、16pt</span><br>  
        <div align="left">字体属性设置： </div>  
        <span style="color:red;background-color:yellow;">前景红色、背景黄色</span><br>  
        <span style="background-image:url('hand.gif');background-repeat:no-repeat;">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~</span><br>  
        图片背景、不重复</span><br>  
    </body>  
</html>
```

这个页面在浏览器中的运行效果入股 3.15 所示。

2. 鼠标样式属性设置

在一些网页中，我们经常会遇到这样一种情况，当把鼠标移到不同区域，或者是在执行不同功能的时候，鼠标的形状都会发生变化。这种功能的实现其实非常简单，就是控制 CSS 中的 `cursor` 属性来实现的，其中 `cursor` 的属性列表如图 3.16 所示。

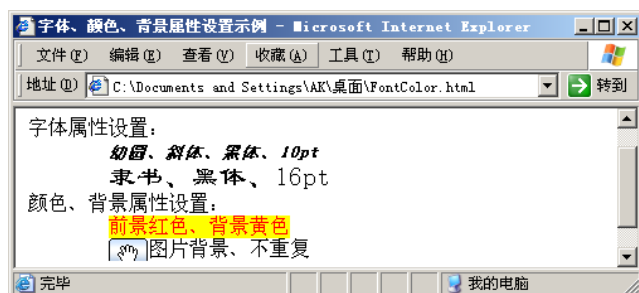


图 3.15 字体、颜色、背景属性设置运行效果

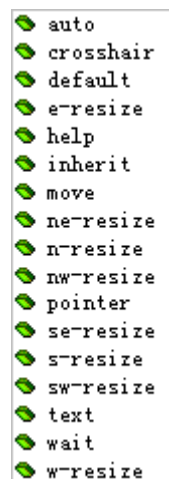


图 3.16 cursor 属性列表

在下面的例子中将要展示几种鼠标样式的设置方法，其他属性的使用方法都是相同。具体的设置方法参考下面的代码。

```
//-----文件名: Cursor.html-----  
<html>  
  <head>  
    <title>设置鼠标样式</title>  
  </head>  
  <body>  
    <div style="font-family:宋体;font-size:10pt;">  
      <span style="cursor:hand">手形状</span><br>  
      <span style="cursor:move">移动</span><br>  
      <span style="cursor:ne-resize">反方向</span><br>  
      <span style="cursor:wait">等待</span><br>  
      <span style="cursor:help">求助</span><br>  
      <span style="cursor:text">文本</span><br>  
      <span style="cursor:crosshair">十字</span><br>  
      <span style="cursor:s-resize">箭头朝下</span>  
    </div>  
  </body>  
</html>
```

上面这个网页在浏览器中打开以后，鼠标移动到不同区域就会变成不同样式。

3.3.3 CSS 绝对定位示例

在 HTML 中布局一般情况下需要使用表格，如果要定位只有通过表格的嵌套来实现，这种方法的确可以在一定程度上解决问题，但是却不能随意定位页面元素，而且对某个元素位置的改变有可能影响到整个页面的布局。

在 CSS 中提供了灵活的定位的方法，所以在页面布局中我们又多了一种可以选择的方案。在 CSS 中常用的定位属性有 position、left、top、width、height、overflow、z-index、visibility 等，其中 position 定义采用绝对定位（absolute）、相对定位（relative）还是静态定位（static），left 和 top 定义横纵坐标的位置，width 和 height 定义宽和高，overflow 定义内容超出的处理方法，z-index 定义立体效果，visibility 定义可见性。在下面的示例程序中就展示了怎么使用 CSS 实现绝对定位。

```
//-----文件名: Locate.html-----
<html>
  <head>
    <title>CSS 定位示例</title>
  </head>
  <body>
    <div style="position:absolute;left=100;top=20;visibility=visible">
      下面这个图片是可见的:
      
    </div>
    <div style="position:absolute;left=100;top=60;visibility=hidden">
      下面这个图片是不可见的:
      
    </div>
  </body>
</html>
```

这个程序在页面上不同的位置放置了两个 DIV，每个 DIV 中都有一个图片，但是第二个 DIV 设置为隐藏的，所在浏览器中的运行的时候只能显示一个图片。

3.3.4 JavaScript+DIV+CSS 实现下拉菜单

在 Web 应用中，下拉菜单的可以说是随处可见，在学习了 JavaScript 和 CSS 以后实现起来毫无难度。其原理就是在用 JavaScript 控制不同 DIV 的显示和隐藏，其中所有的 DIV 都是用 CSS 定位方法提前定义好位置和表现形式，下拉的效果只是当鼠标经过的时候触发一个事件，把对应的 DIV 内容显示出来而已。下面的例子中将会实现一个简单的下拉菜单。

```
//-----文件名: Menu.html-----
<html>
  <head>
    <title>下拉菜单示例</title>
    <script language="javascript">
      //当鼠标移到菜单选项的时候显示对应的 DIV
      function show(menu)
      {
        document.getElementById(menu).style.visibility="visible";
      }
      //当鼠标移出的时候隐藏所有的 DIV
      function hide()
      {
        document.getElementById("menu1").style.visibility="hidden";
        document.getElementById("menu2").style.visibility="hidden";
        document.getElementById("menu3").style.visibility="hidden";
      }
    </script>
  </head>
  <body>
    <div style="position:absolute;left=100;top=20;visibility=visible">
      下面这个图片是可见的:
      
    </div>
    <div style="position:absolute;left=100;top=60;visibility=hidden">
      下面这个图片是不可见的:
      
    </div>
  </body>
</html>
```



```

</script>
</head>
<body>
  <table>
    <tr bgcolor="#9999FF">
      <td width="80" onMouseMove="show('menu1')" onMouseOut="hide()">菜单一</td>
      <td width="80" onMouseMove="show('menu2')" onMouseOut="hide()">菜单二</td>
      <td width="80" onMouseMove="show('menu3')" onMouseOut="hide()">菜单三</td>
    </tr>
  </table>
  <div id="menu1" onMouseMove="show('menu1')"
onMouseOut="hide()" style="background:#9999FF;position:absolute;left=12;top=38;width=80;
visibility:hidden">
    <span>子菜单一</span><br>
    <span>子菜单二</span><br>
    <span>子菜单三</span><br>
  </div>
  <div id="menu2" onMouseMove="show('menu2')"
onMouseOut="hide()" style="background:#9999FF;position:absolute;left=95;top=38;width=80;
visibility:hidden">
    <span>子菜单一</span><br>
    <span>子菜单二</span><br>
    <span>子菜单三</span><br>
  </div>
  <div id="menu3" onMouseMove="show('menu3')"
onMouseOut="hide()" style="background:#9999FF;position:absolute;left=180;top=38;width=80;
visibility:hidden">
    <span>子菜单一</span><br>
    <span>子菜单二</span><br>
    <span>子菜单三</span><br>
  </div>
</body>
</html>

```

在这个程序中可以看出，所有的子菜单都是已经写好的 DIV，当鼠标移到指定区域的时候显示对应的 DIV，当鼠标移出的时候隐藏所有的 DIV，从而就有了下拉菜单的效果。这个程序在浏览器中的效果如图 3.17 所示。



图 3.17 下拉菜单示例程序运行效果

3.3.5 JavaScript+CSS 实现表格变色

在一些 Web 应用中经常会用表格来展示数据，当表格行数比较多的时候，就容易后看错行的情况发生，所以需要一种方法来解决这个问题。

在这里我们采取这样一种措施，当鼠标移到某一行的时候，这行的背景颜色发生变化，这样当前行就会比较突出，不容易出错。具体的实现代码如下。

```
//-----文件名: ColorTable.html-----
<html>
  <head>
    <title>变色表格示例</title>
    <script language="javascript">
      function changeColor(row)
      {
        document.getElementById(row).style.backgroundColor='#CCCCFF';
      }
      function resetColor(row)
      {
        document.getElementById(row).style.backgroundColor="";
      }
    </script>
  </head>
  <body>
    <table width="200" border="1" cellspacing="1" cellpadding="1">
      <tr>
        <th>学校</th>
        <th>专业</th>
        <th>人数</th>
      </tr>
      <tr align="center" id="row1" onMouseOver="changeColor('row1')" onMouseOut="resetColor('row1')">
        <td>北大</td>
        <td>法律</td>
        <td>2000</td>
      </tr>
      <tr align="center" id="row2" onMouseOver="changeColor('row2')" onMouseOut="resetColor('row2')">
        <td>清华</td>
        <td>计算机</td>
        <td>5000</td>
      </tr>
      <tr align="center" id="row3" onMouseOver="changeColor('row3')" onMouseOut="resetColor('row3')">
        <td>人大</td>
        <td>经济</td>
        <td>6000</td>
      </tr>
    </table>
  </body>
</html>
```

这个页面的在浏览器中的运行结果如图 3.18 所示。



图 3.18 变色表格运行效果

3.4 小结

HTML 是组织展示内容的标记语言，JavaScript 是客户端的脚本语言，CSS 是美化页面的样式表，这三种技术结合在一起构成了 Web 开发最基础的知识，所有的 Web 应用开发都是在这个基础之上进行的。

在本章的讲解中，仅仅对这三种技术的大体情况进行了介绍，使读者可以迅速对 Web 开发的基础知识有一个宏观的清楚的认识，从而可以快速进入后面章节的学习，如果读者对这方面基础知识有更进一步了解的需要，就有必要参考相关的专题书籍。

第 4 章 JSP 技术基础知识

JSP (Java Server Page) 是 SUN 公司开发的一种服务器端的脚本语言, 自从 1999 年推出以来, 逐步发展为开发 Web 应用一项重要技术。JSP 可以嵌套在 HTML 中, 而且支持多个操作系统平台, 一个用 JSP 开发的 Web 应用系统, 不用做什么改动就可以在不同的操作系统中运行。

在本章接下来的内容中, 首先将简单介绍 JSP 的运行原理和基本语法, 然后重点介绍在实际开发过程中技巧和方法。

4.1 JSP 简介

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, 可以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。在这个小节中将介绍 JSP 的运行原理、JSP 的优点和其运行环境的搭建。

4.1.1 运行原理

因为本书讲述的主要内容就是基于 JSP 的 Java Web 应用程序开发技术, 为了是读者对 JSP 有一个更加深入的了解, 在这里有必要对 JSP 的运行机制、原理进行深入的介绍。

如果 JSP 页面是第一次被请求运行, 服务器的 JSP 编译器会生成 JSP 页面对应的 Java 代码, 并且编译成类文件。当服务器再次收到对这个 JSP 页面请求的时候, 会判断这个 JSP 页面是否被修改过, 如果被修改过就会重新生成 Java 代码并且重新编译, 而且服务器中的垃圾回收方法会把没用的类文件删除。如果没有被修改, 服务器就会直接调用以前已经编译过的类文件。

下面就是一个简单的 JSP 页面, 在这个页面中没有任何动态代码, 其功能是在页面显示一句话, 这个 JSP 页面的具体代码如下。

```
//-----文件名: Simple.jsp-----
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<html>
  <head>
    <title>简单 JSP 页面示例</title>
  </head>
  <body>
    这是一个简单的 JSP 页面示例 <br>
  </body>
</html>
```

上面这个 JSP 页面在被请求的时候, Web 服务器中的 JSP 编译器会生成对应的 Java 文件, 上面这个 JSP 程序对应的 Java 代码如下所示。

```
//-----文件名: Simple_jsp.java-----
package org.apache.jsp;
```

```

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.util.*;

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

    private static java.util.Vector _jspx_dependants;

    public java.util.List getDependants() {
        return _jspx_dependants;
    }

    public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {

        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        JspWriter _jspx_out = null;
        PageContext _jspx_page_context = null;

        try {
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html;charset=gb2312");
            pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;

            out.write("\r\n");
            out.write("<html>\r\n");
            out.write("    <head>\r\n");
            out.write("        <title>简单 JSP 页面示例</title>\r\n");
            out.write("    </head>\r\n");
            out.write("<body>\r\n");
            out.write("    这是一个简单的 JSP 页面示例 <br>\r\n");
            out.write("</body>\r\n");
            out.write("</html>\r\n");
        } catch (Throwable t) {

```

```

    if (!(t instanceof SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
            out.clearBuffer();
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
    }
} finally {
    if (_jspxFactory != null) _jspxFactory.releasePageContext(_jspx_page_context);
}
}
}

```

上面这段代码就是 Simple.jsp 所对应的 Java 代码，假如你的 Tomcat 安装路径为 C:\Tomcat5.0，Web 应用程序发布的名称为 chap4，那么可以在 C:\Tomcat 5.0\work\Catalina\localhost\chapt4\org\apache\jsp 这个目录下找到 JSP 所对应的 Java 文件和编译出来的类文件。

上面这段程序再本质上就是一个 Servlet，把所有页面的显示的内容都用 out 对象打印出来，包括每个 HTML 标签，所以说 JSP 页面本质上就是 Servlet 的一个种化身，在 JSP 程序种离不开 Servlet 的影子。这段代码的具体语法可以不必深究，这些工作都是有服务器中的 JSP 编译器来完成，这个过程是自动完成的，无需手工干预。

注意：只用被请求过的页面才能生成对应的 Java 文件，当 JSP 页面被修改后，再次对这个页面进行请求才会重新生成对应的 Java 文件。

4.1.2 选择 JSP 的原因

在 Web 应用开发中，可供选择的动态页面语言技术有很多，例如 PHP，ASP，JSP 等，在这些动态页面语言中，JSP 凭借其自身的优点成为开发人员最喜欢的语言之一。下面列出的几条就是开发人员钟爱 JSP 的重要原因。

- ❑ JSP 就是在 HTML 中嵌入 Java 代码，所以在本质上 JSP 程序就是 Java 程序，JSP 程序继承了 Java 的一切优点。JSP 程序有严格的 Java 语法和丰富的 Java 类库支持。
- ❑ JSP 页面在服务器中都会被 JSP 编译器编译成对应的 Servlet，所以就拥有 Java 跨平台的优点，所有的 JSP 程序，无需改动就可以方便地迁移到其他操作系统平台，这是在其他动态脚本语言中所无法想象的。
- ❑ JSP 中可以使用 JavaBean 进行逻辑封装，这样就可以实现逻辑功能代码的重用，从而大大提高系统的可重用性，同时也提高了程序的开发效率。
- ❑ JSP 程序容易上手，如果有 HTML 和 Java 的基本知识，那么学习 JSP 程序就没有任何难度。
- ❑ 在 Java 领域，开源的项目越来越多，这些开源项目是全世界 Java 爱好者的心血的结晶，在我们的 JSP 程序中可以非常方便地使用这些开源工具。在开源项目的支持上，JSP 更是其他动态语言不能相比的。

开发者从对 Java 的热爱延伸到对 JSP 的热爱，同时 JSP 又是 J2EE 体系中最重要，而且是最基础的一个组成部分，如果要体验 J2EE 带来的开发效率和优势，学习 JSP 是一个非常有效的入门方式。

在接下来的章节将开始进入 JSP 的精彩世界。

4.1.3 环境搭建

要运行 JSP 程序，必需为其提供一个 JSP 容器，也就是需要一个 Web 服务器。支持 JSP 的服务器非常多，Tomcat、Resin、Weblogic、WebSphere 等对 JSP 的支持都非常好，但是由于 Weblogic 和 WebSphere 都是功能非常强大的重量级服务器，而且价格昂贵，对计算机的硬件配置要求也比较高，所以在一般情况下，如果只用到 JSP 的技术，是没有必要选择这两个服务器的。

一般情况下我们可以选择 Tomcat 或者 Resin 作为 JSP 容器，这两个服务器对 JSP 的支持都非常好，而且都是开源的，可以免费使用，同时它们对计算机的硬件配置要求也是非常低的。这两个服务器任选其一都可以满足所有 JSP 开发的需要，在这里我们选择 Tomcat 作为 JSP 容器。

Tomcat 的运行需要 JDK 的支持，所以在安装 Tomcat 的时候需要提前安装 JDK，其中 JDK 的安装方法在本书第二章中有详细的介绍。

Tomcat 是 apache 的一个开源项目，可以在其官方网站 <http://tomcat.apache.org> 下载 Tomcat 的各个版本，在这个网站上提供 Tomcat 各个版本下载，下载的文件格式有 zip、tar.gz、exe 三种格式，其中 zip 格式只需要解压到某个目录，然后在环境变量中设置路径即可运行，这种方法比较灵活，但是环境变量设置错误 Tomcat 就不能运行。tar.gz 格式是 Unix、linux 系统上的压缩格式。exe 格式是 Windows 操作系统中的安装程序，这中格式的安装程序提供详细的安装向导，安装过程比较容易，在这里我们选择 exe 的安装个格式。

- (1) 下载 jakarta-tomcat-5.0.28.exe 安装文件，运行这个文件就会进入如图 4.1 所示的安装界面。
- (2) 在图 4.1 中选择“Next”，进入如图 4.2 所示的安装协议界面。



图 4.1 Tomcat 安装初始界面

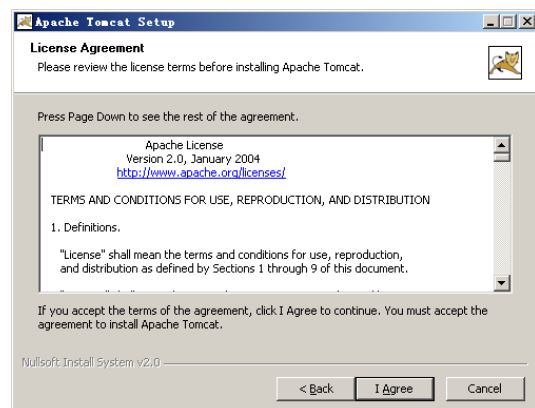


图 4.2 Tomcat 安装协议选择界面

(3) 在图 4.2 中选择“I Agree”进入如图 4.3 所示的组件选择界面，如果不接受协议安装程序就会退出。

(4) 在如图 4.3 的界面中，可以选择安装 Tomcat 的组件，其中安装类型选择 Normal 即可，这时候下面的几个组件会被自动选中，然后选择“Next”就可以进入如图 4.4 的安装路径选择界面。

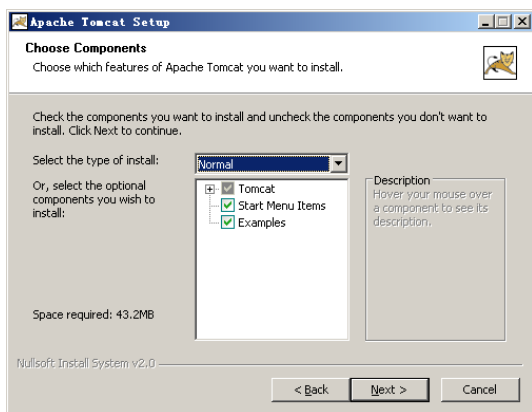


图 4.3 Tomcat 组件选择界面

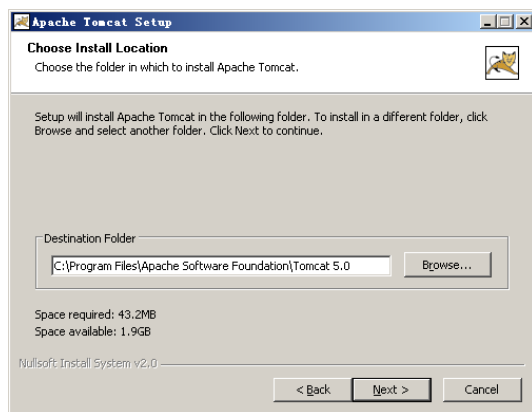


图 4.4 Tomcat 安装路径选择界面

(5) 在如图 4.4 中选择 Tomcat 将要安装到的目标路径，选择“Next”进入如图 4.5 所示界面。

(6) 在如图 4.5 的界面中，可以选择 Tomcat 的访问端口，可以采用默认端口是 8080，在上面这个界面中还可以设置 Tomcat 管理员的用户名和密码，利用这个管理员账号可以管理 Tomcat 的一些运行配置等信息。端口和管理员账号设置完成后选择“Next”进入如图 4.6 所示的 JVM 选择界面。

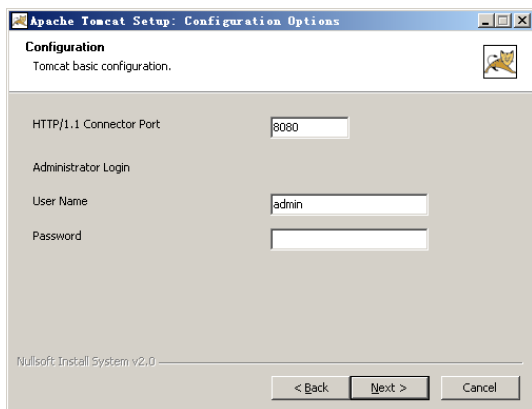


图 4.5 Tomcat 端口选择和管理员密码设置界面

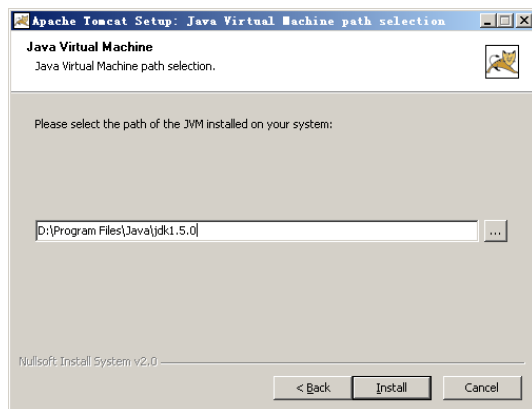


图 4.6 JVM 选择界面

(7) 在图 4.6 中的 JVM 路径不用选择，如果 JDK 已经正确安装配置的话，Tomcat 的安装程序可以自从系统变量中读出这个路径，无需手动配置。选择“Install”开始安装，安装完成以后会出现如图 4.7 所示的安装完成界面。

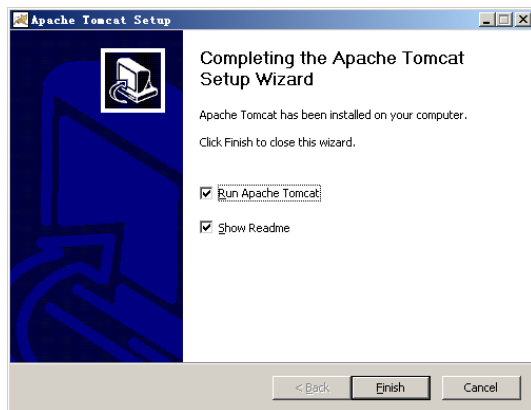


图 4.7 Tomcat 完成安装提示界面

(8) 在图 4.7 的界面中选择“Finish”完成安装。

(9) Tomcat 的启动方式后好多中，最基本的是在 Tomcat 安装目录下的 bin 文件夹中的 startup.bat、shutdown.bat，其中 startup.bat 可以启动 Tomcat 服务器，shutdown.bat 可以关闭 Tomcat 服务器。运行 startup.bat，如果可以看到如图 4.8 所示的启动信息说明 Tomcat 启动成功，

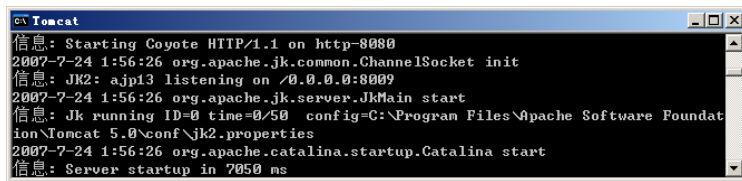


图 4.8 Tomcat 启动信息

(10) 当看到如图 4.8 所示“Server startup in ***** ms”的时候说明 Tomcat 已经成功启动，打开浏览器，在地址栏中输入 <http://localhost:8080/>，如果出现如图 4.9 所示的界面说明 Tomcat 已经成功安装。

(11) Tomcat 还有一个图形界面的管理工具，选择“程序”|“所有程序”|“Apache Tomcat 5.0”|“Monitor Tomcat”，就会出现如图 4.10 所示的图形管理界面。

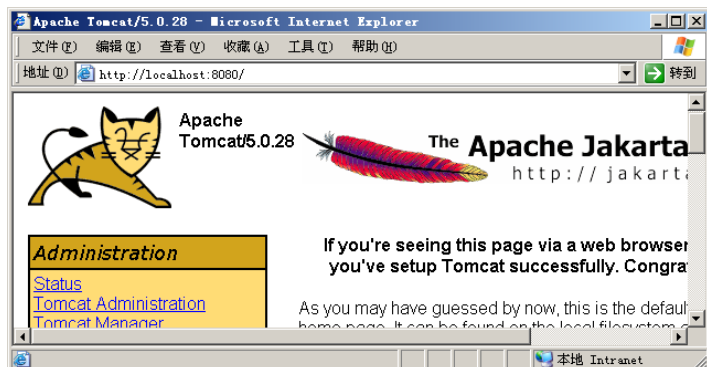


图 4.9 Tomcat 默认欢迎界面

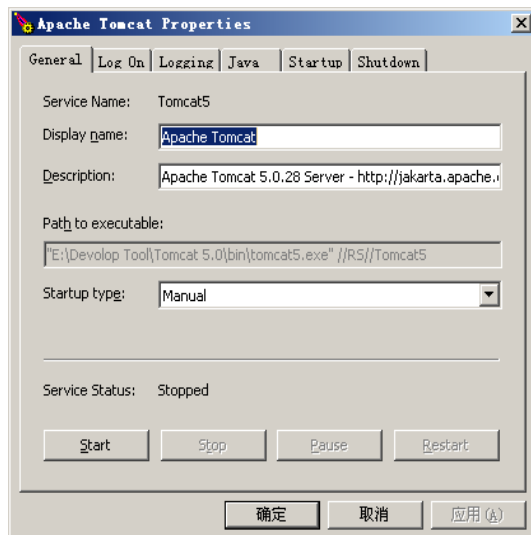


图 4.10 Tomcat 图形管理界面

(12) 如图 4.10 的管理界面中，“General”选项卡中，“Start”可以启动 Tomcat，“Stop”可以关闭 Tomcat，“Pause”可以暂停 Tomcat，“Restart”可以重新启动“Tomcat”。

(13) 在“logging”选项卡中可以对 Tomcat 的日志信息进行配置，在“Java”选项卡中可以对 JVM 进行配置。

注意：如果在启动 Tomcat 的过程中出现了“Address already in use: JVM_Bind”这个错误，则错误的解决方法有两种，一种是关闭其他正在运行的 Tomcat 程序，另一种解决方法就是修改 Tomcat 的端口，可以在 Tomcat 安装目录下的 conf 目录下找到 server.xml 的文件，在这个文件中搜索“<Connector port=“8080””，然后把这个 8080 端口改为其他端口，重新启动 Tomcat 即可，如果这时候还有其他端口冲突，按照同样的方法修改相应端口即可。

4.2 JSP 基本语法

本书的重点内容是介绍基于 JSP 的 Web 开发技术，对于 Java 的语法在此不做详细的介绍，这里所涉及 JSP 语法指的是在 JSP 中所特有的语法规则，在接下来的章节中将假设读者已经了解 Java 的基本语法，只介绍 JSP 的结构、变量声明、表达式、动作、指令等 JSP 的特有语法。如对 Java 语法有疑问的读者可以参考相关语法书籍。

4.3 程序结构

JSP 就是把 Java 代码嵌套在 HTML 中，所以 JSP 程序的结构可以分为两大部分：一部分是静态的 HTML 代码；另一部分是动态的 Java 代码和 JSP 自身的标签和指令；当 JSP 页面第一次被请求的时候，服务器的 JSP 编译器会把 JSP 页面编译成对应的 Java 代码，根据动态 Java 代码执行的结果，生成对应的纯 HTML 的字符串返回给浏览器，这样就可以把动态程序的结果展示给用户。下面就是一个简单的 JSP 程序的代码，现在分析其基本结构。

```
//-----文件名: Add.jsp-----
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%
    int first = 0;
    int second = 0;
    if(request.getParameter("first")!=null&&request.getParameter("first").length()>0)
    {
        first = Integer.parseInt(request.getParameter("first"));
    }
    if(request.getParameter("second")!=null&&request.getParameter("second").length()>0)
    {
        second = Integer.parseInt(request.getParameter("second"));
    }
%>
<html>
<head>
<title>求和 JSP 程序示例</title>
<script type="text/javascript">
    function check()
    {
        if(this.document.forms[0].first.value.length==0)
            alert("请输入第一个整数!");
        else if(this.document.forms[0].second.value.length==0)
            alert("请输入第二个整数!");
        else if (isNaN(this.document.forms[0].first.value))
            alert("输入的的第一个数字必须是整型数据");
        else if (isNaN(this.document.forms[0].second.value))
            alert("输入的第二个数字必须是整型数据");
        else
            this.document.forms[0].submit();
    }
}
```

```

</script>
</head>
<body>
  <form action="Add.jsp" method="post">
    <font size="2">
      这个 JSP 页面的功能是求两个整数的和: <br>
      请输入第一个数: <input type="text" name="first"/><br>
      请输入第二个数: <input type="text" name="second"/><br>
      这两个数的和为: <%= (first+second) %><br>
      <input type="button" value="求和" onclick="check()"/><br>
    </font>
  </body>
</html>

```

在上面这个示例程序中，代码的结构是十分清晰的，至于这个程序的具体实现现在不必深究，在本章内容结束之后自然就会明白，现在只分析这个 JSP 程序的基本代码结构。

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
```

上面这行代码是一个 JSP 的 page 指令，是 JSP 特有的指令。

```

<%
  int first = 0;
  int second = 0;
  if(request.getParameter("first")!=null&&request.getParameter("first").length()>0)
  {
    first = Integer.parseInt(request.getParameter("first"));
  }
  if(request.getParameter("second")!=null&&request.getParameter("second").length()>0)
  {
    second = Integer.parseInt(request.getParameter("second"));
  }
%>

```

上面这段代码是典型的 Java 代码片段，如果要在 JSP 页面中用到 Java 代码，只需要在<% %>这个标签中嵌入 Java 代码片段即可。

```

<script type="text/javascript">
  function check()
  {
    if(this.document.forms[0].first.value.length==0)
      alert("请输入第一个整数!");
    else if(this.document.forms[0].second.value.length==0)
      alert("请输入第二个整数!");
    else if (isNaN(this.document.forms[0].first.value))
      alert("输入的第二个数字必须是整型数据");
    else if (isNaN(this.document.forms[0].second.value))
      alert("输入的第二个数字必须是整型数据");
    else
      this.document.forms[0].submit();
  }
</script>

```

上面这段代码是典型的表单输入验证，这是 JavaScript 代码片段，和 JSP 无关，是属于客户端的语言。

```
这两个数的和为: <%= (first+second) %><br>
```

在上面这行代码中用到了 JSP 的输出标签，在 HTML 页面中直接输出`<%= %>`中表达式的值。这个程序中其他剩余的其他部分就是单纯的静态 HTML 代码。上面这个示例程序的运行结果如图 4.11 所示。



图 4.11 求和 JSP 示例程序运行效果

4.4 JSP 动作指令

在 Web 程序涉及中经常需要用到 JSP 的动作指令，例如在使用 JavaBean 的时候就离不开 userBean 的指令，JSP 的强大功能和它丰富的动作指令标签是分不开的。

在接下来的章节中将对这些指令进行详细的介绍，读者可以仔细体会每个动作的示例程序，在示例程序中掌握这些动作指令的基本用法。

4.4.1 include 动作指令

`include` 动作指令可以在 JSP 页面中动态包含一个文件，这与 `<include>` 指令不同，前者可以动态包含一个文件，文件的内容可以是静态的文件也可以是动态的脚本，而且当包含的动态文件被修改的时候 JSP 引擎可以动态对其进行编译更新。而 `<include>` 指令仅仅是把一个文件简单的包含在一个 JSP 页面中，从而组合成一个文件，仅仅是简答的组合的作用。其功能没有 `include` 动作指令强大。

`include` 动作指令的具体使用方法可以参考下面的示例程序。

```
//-----文件名: IncludeAction.jsp-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>  
<html>  
  <head>  
    <title>include 动作指令使用示例程序</title>  
  </head>  
  <body>  
    <font size="2">  
      <jsp:include flush="true" page="header.txt"></jsp:include>  
      这是一个 JSP 动作标签 include 的使用示例程序。<br>  
      <jsp:include flush="true" page="footer.jsp"></jsp:include>  
    </font>  
  </body>  
</html>
```

在上面这个程序中，用了两次 `include` 动作指令，第一次包含了一个静态的文本文件，第二次用 `include` 指令包含了一个动态的 JSP 文件。`include` 动作标签的使用格式如下。

```
<jsp:include flush="true" page="header.txt"></jsp:include>
```

在上面这行代码中，只需要指定 page 的路径即可，即指明包含的文件为 header.txt 其中 header.txt 的内容如下。

```
//-----文件名: header.txt-----
```

这里用 include 动作指令包含一个静态的文本文件。

在第二次使用 include 的指令时包含了一个 JSP 文件。

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

上面这行代码同样需要指明 page 的值，即指明包含的文件为 footer.jsp，其中 footer.jsp 的内容如下。

```
//-----文件名: footer.jsp-----
```

```
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
```

```
<html>
```

```
<body>
```

```
<%
```

```
out.print("这里用 include 动作指令包含一个动态的 JSP 页面!");
```

```
%>
```

```
</body>
```

```
</html>
```

上面这个示例程序的运行结果如图 4.12 所示。

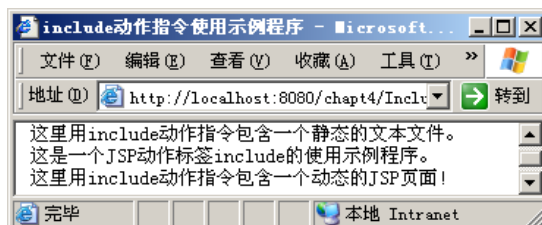


图 4.12 include 动作指令示例程序运行效果

注意：在被包含的 JSP 程序中，如果有所修改，JSP 引擎会及时发现，并重新编译。这就是 include 动作指令与 include 指令的最大不同。

4.4.2 forward 动作指令

forward 动作指令可以用来控制网页的重定向。即从当前页面跳转到另一个页面。

forward 动作的使用方法非常简单，具体使用格式如下。

```
<jsp:forward page="http://www.sohu.com"></jsp:forward>
```

在上面这行代码中，只要指明 page 的值，当 JSP 执行到这行代码的时候就可以直接跳转到对应的网页。forward 动作指令的具体用法可以参考下面的例子。

```
//-----文件名: Forward.jsp-----
```

```
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312" %>
```

```
<html>
```

```
<head>
```

```
<title>forward 动作指令使用示例</title>
```

```
</head>
```

```
<body>
```

```
<jsp:forward page="http://www.sohu.com"></jsp:forward>
```

```
</body>
```

```
</html>
```

上面这个页面运行以后就直接跳转到 www.sohu.com 的主页面。

注意：forward 动作指令和 HTML 中的<a>超链接标签是不同的，在<a>中只有单击链接才能实现页面跳转，在 forward 动作指令中一切都是可以用 Java 的代码进行控制，可以在程序中直接决定页面跳转的方向和时机。

4.4.3 param 动作指令

在上面 forward 动作指令中可以用程序控制页面的跳转，如果需要在跳转的时候同时传递参数，这时候就需要用到 param 动作指令。param 动作指令的具体使用方法可以参考下面的示例程序。

下面这个页面是初始页面，参数从这里开始传递。

```
//-----文件名: PassParam.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>页面跳转并且传递参数示例</title>
  </head>
  <body>
    从这个页面传递参数:
    <jsp:forward page="GetParam.jsp">
      <jsp:param name="param" value="test"/>
    </jsp:forward>
  </body>
</html>
```

在上面这段程序中，把页面跳转到 GetParam.jsp 这个页面，同时向 GetParam.jsp 这个页面传递了一个名称为 param 的参数，这个参数的值为 test。具体设置代码就是上面这段代码。

```
<jsp:forward page="GetParam.jsp">
  <jsp:param name="param" value="test"/>
</jsp:forward>
```

其中 GetParam.jsp 页面的代码如下。

```
//-----文件名: GetParam.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>页面跳转并且传递参数示例</title>
  </head>
  <body>
    <font size="2">
      这个页面接受传递过来的参数: <br>
      前一个页面传递过来的参数为: <%out.print(request.getParameter("param")); %>
    </font>
  </body>
</html>
```

在上面这个页面中，接受从 PassParam.jsp 这个页面传递过来的参数，并且把参数的值打印在页面上。上面这个跳转并且传递参数的示例程序运行效果如图 4.13 所示。



图 4.13 页面跳转并且传递参数示例程序运行效果

注意：在这个示例程序运行的过程中，我们可以发现，在 forward 跳转并且传递参数的过程中，浏览器地址栏中的地址始终是不变的，传递的参数也不会浏览器的地址栏中显示，这也是 forward 动作指令与 HTML 中 `<a>`/`` 超链接的另一个区别。

4.4.4 plugin 动作指令

`<jsp:plugin>` 元素用于在浏览器中播放或显示一个对象（典型的的就是 applet 和 bean），而这种显示需要在浏览器的 java 插件。当 jsp 文件被编译，送往浏览器时，`<jsp:plugin>` 元素将会根据浏览器的版本替换成 `<object>` 或者 `<embed>` 元素。

4.4.5 useBean 动作指令

useBean 动作指令可以在 JSP 中引用 JavaBean，这个动作指令在实际开发过程中经常会用到。在第六章 JavaBean 的讲解过程中将对这个动作指令做详细的介绍。在这里我们仅仅知道其基本用法即可，而且在这里不在用示例程序说明。

useBean 的使用格式如下。

```
<jsp:useBean id=" " class=" " scope=" "></jsp:useBean>
```

其中 id 为所用到的 JavaBean 的实例对象名称，class 是 JavaBean 对应类的包路径，包括包名和类名。scope 是这个 JavaBean 的有效范围，共有 page、request、session、application 四个值可以选择。

4.4.6 setProperty 动作指令

setProperty 一般情况下是和 JavaBean 配合使用的，用来给 JavaBean 的实例对象进行赋值操作，setProperty 的基本方法有以下两种。

```
<jsp:setProperty name="JavaBean 的实例名称" property="属性名" value="属性值"/>
```

上面这种方法是 setProperty 动作指令最基本的用法，用来给 JavaBean 实例对象的某一个属性赋值。

```
<jsp:setProperty name="JavaBean 的实例名称" property="*" />
```

上面这种 JavaBean 的赋值方法也是经常用到的，这个方法可以给 JavaBean 实例对象的所有属性进行赋值，其中在 JSP 的页面请求中有对应与所有属性的输入，这些输入的名称与 JavaBean 的属性名一一对应，所以 JSP 引擎可以根据名称给对应的属性进行赋值。

setProperty 这个动作指令的具体使用方法将在第六章中详细介绍，在这里仅仅知道基本的使用形式即可。

4.4.7 getProperty 动作指令

getProperty 一般情况下也是和 JavaBean 配合使用的，用来取出 JavaBean 实例对象的属性值。这个动作指令的基本使用方法如下。

```
<jsp:getProperty name="JavaBean 的实例名称" property="属性名" value="属性值"/>
```

上面这行代码就可以取出 JavaBean 实例对象的一个属性值，至于 getProperty 这个动作指令更详细的使用方法，在第六章 JavaBean 的讲解中将有详细的实例介绍。

4.5 JSP 指令

JSP 的指令虽然没有动作指令那么丰富，但是其作用却是不容忽视的，例如 page 指令，在设置显示编码、引入类的包路径、设置错误页面等方面都是必不可少的。在接下来的章节中将介绍 JSP 的两个指令标签。

4.5.1 page 指令

page 指令可以用来定义 JSP 页面的全局属性。例如编码、错误页面等。page 指令的属性很多，下面来具体介绍它的各个属性。

1. language

这个属性用来设置页面所使用的语言，对于 JSP 来说当然要选 Java。具体设置方法如下。

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

2. import

import 用来引入用到的包或者是类，这个属性的设置方法如下。

```
<%@ page import="java.util.*" %>
```

在上面这个行代码中，以引入 java.util.* 包为例展示了在 JSP 中引入包或者类的方法。

3. contentType

这个属性设置了 JSP 页面的 MIME 类型，对于还有中文的 JSP 页面可以按照下面这种方式设置。

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

经过这样的设置，页面显示编码方案设置为 gb2312，这种编码格式可以正确显示中文。

4. session

设置在 JSP 页面中是否可以使用 session 对象，默认为 true。

5. buffer

用来设置 out 对象缓冲区的大小，可以选择 none、也可以设置为指定的大小，单位为 KB。

6. autoFlush

当在 JSP 页面设置了可以使用缓冲区的时候，才可以设置这个属性，这个属性设置为 true 的时候，缓冲区一旦满了就会自动刷新。如果设置为 false，缓冲区就满了以后就会报溢出错误。

7. isThreadSafe

设置当前 JSP 页面是否是线程安全的，默认是 true，可以同时相应多个请求。

8. info

此属性设置当前 JSP 页面的描述信息，不常用。

9. errorPage

此属性设置错误处理页面，当页面出错的时候可以跳转到这个错误处理页面。

10. isErrorPage

设置当前页面是否为错误处理页面，默认为 false。

4.5.2 include 指令

include 指令可以在当前的 JSP 页面中包含一个文件，从而和当前页面组成一个整体的文件。这中包
含仅仅是静态包含。

include 指令的具体使用方法实例如下。

```
//-----文件名: Include.jsp-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>  
<html>  
  <head>  
    <title>include 指令使用示例程序</title>  
  </head>  
  <body>  
    <font size="2">  
      这是 include 指令的使用示例程序。<br>  
      <%@ include file="inc.txt"%>  
    </font>  
  </body>  
</html>
```

在这个页面中静态引入了一个文件 inc.txt，其中 inc.txt 的内容如下。

```
//-----文件名: inc.txt-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>  
这是一个简单文本文件
```

上面这个程序的运行结果如图 4.14 所示。



图 4.14 include 指令实例程序运行效果

4.6 JSP 内置对象简介

JSP 内置对象即无需声名就可以直接使用的对象实例，在实际的开发过程中，比较常用的 JSP 内置

对象有 request、response、session、out、application 等，在接下来的章节中将详细介绍这几个 JSP 内置对象的使用方法。JSP 其他的几个内置对象在实际的开发中并不十分常用，在这里不做具体介绍。

4.7 request 对象

request 对象代表这从用户发送过来的请求，从这个对象中间可以取出客户端用户提交的数据或者是参数。这个对象只有接受用户请求的页面才可以访问。

4.7.1 request 对象使用场合

如果要与用户的互动，必须要知道用户的需求，然后根据这个需求生成用户期望看到的结果。这样才能实现与用户的互动。在 Web 应用中，用户的需求就抽象成一个 request 对象，这个对象中间包括用户所有的请求数据，例如通过表单提交的表单数据，或者是通过 URL 等方式传递的参数，这些就是用户的需求。request 正是用来收集类似这些用户的输入数据和参数。

同时，request 对象中还包括一些服务器的信息，例如端口、真实路径、访问协议等信息，通过 request 对象可以取得服务器的这些参数。

4.7.2 request 对象主要方法

request 对象的方法非常多，在这里我们只介绍其中最常用的几种方法，其他方法可以参考相关类库的介绍。

1. `getAttribute(String name)`

这个方法可以取出指定名称的这个属性的值，这个属性可以用 `setAttribute (String name,Object o)` 这个方法进行赋值，如果没有对这个属性赋值的话取值的操作返回 `null`。

2. `getContextPath ()`

这个方法可以获取的服务器上下文路径。

3. `getCookies ()`

这个方法可以取出客户端的 Cookies。

4. `getHeader (String name)`

这个方法可以取得指定名称的 HTTP 报头的属性值。

5. `getParameter (String name)`

这个方法可以取出客户端提交到服务器的参数。

6. `getServerName ()`

这个方法可以取得服务器的名称

7. `getServerPort ()`

这个方法可以取得服务器的访问端口。

8. `setAttribute (String name,Object o)`

这个方法对指定名称的属性进行赋值。

9. removeAttribute (String name)

这个方法可以移除指定名称的一个属性。

10. getRemoteAddr ()

这个方法返回客户端机器的 IP 地址。

下面是这些方法的调用实例。

```
//-----文件名: Request.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>request 主要方法调用示例</title>
  </head>
  <body>
    <font size="2">
      request 主要方法调用示例: <br>
    <%
      request.setAttribute("attr","Hello!");
      out.println("attr 属性的值为: "+request.getAttribute("attr")+"<br>");
      out.println("上下文路径为: "+request.getContextPath()+"<br>");
      out.println("Cookies:"+request.getCookies()+"<br>");
      out.println("Host:"+request.getHeader("Host")+"<br>");
      out.println("ServerName:"+request.getServerName()+"<br>");
      out.println("ServerPort:"+request.getServerPort()+"<br>");
      out.println("RemoteAddr:"+request.getRemoteAddr()+"<br>");
      request.removeAttribute("attr");
      out.println("属性移除操作以后 attr 属性的值为: "+request.getAttribute("attr")+"<br>");
    %>
  </font>
</body>
</html>
```

在这个程序中对 request 常用的方法进行展示，其他方法的使用和这些基本的方法类似。可以参考这些例子进行尝试。上面这个示例程序的运行结果如图 4.15 所示。

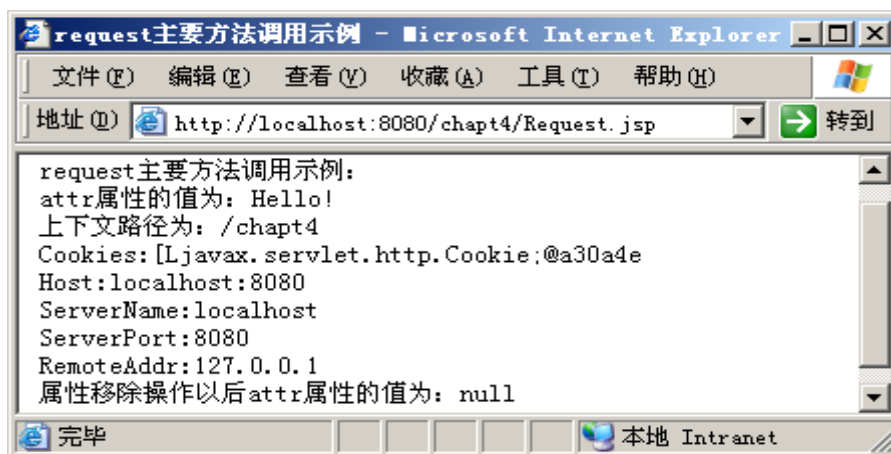


图 4.15 request 对象主要方法调用示例运行效果

4.7.3 request 对象使用示例

1. 使用 request 对象取得表单数据

request 获取用户数据的一个主要方式就是获取表单数据，下面是一个简单的表单，在这里为了方便说明，我们在把表单提交到这个页面自身，并在这个页面中取出提交的表单的数据。

```
//-----文件名: Form.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>request 获取表单数据示例</title>
  </head>
  <body>
    <font size="2">
      下面是表单内容:
      <form action="Form.jsp" method="post">
        用户名: <input type="text" name="userName" size="10"/>
        密 码: <input type="password" name="password" size="10"/>
        <input type="submit" value="提交">
      </form>
      下面是表单提交以后用 request 取到的表单数据: <br>
      <%
        out.println("表单输入 userName 的值:"+request.getParameter("userName")+"<br>");
        out.println("表单输入 password 的值:"+request.getParameter("password")+"<br>");
      %>
    </font>
  </body>
</html>
```

在上面这个程序中，当然可以把表单提交给其他的页面，那样只需把接受参数的方法放在对应的页面即可。在这里只是为了方便展示问题的才提交给页面自身。上面这个程序的运行结果如图 4.16 所示。



图 4.16 request 获取表单数据示例程序运行效果

2. 使用 request 对象取得页面传递的参数

在实际的开发过程中，经常遇到这样的情景，在页面跳转中需要添加相对应的参数。例如在在论坛

中需要参看一条帖子的时候，单击链接会进入一个新页面，在新页面中显示帖子的内容，这个操作过程即需要把帖子的编号传到新页面。

对与参数的处理我们还是采用 request 来处理，下面的例子就展示了如何使用 request 对象来获取参数的值。

```
//-----文件名: URL.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>request 对象取得页面传递参数示例</title>
  </head>
  <%
    String param = request.getParameter("param");
  %>
  <body>
    <font size="2">
      <a href="URL.jsp?param=Hello">请单击这个链接</a><br>
      你提交的参数为: <%=param%>
    </font>
  </body>
</html>
```

在这个页面中我们同样把参数传递给这个页面自身，道理和前面表单的处理是一样的。读者也可以尝试参考这个例子把参数传递到一个不同的页面。这个程序运行的结果如图 4.17 所示。



图 4.17 request 对象取得页面传递参数示例程序运行效果

4.8 response 对象

response 对象是服务器端向客户端返回的数据，从这个对象中间可以取出一部分与服务器互动的数据和信息。只有接受这个对象的页面才可以访问这个对象。

4.8.1 response 对象使用场合

既然用户可以对服务器发出请求，服务器就需要对用户的请求做出反应。这里服务器就可以使用 response 对象向用户发送数据。response 是对应 request 的一个对象。

如果需要获取服务器返回的处理信息，就可以对 response 进行操作，同时当服务器需要再客户端进行某些操作的时候也需要用到 response 对象，例如服务器要在客户端生成 Cookies，那么这时候 response

对象就是一个很好的选择。

4.8.2 response 对象主要方法

response 的方法也很多，但是常用的也就其中的几个，下面介绍比较常用的几个方法。

1. addCookie (Cookie cookie)

这个方法可以添加一个 Cookie 对象，用来保存客户端的用户信息。

2. containsHeader (String name)

这个方法判断指定的头信息是否存在。

3. encodeRedirectURL (String url)

这个方法可以对 URL 进行进行编码。

4. encodeURL (String url)

这个方法可以对 URL 进行进行编码。

5. flushBuffer ()

这个方法可以清空缓存的内容。

6. sendError (int error)

这个方法可以向客户端浏览器发送错误代码。如 500 为服务器内部错误。

7. sendRedirect (String location)

这个方法可以把当前页面转发到其他的页面，实现页面的跳转。

4.8.3 response 对象使用示例

response 的用法很多，在这里我们用 response 来实现一个页面的重定向，具体的代码如下。

```
//-----文件名: Redirect.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>response 对象页面重定向示例</title>
  </head>
  <body>
    <%
      response.sendRedirect("http://www.sohu.com");
    %>
  </body>
</html>
```

上面这个页面运行的时候直接重定向到 www.sohu.com 的首页。

4.9 session 对象

session 对象维护着客户端用户和服务端的状态，从这个对象中间可以取出用户和服务端交互的过程中的数据和信息。这个对象在用户关闭浏览器离开 Web 应用之前一直有效。

4.9.1 session 对象使用场合

session 对象中保存的内容是用户与服务端整个交互过程中的信息，如果是想在整个交互的过程中都可以访问到的信息，就可以选择存放在 session 对象中。

例如在用户登录的过程中，可以在 session 中记录用户的登录状态，这样用户就不必在每个页面都重新登录，只要用户没有离开当前的 Web 应用系统，就可以一直保存登录的状态。

4.9.2 session 对象主要方法

session 所提供的方法并没有前面几个内置对象那么多，但是基本都是非常常用的。

1. `getAttribute (String name)`

这个方法可以获取指定属性的值。

2. `getCreationTime ()`

这个方法可以获取 session 对象创建的时间。

3. `getLastAccessedTime ()`

这个方法可以获取 session 对象上次被访问的时间。

4. `invalidate ()`

这个方法可以失 session 对象失效。

5. `removeAttribute (String name)`

这个方法可以移除指定的属性。

6. `setAttribute (String name, Object value)`

这个方法可以给指定名称的属性赋值。

4.9.3 session 对象使用示例

在这里我们模拟一个简单的用户登录动作，在这个示例程序中，我们不对提交的登录信息做具体的验证，只要用户名和密码都不为空就可以登录系统，这样处理只是为了方便展示 session 的使用方法，在具体的开发中必须要对登录信息进行验证的。

这个登录注册的示例程序代码如下。

```
//-----文件名: Login.jsp-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>  
<html>  
  <head>  
    <title>用户登录界面</title>
```

```

</head>
<body>
    <font size="2">
        <form action="LoginCheck.jsp" method="post">
            用户名: <input type="text" name="userName" size="10"/>
            密 码: <input type="password" name="password" size="10"/>
            <input type="submit" value="提交">
        </form>
    </font>
</body>
</html>

```

上面这个 JSP 页面向 LoginCheck.jsp 提交了一个登录表单，表单中有用户名和密码。下面是 LoginCheck.jsp 的内容。

```

//-----文件名: LoginCheck.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
    <head>
        <title>用户登录验证页面</title>
    </head>
    <%
        String userName = request.getParameter("userName");
        String password = request.getParameter("password");

        if(userName.length()>0&&password.length()>0)
        {
            session.setAttribute("status","Login");
            response.sendRedirect("Main.jsp");
        }else
            response.sendRedirect("Login.jsp");

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

```

在上面这个页面中，从表单中取出用户名和密码，如果用户名和密码都不为空就允许登录，否则就重定向到登录页面，让用户重新登录。如果用户登录成功，就在 session 中设置一个 status 属性，然后把用户重定向到系统的主页面。在主页面中可以访问 session 中的值。下面是 Main.jsp 的具体代码。

```

//-----文件名: Main.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
    <head>
        <title>系统主界面</title>
    </head>
    <body>
        <font size="2">
            <%
                Object status = session.getAttribute("status");
                if(session.getAttribute("status")!=null)
                    out.print("用户已经登录! ");
            %>
        </font>
    </body>
</html>

```



```
        else
            response.sendRedirect("Login.jsp");
    %>
</font>
</body>
</html>
```

在上面这个页面中，对用户的状态进行判断，如果从 `session` 中可以取出对应的属性值就说明用户已经登录，如果没有取到指定属性的值，说明用户没有登录，这时就重定向到登录页面，让用户重新登录。其中 `session` 的值在用户离开系统之前的任何页面都可以访问。

注意：当用户打开系统的一个网页，服务器就生成一个 `session` 对象，在用户离开之前，这个对象一直伴随用户。在这个过程中可以在这个 `session` 中间存储用户的信息，这些信息在所有页面中都是可以访问的。只有当用户关闭浏览器的时候，这个 `session` 对象就会被注销。

4.10 out 对象

这个对象是在 Web 应用开发过程中使用最多的一个对象，其功能就是动态的向 JSP 页面输出字符流，从而把动态的内容转化成 HTML 形式来展示。这个对象在任何 JSP 页面中都可以任意访问。

4.10.1 out 对象使用场合

`out` 对象的功能就是向 JSP 页面输出数据信息。所以当有动态信息要展示给用户的时候就要用到 `out` 对象。在前面的很多示例中已经多次用到这个对象，读者从中可以很清楚的看到，`out` 对象就是用来输入动态内容信息的。

4.10.2 out 对象主要方法

在这里只介绍 `out` 对象最常用的方法。

1. `clear ()`

这个方法可以清除缓冲区的数据，但是仅仅是清除，并不向用户输出。

2. `clearBuffer ()`

这个方法可以清除缓冲区的数据，同时把这些数据向用户输出。

3. `close ()`

这个方法可以关闭 `out` 输出流。

4. `flush ()`

这个方法可以输出缓冲区的内容。

5. `isAutoFlush ()`

这个方法可以判断是否为自动刷新。

6. `print (String str)`

这个方法在 out 对象中最常用的，可以向 JSP 页面输出数据，其中数据格式可以是 int、boolean、Object 等。用法都是类似的。

4.10.3 out 对象使用示例

out 对象在前面的示例中已经多次使用到，在这里就不再针对这个对象举例说明。

4.11 application 对象

application 对象保存着整个 Web 应用运行期间的全局数据和信息，从 Web 应用开始运行开始，这个对象就会被创建，在整个 Web 应用运行期间可以在任何 JSP 页面中访问这个对象。

4.11.1 application 对象使用场合

application 中保存的信息可以在整个应用的任何地方访问，这个 session 对象类似，但和 session 对象还是有所区别的。只要 Web 应用还在正常运行，application 对象就可以访问，而 session 对象在用户离开系统就被注销。

所以如果要保存在真个 Web 应用运行期间都可以访问的数据，这时候就要用到 application 这个对象。

4.11.2 application 对象主要方法

下面介绍 application 对象的最常用的主要方法。

1. **getAttribute (String name)**

这个方法可以获取指定属性的值。

2. **getServerInfo ()**

这个方法可以取得服务器的信息。取出的值是类似 Apache Tomcat/5.0.28 这样的形式。

3. **removeAttribute (String name)**

这个方法可以移除指定的属性。

4. **setAttribute (String name, Object o)**

这个方法可以给一个指定名称的属性赋值。

4.11.3 application 对象使用示例

在这里我们要实现一个简单的计数器，这个计数器就是利用 application 对象来储存计数器的值，用来统计服务器开始运行以来的访问量。

```
//-----文件名: Counter.jsp-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
```

```

<html>
<head>
<title>利用 application 对象实现的计数器示例</title>
</head>
<body>
<font size="2">
<%
    int count=0;
    if(application.getAttribute("count")==null)
    {
        count = count + 1;
        application.setAttribute("count",count);
    }else
    {
        count = Integer.parseInt(application.getAttribute("count").toString());
        count = count + 1;
        application.setAttribute("count",count);
    }

    out.println("您是本系统的第"+count+"访问者！");
%>
</font>
</body>
</html>

```

在上面这个程序中，当第一次访问时候把 count 的初始值设置为 1，以后每次刷新的时候累加 count 的值。上面这个计数器的运行过程中，多个页面之间共享计数器的值，而且关闭浏览器然后再新窗口的时候，以前计数器的值还保留。这就是 application 与 session 最大的区别。

4.12 JSP 中文问题完全解决方案

在 Java 开发中，中文乱码是一个最让人头疼的问题，如果不对中文做特殊的编码处理，这些中文字符就会变成乱码或者是问号。而在不同情况下对这些乱码的处理方法又各不相同，这就导致很多初学者对中文乱码问题束手无策。其实造成这种问题的根本原因是 Java 中才用的默认编码方式是 Unicode，而中文的编码方式一般是 GB2312，因为编码格式的不同，导致在中文不能正常显示。

对于中文乱码问题，在不同的 JDK 版本和不同的应用服务器中的处理方法是不同的。但是其本质上都是一样的，就是把中文字符转化成合适的编码方式，或者是把在显示中文的环境中声名采用 GB2312 的编码。统一编码方案之后自然可以正常显示。

在接下来的章节中，将对 JSP 开发过程中的中文乱码问题进行详细的介绍，对各种乱码提供对应的解决方法，在各种编码方案中，UTF-8、GBK、GB2312 都是支持中文显示的，在本章中我们统一采用 GB2312 的编码格式来支持中文。

4.12.1 JSP 页面中文乱码

在 JSP 页面中，中文显示乱码有两种情况：一种是 HTML 中的中文乱码，另一中是在 JSP 中动态

输出的中文乱码。下面来看这样一个 JSP 程序。

```
//-----文件名: PageCharset.jsp-----
<%@ page language="java" import="java.util.*"%>
<html>
  <head>
    <title>中文显示示例</title>
  </head>
  <body>
    这是一个中文显示示例:
    <%
      out.print("这里是用 JSP 输出的中文。");
    %>
  </body>
</html>
```

上面这个 JSP 程序看起来好像是在页面显示几句中文，而且标题栏也是中文，但是在浏览器中的运行结果却如图 4.18 所示。

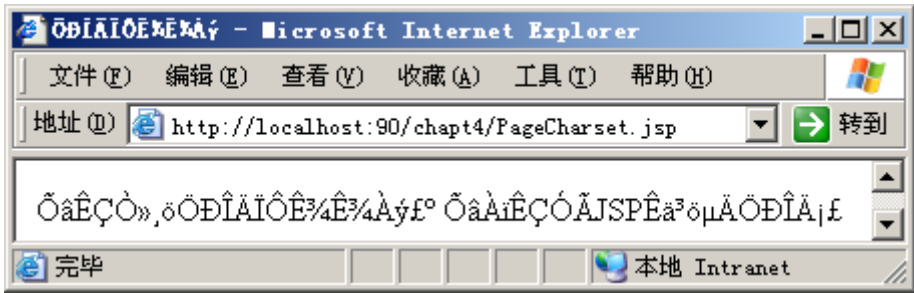


图 4.18 JSP 页面运行乱码效果

显然，图 4.18 中所示并非我们预期的效果，在我们 JSP 源代码中清清楚楚看到的是中文，在这里为什么就成了乱码，造成这种原因的可能就是出在浏览器端的字符显示设置上，所以我们可以把上面的程序进行如下改进。

```
<%@ page language="java" import="java.util.*"%>
```

在上面这行代码中就是对整个页面的起作用的 page 指令，我们可以在这个指令中间设置支持中文显示的编码方式，具体修改方法如下所示。

```
<%@ page language="java" import="java.util.*"contentType="text/html;charset=gb2312"%>
```

上面这行代码中，向 page 指令中添加了页面内容和显示方式的设置，其中采用 gb2312 的编码方式来显示 HTML 页面的内容，所以中文可以正常显示，改进以后的程序运行效果如图 4.19 所示。



图 4.19 JSP 页面正常运行效果

4.12.2 URL 传递参数中文乱码

在一般情况下，可以用类似 `http://localhost:8080/chapt4/URLCharset.jsp?param='中文'` 这种形式来传递参数，而且 HTML 在处理表单的时候，当表单的 method 采用 get 方法的时候，传递参数的形式与 URL 传递参数的形式基本一样。下面就是采用 URL 传递参数的一个示例程序。

```
//-----文件名: URLCharset.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>URL 传递参数中文处理示例</title>
  </head>
  <%
    String param = request.getParameter("param");
  %>
  <body>
    <a href="URLCharset.jsp?param='中文'">请单击这个链接</a><br>
    你提交的参数为: <%=param%>
  </body>
</html>
```

上面这个 JSP 程序的功能就是通过一个 URL 链接向自身传递一个参数，这个参数是中文字符串，这个程序的运行效果如图 4.20 所示。

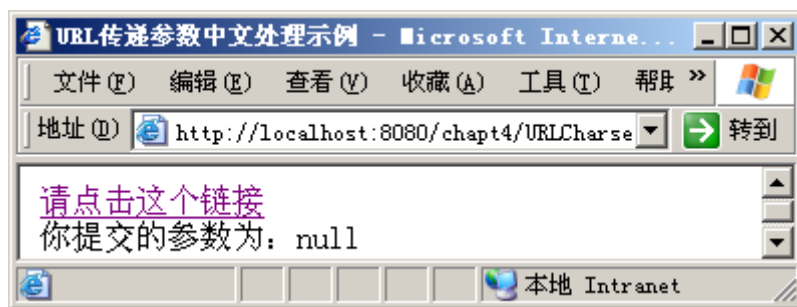


图 4.20 URL 传递中文参数初始界面

在图 4.21 所示的界面中单击链接，可以得到如图 4.21 所示的运行效果。

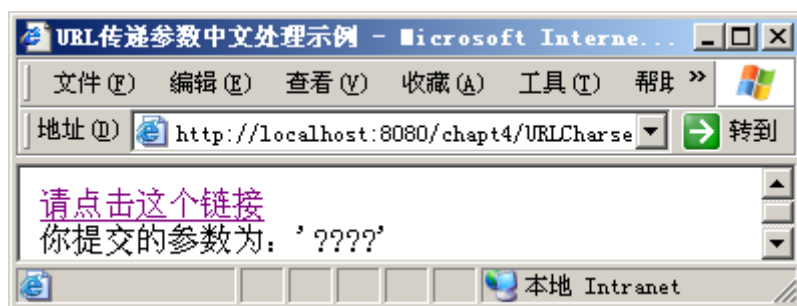


图 4.21 URL 传递中文参数出现乱码

对于 URL 传递中文参数乱码这个问题，其处理方法比较独特，仅仅转换这个中文字符串的编码，或者设施 JSP 页面显示编码都是不能解决问题的。在这里需要多 Tomcat 服务器的配置文件进行修改才可以解决问题。在这里需要修改 Tomcat 的 conf 目录下的 server.xml 配置文件。具体修改方法如下。

```
<Connector port="8080" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
```

```
enableLookups="false" redirectPort="8443" acceptCount="100"
debug="0" connectionTimeout="20000"
disableUploadTimeout="true" />
```

在上面的这段代码中间添加 URI 编码的设置即可，即在 `port="8080"` 后面添加 URI 编码设置 `URIEncoding="gb2312"` 即可。重新启动 Tomcat 服务器，可以得到如图 4.22 所示的界面。



图 4.22 URL 传递中文参数正常运行效果

注意: URL 中文乱码的处理方法比较特殊，后续章节中的转码和过滤器的方法对这个问题都不起作用，这个问题的解决只有更改 Tomcat 的配置文件。

4.12.3 表单提交中文乱码

对于表单中提交的数据，可以使用 `request.getParameter("")` 的方法获取。但是当表单中如果出现中文数据的时候就会出现乱码。请看下面这个简单的表单。

```
//-----文件名: FormCharset.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
<head>
<title>Form 中文处理示例</title>
</head>
<body>
<font size="2">
下面是表单内容:
<form action="AcceptFormCharset.jsp" method="post">
    用户名: <input type="text" name="userName" size="10"/>
    密 码: <input type="password" name="password" size="10"/>
    <input type="submit" value="提交">
</form>
</font>
</body>
</html>
```

在上面这个表单中，向 `AcceptFormCharset.jsp` 这个页面提交两项数据，下面是 `AcceptFormCharset.jsp` 的内容。

```
//-----文件名: AcceptFormCharset.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
<head>
<title>Form 中文处理示例</title>
</head>
```

```

<body>
<font size="2">
  下面是表单提交以后用 request 取到的表单数据: <br>
  <%
    out.println("表单输入 userName 的值:"+request.getParameter("userName")+"<br>");
    out.println("表单输入 password 的值:"+request.getParameter("password")+"<br>");
  %>
</font>
</body>
</html>

```

在上面这个程序中，如果表单中输入的内容没有中文，则可以正常显示，当输入的数据中有中文的时候，假如有如图 4.23 所示的数据输入。



图 4.23 用户名为中文的表单输入

在上面这个表单中，用户名的输入为中文字符，当提交这个表单的时候，就会得到如图 4.24 所示的结果。

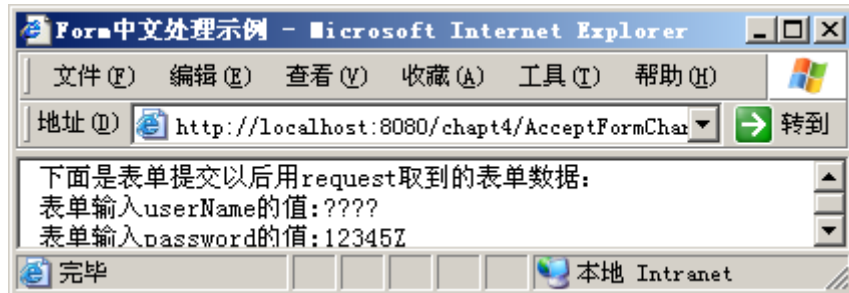


图 4.24 表单输入中文乱码

在图 4.25 所示的界面中可以清楚看到，输入的中文用户名在用 request 取出以后全部变成了乱码，造成这个问题的原因是：在 Tomcat 中，对于以 POST 方法提交的表单采用的默认编码为 ISO-8859-1，而这种编码格式不支持中文字符。对于这个问题，可以采用转换编码格式的方法来解决，现在对 AcceptFormCharset.jsp 改造如下。

```

<%
  out.println("表单输入 userName 的值:"+request.getParameter("userName")+"<br>");
  out.println("表单输入 password 的值:"+request.getParameter("password")+"<br>");
%>

```

上面这两行是对表单数据的处理，可以在这里进行数据的转码，具体的转换编码的方法如下。

```

<%
  String userName = request.getParameter("userName");
  String password = request.getParameter("password");
  out.println("表单输入 userName 的值:"+new String(userName.getBytes("ISO-8859-1"),"gb2312")+"<br>");

```

```
out.println("表单输入 password 的值:"+new String(password.getBytes("ISO-8859-1"),"gb2312")+"<br>");  
%>
```

经过这样的转换编码以后，所有的中文输入都可以用 request 对象正常取出。在上面这个程序中，new String(userName.getBytes("ISO-8859-1"),"gb2312")这句代码是转换编码格式的关键，先从 ISO-8859-1 格式的字符串中取出字节内容，然后再用 gb2312 的编码格式重新构造一个新的字符串。这样就可以支持中文的表单输入的正常取值和显示。改进以后的程序运行结果如图 4.25 所示。



图 4.25 表单中文输入正常取值并显示

注意：new String(userName.getBytes("ISO-8859-1"),"gb2312")这个转码的方法中，是从编码格式为 ISO-8859-1 的字符串中取出字节内容，然后转换成 gb2312，如果原来字符的编码格式不是 ISO-8859-1 的时候这个方法就失效。例如在下面的过滤器处理中，把所有的编码都改成了 gb2312，这个时候如果再使用 new String(userName.getBytes("ISO-8859-1"),"gb2312")这个方法进行转码就会发生错误。

经过上面的更改编码格式的处理，表单的中文输入乱码问题已经解决。但是同时暴露了另一个问题，在上面这个表单中，输入项只有两项，所以对每个输入项都进行编码的转化也不是很麻烦，但是试想一下，现在如果有一个很大的表单，表单的输入项有几十个之多，这个时候对每个输入项都进行转码就很麻烦了，同时，在一个应用系统中，表单的个数也是非常多的，那么对于每个表单都得做同样的转码处理，这样的重复工作是最让人头疼的。

在这种情况下，有一种比较简便的方法，那就是使用过滤器 filter，过滤器的原理和使用方法在介绍 servlet 的时候会做详细的介绍，在这里由于需要用它处理中文问题，提前知道基本的用法即可。

过滤器的基本原理就是对于每一个用户请求，都必须经过过滤器的处理才能继续发送到目的页面，在 JSP 中，以 POST 方式提交的表单在本质上就是封装在 request 对象中，而 request 对象是必须要经过过滤器的处理的，所以，对于表单的中文问题来说，可以在 filter 中对所有的请求进行编码格式的处理，这样就不必在每个表单中都做转码处理，节省大量的时间和精力。

下面的代码就是中文处理过滤器的具体代码。

```
//-----文件名: SetCharacterEncodingFilter.java-----  
import java.io.IOException;  
import javax.servlet.Filter;  
import javax.servlet.FilterChain;  
import javax.servlet.FilterConfig;  
import javax.servlet.ServletException;  
import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;  
import javax.servlet.UnavailableException;  
public class SetCharacterEncodingFilter implements Filter {  
  
    protected FilterConfig filterConfig;  
    protected String encodingName;
```



```

protected boolean enable;

public SetCharacterEncodingFilter() {
    this.encodingName = "gb2312";
    this.enable = false;
}

public void init(FilterConfig filterConfig) throws ServletException {
    this.filterConfig = filterConfig;
    loadConfigParams();
}

private void loadConfigParams() {
    this.encodingName = this.filterConfig.getInitParameter("encoding");
    String strIgnoreFlag = this.filterConfig.getInitParameter("enable");
    if (strIgnoreFlag.equalsIgnoreCase("true")) {
        this.enable = true;
    } else {
        this.enable = false;
    }
}

public void doFilter(ServletRequest request, ServletResponse response,
                    FilterChain chain) throws IOException, ServletException {
    if(this.enable) {
        request.setCharacterEncoding(this.encodingName);
    }
    chain.doFilter(request, response);
}

public void destroy() {
}
}

```

filter 本质上就是一个 servlet，如果这个 servlet 要起作用的话就需要在 web.xml 中进行配置，对于这个 filter 可以在 web.xml 中添加下面的内容。

```

<filter>
    <filter-name>SetCharacterEncoding</filter-name>
    <filter-class>
        SetCharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>gb2312</param-value>
    </init-param>
    <init-param>
        <param-name>enable</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>SetCharacterEncoding</filter-name>
    <url-pattern>/*</url-pattern>

```

```
</filter-mapping>
```

经过这样的处理，所有的请求都会被转换成 gb2312 的编码格式，这样表单中的中文输入就可以正常获取并显示，不必对每个输入项再做转换编码的处理。

注意：web.xml 是每个 Web 应用系统必需的一个配置文件，在这里可以配置应用系统中的一些基本信息，还可以配置应用系统中的 servlet 等。这个文件改动以后，需要重新启动 Tomcat 以后才能生效。

4.12.4 数据库操作中文乱码

在建立数据库的时候，应该选择支持中文的编码格式，最好能和 JSP 页面的编码格式保持一致，这样就可以尽可能减少数据库操作的中文乱码问题。同时在 JDBC 连接数据库的时候可以使用类似下面这种形式的 URL。

```
jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=pubs;useUnicode=true;characterEncoding=gb2312
```

在上面这行代码中，指定了访问数据所使用的编码策略。在这里我们选择的数据库驱动程序为微软提供的官方驱动，连接的数据库为 SqlServer 2000 自带的 pubs 数据库，选择的编码为 gb2312。

上面所说的方法仅仅适用于创建数据库的时候已经选择支持中文的编码，但是当数据库已经创建，而且编码格式已经是 ISO-8859-1 的时候，这种情况下通过重新创建数据库显然是不显示的，尤其是当数据库中已经有大量数据的时候，这时候用上面介绍的方法可以正常向数据库中写入中文数据，但是读出的数据格式是 ISO-8859-1，如果有中文内容就会显示乱码。这时候只有在读取数据库的时候进行转码，使用的方法还是上一节中的转码方法，在这里把它整理成一个转码的函数。

```
public String encoder(String str) throws UnsupportedEncodingException
{
    String result = new String(str.getBytes("ISO-8859-1"), "gb2312");
    return result;
}
```

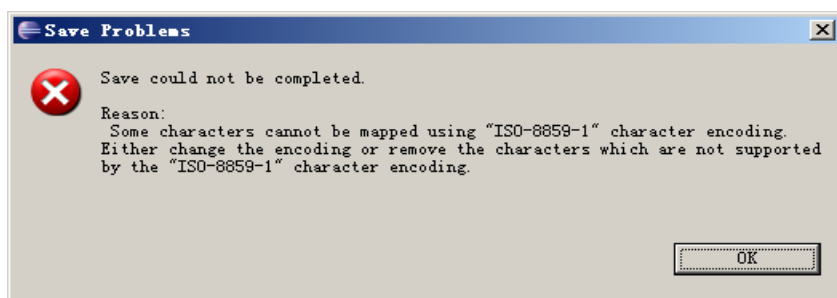
有了这个转码函数，在读取数据库的时候就可以使用 encoder(rs.getString("列名"))，这样取得中文字符串的内容就可以正常显示。

4.12.5 Eclipse 开发工具中 JSP 文件中文不能保存

在 Eclipse 中，JSP 文件默认的编码格式为 ISO-8859-1，所以在 JSP 代码中间如果出现中文就不能保存，像下面这段 JSP 代码就不能保存。

```
<%@ page language="java" import="java.util.*" %>
<html>
<head>
<title>中文显示示例</title>
</head>
<body>
这是一个中文显示示例:
</body>
</html>
```

这段代码在保存的时候会有如图 4.26 所示的提示。



对与这个问题，只要在 JSP 页面中指明页面编码即可。

```
<%@ page language="java" import="java.util.*" %>
```

把上面这行代码添加支持中文的页面编码就能保存，具体代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
```

其中 `pageEncoding="gb2312"` 指明了 JSP 页面编码采用 `gb2312`, 这样就可以正常保存 JSP 的源文件。

4.12.6 Eclipse 开发工具中中文显示乱码

在 Eclipse 中，由于默认的 JSP 编码格式为 ISO-8859-1，所以当打开由其他编辑器编辑的 JSP 页面就会出现乱码，下面这个 JSP 程序就是用 UltraEdit 编辑的页面，中文完全可以显示。

```
<%@ page language="java" import="java.util.*"%>
<html>
<head>
<title>中文显示示例</title>
</head>
<body>
这是一个中文显示示例:
<%
    out.print("这里是用 JSP 输出的中文。");
%>
</body>
</html>
```

但是在 Eclipse 中打开以后却成为下面这样。

```
<%@ page language="java" import="java.util.*"%>
<html>
<html>
    <head>
        <title>?D?????"?"ay</title>
    </head>
    <body>
        ?a?"?????D?????"?"ayjêo
    <%
        out.print("?a?"?"®?JSP"?3?|!??D???jê");
    %>
    </body>
</html>
```

在上面这个 JSP 中，所有的中文都变成乱码。造成这个问题的原因是两个编辑器保存源文件的编码

格式不同，在 UltraEdit 中可以支持中文，但是在 Eclipse 中对 JSP 文件的保存方式为 ISO-8859-1，而这种编码格式对中文不支持，所以就出现了乱码的问题。

(1) 对于这个问题的解决方法是更改 Eclipse 默认的编码方案。在 Eclipse 中选择“Window”|“Preferences”会弹出如图 4.27 所示的窗口。

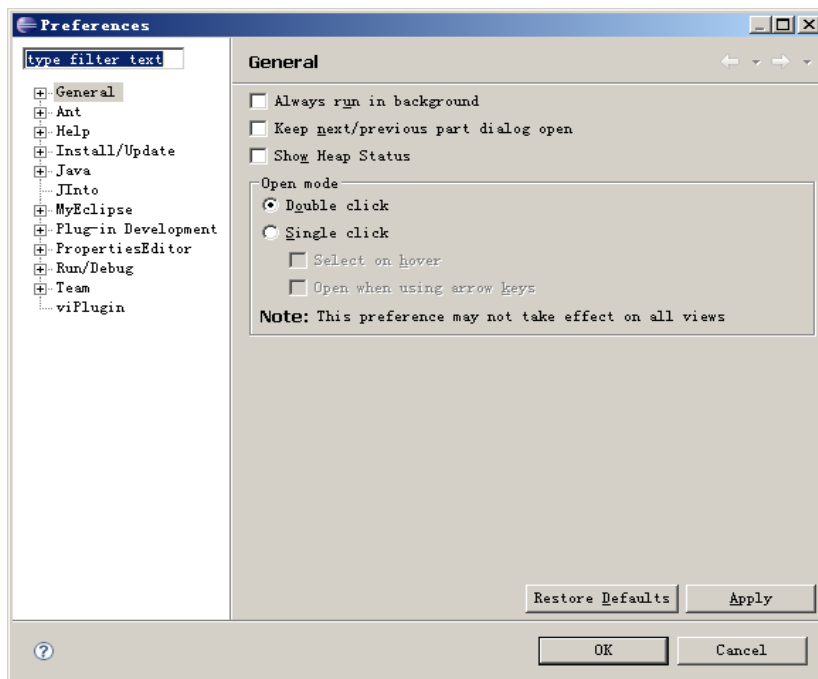


图 4.27 Eclipse 选项定制窗口

(2) 在上面这个窗口中选择“General”|“Content Types”会切换到如图 4.28 所示的界面。

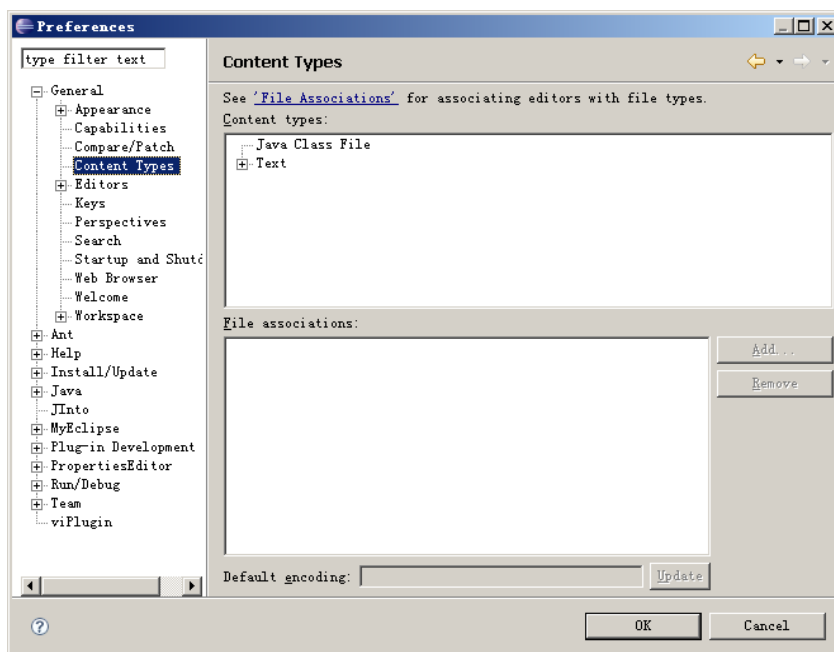


图 4.28 Eclipse 内容类型设置窗口

(3) 在上面的窗口的右上部分，选择“Text”|“JSP”在上面窗口的右下部分就会出现各种 JSP

文件的列表，选择*.jsp，然后按照图 4.29 设置，然后单击“Update”更新设置，最后重新启动 Eclipse 中文就可以在 Eclipse 中正常显示。

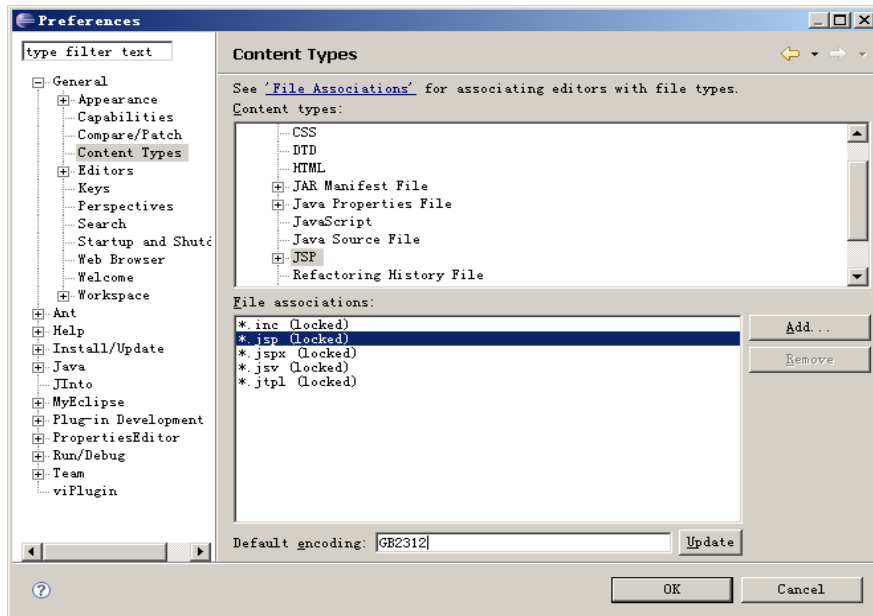


图 4.29 Eclipse 中 JSP 文件编码设置

4.12.7 JSP 下载中文文件名乱码

在实现文件下载功能的时候，如果出现中文文件名，如果不进行特殊的处理，下载下来的中文文件名会变成乱码，在下载前，就需要对这个文件名进行处理，然后才能正常显示中文的文件名，具体处理方法示例如下。

```
//-----文件名: FileNameCharset.jsp-----
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%@ page import="java.io.*" %>
<%
    String fname = "中文";
    OutputStream os = response.getOutputStream();//取得输出流
    response.reset();//清空输出流

    //下面是对中文文件名的处理
    response.setCharacterEncoding("UTF-8");
    fname = java.net.URLEncoder.encode(fname, "UTF-8");
    response.setHeader("Content-Disposition", "attachment; filename="+ new String(fname.getBytes("UTF-8"),
"gb2312") + ".xls");
    response.setContentType("application/msexcel");//定义输出类型
    os.close();

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

经过上面的处理，下载下来的中文文件名称可以正常显示。

总之，在 JSP 中出现中文乱码的问题，在不同的服务器中有不同的原因，但最根本的原因都是相通的，那就是编码格式的冲突，只要知道使用的服务器采用的编码格式，然后再根据这个编码格式把中文字符串转码即可。读者要在上面的这些例子中理解中文处理的关键所在，要做到以不变应万变。

4.13 其他 JSP 开发技巧

4.13.1 自定义错误页面

在 JSP 中，如果出现在代码的错误，就会直接在页面上打印类似如图 4.30 所示的错误信息。

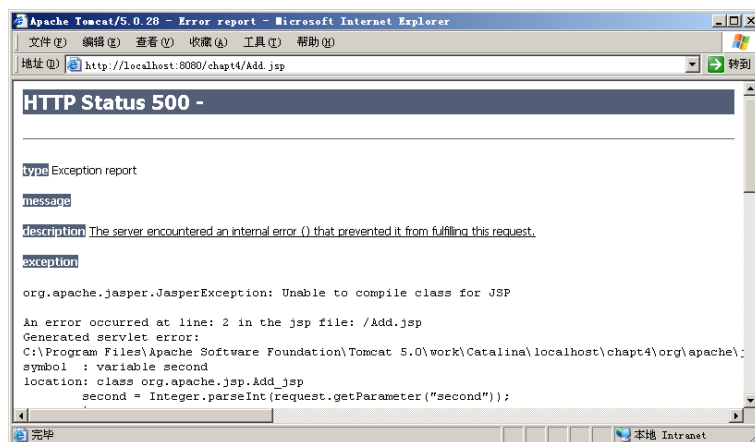


图 4.30 JSP 代码错误的页面

当用户看到这样的页面，肯定会不知所措，用户不会明白出现了什么问题，所以有必要在 JSP 页面中统一指定错误处理页面。例如在 JSP 页面中的 page 指令中，可以设置错误页面，具体设置方法如下。

```
<%@page language="java" contentType="text/html; charset=gb2312" errorPage="error.jsp"%>
```

上面这行代码中 errorPage="error.jsp"指定了这个页面出错以后会转向 error.jsp，我们可以在 error.jsp 中详细描述出错信息，让用户可以明白发生了什么情况。

上面介绍的这种方法当然可以很好的解决问题，但是由于在每个 JSP 页面都要添加错误页面的设置，当页面比较多时，工作量就相当大。所以我们提供一种更方便的解决办法。

在前面章节中介绍过，每个 Web 应用都需要一个 web.xml 文件，这个文件中是 Web 应用的基本配置信息，同样我们可以在这个文件中统一配置错误处理页面，具体操作方法就是在 web.xml 中添加如下内容。


```
<error-page>
  <error-code>500</error-code>
  <location>error.jsp</location>
</error-page>
```

上面这段代码指定了 500 错误时的错误处理页面，其中 500 错误即服务器内部错误，读者可以根据这个示例尝试配置其他错误类型的处理页面，例如 400 没有找到请求页面的错误处理页面。

4.13.2 在 MyEclipse 中快速部署项目

在 Web 应用开发的过程中，部署项目往往十分麻烦，虽然在后续章节中介绍的 Ant 可以非常方便的完成这个任务，但是 Ant 复杂的操作不适合初学者，在这里我们使用前面推荐的 MyEclipse 这个集成开发工具来部署项目。

MyEclipse 的安装在前面第二章中已经详细介绍，在这里直接开始介绍如何发布 Web 应用项目。要想发布部署一个项目，首要任务就是把 MyEclipse 和服务器 Tomcat 联系起来，下面就是详细的配置步骤。

- (1) 在 MyEclipse 的工具栏中单击这个服务器的标志，就会看到如图 4.31 所示的界面。
- (2) 在图 4.31 中可以清楚的看到，这时候还没有可用的服务器，所以需要进行服务器的设置。单击 “No enabled servers available.....” 这个区域，就会出现如图 4.32 所示的界面。

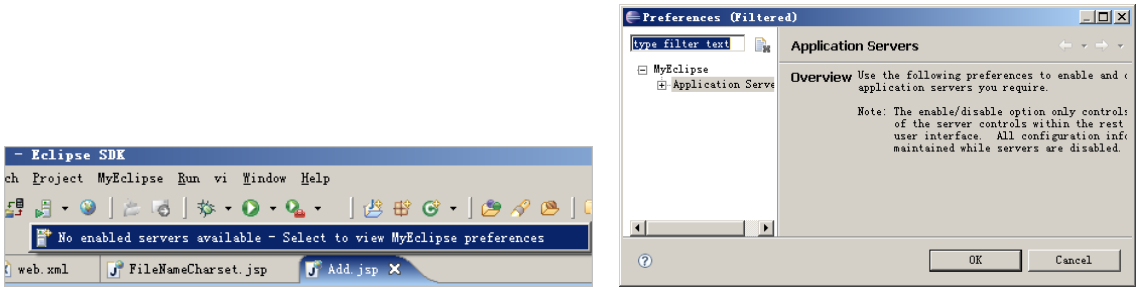


图 4.31 MyEclipse 中没有配置服务器情况下的界面 图 4.32 MyEclipse 服务器配置页面

- (3) 在如图 4.32 所示的界面中，选择 “Application Servers” | “Tomcat 5” 可以看到如图 4.33 所示的界面。

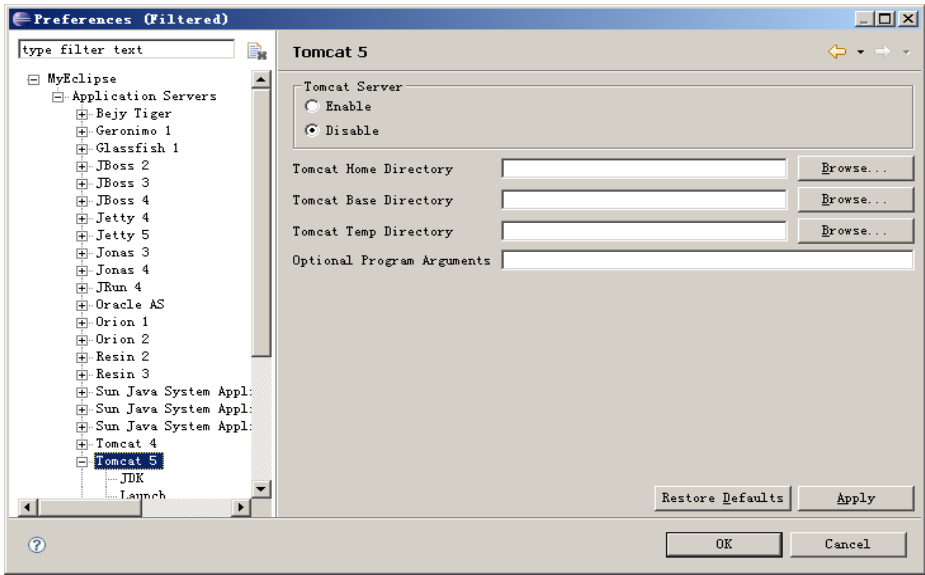


图 4.33 MyEclipse 中的 Tomcat 配置界面

- (4) 按照如图 4.34 所示的详细配置进行设置。

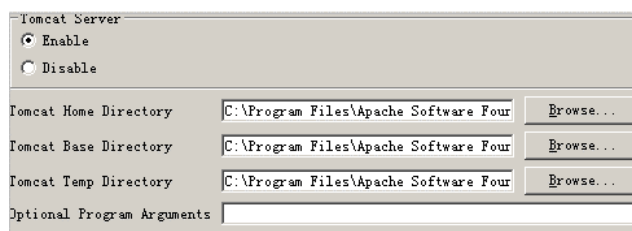



图 4.34 MyEclipse 中 Tomcat 的详细配置

在图 4.34 中，需要设置的只有两处，第一需要选择 Tomcat Server 下面的 Enable，这个选项可以激活对应的 Tomcat 服务器。第二处设置是选择 Tomcat Home Directory，即选择 Tomcat 在硬盘中的安装位置，这个属性选择以后，其他两个属性都可以自动生成。

(5) 按照图 4.34 设置完成之后，单击如图 4.33 所示的“Apply”，到这里在 MyEclipse 中配置 Tomcat 的设置就结束了。

4.13.3 测试配置是否成功

下面来测试下配置是否成功，在 MyEclipse 的工具栏中单击  图标，会出现如图 4.35 所示的界面。在图 4.35 所示的界面中，选择 Tomcat 5，会出现如图 4.36 所示的界面。

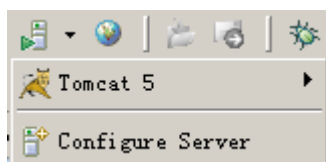


图 4.35 MyEclipse 中的 Tomcat 菜单

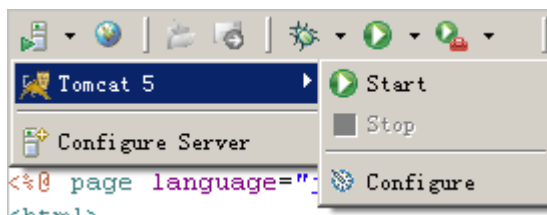


图 4.36 MyEclipse 中的 Tomcat 启动菜单

在如图 4.36 所示的界面中选择“Start”就可以启动 Tomcat 服务器，如果在 MyEclipse 的控制台看到如图 4.37 所示的界面就说明 Tomcat 在 MyEclipse 中配置成功。

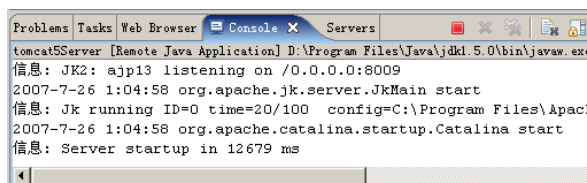



图 4.37 MyEclipse 中 Tomcat 的启动信息

通过上面一系列的设置，现在已经可以在 MyEclipse 开发环境中调用 Tomcat 服务器。接下来将介绍如何在 MyEclipse 中快速部署项目。

- (1) 在 MyEclipse 的工具栏中选择  这个图标，就会得到如图 4.38 所示的界面。
- (2) 在如图 4.38 所示的界面中选择“Add”，就可以得到如图 4.39 所示的界面。

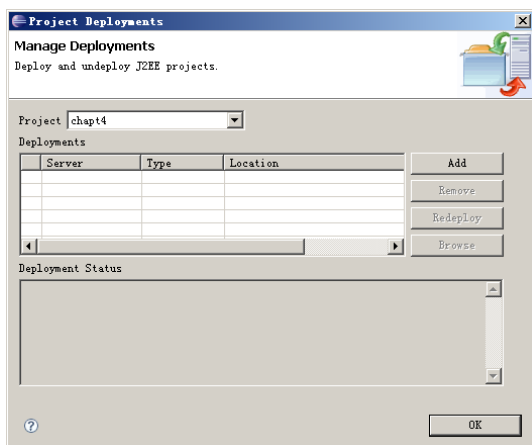


图 4.38 MyEclipse 中的项目部署界面

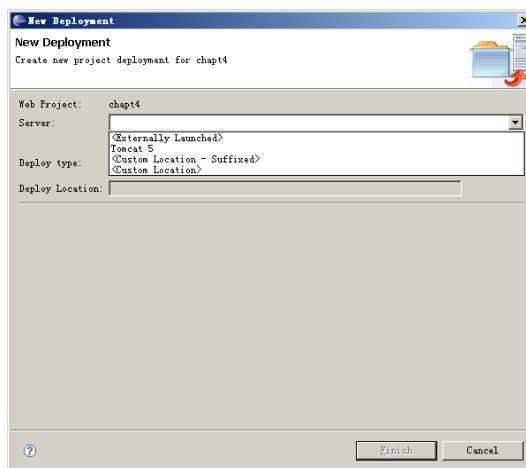


图 4.39 创建新的项目部署界面

在图 4.39 所示的界面中，从 Server 的下拉列表中选择 Tomcat 5，这个 Tomcat 服务器就在前面的设置中已经成功配置。选择 Tomcat 以后，上面这个界面中的 Finish 按钮就被激活，选择“Finish”就结束了部署，然后启动 Tomcat 就可以访问部署成功的应用。

注意：在使用 MyEclipse 快速部署项目的时候，默认把工程名称当作应用部署时候的名称。

4.13.3 在 MyEclipse 中调试 Web 应用程序

在 MyEclipse 中，对 JSP 页面进行调试也是非常方便的，如果需要调试 JSP 页面，只需要在 JSP 页面源代码的左侧双击鼠标左键，当出现如图 4.40 所示的小圆点的时候说明断点设置成功。

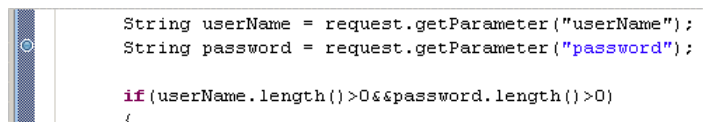


图 4.40 在 MyEclipse 中设置断点

断点设置成功以后，再次访问这个页面的时候，程序运行到断点位置就会停止，并且把 MyEclipse 切换到如图 4.41 所示的 Debug 视图。

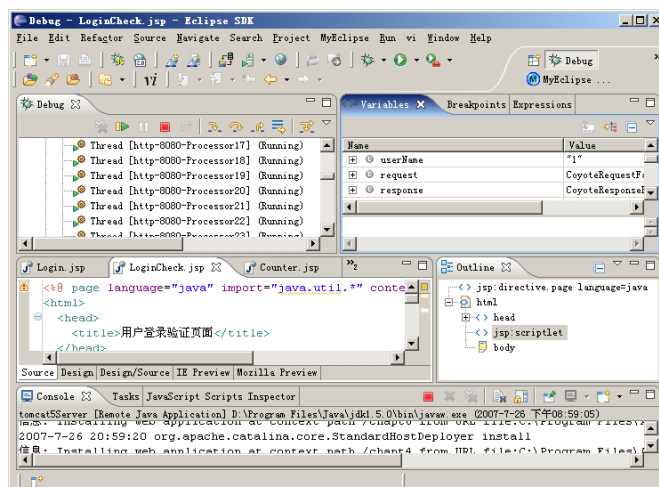


图 4.41 MyEclipse 调试界面

在上面这个界面中，右上角可以查看变量值，断点位置，表达式等信息，中间的两个窗口是源代码窗口源代码大纲，最下面的窗口是控制台信息输出窗口。

当然，我们可以在程序中设置多个断点，不仅可以为 JSP 程序设置断点，而且还可以给 Java 代码设置断点，MyEclipse 中 Debug 有几个常用的快捷键 F8 恢复，F5 进入方法，F6 单步执行跳过语句，F7 跳出方法。

4.13.4 学习使用日志 Log4j

在一般情况下，我们可以把错误信息，或者是调试的信息输入到控制台，这时候要用 `System.out.println()` 这个方法，这个方法固然可行，但是带来了很多不便，第一个问题就是在每个需要输入信息，以便调试维护的地方都需要这样的语句，这就增加了很多工作量，第二个问题是，当系统正式交付的时候需要把这些调试信息去掉，这个时候就需要一个一个找到这些输出语句，并且删除，这样带来的还是不必要的额外工作量。

在 JSP Web 开发中，有很多方便的日志工具可供选择，利用这些日志工具可以很方便的对系统中的错误信息进行管理，在这里我们选择使用 Log4j，Log4j 是目前 JSP 开发中使用最多的日志工具。Log4j 按照严重程度给日志风味 5 个等级：DEBUG（调试）、INFO（提示）、WARN（警告）、ERROR（错误）、FATAL（严重错误）。我们只需要简单的配置文件 `log4j.properties` 来就 Log4j 进行简单的配置，然后再把 `log4j.jar` 添加到项目中 WEB-INF/lib 文件夹中，就可以在程序中调用其强大的日志功能，下面是一个示例的 Log4j 的配置代码。

```
//-----文件名: log4j.properties-----  
log4j.rootLogger=ERROR,console,file  
log4j.appender.console=org.apache.log4j.ConsoleAppender  
log4j.appender.file=org.apache.log4j.FileAppender  
log4j.appender.file.File=system.log  
log4j.appender.console.layout=org.apache.log4j.SimpleLayout  
log4j.appender.file.layout=org.apache.log4j.SimpleLayout
```

上面这个配置文件配置了日志的级别为错误级别，日志信息在控制台和文件中输出，其中输入到文件的名称为 `system.log`。经过上面这样简单的配置，就可以在程序中使用类似下面的方法使用这个日志工具。

```
public class Hello{  
    private static Log logger = LogFactory.getLog(Hello.class);  
    public sayHello ( ){  
        logger.error(" sayHello method error!");  
    }  
}
```

4.14 小结

在本章中，对 JSP 的基本语法和对象等知识进行了系统的介绍，而且对于其中大部分的知识点都给出了具体示例，这些示例在具体的开发过程中都有很大的参考价值，读者可以在这些示例程序的基础上进行尝试，试着修改其中的功能，只有这样才能对其运行原理有更深入的了解和体会，这就是学习程序语言的最基本最有效的方法。

第 5 章 Servlet 技术基础知识

Servlet 是一种服务器端的编程语言，是 J2EE 中比较关键的组成部分，Servlet 技术的推出，扩展了 Java 语言在服务器端开发的功能，巩固了 Java 语言在服务器端开发中的地位，而且现在使用非常广泛的 JSP 技术也是基于 Servlet 的原理，JSP+JavaBeans+Servlet 成为实现 MVC 模式的一种有效的选择。在本章中将介绍 Servlet 的基础知识，并通过具体的示例介绍 Servlet 的强大功能。

5.1 Servlet 简介

Servlet 在本质上就是 Java 类，编写 Servlet 需要遵循 Java 的基本语法，但是与一般 Java 类所不同的是，Servlet 是只能运行在服务器端的 Java 类，而且必需遵循特殊的规范，在运行的过程中有自己的生命周期，这些特性都是 Servlet 所独有的。另外 Servlet 是和 HTTP 协议是紧密联系的，所以使用 Servlet 几乎可以处理 HTTP 协议各个方面的内容，这也正是 Servlet 收到开发人员青睐的最大原因。

5.1.1 Servlet 的工作原理

Servlet 需要在特定的容器中才能运行，在这里所说的容器即 Servlet 运行的时候所需的运行环境，一般情况下，市面上常见的 Java Web Server 都可以支持 Servlet，例如 Tomcat、Resin、Weblogic、WebSphere 等，在本书中采用 Tomcat 作为 Servlet 的容器，由 Tomcat 为 Servlet 提供基本的运行环境。

Servlet 容器环境在 HTTP 通信和 web 服务器平台之间实现了一个抽象层。Servlet 容器负责把请求传递给 Servlet，并把结果返回给客户。容器环境也提供了配置 Servlet 应用的简单方法，并且也提供用 XML 文件配置 Servlet 的方法。当 Servlet 容器收到对用户 Servlet 请求的时候，Servlet 引擎就会判断这个 Servlet 是否是第一次被访问，如果是第一次访问，Servlet 引擎就会初始化这个 Servlet，即调用 Servlet 中的 `init()` 方法完成必要的初始化工作，当后续的客户请求 Servlet 服务的时候，就不再调用 `init()` 方法，而是直接调用 `service()` 方法，也就是说每个 Servlet 只被初始化一次，后续的请求只是新建一个线程，调用 Servlet 中的 `service()` 方法。

在使用 Servlet 的过程中，并发访问的问题由 Servlet 容器处理，当多个用户请求同一个 Servlet 的时候，Servlet 容器负责为每个用户启动一个线程，这些线程的运行和销毁由 Servlet 容器负责，而在传统的 CGI 程序中，是为每一个用户启动一个进程，因此 Servlet 的运行效率就要比 CGI 的高出很多。

5.1.2 Servlet 的生命周期

Servlet 是运行在服务器端的程序，所以 Servlet 的运行状态完全由 Servlet 容器维护，一个 Servlet 的生命周期一般有三个过程。

1. 初始化

当一个 Servlet 被第一次请求的时候，Servlet 引擎就初始化这个 Servlet，在这里是调用 `init()` 方法完

成必需的初始化工作。而且这个对象一致在内存中活动，Servlet 为后续的客户请求新建线程，直接调用 Servlet 中的 service（）方法提供服务，不再初始化 Servlet。

2. 提供服务

当 Servlet 对象被创建以后，就可以调用具体的 service（）方法为用户提供服务。

3. 销毁

Servlet 被初始化以后一直再内存中保存，后续的访问可以不再进行初始化工作，当服务器遇到问题需要重新启动的时候，这些对象就需要被销毁，这时候 Servlet 引擎就会调用 Servlet 的 destroy（）方法把内存中的 Servlet 对象销毁。

5.1.3 简单 Servlet 开发配置示例

Java Servlet API 包括两个基本的包，javax.servlet 和 javax.servlet.http，其中 javax.servlet 提供了用来控制 Servlet 生命周期所需的类和接口，是编写 Servlet 必需要实现的。javax.servlet.http 提供了处理与 HTTP 相关操作的类和接口，每个 Servlet 必需实现 Servlet 接口，但是在实际的开发中，一般情况都是通过继承 javax.servlet.http.HttpServlet 或者 javax.servlet.GenericServlet 来间接实现 Servlet 接口。

下面这个简单的示例程序就是继承了 javax.servlet.http.HttpServlet，从而实现 Servlet 接口，具体的代码如下。

```
//-----文件名: HelloWorld.java-----
package servlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

在上面这个简单的示例程序中，继承了 HttpServlet，而 HttpServlet 是一个实现了 Servlet 接口的类，所以这个 Servlet 就间接地实现了 Servlet 的接口，从而可以使用接口提供的服务。

在这个程序中，并没有具体的 init（）方法和 destroy（）方法，这里使用 Servlet 容器默认的方式对这个 Servlet 进行初始化和销毁动作，而在这里的 doGet（）方法就是具体的功能处理方法，这个方法可

以对以 get 方法发起的请求进行处理，在这里这个方法的功能就是打印出一个 HTML 页面。

上面这个程序编译以后会生成对应的 Servlet 类文件，把编译生成的类文件拷贝到当前应用项目的 WEB-INF/classes 文件夹中，Servlet 引擎会到这个目录下面寻找 Servlet 的类文件。这里需要注意，如果 Servlet 有包名，这时候把需要把整个包拷贝，而不是仅仅拷贝类文件。负责 Servlet 引擎就会找不到 Servlet 类文件

Servlet 引擎需要通过配置文件来找到具体的 Servlet，在一般情况下都需要在当前应用项目的 web.xml 配置文件中对各个 Servlet 进行配置，其中 web.xml 文件的位置在当前项目应用的 WEB-INF 文件夹下。对于上面这个简单的 Servlet 示例，可以在 web.xml 中进行如下的配置。

```
<servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>servlets.HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

对于每一个 Servlet 都需要像上面这样的配置信息，这时 Servlet 引擎初始化 Servlet 所必需的信息，这个配置信息可以分为两个部分，第一部分是配置 Servlet 的名称和类，第二部分是配置 Servlet 的访问路径，其中<servlet-name>是这个 Servlet 的名称，这个名字可以任意命名，但是要和<servlet-mapping>节点中的<servlet-name>保持一致，<servlet-class>是 Servlet 对应类的路径，在这里要注意，如果有 Servlet 带有包名，一定要把包路径写全，否则 Servlet 引擎就找不到对应的 Servlet 类。<servlet-mapping>节点是对 Servlet 的访问路径进行配置，<url-pattern>定义了 Servlet 的访问路径。

经过上面的步骤，就可以使用类似 <http://localhost:8080/chapt5/HelloWorld> 这样的网址访问这个 Servlet，其中 <http://localhost:8080/> 是本地 Tomcat 服务器的访问地址，chapt5 是当前应用项目的名称，HelloWorld 是 Servlet 的访问路径，当 Servlet 引擎接收到这样的请求的时候，就会初始化 HelloWorld 这个 Servlet，然后调用其中的方法为用户提供服务。

上面这个简单的示例程序的运行效果如图 5.1 所示。



图 5.1 HelloWorld 示例程序运行效果

总之，编写一个 Servlet 要经过以下几个步骤。

- (1) 编写 Servlet 的功能代码，即实现功能的代码类。
- (2) 把编译成功的 Servlet 功能代码类文件拷贝到当前应用项目的 WEB-INF/classes 目录下。
- (3) 在当前应用项目的 web.xml 文件中对 Servlet 进行配置，即在 web.xml 中添加配置信息。

经过这样三个步骤就可以通过浏览器访问这个 Servlet。

注意：Servlet 的配置信息需要添加在<web-app></web-app>标签之间。

5.1.4 使用 Servlet 实现 MVC 开发模式

Java 语言之所以受到开发人员支持，是因为 Java 语言实现科学方便的开发模式，在这些开发模式中，

最出色而且应用最广的就是 MVC 模式，对于 MVC 模式的研究由来已久，但是一直没有得到很好的推广和应用，随着 J2EE 技术的成熟，MVC 逐渐成为了一种常用而且重要的设计模式。

MVC（Model-View-Controller）把应用程序的开发分为三个层面：视图层、控制层、模型层。其中视图层负责从用户获取数据和向用户展示数据，在这层中不负责业务逻辑的处理和数据流程的控制。而模型层负责处理业务逻辑和数据库的底层操作，其中视图层和模型层之间没有直接的联系。控制层主要负责处理视图层和模型层的交互，控制层从视图层接收请求，然后从模型层取出对请求的处理结果，并把结果返回给视图层。在控制层中只负责数据的流向，并不涉及具体的业务逻辑处理。

MVC 三层结构的内部关系如图 5.2 所示。

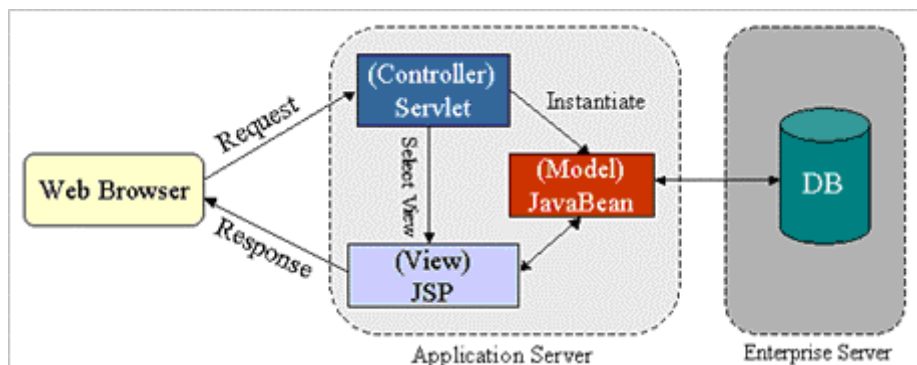


图 5.2 MVC 三层结构的内部关系

从图 5.2 中可以看出，Servlet 在 MVC 开发模式中承担着重要的角色，在 MVC 结构中，控制层就是依靠 Servlet 实现，Servlet 可以从浏览器端接收请求，然后从模型层取出处理结果，并且把处理结果返回给浏览器端的用户。在整个结构中，Servlet 负责数据流向控制的功能。

虽然现在用很多开源框架都很好的实现了 MVC 的开发模式，例如 Struts、WebWork 等，这些开源框架对 MVC 的实现都是非常出色的，在这些框架中，处理数据控制流向的时候，采用的还是 Servlet，例如在 Struts 中，对应每一个用户请求都有一个 Action，这个 Action 就是继承了 Servlet 的类，所以在 MVC 架构中，Servlet 是不可替代的。

5.2 JSP 页面调用 Servlet 的方法

在上面 HelloWorld 的示例程序中，我们直接在浏览器中输入具体的地址进行访问，在实际的应用中，不可能让用户在浏览器中直接输入 Servlet 的地址进行访问，一般情况下，可以通过调用 Servlet 进行访问，在这里介绍通过提交表单和超链接两种方式调用 Servlet。

5.2.1 通过表单提交调用 Servlet

在通过提交表调用 Servlet 的时候，只需要把表单的 action 指向对应的 Servlet 即可，下面是一个简单的表单，通过这个表单可以调用指定的 Servlet。

```
//-----文件名: form.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
```

```

<title>Servlet 接收表单示例</title>
</head>
<body>
    <font size="2">
        <form action="AcceptForm" method="post">
            姓名: <input type="text" name="name"/><br>
            省份: <input type="text" name="province"><br>
            <input type="submit" value="提交">
        </form>
    </font>
</body>
</html>

```

在上面这个表单中,指明表单的处理程序是 AcceptForm 这个 Servlet,下面是 AcceptForm 这个 Servlet 的具体代码。

```

//-----文件名: AcceptForm.java-----
package servlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AcceptForm extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("name");
        String province = request.getParameter("province");

        out.println("<font size='2'>");
        out.print("提交的表单内容为:<br>");
        out.print("姓名:" + name + "<br>");
        out.print("省份:" + province + "<br>");
        out.print("</font>"); }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        doGet(request, response);
    }
}

```

在上面这个 Servlet 中,只对 doGet () 方法进行重写,在 doPost () 方法中直接调用 doGet () 方法的具体内容。

下面来介绍这个 Servlet 的关键代码。

```
response.setContentType("text/html");
```

上面这行代码设置了服务器响应的内容格式为 HTML 文档格式。

```
response.setCharacterEncoding("gb2312");
```

这里设置服务器响应内容的字符编码格式为 gb2312。用来支持中文显示。

```
PrintWriter out = response.getWriter();
```

上面这行代码取出输出对象，用来在页面上输出要显示的内容。

```
String name = request.getParameter("name");  
String province = request.getParameter("province");
```

上面这两行代码取出表单中的输出内容。

在前面 HelloWorld 的例子中已经介绍过，每个 Servlet 必需在 web.xml 中进行配置，然后 Servlet 引擎才能通过访问路径找到对应的 Servlet 类文件。上面这个 Servlet 的配置信息如下。

```
<servlet>  
    <servlet-name>AcceptForm</servlet-name>  
    <servlet-class>servlets.AcceptForm</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>AcceptForm</servlet-name>  
    <url-pattern>/AcceptForm</url-pattern>  
</servlet-mapping>
```

上面这个程序的执行效果如图 5.3 和图 5.4 所示。

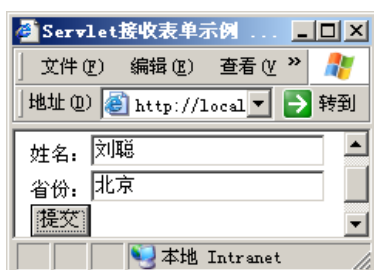


图 5.3 提交表单内容



图 5.4 Servlet 处理表单结果

注意：在这里设置字符编码格式的操作要在取得输出对象之前，即在取出 PrintWriter 对象之前就要设置好字符的编码格式，如果在这之后设置，中文字符还是不能正常显示的。

5.2.2 通过超链接调用 Servlet

在上面这个例子中，用户有输入的内容需要提交给服务器，所以需要表单来调用 Servlet，但是在没有输入的数据内容需要提交的情况下，使用表单就不是很合理了，在这里介绍 Servlet 的第二种调用方法，直接通过超链接的方式来调用 Servlet，在这种情况下还可以给 Servlet 传递参数。

下面的例子中就是使用超链接的方式调用 Servlet，并且向这个 Servlet 传递一个参数。这个调页面的代码如下。

```
//-----文件名: link.jsp-----  
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>  
<html>  
    <head>  
        <title>Servlet 接收链接传递参数示例</title>  
    </head>  
    <body>  
        <font size="2">  
            单击下面的链接: <br>  
            <a href="AcceptLink?name=Bill">调用 Servlet,并传递参数</a>  
        </font>  
    </body>
```


</html>

在上面这个 JSP 页面中，通过这条超链接语句可以生成类似下面<http://localhost:8080/chapt5/AcceptLink?name=Bill> 这样的访问地址，其中 AcceptLink 就是这个链接调用的 Servlet，并且这个链接还向 AcceptLink 这个 Servlet 传递了一个名为 name 的参数，这个参数的值为 Bill。AcceptLink 这个 Servlet 的具体代码如下。

```
//-----文件名: AcceptLink.java-----
package servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class AcceptLink extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("name");

        out.println("<font size='2'>");
        out.println("链接传递过来的参数为: <br>");
        out.println("参数名称: name<br>");
        out.println("参数值: " + name + "<br>");
        out.print("</font>");
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        doGet(request, response);
    }
}
```

在这个 Servlet 中，取出调用页面传递过来的参数，然后在页面打印，其中获取参数的方法仍然是 request.getParameter("name")。这个 Servlet 的配置信息如下。

```
<servlet>
    <servlet-name>AcceptLink</servlet-name>
    <servlet-class>servlets.AcceptLink</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>AcceptLink</servlet-name>
    <url-pattern>/AcceptLink</url-pattern>
</servlet-mapping>
```

上面这个示例程序的运行效果如图 5.5 和 5.6 所示。



图 5.5 超链接调用 Servlet 页面

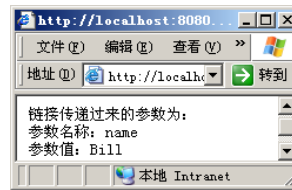


图 5.6 Servlet 处理链接传递参数页面

5.3 Servlet 中的文件操作

在 JSP 的开发过程中，经常会遇到需要把相关内容存储为文件的情况，在 JSP 中是用输入输出流进行操作的，在 Servlet 中也可以使用输入输出流实现对文件的读写，同时，使用 Servlet 还可以很方便的实现文件的上传下载。接下来的内容将通过具体的示例展示 Servlet 文件操作的方法。

5.3.1 Servlet 读取文件

在这个例子中将要读取一个文本文件的内容，并且在页面上打印文件的内容。这个 Servlet 的代码如下。

```
//-----文件名: FileRead.java-----
package servlets;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FileRead extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        String fileName = "content.txt";
        String realPath = request.getRealPath(fileName);

        File file = new File(realPath);
        if(file.exists())
        {
```

```

        FileReader reader = new FileReader(file);
        BufferedReader bufferReader = new BufferedReader(reader);
        String line = null;
        while((line = bufferReader.readLine())!=null)
        {
            out.print("<font size='2'>"line+"</font><br>");
        }
    }else
    {
        out.print("文件不存在！ ");
    }
}

public void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException
{
    doGet(request, response);
}
}

```

在上面这个 Servlet 中，首先从一个文本文件 content.txt 中读取内容，然后把读取的内容打印在页面。下面解释这个 Servlet 的关键代码。

```

response.setContentType("text/html");
response.setCharacterEncoding("gb2312");

```

上面两行代码设置服务器的响应内容类型是 HTML 格式，字符编码格式为 gb2312，用来支持中文的显示，如果没有在这里设置字符编码格式，页面上打印的中文内容就不能正常显示。

```

PrintWriter out = response.getWriter();

```

上面这行代码取得输出对象，用来在后面的操作中在页面上输出文件的内容。

```

String realPath = request.getRealPath(fileName);

```

上面这行代码把文件名转换为绝对路径，例如在这里文件名称为 content.txt，这个文本文件的位置在当前应用的根目录，那么通过上面这行代码就可以取得这个文件的绝对路径，转化以后的路径格式为 C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\chapt5\content.txt,取得文件的绝对路径之后就可以使用 Java 的 API 尽心文件的读写操作。

```

File file = new File(realPath);

```

这行代码用文件的绝对路径构造出一个文件对象，后续的操作都可以在这个文件对象上面进行。

```

FileReader reader = new FileReader(file);

```

上面这行代码用文件对象构造了一个文件读取对象。

```

BufferedReader bufferReader = new BufferedReader(reader);

```

在 Java 文件操作中，为了提高文件读取的效率，提供了一个字符输入缓冲流类，这个类就是 BufferedReader，一般情况下 BufferedReader 和 FileReader 结合使用，在上面这行代码中，使用一个 FileReader 对象构造出一个 BufferedReader 对象。

```

while((line = bufferReader.readLine())!=null)

```

BufferedReader 类中有一个 readerLine() 方法，这个方法每次从 BufferedReader 对象中读取一行字符，在上面这行代码中循环读出 BufferedReader 中的每一行内容。

这个 Servlet 的配置信息如下。

```

<servlet>
    <servlet-name>FileRead</servlet-name>

```

```

<servlet-class>servlets.FileRead</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>FileRead</servlet-name>
  <url-pattern>/FileRead</url-pattern>
</servlet-mapping>

```

上面这个 Servlet 可以用类似这样的地址进行访问 <http://localhost:8080/chapt5/FileRead>，其中 <http://localhost:8080/> 是本地计算机的访问路径，chapt5 是当前应用项目的名称，FileRead 是文件读取 Servlet，这个示例程序的运行效果如图 5.7 所示。



图 5.7 文件读取 Servlet 运行效果

5.3.2 Servlet 写文件

Servlet 写文件的处理方法和读取文件的处理方法非常类似，只是把文件输入流换成文件输出流，在下面这个示例程序中，将在指定位置生成文件。这个 Servlet 的代码如下。

```

//-----文件名: FileWrite.java-----
package servlets;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FileWrite extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        String fileName = "new.txt";
        String realPath = request.getRealPath(fileName);

        File file = new File(realPath);
        FileWriter writer = new FileWriter(file);
    }
}

```

```

        BufferedWriter bufferWriter = new BufferedWriter(writer);
        bufferWriter.write("计算机网络");
        bufferWriter.newLine();
        bufferWriter.write("高等数学");

        bufferWriter.flush();
        bufferWriter.close();
        writer.close();
        out.print("<font size='2'>文件写入成功，路径为:"+file.getAbsolutePath()+"</font>");
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        doGet(request, response);
    }
}

```

上面这个 Servlet 可以在指定位置生成一个文件，并在这个文件中写入字符串，下面解释这个 Servlet 中的关键代码。

```

String fileName = "new.txt";
String realPath = request.getRealPath(fileName);

```

上面这两行代码实现的功能还是把文件名转换成一个绝对路径。

```

File file = new File(realPath);

```

上面这行代码根据文件路径构造一个文件对象。需要注意的是，这个操作并没有在硬盘上生成指定的文件，这时候的文件对象仅仅存在内容中，并没有在硬盘上创建。

```

FileWriter writer = new FileWriter(file);

```

上面这行代码根据文件对象构造了一个 FileWriter 文件读取对象。

```

BufferedWriter bufferWriter = new BufferedWriter(writer);

```

同文件读取操作类似，为了提高文件写操作的效率，Java 语言提供了一个提供了一个字符输入缓冲流类，即 BufferedWriter，一般情况下 BufferedWriter 和 FileWriter 结合使用，在上面这行代码中，根据 FileWriter 对象构造一个 BufferedWriter 对象。

```

bufferWriter.write("计算机网络");

```

上面这行代码在 BufferedWriter 对象中写入一个字符串，这时候的操作还是在内存中进行，并不是在硬盘上的文件中写入内容，这个时候文件还没有生成。

```

bufferWriter.newLine();

```

上面这行代码实现的是换行操作。

```

bufferWriter.flush();

```

上面这行代码刷新字符输入缓冲区中的内容，这个时候才把内容中的文件对象输出到硬盘，即此时才在硬盘中生成指定的文件。

```

bufferWriter.close();
writer.close();

```

文件操作结束以后要对相关资源进行释放，上面两行代码分别关闭 FileWriter 对象和 BufferedWriter 对象，这两个对象释放以后文件操作就已经结束，如果在这两个对象关闭以后仍然在这两个对象上进行操作就会抛出异常。

上面这个 Servlet 的配置信息如下。

```

<servlet>

```

```
<servlet-name>FileWrite</servlet-name>
<servlet-class>servlets.FileWrite</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>FileWrite</servlet-name>
  <url-pattern>/FileWrite</url-pattern>
</servlet-mapping>
```

在配置工作完成以后就可以用类似 <http://localhost:8080/chapt5/FileWrite> 这样的地址进行访问，其中 <http://localhost:8080/> 是本机计算机的访问路径，chapt5 是当前应用项目的名称，FileWrite 是写文件的 Servlet，这个示例程序的运行效果如图 5.8 所示。

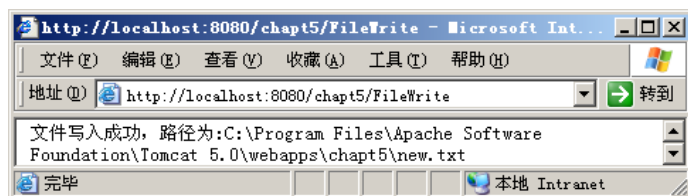


图 5.8 Servlet 写文件示例运行效果

5.3.3 Servlet 上传文件

文件的上传下载在 Web 开发中会经常遇到，使用基本的 IO 输入输出流当然可以完成这项操作，但是出于对开发的效率和程序运行的效率方面的考虑，在实际的开发过程中一般采用第三方的组件来完成这个上传的功能。

在实际开发过程中用的比较多的是 commons-fileupload 组件和 jspSmartUpload 组件，这两个组件都可以很好地完成文件上传的功能，在本书中选择使用 commons-fileupload 组件，这个组件是 Apache 基金开发维护的，可以在 <http://jakarta.apache.org/site/downloads> 找到各种版本的下载，目前的版本为 1.2，下载下来的文件是一个压缩文件 commons-fileupload-1.2-bin.zip，解压这个文件可以得到如图 5.9 所示的目录结构。



图 5.9 commons-fileupload 组件的目录结构

在如图 5.9 所示的 lib 文件夹中有一个名为 commons-fileupload-1.2.jar 的文件，这个就是 commons-fileupload 组件的类库，把这个文件拷贝到当前应用项目的 WEB-INF\lib 文件夹中。

使用 commons-fileupload 组件的时候，需要 Apache 另外一个组件 commons-io 的支持，commons-io 组件也可以在 <http://jakarta.apache.org/site/downloads> 上找到，当前最新的版本为 1.3.2，下载下来的文件为 commons-io-1.3.2-bin.zip，把这个文件解压后可以得到如图 5.10 所示的目录结构。

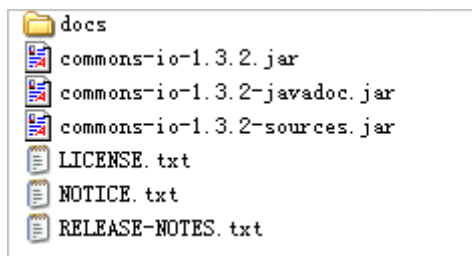


图 5.10 commons-io-1.3.2 组件的目录结构

把如图 5.10 所示的 commons-io-1.3.2.jar 拷贝到当前应用项目的 WEB-INF\lib 的文件夹下，这个时候 commons-fileupload 组件的配置工作就完成了，可以在项目中开始使用 commons-fileupload 组件提供的文件上传功能。

在下面这个例子中将详细介绍如何使用这个组件完成文件上传的功能。

下面这个页面是上传文件选择表单页面。

```
//-----文件名: upload.jsp-----
<%@ page contentType="text/html; charset=gb2312" %>
<html>
<head>
<title>commonfileupload 上传文件示例</title>
</head>
<body>
<font size="2">
commonfileupload 上传文件示例 <br>
<form method="post" action="FileUpload" ENCTYPE="multipart/form-data">
文件:<input type="file" name="file">
<input type="submit" value="上传" name="submit">
</form>
</font>
</body>
</html>
```

在这里需要注意的是，上传文件的时候表单中需要添加 ENCTYPE="multipart/form-data"的属性，而且最好使用 POST 方法提交表单。

下面是接收表单实现文件上传的 Servlet。

```
//-----文件名: FileUpload.java-----
package servlets;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Iterator;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
```

```

import org.apache.commons.fileupload.servlet.ServletFileUpload;
public class FileUpload extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        boolean isMultipart = ServletFileUpload.isMultipartContent(request);
        if (isMultipart) {
            FileItemFactory factory = new DiskFileItemFactory();
            ServletFileUpload upload = new ServletFileUpload(factory);
            Iterator items;
            try {
                items = upload.parseRequest(request).iterator();
                while (items.hasNext()) {
                    FileItem item = (FileItem) items.next();
                    if (!item.isFormField()) {
                        //取出上传文件的文件名称
                        String name = item.getName();
                        String fileName = name.substring(name.lastIndexOf("\\")+1,name.length());
                        String path = request.getRealPath("file")+File.separatorChar+fileName;
                        //上传文件
                        File uploadedFile = new File(path);
                        item.write(uploadedFile);

                        //打印上传成功信息
                        response.setContentType("text/html");
                        response.setCharacterEncoding("gb2312");
                        PrintWriter out = response.getWriter();
                        out.print("<font size='2'>上传的文件为: "+name+"<br>");
                        out.print("保存的地址为: "+path+"</font>");
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

下面详细解释这个程序中的关键代码。

```
boolean isMultipart = ServletFileUpload.isMultipartContent(request);
```

上面这行代码可以判断出提交过来的表单是否为文件上传表单，如果不是文件上传表单，在后续的处理中就不再用文件上传功能来处理这个表单。

```
FileItemFactory factory = new DiskFileItemFactory();
ServletFileUpload upload = new ServletFileUpload(factory);
```

上面这两行代码构造了一个文件上传处理对象。

```
items = upload.parseRequest(request).iterator();
```

上面这行代码解析出表单中提交的所有文件内容。

```
String name = item.getName();
String fileName = name.substring(name.lastIndexOf("\\")+1,name.length());
String path = request.getRealPath("file")+File.separatorChar+fileName;
```

上面这段代码可以取出文件名，并且得出这个文件上传到服务器以后存储的路径。上传的文件将被

存储再当前应用项目的 file 文件夹中。

```
File uploadedFile = new File(path);
item.write(uploadedFile);
```

上面这两行代码把上传的文件存储到服务器中，完成上传操作。

这个文件上传 Servlet 的配置信息如下。

```
<servlet>
    <servlet-name>FileUpload</servlet-name>
    <servlet-class>servlets.FileUpload</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>FileUpload</servlet-name>
    <url-pattern>/FileUpload</url-pattern>
</servlet-mapping>
```

配置工作完成以后就可以浏览器中进行文件上传操作，这个示例程序的运行效果如图 5.11 和 5.12 所示。

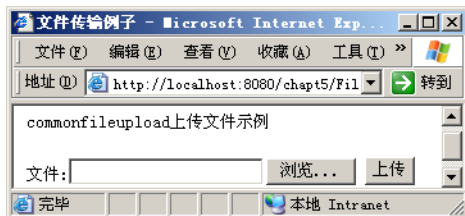


图 5.11 文件选择界面

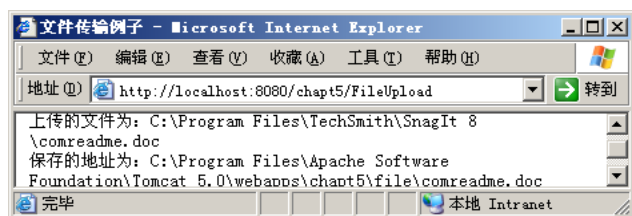


图 5.12 文件上传成功界面

在这个示例程序中，接收上传文件的 Servlet 可以分析出表单中所有的文件内容，所以如果要上传多个文件的，只需要表单中添加文件选择输入框即可，其中 name 属性可以任意命名。

5.3.4 Servlet 下载文件

用 Servlet 下载文件的时候，并不需要第三方组件的帮助，只需要对服务器的响应对象 response 进行简单的设置即可，下面的就是一个文件下载的示例程序，这个程序将从当前应用项目的根目录下载一个名为 test.xls 的 Excel 文档，具体代码如下。

```
//-----文件名: FileDownload.java-----
package servlets;

import java.io.FileInputStream;
import java.io.OutputStream;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FileDownload extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        try {
            String fname = "test.xls";
            response.setCharacterEncoding("UTF-8");
            fname = java.net.URLEncoder.encode(fname, "UTF-8");
```

```

        response.setHeader("Content-Disposition", "attachment; filename="+fname);
        response.setContentType("application/msexcel");// 定义输出类型
    } catch (Exception e) {
        System.out.println(e);
    }
}
}

```

下面来解释这个 Servlet 中的关键代码。

```

response.setCharacterEncoding("UTF-8");
fname = java.net.URLEncoder.encode(fname, "UTF-8");

```

上面这两行代码是对字符编码的设置，用来支持中文的文件名。

```

response.setHeader("Content-Disposition", "attachment; filename="+fname);

```

上面这行代码指明了这个 Servlet 的功能是输出文件，并且指明文件的位置。

```

response.setContentType("application/msexcel");// 定义输出类型

```

上面这行代码指明了要输入文件的类型，其中 application/msexcel 就是 Excel 文件的 MIME 类型描述。

这个 Servlet 的配置信息如下。

```

<servlet>
    <servlet-name>FileDownload</servlet-name>
    <servlet-class>servlets.FileDownload</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>FileDownload</servlet-name>
    <url-pattern>/FileDownload</url-pattern>
</servlet-mapping>

```

上面这个 Servlet 可以用类似这样的地址进行访问 <http://localhost:8080/chapt5/FileDownload>，其中 <http://localhost:8080/> 是本地计算机的访问路径，chapt5 是当前应用项目的名称，FileDownload 是文件下载 Servlet，这个示例程序的运行效果如图 5.13 所示。

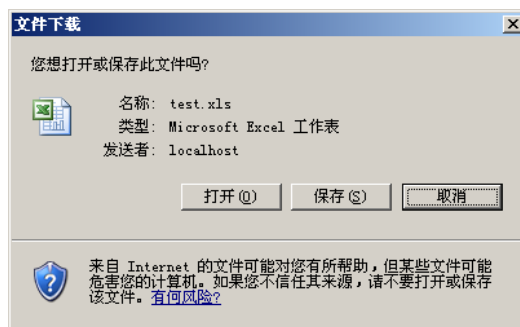


图 5.13 文件下载对话框

5.4 Servlet 过滤器

在 Web 应用中可以使用过滤器对所有的访问和请求进行统一的处理，IP 访问限制，用户发送请求的字符编码转换等，在进行具体的业务逻辑处理之前，首先要经过过滤器的统一处理，然后才开始进入真正的逻辑处理阶段。在本节内容中，将介绍过滤器的原理的实际应用。

5.4.1 过滤器的基本原理

过滤器的功能就是在服务器和客户中间增加了一个中间层，可以对两者之间的交互进行统一的处理，每一个从客户端提交的请求都需要通过过滤器的处理，然后再进行其他的操作。

在实际开发中，过滤器器可以用来对用户进行统一的身份判断、IP 访问限制，用户发送请求的字符编码转换、对请求和响应进行加密和解密、记录用户登录日志等。当然过滤器的用途不仅仅这些，读者可以根据过滤器的实现原理，思考过滤器更多的用途。

5.4.2 IP 访问 filter

在实际的应用中，可能会遇到这样的情况，需要对某些 IP 进行访问限制，不让非法的 IP 访问应用系统，这个时候就需要用到过滤器进行限制，当一个用户发出访问请求的时候，首先通过过滤器进行判断，如果用户的 IP 地址被限制，就禁止访问，只有合法的 IP 才可以继续访问。

这个 IP 访问限制过滤器的具体代码如下。

```
//-----文件名: IPFilter.java-----
package filters;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.UnavailableException;

public class IPFilter implements Filter {
    protected FilterConfig filterConfig;
    protected String ip;

    public void init(FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
        this.ip = this.filterConfig.getInitParameter("ip");
    }

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        String remoteIP = request.getRemoteAddr();
        if(remoteIP.equals(ip))
        {
            response.setCharacterEncoding("gb2312");
            PrintWriter out = response.getWriter();
            out.println("<b>你的 IP 地址被禁止访问。</b>");
        }else
        {
            chain.doFilter(request,response);
        }
    }
}
```

```

    }
}
public void destroy() {}
}

```

在上面这个过滤器中，主要的方法就是 `init()` 和 `doFilter()` 这两个方法，`init()` 方法是这个过滤器初始化的时候调用的，在这个过滤器中初始化的工作就是从配置文件中读取参数的内容，而 `doFilter()` 方法的功能就是这个过滤器真正要执行的处理功能，在这个示例程序中，对 IP 的限制就是在这个方法中实现的。

```
String remoteIP = request.getRemoteAddr();
```

这行代码可以取出访问系统的用户的 IP 地址。

```

if(remoteIP.equals(ip))
{
    response.setCharacterEncoding("gb2312");
    PrintWriter out = response.getWriter();
    out.println("<b>你的 IP 地址被禁止访问。</b>");
}

```

上面这段代码的功能是，如果用户的 IP 地址和配置文件中限制的 IP 相同，就拒绝用户的访问，并在页面上打印错误信息。

```
chain.doFilter(request,response);
```

上面这行代码的意思是，如果 IP 合法，就允许访问，用户可以进入应用系统进行其他的操作。

过滤器和 Servlet 很类似，都需要在 `web.xml` 文件中进行配置，但是过滤器的配置明显要比 Servlet 的复杂很多，下面是这个 IP 限制过滤器的配置信息，这段配置内容需要添加在 `web.xml` 文件的 `<web-app></web-app>` 标签之间。

```

<filter>
  <filter-name>IPFilter</filter-name>
  <filter-class>filters.IPFilter</filter-class>
  <init-param>
    <param-name>ip</param-name>
    <param-value>127.0.0.1</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>IPFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

```

在上面这个配置文件中，`<filter>` 节点是对过滤器自身的属性进行配置，`<filter-mapping>` 节点是对过滤器访问路径的配置，其中 `<filter-name>` 是过滤器的名称，这个属性可以自己命名，但是要和 `<filter-mapping>` 中的 `<filter-name>` 保持一致，`<filter-class>` 指的是过滤器所对应的 Java 类文件，这里一定要注意要把包名和类名写全。`<init-param>` 定义了这个过滤器在初始化的时候加载的参数，`<param-name>` 是初始化参数的名称，`<param-value>` 是初始化参数的值，在一个过滤器中可以没有参数，也可以有一个到多个参数。`<url-pattern>` 指明了这个过滤器对那些路径的访问有效，在这里“`/*`”的意思对于所有请求，都必需经过这个过滤器的处理。

同时，在一个应用系统中，可以没有过滤器，也可以定义一个或者多个过滤器。

总之，开发一个过滤器需要通过下面三个基本步骤。

(1) 编写过滤器的功能代码，即实现功能的代码类。

(2) 把编译成功的过滤器功能代码类文件拷贝到当前应用项目的 WEB-INF/classes 目录下。

(3) 在当前应用项目的 web.xml 文件中对过滤器进行配置，即在 web.xml 中添加配置信息。

经过这样三个步骤这个过滤器就可以发挥作用。

上面这个 IP 访问限制的示例程序中，设置禁止访问的 IP 地址为 127.0.0.1，这个 IP 地址就是本机 IP，所以在上面这个过滤器起作用以后，再访问上面任何一个页面或者是 Servlet，都会得到如图 5.14 所示的界面。



图 5.14 IP 访问过滤器运行效果

注意：在这个 IP 访问限制的过滤器中，配置文件中<init-param>的值可以设置成任意要限制的 IP，此处仅仅是为了展示方便，才设置为本机默认的 IP。

5.4.3 转换字符编码 filter

在 Java 语言中，默认的编码方式是 ISO-8859-1，这种编码格式不支持中文的显示，我们可以用类似 `<%@ page contentType="text/html; charset=gb2312"%>` 这样的方式来规定页面字符编码格式，但是如果显示的内容是表单提交、或者是经过 Servlet 处理，这时候字符内容本身的编码格式就是 ISO-8859-1，所以尽管页面指定的字符编码方案为 gb2312，在这种情况下中文内容仍然不能正常显示。在第四章中已经对中文处理的问题做了详细的介绍，所以在本章仅仅对其中使用过滤器解决中文乱码问题进行详细的分析。

这个转换中文字符编码过滤器的代码如下。

```
//-----文件名: SetCharacterEncodingFilter.java-----
package filters;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.UnavailableException;

public class SetCharacterEncodingFilter implements Filter {

    protected FilterConfig filterConfig;
    protected String encodingName;
    protected boolean enable;

    public SetCharacterEncodingFilter() {
        this.encodingName = "gb2312";
        this.enable = false;
    }
}
```

```

    }
    public void init(FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
        loadConfigParams();
    }
    private void loadConfigParams() {
        this.encodingName = this.filterConfig.getInitParameter("encoding");
        String strIgnoreFlag = this.filterConfig.getInitParameter("enable");
        if (strIgnoreFlag.equalsIgnoreCase("true")) {
            this.enable = true;
        } else {
            this.enable = false;
        }
    }
}

public void doFilter(ServletRequest request, ServletResponse response,
                    FilterChain chain) throws IOException, ServletException {
    if(this.enable) {
        request.setCharacterEncoding(this.encodingName);
    }
    chain.doFilter(request, response);
}

public void destroy() {
}
}

```

在上面这个过滤器中，init（）方法从配置文件中取出字符编码格式的参数，在 doFilter（）方法中使用 request 对象对所有的请求统一编码格式。

这个字符编码设置过滤器的配置信息如下。

```

<filter>
    <filter-name>SetCharacterEncodingFilter</filter-name>
    <filter-class>filters.SetCharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>gb2312</param-value>
    </init-param>
    <init-param>
        <param-name>enable</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>SetCharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

在上面这个过滤器中，初始化的时候提供了两个参数，参数 encoding 指明了编码格式为 gb2312，参数 enable 指明是否使用这个过滤器处理字符编码，在这里设置为 true，即启动这个字符编码过滤器，需要注意的是，这里的 enable 参数并不是所有过滤器必需的，只是在这个字符处理的程序中自己设定的

一个参数，没有通用的含义。<url-pattern>/*</url-pattern>指明了对所有的请求都使用字符编码过滤器进行转码处理。这样就避免了对每个请求都进行字符编码设置，从而大大减少了工作量。

经过这个字符编码格式转换过滤器的处理，所有来自客户的请求数据都被转换成 gb2312 的格式，这样就可以解决中文编码格式不同带来的乱码问题，这里需要注意，在 Tomcat 中，对 URL 和 GET 方法提交的表单是按照 ISO-8859-1 的格式进行编码的，过滤器对这种情况并不起作用，这时候就需要修改 Tomcat 的配置文件来解决，这在第四章中的中文乱码解决方案中已经提供详细的解决方法。

5.5 Servlet 应用示例

在上面的内容中已经提到，Servlet 是与 HTTP 协议紧密结合的，使用 Servlet 几乎可以处理 HTTP 协议各个方面的内容，在本节的几个示例程序中，将集中展示 Servlet 在 HTTP 方面的具体应用。

5.5.1 获取请求信息头部内容

当客户访问一个页面的时候，会提交一个 HTTP 请求给服务器的 Servlet 引擎，在这个请求中有 HTTP 的文件头信息，其中包含这个请求的详细属性信息，在下面这个示例 Servlet 中将取出 HTTP 头部内容，并在页面打印，这个 Servlet 的具体代码如下。

```
//-----文件名: RequestHeader.java-----
package servlets;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class RequestHeader extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Enumeration e = request.getHeaderNames();
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = request.getHeader(name);
            out.println(name + " = " + value + "<br>");
        }
    }
}
```

在上面这个 Servlet 中，首先使用 request.getHeaderNames()取出所有的 HTTP 头信息的名称，然后循环使用 request.getHeader(name)取出对应的头信息的值，并且在页面打印。

这个 Servlet 的配置信息如下。

```
<servlet>
    <servlet-name>RequestHeader</servlet-name>
    <servlet-class>servlets.RequestHeader</servlet-class>
```

```

</servlet>
<servlet-mapping>
    <servlet-name>RequestHeader</servlet-name>
    <url-pattern>/RequestHeader</url-pattern>
</servlet-mapping>

```

这个 Servlet 的配置信息和上面其他的 Servlet 的配置基本一样，在配置工作完成以后就可以用类似 `http://localhost:8080/chapt5/ RequestHeader` 这样的地址进行访问，其中 <http://localhost:8080/> 是本地计算机的访问路径，chapt5 是当前应用项目的名称，RequestHeader 是取 HTTP 头信息的 Servlet，上面这个示例 Servlet 的运行效果如图 5.15 所示。

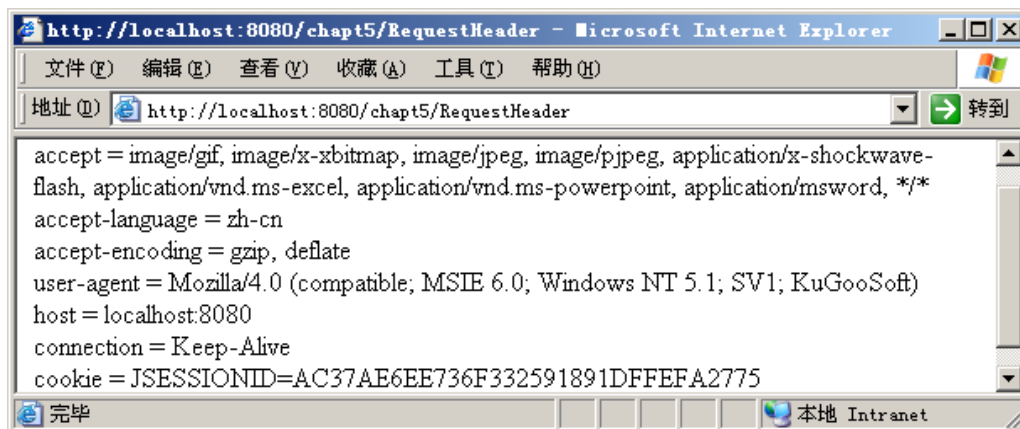


图 5.15 获取 HTTP 头部信息示例程序运行效果

如图 5.15 所示，等号左边的是 HTTP 头信息项的名称，等号右边是这项的值。而且具体 HTTP 头文件信息的内容不尽相同，在不同的浏览器中会有所不同。

5.5.2 获取请求信息

在上面这个 Servlet 示例中，我们取出所有的 HTTP 文件头信息，在 Servlet 中还可以很方便取出客户发出请求对象自身的信息。这些信息是和客户的请求密切相关的，例如客户提交请求所使用的协议，客户提交表单的方法是 POST 还是 GET 等，在下面这个示例程序中介绍集中常见属性的取值方法。这个示例程序的具体代码如下。

```

//-----文件名: RequestInfo.java-----
package servlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class RequestInfo extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
    }
}

```



```

        out.println("<head>");
        out.println("<title>请求信息示例</title>");
        out.println("</head>");
        out.println("<body><font size='2'>");
        out.println("<b>请求信息示例</b><br>");
        out.println("Method: " + request.getMethod()+"<br>");
        out.println("Request URI: " + request.getRequestURI()+"<br>");
        out.println("Protocol: " + request.getProtocol()+"<br>");
        out.println("Remote Address: " + request.getRemoteAddr()+"<br>");
        out.println("</font></body>");
        out.println("</html>");
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        doGet(request, response);
    }
}

```

下面解释上面这个 Servlet 的关键代码。

```

response.setContentType("text/html");
response.setCharacterEncoding("gb2312");

```

上面这两行代码这时服务器响应的文件格式和字符编码格式。其中 `gb2312` 的编码格式可以用来支持中文字符的显示。

```

out.println("Method: " + request.getMethod()+"<br>");

```

上面这行代码取出表单提交的方法，可以是 POST 或者是 GET。

```

out.println("Request URI: " + request.getRequestURI()+"<br>");

```

上面这行代码取出这个客户请求的 URI，即相对的访问路径，在本例中的值为 `/chapt5/RequestInfo`。

```

out.println("Protocol: " + request.getProtocol()+"<br>");

```

上面这行代码取出客户发出请求的时候使用的协议，在本例中的值为 `HTTP/1.1`。

```

out.println("Remote Address: " + request.getRemoteAddr()+"<br>");

```

上面这行代码取出客户的 IP 地址。

上面这个 Servlet 示例程序的配置信息如下。

```

<servlet>
    <servlet-name>RequestInfo</servlet-name>
    <servlet-class>servlets.RequestInfo</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>RequestInfo</servlet-name>
    <url-pattern>/RequestInfo</url-pattern>
</servlet-mapping>

```

在配置工作完成以后就可以用类似 `http://localhost:8080/chapt5/ RequestInfo` 这样的地址进行访问，其中 <http://localhost:8080/> 是本机计算机的访问路径，`chapt5` 是当前应用项目的名称，`RequestInfo` 是获取请求信息的 Servlet，上面这个示例 Servlet 的运行效果如图 5.16 所示。

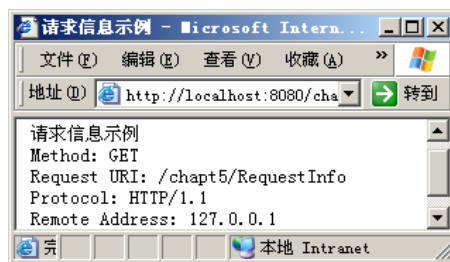


图 5.16 获取请求信息示例运行效果

在上面这个示例程序中，仅仅展示了有限的几个请求信息的获取方法，限于篇幅，其他请求信息的获取方法不在展示，它们的使用方法都是基本类似的，读者可以尝试自己进行思考、摸索。

5.5.3 获取参数信息

在 Servlet 中，同样可以很方便的取出用户请求中的参数信息，这种参数包括以 POST 方法或者是 GET 方法提交的表单，也包括直接使用超链接传递的参数，Servlet 都可以取出这些信息并且加以处理，在下面的例子中将具体展示 Servlet 获取各种参数的方法。

首先来看这样一个表单，这个表单就是在 5.3.1 中那个简单的用户信息表单，在这个例子中我们会分别使用 POST 方法和 GET 方法提交用户的请求信息。

这个表单的具体内容如下。

```
//-----文件名: paramForm.jsp-----
<%@ page language="java" import="java.util.*" contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <title>Servlet 接收表单示例</title>
  </head>
  <body>
    <font size="2">
      <form action="RequestParam" method="post">
        姓名: <input type="text" name="name"/><br>
        省份: <input type="text" name="province"><br>
        <input type="submit" value="提交">
      </form>
    </font>
  </body>
</html>
```

接收这个表单请求的 Servlet 的具体代码如下。

```
//-----文件名: RequestParam.java-----
package servlets;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class RequestParam extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```

throws IOException, ServletException
{
    response.setContentType("text/html");
    response.setCharacterEncoding("gb2312");
    Enumeration e = request.getParameterNames();
    PrintWriter out = response.getWriter ();
    out.print("<font size='2'>");
    out.print("下面是用 GET 方法传递过来的参数:<br>");

    while (e.hasMoreElements()) {
        String name = (String)e.nextElement();
        String value = request.getParameter(name);
        out.println(name + " = " + value+"<br>");
    }
    out.print("</font>");

}

public void doPost(HttpServletRequest request, HttpServletResponse res)
throws IOException, ServletException
{
    res.setContentType("text/html");
    res.setCharacterEncoding("gb2312");
    Enumeration e = request.getParameterNames();
    PrintWriter out = res.getWriter ();
    out.print("<font size='2'>");
    out.print("下面是用 POST 方法传递过来的参数:<br>");
    while (e.hasMoreElements()) {
        String name = (String)e.nextElement();
        String value = request.getParameter(name);
        out.println(name + " = " + value+"<br>");
    }
    out.print("</font>");
}
}

```

在这个 Servlet 中可以看到，我们分别处理 doGet 和 doPost 方法，而不是像前面的示例程序中只实现 doGet 方法，然后在 doPost 中简单调用，在这里因为要处理不同方法提交的参数，需要有不同的处理方法，所以在这里分别实现 doGet 和 doPost 方法。

这个 Servlet 的配置信息如下。

```

<servlet>
    <servlet-name>RequestParam</servlet-name>
    <servlet-class>servlets.RequestParam</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>RequestParam</servlet-name>
    <url-pattern>/RequestParam</url-pattern>
</servlet-mapping>

```

在配置工作完成以后就可以用类似 <http://localhost:8080/chapt5/paramForm.jsp> 这样的地址进行访问，其中 <http://localhost:8080/> 是本地计算机的访问路径，chapt5 是当前应用项目的名称，paramForm.jsp 是填

写表单的页面，这个 Servlet 的运行效果如图 5.17 所示。



图 5.17 Servlet 接收 POST 方法提交的参数效果

当把上面这个表单的提交方法改为 GET 的时候，这个 Servlet 的运行效果如图 5.18 所示。

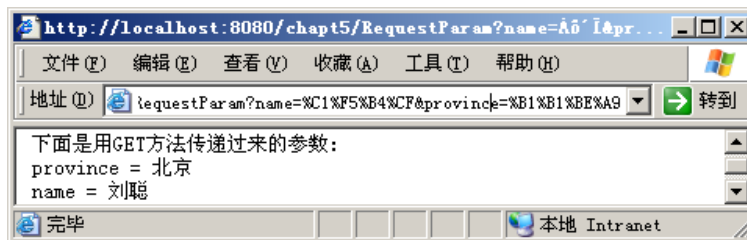


图 5.18 Servlet 接收 GET 方法提交参数的效果

在上面这两个效果图中可以看出，浏览器地址栏中的内容是不同的，用 POST 方法提交的时候参数的内容不在地址栏中显示，而使用 GET 方法提交的时候，所有的参数都在地址栏中显示，在这里因为 Tomcat 对 URL 的字符编码格式为 ISO-8859-1，所以在地址栏中的中文字符不能正常显示，不过我们已经按照第四章中处理中文乱码的方式修改了 Tomcat 的配置，所以在页面上还是可以正常显示出中文参数的内容。

超链接传递参数的处理方法和处理 GET 方法提交表单的参数手段是一样的，因为单击超链接以后，参数的内容也是在地址栏中显示的，其道理是和 GET 方法提交表单中的参数是相同的，所以在这里就不在赘述。

5.5.4 Cookies 操作

Cookies 是指在 Web 应用中，为了辨别用户身份而存储在用户本地计算机上的数据。Servlet API 提供了 Cookie 操作类，封装了操作 Cookie 常用的方法，在下面的例子中，将展示 Servlet 操作 Cookie 的方法。这个 Servlet 的具体代码如下。

```
//-----文件名: Cookies.java-----
package servlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Cookies extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
    }
}
```

```

        PrintWriter out = response.getWriter();

        //设置一个 Cookie
        Cookie cookie = new Cookie("name", "Gates");
        response.addCookie(cookie);

        //打印 Cookie 的内容
        Cookie[] cookies = request.getCookies();
        out.print("<font size='2'>");
        out.print("下面是 Cookie 的内容:<br>");
        for (int i = 0; i < cookies.length; i++) {
            Cookie c = cookies[i];
            String name = c.getName();
            String value = c.getValue();
            out.println(name + " = " + value+"<br>");
        }
        out.print("</font>");
    }
}

```

下面介绍 Servlet 中操作 Cookie 的方法。

```
Cookie cookie = new Cookie("name", "Gates");
```

上面这行代码创建了一个 Cookie 对象，这个 Cookie 的名字为 name，存储的信息为 Gates。

```
response.addCookie(cookie);
```

上面这行代码把这个 Cookie 写入到用户本地的计算机中。

```
Cookie[] cookies = request.getCookies();
```

上面这行代码取出用户机器中的 Cookie，用于接下来的打印操作。

```
Cookie c = cookies[i];
```

```
String name = c.getName();
```

```
String value = c.getValue();
```

上面这段代码可以取出一个 Cookie 的名称和值。

这个示例程序的配置信息如下。

```

<servlet>
    <servlet-name>Cookies</servlet-name>
    <servlet-class>servlets.Cookies</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Cookies</servlet-name>
    <url-pattern>/Cookies</url-pattern>
</servlet-mapping>

```

在配置工作完成以后就可以用类似 [http://localhost:8080/chapt5/ Cookies](http://localhost:8080/chapt5/Cookies) 这样的地址进行访问，其中 <http://localhost:8080/> 是本地计算机的访问路径，chapt5 是当前应用项目的名称，Cookies 是操作 Cookie 的 Servlet，这个示例程序的运行效果如图 5.19 所示。



图 5.19 Cookie 操作示例程序运行效果

注意：在运行 Cookie 操作这个示例程序的时候，如果程序不能正常运行，需要检查浏览器是否禁用 Cookie。

5.5.5 Session 操作

在 JSP 中有内置的 Session 对象，可以用来保持服务器与用户之间的会话状态，在 Servlet 中间，同样可以对 Session 进行方便的操作，在现面的例子中，将详细介绍 Servlet 中处理 Session 的具体方法。这个 Servlet 的具体代码如下。

```
//-----文件名: RequestInfo.java-----
package Sessions;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Sessions extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession(true);
        // 打印当前 Session 的具体属性信息
        out.println("<font size='2'>");
        Date created = new Date(session.getCreationTime());
        Date accessed = new Date(session.getLastAccessedTime());
        out.println("Session 的 ID 为: " + session.getId()+"<br>");
        out.println("Session 创建的时间为: " + created+"<br>");
        out.println("Session 上次访问时间为: " + accessed+"<br>");
        // 在这里可以设置一个 Session
        session.setAttribute("msg", "Hello");
        //打印 Session 的具体内容
        Enumeration e = session.getAttributeNames();
        out.println("Session 的内容如下: ");
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = session.getAttribute(name).toString();
```

```

        out.println(name + " = " + value+"<br>");
    }
    out.println("</font>");
}
}

```

接下来详细介绍这个 Servlet 的关键代码。

```
HttpSession session = request.getSession(true);
```

上面这行代码从 request 对象中取出当前的 session 对象。

```
Date created = new Date(session.getCreationTime());
```

上面这行代码取出 session 的创建时间。

```
Date accessed = new Date(session.getLastAccessedTime());
```

上面这行代码取出 session 上次被访问的时间。

```
out.println("Session 的 ID 为: " + session.getId()+"<br>");
```

上面这行代码取出 session 的 id, 并且在页面上打印。

```
session.setAttribute("msg", "Hello");
```

上面这行代码在 session 中设置了一个名为 msg 的属性变量, 这个变量的值为 Hello。

```
Enumeration e = session.getAttributeNames();
```

上面这句话从 session 中取出所有的属性, 并放在一个集合中间。

```

while (e.hasMoreElements()) {
    String name = (String)e.nextElement();
    String value = session.getAttribute(name).toString();
    out.println(name + " = " + value+"<br>");
}

```

上面这段程序中, 循环取出 session 中的内容, 并在页面打印。

上面这个 Servlet 的配置信息如下。

```

<servlet>
    <servlet-name>Sessions</servlet-name>
    <servlet-class>servlets.Sessions</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Sessions</servlet-name>
    <url-pattern>/Sessions</url-pattern>
</servlet-mapping>

```

在配置工作完成以后就可以用类似 <http://localhost:8080/chapt5/Sessions> 这样的地址进行访问, 其中 <http://localhost:8080/> 是本地计算机的访问路径, chapt5 是当前应用项目的名称, Sessions 是操作 session 的 Servlet, 上面这个示例 Servlet 的运行效果如图 5.20 所示。



图 5.20 Session 操作示例运行效果

在上面这个示例程序中, 只要不关闭浏览器, 在整个用户与服务器的交互期间, 这个 session 对象

是始终存在的，这一点可以从 session 的 id 和创建时间看出。一旦关闭浏览器，这个 session 对象就会被销毁，再次访问的时候服务器重新生成一个 session 用来维护与用户之间的状态。

5.6 小结

在本章的内容中，详细讲解了 Servlet 的工作原理，并且通过实际的示例程序详细介绍了 Servlet 的调用方法，对 Servlet 常见的文件操作也做了比较详细的介绍，Servlet 是和 HTTP 协议密切联系的，所以在本章最后的部分对 Servlet 的 HTTP 操作方法做了细致的讲解。

通过本章内容的讲解，读者已经可以对 Servlet 有一个总体上的把握，Servlet 在本质上就是 Java 类，在了解这 Servlet 的基本原理和基本使用方法以后，如果要想在 Servlet 领域有更大的提高，还是需要回头巩固 Java 的基础，这才是学习 Servlet 的最根本的途径。