第4章 Servlet

4.1 Web 应用概述

4.1.1 Java Web 技术

Web 是一种典型的分布式应用架构 :Web 应用中的每一次信息交换都要涉及到客户端和服务器端两个层面。因此 Web 开发技术大体上也可以被分为客户端和服务器端两个大类。

Java Web,是用 Java 技术来解决相关 web 互联网领域的技术总和。web 包括:web 服务器和 web 客户端两部分。Java 在客户端的应用有 java applet 不过现在使用的很少,Java 在服务器端的应用非常的丰富,比如 Servlet,JSP 和第三方框架等等。Java 技术对 Web 领域的发展注入了强大的动力。

Web 服务器端主要是响应客户的请求 :最早的 web 服务器仅是简单的响应浏览器发来的 http 请求,并将存储在服务器上的 html 文件返回给客户端浏览器。这种静态的服务功能非常有限,随着 web 技术的发展,逐渐出现了各种可以动态响应客户端请求的技术,从最早的 CGI 技术到目前的 PHP,ASP,JSP/Servlet 等。通过这些动态的 web 服务器端技术人们就可以享受到信息检索、信息交换、信息处理等更为便捷的动态信息服务了。

Servlet 是 SUN 公司针对 Web 服务器端开发的技术,它利用 Java 平台的优越性,自一开始发布就受到开发者的青睐。利用 Servlet,Jsp 组合技术不但可以同时拥有类似 CGI 程序的集中处理功能和类似 PHP 的 HTML 嵌入功能,还可以享用 Java 语言一次编译处处运行所带来的各种优势。

4.1.2 Http 通信协议

HTTP 协议(Hypertext Transfer Protocol,超文本传输协议),是用于从 WWW 服务器传输超文本到本地浏览器的传送协议;是浏览器与 WEB 服务器之间的一问一答的交互过程遵循的规则。该协议是一个无状态的基于请求和响应的协议。它是 TCP/IP 协议集中的一个应用层协议,用于定义浏览器与 WEB 服务器之间交换数据的过程以及数据本身的格式,大家平常

通过浏览器访问 Internet 的某一个网页的过程就是借助 HTTP 协议来完成的。

4.2 Servlet 简介

4.2.1 Servlet 的概念及其作用

Servlet 就是普通的运行在服务器端的 Java 程序,主要用来拓展基于请求/响应模式的服务端的功能。尽管 Servlet 可以响应任何类型的协议,但是它们通常用于拓展基于 Web 的应用程序。在这种应用程序中,主要是响应 HTTP 协议,所以针对这种情况 Java Servlet 技术定义了特定的 HTTP Servlet 类。它的特点和优点如下:

✧ Servlet API 提供了为 Web 应用程序定制的接口;

✧ Servlet 是独立于服务器和平台的,这使得 Servlet 可移植并且可重用;

✧ Servlet 是高效的和可扩展的;

✧ Servlet 运行在服务器内,所以可以为某种目的委托服务器执行特定的功能,比如用户身份验证。

一个 Servlet 是实现 javax. servlet. Servlet 接口的类的一个实例,然而大多数 Servlet 扩展了这个接口的一个实现类:javax. servlet. GenericServlet 或 javax. servlet. http. HttpServlet。Servlet 的接口声明了管理 Servlet 以及客户端通信的方法。作为一个 Servlet 开发人员,我们可以重载部分或是全部这些方法来开发我们自己的 Servlet。

GenericServlet 只有很有限的功能,所以我们只来探讨更为有用的 HttpServlet 类。它是一个抽象类,可以从它这里来继承,以创建一个适合我们某一 Web 站点的 Http Servlet。若要实现这个目的,一个 HttpServlet 必须能够访问特定于 HTTP 的调用的库。

一个 Servlet 被装载后,它通常会驻留在服务器的内存中。绝大多数情况下,只有单个 Servlet 实例会被创建,而且为了支持并发的页面访问,这个 Servlet 是运行于多线程的。这就避免了针对每次相同页面访问而创建新的 Servlet 处理的开销,也就节省了内存,也使得页面访问变得十分高效。

4.2.2 Servlet 基本工作原理

当客户端发送请求到服务器端后,web 服务器会根据请求路径信息,调度相应的 Servlet 对象来处理,但是如果是多个并发的用户发送相同的请求,web 服务器会采用线程的机制来给客户端分配资源,然后让其并发的来访问某一个特定的 Servlet 对象。Servlet 的具体工作流程如下:

(1)客户端(例如 Web 浏览器)通过 HTTP 发送请求(请求)。

(2)Web 服务器接收该请求并将其发给相应的 Servlet。如果这个 Servlet 尚未被加载,Web 服务器将把它加载到 Java 虚拟机并且执行它。

(3)Servlet 将接收该 HTTP 请求并执行相应的业务处理。

(4)Servlet 处理完毕后,向 Web 服务器返回应答。

(5)Web 服务器将从 Servlet 收到的应答发送给客户端(响应)。

(6)至此一个请求过程处理完毕。

图 4.1 显示了 Servlet 的生命周期。

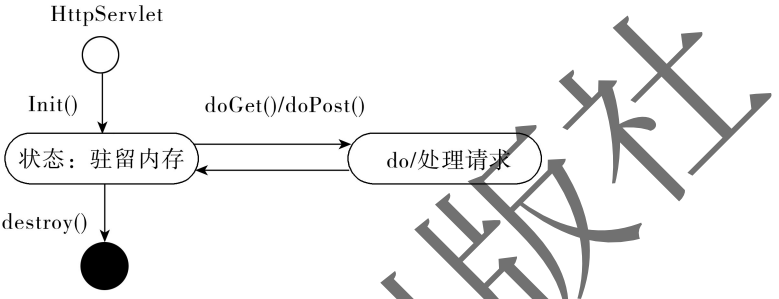

图 4.1　Servlet 生命周期

当实例化 Servlet 时,服务器调用 servlet 的 init()方法,这个方法能且只能被调用一次,来初始化 servlet 资源和 servlet 实例变量。Servlet 可以在 init()方法中访问它的上下文。init()方法必须在 servlet 可以服务于任何请求前完成,若 init()方法抛出一个 ServletException 异常,相应的 servlet 将无法进行服务。该方法是在超类中被定义的,也就是只有我们需要对 servlet 进行初始化时,才需要重载 init()方法。init()方法声明如下:

```
public void init(ServletConfig config)throws ServletException
public void init()throws ServletException
```

当初始化工作完毕后,Servlet 对象就可以响应并处理用户请求了,在 Servlet 的生命周期中,大部分的时间是用来处理请求的,当一个请求到来时,Web 服务器将会调用 Servlet 对象的 service 方法,service 方法声明如下:

```
public void service(ServletRequest request,ServletResponse response)throws ServletException
```

它的参数 request,response 都是由 Servlet 容器创建并传递给 service()方法使用的;在 HttpServlet 中,service()方法将会区分不同的 HTTP 请求类型,调用相应的 doXXX()方法进行处理,比如请求的是 HTTP GET 方法,将会调用 doGet(),而 POST 则会调用 doPost()。所以当我们实现一个针对 http 协议的 Servlet 时,我们只需要覆盖相应的 doGet()或 doPost()方法,实现我们的业务处理逻辑即可。

当从服务器中删除 servlet 时,服务器会先调用 servlet 的 destroy()方法。由于服务器可以在任何时候销毁一个 servlet,所以该方法不仅可以用来释放资源,比如:如果我们之前在 init()方法中打开的数据库连接或者 socket 连接,在 destroy()方法中都需要关闭;并且可以用来保存数据和状态等信息,这些数据和状态信息在 servlet 再次调用的时候可能会被用到。在 Servlet 的生命周期中,由于 Servlet 只会被卸载一次,所以 destroy 方法也只会被执行一次,则与 init()方法是一样的。方法声明如下:

```
public void destroy() throws ServletException
```

4.2.3 Servlet 运行环境

Servlet 是一组 java 类,这些类可以被支持 java 的 web 服务器动态装载并运行。向 servlet 提供扩展支持的 web 服务器被称为“容器”(也叫做 servlet 引擎)。Web 客户端(通常都是 web 浏览器来扮演的角色)使用 HTTP 请求/响应协议与服务器进行通信、交互。Servlet 容器通常都提供下列服务和功能:

✧ 用来发送请求和响应的网络服务;

✧ 为 servlet 注册一个或多个 URL;

✧ 管理 servlet 的生命周期;

✧ 解码基于 MIME 的请求;

✧ 构造基于 MIME 的响应;

✧ 支持 HTTP 协议(也可以支持其他协议)。

4.2.4 Servlet 包结构

我们要编写和开发的 HTTP servlet 是需要扩展、继承 javax. servlet. HttpServlet 类的,而这个类其实又扩展了 javax. servlet. GenericServlet。而 GenericServlet 实际上是对javax. servlet. Servlet接口、javax. servlet. ServletConfig 接口、及序列化接口 java. io. Serializable 的实现。

HttpServlet 类层次图如 4.2 所示。

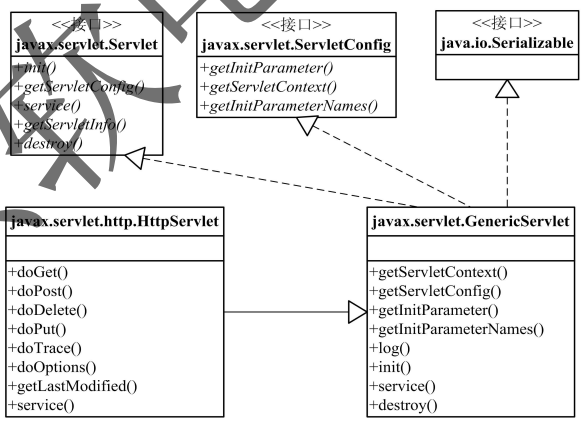

图 4.2 HttpServlet 类层次结构

其中的各个方法,都非常有用,建议掌握。

4.3　Web 应用程序结构

4.3.1　文件和目录结构

我们知道，web 应用程序是必须要运行在 web 应用服务器之上的。一个 web 应用程序它一般都是包括了如 HTML 静态页面、CSS 式样表文件、javascript 脚本文件、图片、servlet、后台的支持类、javabean 以及其他所有的被绑定在一起的一个完整的应用程序资源。那么这些种类各异的文件应该以怎样的规则组织在一起呢？下面我们来讨论一个简单的 web 应用的文件及目录结构。

Java servlet 规范定义了目录的结构层次，这些目录被用于部署和打包所有这些必须的文件。每一个 web 应用程序都有一个根目录，我们称之为环境路径。在相同的 web 服务器上，任何两个应用程序的环境路径都不能相同。

在项目的根目录下有个一个非常特殊的目录就是 WEB-INF，在该目录下包含以下内容：

✧ web. xml 文件：web 应用程序的核心部署描述文件；

✧ /classes 目录：专门用来存储 servlet 类，实用程序类的编译后的类文件（如后台支持类、javabean 等类）；

✧ /lib 目录：依赖的第三方 jar 资源文件包，如数据库驱动程序文件、struts 应用开发包、spring 应用开发包等等。

有时候可能还会有一个 META-INF 目录，它包含特定于实现的部署描述文件。如基于 tomcat 应用服务器创建的数据库连接池、数据源配置文件，就要放置在这里。如图 4.3 所示为 web 应用目录结构示例。

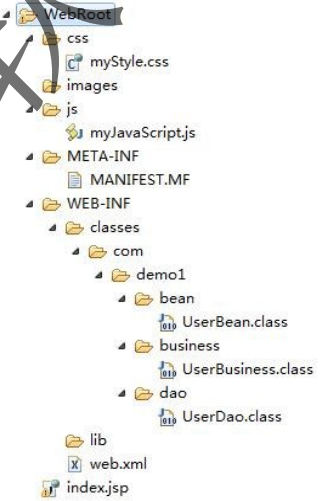

图 4.3　web 应用目录结构示例

4.3.2 部署描述文件结构

Web.xml 文件是一个 web 应用程序的核心,它是整个 web 应用资源、安全等相关信息的部署描述。该文件以<? xml>标记开头,定义 XML 的版本和字符集编码。接着是该文件的根标记<web-app>,引用用于验证文档结构的 XML Schema,所有对资源的描述都应该放在<web-app></ web-app >之间。web 应用部署描述文件【4-1】web.xml

```
<? xml version="1.0" encoding="UTF-8"? >
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<servlet>
    <servlet-name>user_Login</servlet-name>
    <servlet-class>com.demo1.servlet.UserLoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>user_Login</servlet-name>
    <url-pattern>/login</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<error-page>
    <error-code>404</error-code>
    <location>/errorPages/error404.html</location>
</error-page>
</web-app>
```

部署描述文件中常用的关键标签的功能作用描述如下:

1. <servlet>该元素用来在 Web 应用中定义一个 Servlet。

(1)<servlet-name>以<servlet>元素的子元素出现,表明此 servlet 对象在容器创建 servlet 对象时所对应的名字,此名称要求在一个 web 应用是唯一的。

(2)<servlet-class>以<servlet>元素的子元素出现,表示 servlet 所在的完全类路径 web 服务器将根据此路径创建 servlet 对象。

(3)<load-on-startup>以<servlet>元素的子元素出现,该元素告诉容器此 servlet 在整个 web 应用中装载的次序。当一个应用有多个 servlet 时,容器在加载这些 servlet 对象时的次序时随机的。如果我们在 servletA 初始化中需要用到 servletB 中初始化的一些参数,那么我们就要在容器装载 A 之前先装载 B。数值越小越先被装载。所以我们只需将 servletB 所对应的值小于 servletA 的即可。但是如果是一个负数或是没有配置该元素,那么加载的顺序和时间都是随机的。

(4)<init-param>以<servlet>元素的子元素出现,该元素用于定义 Servlet 中需要的初

始化参数如下：

```
<init-param>
    <param-name>driver</param-name>
    <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
```

<param-name>以<init-param>元素的子元素出现，表示参数的名字；

<param-value>以<init-param>元素的子元素出现，表示参数的值。

获取上面配置的参数代码如下：

```
public void init() throws ServletException{
  ServletConfig config =getServletConfig();
  String driver=config.getInitParameter("driver");
}
```

(5)<context-param>该元素定义的初始化参数，将被存储到 ServletContext 对象中，此对象被同一个 web 应用下的所有的资源所共享，所以都可以访问到此中的参数。它与<init-param>元素的区别在于，通过<init-param>元素定义的参数只能在当前 Servlet 中获取，而其他的 Servlet 和资源却无法获取。

```
<context-param>
    <param-name>user_name</param-name>
    <param-value>eddyyang</param-value>
</context-param>
```

<param-name>以<context-param>元素的子元素出现，表示参数的名字；

<param-value>以<context-param>元素的子元素出现，表示参数的值。

可以使用 ServletContext 对象的如下两个方法取得定义的参数：

```
public String getInitParameter(String name);
public java.util.Enumeration getInitParameterNames();
```

第一个方法返回给定参数名所对应的值；第二个方法取得<context-param>元素中配置的所有初始化参数的名字集合。若希望获取到参数名为“user_name”所对应的参数值，我们可以使用方法之一：

```
Stringv_name=getServletContext().getInitParameter("user_name");
```

2.<servlet-mapping>该元素提供了能够使用某一 Servlet 处理请求的映射地址，即，当容器创建一个 servlet 对象后，它还要知道该 servlet 用来处理什么样的请求，也就是说当 web 服务器接收到用户请求后，需要知道将该请求交给哪个 servlet 来处理。

(1)<servlet-name>以<servlet-mapping>元素的子元素出现，用于定义接收并处理用户请求的 servlet 对象名字，它需要和<servlet>元素中的某个<servlet-name>对应。

(2)<url-pattern>表示该 servlet 所要处理的 url 匹配路径。

3.<error-page>该元素可以通过定义特定的错误页面，当服务器产生错误时会选择相应

的页面展示给用户。

```
<error-page>
    <error-code>404</error-code>
    <location>/404.html</location>
</error-page>
<error-page>
    <exception-type>java.lang.NullPointerException</exception-type>
    <location>/exception.html</location>
</error-page>
```

4. <welcome-file-list>该元素用于定义当用户访问、提交的是一个目录而不是一个显式的文件时的一个文件列表。一般默认的欢迎页面文件路径都会配置在这里。

【课堂实训 4-1】通过创建一个 web 项目【ch4_1】,我们先通过一个 servlet 请求路径向其发出请求及它的响应情况。

1. 通过使用 MyEclipse 或 eclipse-jee-indigo-SR1-win32 版本的 Eclipse 创建一个 web project,步骤如下:单击菜单栏 File→New→Web Project→在 project name 输入域中输入你的合法自定义项目名称→Finish,如图 4.4 Create a Web Project 窗口所示。

2. 集成开发工具帮助我们创建了该项及默认项目结构,如图 4.5 web 项目【ch4_1】默认项目结构所示。

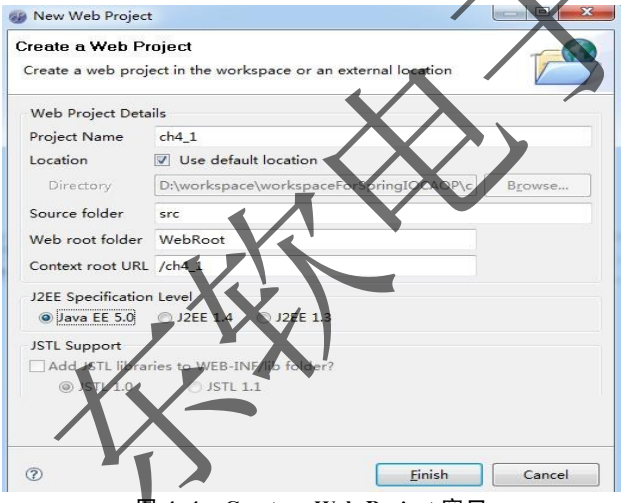

图 4.4 Create a Web Project 窗口

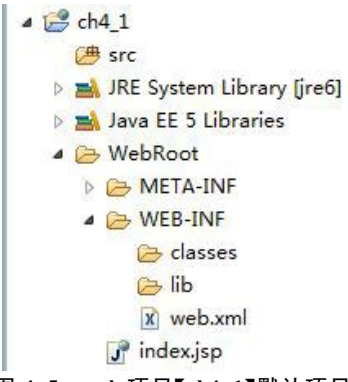

图 4.5 web 项目【ch4_1】默认项目结构

这里,我们所有该项目所需编写的 java 源代码,请在 src 目录下事先创建的各个包中进行编写,web 资源是需要在 WebRoot 下进行编写、创建的。当我们编写完毕后的 java 原文件,经过保存、自动编译成为的 class 文件,会被我们所使用的 MyEclipse 或 eclipse-jee-indigo-SR1-win32版本的 Eclipse 自动放置在 WebRoot 下的 WEB-INF 中的 classes 目录下。下面会逐步给大家演示和展示。

3. 创建我们所需要的 HttpServlet 文件、及其他类文件,并在 web.xml 文件中,对该 servlet 进行配置、部署、描述。如图 4.6【ch4_1】完整的项目文件结构所示。

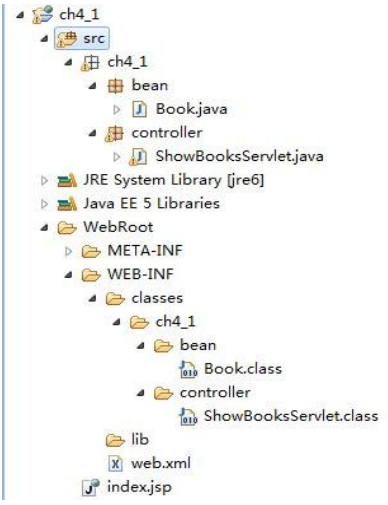

图 4.6　【ch4_1】完整的项目文件结构

首先我们完成 javabean【4-2】Book.java 的编写：

```
package ch4_1.bean;
public class Book {
    private String id;
    private String cateid;
    private String name;
    private String publisher;
    private float price;
    private String desc;
    public Book(){}
    public Book(String id,String cateid,String name,String pub,float price,String desc){
        this.id=id;
        this.cateid=cateid;
        this.name=name;
        this.publisher=pub;
        this.price=price;
        this.desc=desc;
    }
    public String getCateid() {
        return cateid;
    }
    public void setCateid(String cateid) {
        this.cateid=cateid;
    }
    public String getDesc() {
        return desc;
    }
    ......
```

4. 编写我们将要向之发出请求的 Servlet 类【4-3】ShowBooksServlet.java。

```
package ch4_1.controller;
......
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import ch4_1.bean.Book;
public class ShowBooksServlet extends HttpServlet{
    @Override
    protected void doGet(HttpServletRequest requset, HttpServletResponse response)
            throws ServletException, IOException {
        //创建若干个 book 实例
        Book book1=new Book("1001","Computer Software","Java 与模式","**电子出版社",
                88.88f,"很好的一本书...");
        Book book2=new Book("1002","Computer Software","Java 编程思想","**工业出版社",
                188.88f,"很好的一本书...");
        Book book3=new Book("1003","Computer Software","Jsp 与 Servlet","**大学出版社",
                288.88f,"很好的一本书...");
        //将若干 book 对象放入集合容器中
        List<Book> bookList=new ArrayList<Book>();
        bookList.add(book1);
        bookList.add(book2);
        bookList.add(book3);
        //当有请求时,遍历该集合并在控制台上打印输出
        Iterator<Book> it=bookList.iterator();
        while(it.hasNext()){
            Book b=it.next();
            System.out.println("书名:"+b.getName()+"编号:"+b.getId()+"价格:"+b.
        getPrice());
        }
    }
    @Override
    protected void doPost(HttpServletRequest requset, HttpServletResponse response)
            throws ServletException, IOException {
        this.doGet(requset, response);
    }
}
```

当我们通过一个请求路径发出请求时,该 servlet 将会在控制台上打印输出集合中的信息。

5. 编写本项目的部署描述文件【4-4】web.xml。这里,我们通过标签<url-pattern>定义的

该 servlet 的请求路径是/showMe。

```
<? xml version="1.0" encoding="UTF-8"? >
<web-app version="2.5"
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
      <servlet-name>Show_Books</servlet-name>
      <servlet-class>ch4_1.controller.ShowBooksServlet</servlet-class>
  </servlet>
  <servlet-mapping>
      <servlet-name>Show_Books</servlet-name>
      <url-pattern>/showMe</url-pattern>
  </servlet-mapping>
</web-app>
```

6. 以上工作都做完毕之后，接下来就是将该项目发布到指定的 tomcat 服务器（本章节中我们默认使用 tomcat6.0 版本）的规定路径下了。一般，我们可以通过手动方式自行发布，也可以通过一些集成开发工具自动发布。若是手动方式发布，我们需要将我们开发完毕的 web 项目的的 WebRoot 下的所有资源，放入一个指定的、或是自定义名称的文件夹内（也可以打包），然后将其放入进 tomcat 服务器的 webapps 目录下即可。

7. 将项目部署到服务器的指定位置后，启动服务器。步骤是，进入 tomcat6.0 的 bin 目录，双击 startup.bat。若我们的 java 环境配置正确无误的话，即可看到如下启动成功界面，如图 4.7所示。

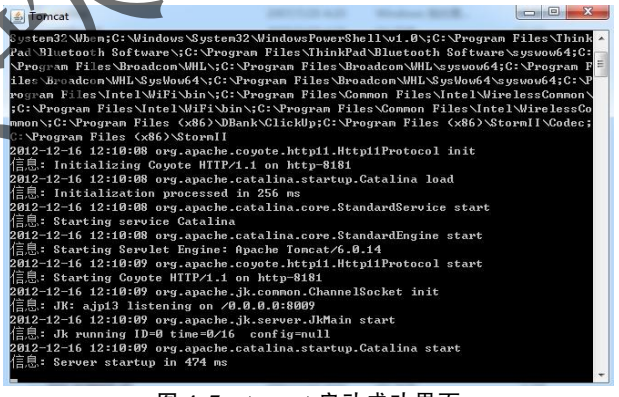

图 4.7　tomcat 启动成功界面

双击 shutdown.bat，即可安全的关闭 tomcat 服务器。

8. 当服务器启动成功后，我们即可通过浏览器，在地址栏处输入请求协议 http://；服务器所在地址，若服务器在本地机器上，我们可以输入 127.0.0.1 或 localhost；后面跟上端口号，默

认是 8080,若该 http 请求端口没有被修改,我们也可以不必输入 8080 端口;然后需准确无误的写明我们要请求的项目名称,这里即为 ch4_1,最后加上某个我们将要请求的 servlet 的路径,点击回车即可,如:

```
http://localhost:8181/ch4_1/showMe
```

9. 当指定的地址下的服务器接收到我们的特定的请求路径后,该路径所对应的唯一的 servlet 将会被运行,我们这个练习项目,并没有要求该 servlet 向我们的浏览器进行请求内容的响应,仅仅是将我们希望的运行结果在控制台上打印输出而已。后续会继续讲解和演示 servlet 的响应方式和技术。如图 4.8 所示为【ch4_1】请求运行结果。

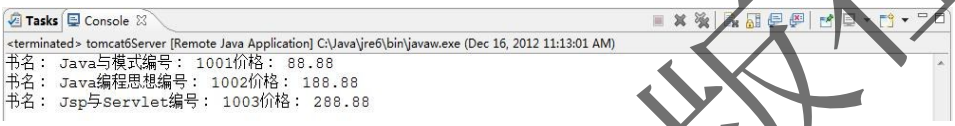

图 4.8 【ch4_1】请求运行结果

4.4 Servlet 应用模型及解析

4.4.1 Servlet 如何接收 http 请求

对于请求而言客户端主要通过以下几种方式给服务器端发送请求:

✧ 用户点击 HTML 页面上的一个超链接;

✧ 通过点击一个表单的提交按钮提交表单;

✧ 用户在 URL 地址栏直接键入服务器端的 URL 地址。

当用户发送请求到 web 服务器后,web 服务器确定如果是一个 Servlet 请求的话,则将该请求交给 Servlet 容器处理,Servlet 容器是 Web 服务器的一部分,是运行在 Web 服务器中的组件。Servlet 容器将根据请求路径来确定具体要交给哪个 Servlet 处理。

ServletRequest 是定义在 javax.servlet 包中的一个接口,ServletRequest 类与具体的协议无关。从客户端发送的请求信息将会被封装在 ServletRequest 对象当中。通常我们可以通过下面的方法获取封装的数据:

✧ 获取单个参数的值,或是多个参数当中的某一个,如果对应参数不存在则返回 null:

```
public String getParameter(String name)
```

✧ 获取一个参数对应的多个值:

```
public String[] getParameterValues(String name)
```

✧ 返回所有参数的名字:

```
public java.util.Enumeration getParameterNames()
```

HttpServletRequest 是 ServletRequst 的扩展子类，它添加了一些专用于 HTTP 协议的方法，HttpServletRequest 对象由 Servlet 容器为我们创建，并以参数的形式传递给 doGet( )和 doPost( )方法供 Servlet 使用。常见的方法如下：

✧ public String getMethod( )；

通过这个方法可以获取 HTTP 请求类型的名字，例如 GET 或 POST 等。

✧ public String getHeader(String name)；

此方法返回指定标题名字所对应的值，例如：

```
String  host=request.getHeader("host");
```

✧ public Enumeration getHeaderNames( )；

获取所有头信息中标题名，具体对应的值可以通过上面的方法获取。

✧ public Enumeration getHeaders(String name)；

获取指定标题名的所有对应值。

✧ Public String getProtocol( )；

用于获取请求所使用的协议的名称及版本等信息。

【课程实训 4-2】实践通过如上方法如何获取表单的请求参数的值及客户端浏览器等的请求信息。

1. 我们对项目【ch4_1】进行修改，为本项目新增一个 html 文件【4-5】userRegister. html，用来用户注册个人信息。

```
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head><title>新用户注册</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  </head>
  <body  style="font-size:14px;color:#006699;">
    <center>
    <fieldset style="border:1.5px solid #0033FF;margin-bottom:10px;width:60%">
    <legend><font size="3" color="red">用户注册</font></legend>
    <form action="/ch4_1/user_reg" method="post">
      <table>
        <tr>
        <td>登录 ID:</td>
        <td><input type="text" name="userid"/></td>
        </tr>
        <tr>
        <td>登录密码:</td>
        <td><input type="password" name="userpwd"/></td>
        </tr>
        <tr>
        <td>用户性别:</td>
        <td><input type="radio" name="sex" value="male"/ checked="checked">男
            <input type="radio" name="sex" value="female"/>女</td>
```

```
        </tr>
        <tr>
        <td>学历:</td>
        <td><select name="education">
            <option value="doctor"/>博士
            <option value="master"/>硕士
            <option value="bachelor"/>学士
            <option value="other"/>其他
            </select></td>
        </tr>
<tr><td>爱好:</td>
        <td>
            <input type="checkbox" name="hobby" value="shopping"/>购物
            <input type="checkbox" name="hobby" value="sporting"/>运动
            <input type="checkbox" name="hobby" value="sleepping"/>睡觉
            <input type="checkbox" name="hobby" value="coding"/>编程</td>
    </tr>
    <tr>
        <td>个人简介:</td>
            <td><textarea rows="10" cols="25">
                                Write something about yourself...</textarea>
            </td>
            </tr>
            <tr><td colspan="2" align="center">
                <input type="submit" value="提交"/>
                <input type="reset" value="重置"/></td>
            </tr>
        </table>
        </form>
    </fieldset>
    </center>
  </body>
</html>
```

用户注册页面运行效果如图 4.9 所示。

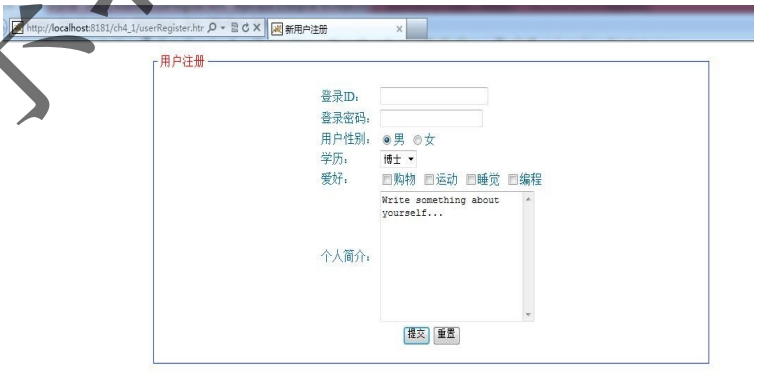

图 4.9　用户注册页面

2. 在 ch4_1. controller 包下创建一个 Servlet 文件【4-6】GetRequestInforServlet. java。

```
package ch4_1.controller;
import java.io.IOException;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class GetRequestInforServlet extends HttpServlet{
    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)throws ServletException,
            IOException {
        String u_id=request.getParameter("userid");
        String u_pwd=request.getParameter("userpwd");
        String u_sex=request.getParameter("sex");
        String u_edu=request.getParameter("education");
        String u_intro=request.getParameter("introduce");
        String[] u_hobby=request.getParameterValues("hobby");
        String u_hobby_str="";
        for(int i=0;i<u_hobby.length;i++){
            u_hobby_str += u_hobby[i]+";";
        }
        System.out.println("客户端提交的用户信息如下：登录 id 是："+u_id+"\r"+
            " 登录密码是："+u_pwd+"\r"+" 用户性别是："+u_sex+"\r"+
            " 用户学历是："+u_edu+"\r"+" 用户简介是："+u_intro+"\r"+
            " 用户兴趣爱好有："+u_hobby_str);
        //获得客户端提交请求的方法名称
        String method=request.getMethod();
        System.out.println("客户端发出请求的方法是："+method);
        //获得客户端使用的请求协议类型
        String protocol=request.getProtocol();
        System.out.println("客户端发出请求使用的协议是："+protocol);
        /*获取全部客户端请求头部信息内容
         *先获取请求头部结构的名称
         */
        Enumeration header_name=request.getHeaderNames();
        //遍历存储了请求头部结构名称的枚举器,得到每个名称所对应的值
        while(header_name.hasMoreElements()){
            String name=(String)header_name.nextElement();
            String head_value=request.getHeader(name);
            System.out.println("请求头部结构的名称是："+name+"  值是："+head_value);
        }
    }
    ......
```

3. 在项目【ch4_1】的原有配置文件 web. xml 中增加以下内容,即将 Servlet【4-6】配置进去。

```
<servlet>
    <servlet-name>get_information</servlet-name>
    <servlet-class>ch4_1.controller.GetRequestInforServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>get_information</servlet-name>
    <url-pattern>/user_reg</url-pattern>
</servlet-mapping>
```

这里为该 Servlet 配置的请求路径为 /user_reg,那么我们也需确保在用户注册页面【4.5】的表单中,属性 action 的请求路径也需要是 /user_reg,进而才能保证当我们客户端浏览器页面上点击“提交”按钮时,将我们的请求交给该路径所对应的 Servlet 来进行处理。

4. 重新发布项目【ch4_1】至 tomcat 服务器,重新启动服务器,在浏览器地址栏中键入如下请求路径。

```
http://localhost:8181/ch4_1/userRegister.html
```

将出现图 4.9 用户注册页面,我们在表单中输入、或是点选相应的选项,最后点击“提交”,后台服务器端将接收到我们的请求,并且服务器会将我们的客户端请求参数自动的封装在 HttpServletRequest 对象中,并以参数的形式传给我们的 Servlet 的 doGet 或 doPost 方法,我们在后台即可利用请求对象的相应的很多方法获取到我们想要的请求参数值。

5. 控制台打印输出我们的请求处理结果,如图 4.10 所示为通过表单提交请求处理结果。

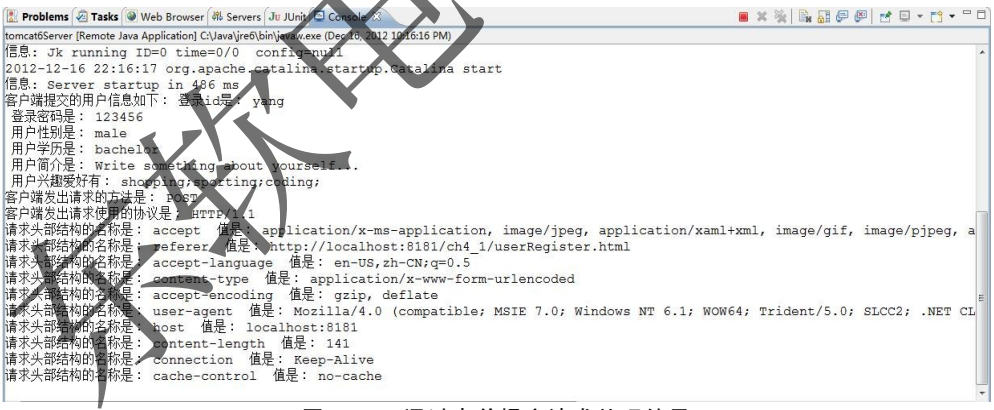

图 4.10 通过表单提交请求处理结果

4.4.2 Servlet 如何响应 http 请求

Servlet 主要通过 HttpServletResponse 对象封装对用户的响应信息,再由 Web 服务器发送给客户端。HttpServletResponse 是 ServletResponse 的一个子接口,定义在 javax. servlet. http 包中,它添加了一些针对 HTTP 协议响应的处理方法。其实,无论我们返回给客户端的信息是一些简单的文本还是一些复杂的流式文件,都需要获取输出流对象,在 Servlet 中可以通过

javax.servlet.ServletResponse 的如下两种方法获取一个输出流：

✧ public ServletOutputStream getOutputStream() throws IOException;

✧ public PrintWriter getWriter() throws IOException;

第一种输出流主要是用来向客户端发送一些二进制数据流，如：要返回给客户端一个文件或是其他的应用程序。我们可以通过 ServletOutputStream 对象向客户端响应输出 PDF 文件内容：

```
    public  void  doGet(HttpServletRequest request,HttpServletResponse response)throws
ServletException,IOException{
        response.setContentType("application/pdf");
    FileInputStream pdf=new FileInputStream("c://mytest.pdf");
        ServletOutputStream out=response.getOutputStream();
        byte[] buffer=new byte[1024];
        int count;
        while((count=pdf.read(buffer))! =-1){
            out.write(buffer,0,count);
        }
        out.flush();
        out.close();
        pdf.close();
    }
```

第二种方法则主要是返回一些字符文本信息，如需要返回浏览器现实的 HTML 页面。

另外，当 Servlet 接收到请求后，如果需要返回一定的数据信息给客户端，需要使用 ServletResponse 对象可以获得用于：

✧ 发送给客户端数据的输出流（上面已经阐述过）；

✧ 设定返回内容的类型和长度；

✧ 以及设定返回文本的字符编码等。

```
    public void doGet(HttpServletResquest request,HttpServletResponse response)
    throws ServletException ,IOException{
    response.setContentType("text/html;charset=GBK");
        PrintWriter  out= response.getWriter();
        out.println("<html>");
        out.println("<head><title>");
        out.println("测试页面");
        out.println("</title></head>");
        out.println("<body><center><b>);
        out.println("我的第一个输出页面");
            out.println("</center></b>");
        out.println("</body></html>");
        out.close();
    }
```

当客户端访问服务器端有固定后缀的资源时，服务器一般能根据其后缀来区分需要返回的

信息内容的类型。

比如我们访问 Web 服务器的 index.html,服务器会根据后缀知道是一个 html 文件,当客户请求的是 mypic.gif 时,可以被识别为一个图片信息。但是,当访问的是一个 servlet 时,因为没有固定的后缀,客户端浏览器则不知道它可能会返回什么类型的信息,所以在 Servlet 返回给客户端信息时一定要设置返回的类型。在 ServletResponse 中我们可以通过 setContentType()方法和 setContentLength()方法设定返回内容的类型和长度。

比如:我们设定内容为"text/html;charset=gb2312",可以使用 response.setContentType("text/html;charset=gb2312");或者要响应、返回一个 pdf 文件,我们可以设置 response.setContentType("application/pdf");表明响应、返回的文件类型是 PDF 格式的。

【课堂实训 4-3】通过客户端发出请求,让 web 服务器向客户端浏览器响应超文本信息内容。

让我们对项目【ch4_1】进行如下修改:

1.修改 Servlet 类【4-6】GetRequestInforServlet.java,首先在该类的 doGet 方法里增加代码:

```
response.setContentType("text/html; charset=UTF-8");
PrintWriter out=response.getWriter();
```

2.然后利用 PrintWriter 类型的对象 out,将文本和超文本向客户端输出,继续修改类【4-6】,部分代码如下:

```
response.setContentType("text/html; charset=UTF-8");
PrintWriter out=response.getWriter();
out.println("<html>");
out.println("<head><title>你请求信息及你的注册信息响应页面</title></head>");
out.println("<body>");
out.println("<center>");
out.println("你的注册信息内容如下列表,请核对:");
out.println("<table border=1>");
out.println("<tr><td bgcolor=gray>你要注册的登录 id 是:</td>");
out.println("<td>"+u_id+"</td></tr>");
out.println("<tr><td bgcolor=gray>你要注册的登录密码是:</td>");
out.println("<td>"+u_pwd+"</td></tr>");
......
out.println("<td>"+u_intro+"</td></tr>");
out.println("<tr><td bgcolor=gray>你要注册的兴趣爱好有:</td>");
out.println("<td>"+u_hobby_str+"</td></tr>");
out.println("<tr><td colspan=2 bgcolor=#99FFFF align=center>
```

```
以下是客户端请求的其他请求头信息：
</td></tr>");
Enumeration header_name=request.getHeaderNames();
while(header_name.hasMoreElements()){
    String name=(String)header_name.nextElement();
    String head_value=request.getHeader(name);
    out.println("<tr><td bgcolor=gray>"+name+"</td>");
    out.println("<td>"+head_value+"</td></tr>");
}
    out.println("</table>");
    out.println("</center>");
    out.println("</body>");
    out.println("</html>");
```

3. 重新发布项目【ch4_1】到指定 web 服务器后，启动服务器，在浏览器客户端同样输入地址。

```
http://localhost:8181/ch4_1/userRegister.html
```

在个输入域输入需要的值，点击提交。

4. 运行结果经过处理后，响应给客户端浏览器的结果，如图 4.11 所示为服务器响应页面。

你的注册信息内容如下列表，请核对：

你要注册的登录id是：	yang
你要注册的登录密码是：	123
你要注册的用户性别是：	male
你要注册的用户学历是：	doctor
你要注册的用户简介是：	Write something about yourself...
你要注册的兴趣爱好有：	shopping;sleepping;
以下是客户端请求部分信息：	
请求方法是：	POST
请求协议类型是：	HTTP/1.1
以下是客户端请求的其他请求头信息：	
accept	text/html, application/xhtml+xml, */*
referer	http://localhost:8181/ch4_1/userRegister.html
accept-language	en-US,zh-CN;q=0.5
user-agent	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
content-type	application/x-www-form-urlencoded
accept-encoding	gzip, deflate
host	localhost:8181
content-length	124
connection	Keep-Alive
cache-control	no-cache

图 4.11　服务器响应页面

4.4.3　Servlet 的生命周期

Servlet 对象是运行在 Servlet 容器中并由容器管理和调度的，那么容器在何时创建 Servlet 对象？何时调用 Servlet 对象中相应的方法呢？又是何时销毁 Servlet 对象的呢？所有这些问题都涉及到 Servlet 的生命周期问题。一个 Servlet 生命周期要经历 4 个阶段：

✧ 生成与装载；

✧ 初始化；

✧ 执行请求处理；

✧ 卸载。

当开发完毕具体的 Servlet 类，并且部署到 Web 服务器上以后，是由 Web 服务器为我们装载并创建 Servlet 对象的，一般来说，Servlet 可以在三种情况下装载到内存并被实例化：

✧ Web 服务器启动时；

✧ 系统管理员向 Web 服务器部署 Servlet 应用时；

✧ 通过浏览器第一次访问 Servlet 时。

Web 服务器装载一个 Servlet 对象时，需要明确知道 Servlet 的类名。比如一个名为 com.demo1.MyServlet 的类，当服务器装载其到内存并创建对象时将会使用 Class.forName("myweb.HelloServlet").newInstance()方法。采用这种方法创建对象需要类中有个不带参数的构建器。所以我们在开发 Servlet 时需要提供这样的构造器，或者干脆不写构造器，采用默认的构造即可。

一旦 Web 服务器创建完 servlet 对象以后，将会立即调用 Servlet 的 init()方法，对 servlet 进行初始化，所以就算我们在 Servlet 中使用了默认的构建器，我们对 Servlet 的初始化仍然可以放到 init()方法中。init 方法的具体声明如下：

```
public void init(ServletConfig config)throws ServletException
public void init()throws ServletException
```

作为参数的 javax.servlet.ServletConfig 接口类型对象封装了用于初始化的参数，它是在容器调用 init()方法时传递的参数。我们可以通过 ServletConfig 对象的以下方法获得初始化参数以及它们的值：

```
public java.util.Enumeration getInitParameterNames();
public String getInitParameter(String name);
```

这些参数是配置在 Web 应用程序的配置文件 web.xml 中<servlet>元素内的。对于这两个 init()方法，我们在应用时继承 GenericServlet 或 HttpServlet 只需要覆盖其中的一个 init()方法就可以。

【课堂实训 4-4】使用 ServletConfig 接口提供的方法，获取在配置文件中配置的参数所对应的值。【见实例：TestServletConfig.java】

1. 在项目【ch4_1】中创建一个 Servlet【4-7】GetInitParamValuesServlet.java。

2. 在本项目的部署描述文件【4-4】web.xml 内将【4-7】配置进来，并为它增加若干初始化参数。

```
<servlet>
    <servlet-name>get_init_values</servlet-name>
    <servlet-class>ch4_1.controller.GetInitParamValuesServlet
</servlet-class>
    <init-param>
        <param-name>driver</param-name>
        <param-value>oracle.jdbc.driver.OracleDriver</param-value>
    </init-param>
    <init-param>
        <param-name>user_name</param-name>
        <param-value>scott</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>get_init_values</servlet-name>
    <url-pattern>/getinit</url-pattern>
</servlet-mapping>
```

3. 接下来我们继续完善【4-7】GetInitParamValuesServlet. java，重写父类的 init (ServletConfig config)方法，用于获取部署描述文件中的初始化参数对应的值，并在重写的 doGet( )方法中对这些值进行打印输出，以检验我们是否获取到了这些值。

```
package ch4_1.controller;
import java.util.Enumeration;
import javax.servlet.ServletConfig;
.....
public class GetInitParamValuesServlet extends HttpServlet{
    private ServletConfig config;
    @Override
    public void init(ServletConfig config) throws ServletException {
        this.config=config;
    }
    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)throws ServletException, IOException {
        Enumeration e=config.getInitParameterNames();
        while(e.hasMoreElements()){
            String init_name=(String)e.nextElement();
            String init_value=config.getInitParameter(init_name);
                    System.out.println("获取到的初始化参数名："+init_name+"
该参数对应的值：  "+init_value);
        }
    }
    @Override
    protected void doPost(HttpServletRequest request,
            HttpServletResponse response)throws ServletException,
            IOException {
        this.doGet(request, response);
    }
}
```

4.将本项目重新发布到 web 服务器的指定路径上,启动服务器,打开浏览器并在地址栏处输入该 Servlet【4-7】的请求路径:

```
http://localhost:8181/ch4_1/getinit
```

5.获取初始化参数运行结果如图 4.12 所示。

图 4.12 获取初始化参数运行结果

当 Servlet 初始化完毕以后,Servlet 对象就可以响应并处理用户请求了,在 Servlet 的生命周期中,大部分的时间是用来处理请求的,当一个请求到来时,Web 服务器将会调用 Servlet 对象的 service 方法,service 方法声明如下:

```
public abstract void service(ServletRequest request, ServletResponse respose) throws ServletException, IOException;
```

它的参数 request, response 都是由 Servlet 容器创建并传递给 service()方法使用的。在 HttpServlet 中,service()方法将会区分不同的 HTTP 请求类型,调用相应的 doXXX()方法进行处理,比如请求的是 HTTP GET 方法,将会调用 doGet(),而 POST 则会调用 doPost()。所以当我们实现一个针对 http 协议的 Servlet 时,我们只需要覆盖相应的 doGet()、doPost()方法,实现我们的业务处理逻辑即可。

当 Servlet 容器决定卸载一个 Servlet 时,比如:由于 Web 服务器考虑到 Web 应用的性能问题或者管理员发送卸载请求,或者系统将要关闭,Web 服务器将会卸载容器中的 Servlet 对象,卸载之前将会调用 Servlet 的 destroy()方法,一般来说,Servlet 的 destroy()方法是用来释放 Servlet 所持有的资源。比如:如果我们之前在 init()方法中打开的数据库连接或者 socket 连接,都可以在 destroy()方法中进行关闭、释放。在 Servlet 的生命周期中,由于 Servlet 只会被卸载一次,所以 destroy 方法也只会被执行一次,则与 init()方法是一样的。

4.4.4 如何保留客户端状态及信息

Web 应用程序在绝大多数情况下都可能需要在不同页面请求之间保留客户的相关信息,除非是非常简单的 web 应用。为此称之为"会话",但是由于客户端是通过浏览器与服务器端交互的,采用的是 HTTP 协议,而 HTTP 协议是无状态的协议,那么,如何保存用户的会话信息呢?在不同的 web 组件之间又是如何共享数据对象信息的呢?

实际上在 Web 应用中可以通过相应的数据存储机制来管理这些数据信息。在 Web 应用中,不同的 Web 组件之间传递数据信息可以采用不同的对象类型,主要有以下几种:

✧ Cookies 对象,可以用来通过浏览器在客户端存储相关用户信息。

✧ HttpServletRequest 存储与传递对象信息。我们知道 HttpServletRequest 对象是用来封装客户端提交数据信息的，是由服务器创建的。而它的另一个功能则是可以在一个请求的处理过程中，在不同的 Web 组件之间传递对象信息。

✧ HttpSession 技术，一般称为 Servlet 会话技术，用来维持客户端用户的状态信息。

✧ ServletContext 对象可以用来存储整个 Web 应用的相关信息，对于全局共享的数据可以存放其中。

4.4.4.1　使用 Cookie

Cookie 是 Web 服务器发送到浏览器，并且存储在浏览器中的一些文本信息。当浏览器再一次发送一个请求到该 Web 服务器时，浏览器会将这些文本信息重新原封不动地返回给该服务器，通过让服务器读取它以前曾经发送到客户端的信息，服务器可以向客户端提供很多的方便。但是用户可能会在浏览器上禁用 Cookie 功能，这样就会使浏览器不能识别、接收 Cookie。从而使得保留或跟踪客户端的状态遇到麻烦。

Java Servlet API 包中包含一个类 javax. servlet. http. Cookie，通过此类可以创建一个 Cookie 对象，Cookie 的构造器为：

```
public Cookie (String name,String value);
```

通过 Cookie 类声明的以下方法可以获取 Cookie 的名字和值：

```
public String getName()
public String getValue()
```

当创建好 Cookie 对象以后，可利用 httpServletResponse 的 addCookie()方法，将 Cookie 插入到 http 响应头中，例如：

```
Cookie cookie=new Cookie("userName","tom");
response.addCookie(cookie);
```

当在客户端设置了 Cookie 以后，为了读取从客户端返回的 Cookie，可调用 HttpServletRequest 对象的 getCookies()方法，该方法返回一个 Cookie 对象的数组，如果该请求中不包含任何的 Cookie 信息，将返回 null。一旦获取这个数组，就可以对其进行迭代遍历，调用每个 Cookie 的 getNames()方法，获取所需要的 Cookie 对象，然后调用该 Cookie 的 getValue()方法获取所对应的值。例如下面代码所演示：

```
    Cookie[] cookies=request.getCookies();
if(cookies ! = null){
    Cookie cookie=null;
    String name=null;
    for(int i=0;i<cookies.length;i++){
        cookie=cookies[i];
        name=cookie.getName();
        if("userName".equals(name)){
            System.out.println("userName is :"+cookie.getValue());
        }
    }
}
```

当浏览器发送一个请求时,并不是发送所有的 Cookie。每一个 Cookie 都有各自的域名和路径,浏览器只会发送与服务器的域名和路径相匹配的 Cookie。例如:如果浏览器正在访问:http://java. yourserver. com/examples/loginServlet,同时浏览器刚好有一个域为".yourserver. com",路径为"examples" 的 Cookie,那么浏览器将该 Cookie 发送到服务器。但是如果浏览器有一个域名一致,而路径为"/otherapp",那么浏览器就会因为路径不匹配的原因,而不会发送该 Cookie. 如果浏览器有一个域为"other. yourserver. com"而路径为"/example",那么浏览器也会由于域不匹配而不发送该 Cookie。由于 Cookie 技术现在使用的不多了已经,这里就不再进行过多的阐述。

4.4.4.2 使用 HttpServletRequest

HttpServletRequest 是由 Web 服务器创建,并传递给 Servlet 使用的,它封装了客户端提交的数据。除了以上功能, HttpServletRequest 对象可以作为一个领域,在不同的 Servlet 或 JSP 间传递数据。作为一个存储领域,它所提供的方法有:

✧ public void setAttribute(String name,Object value)向 request 对象中存入一对键值对;

✧ public Object getAttribute(String name)根据传入的属性名,取出相应的值对象;

✧ public void removeAttribute(String name)根据传入的属性名,删除相应的值对象;

✧ public java. util. Enumeration getAttributeNames()用来获取存储在 request 对象中的所有属性名,进而通过得到的这些属性名,可以利用方法 public Object getAttribute(String name)得到相应的值对象。

需要注意的是在使用 request 对象存储数据时,不同的 web 组件之间必须接收的是同一个请求,否则不能共享存储的信息。那么也就需要我们确保不同的 web 组件之间的跳转关系是"转发"。

【课堂实训 4-5】通过一个 Servlet 向 HttpServletRequest 对象存储一些值,而在另一个 Servlet 或是 Jsp 中得到这些值的应用,展示 HttpServletRequest 对象是如何作为一个存储领域而发挥作用的。

1. 创建项目【ch4_2】,并创建完毕简单目录结构。在指定包下创建一个 Servlet【4-8】StoreValuesServlet. java。

```
package ch4_2.servlet;
import javax.servlet.RequestDispatcher;
......
public class StoreValuesServlet extends HttpServlet {
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String req_name=request.getParameter("name");
    String req_pwd=request.getParameter("pwd");
    request.setAttribute("key1", req_name);
    request.setAttribute("key2", req_pwd);
    RequestDispatcher rd=request.getRequestDispatcher("/getValues");
    rd.forward(request, response);
}
......
}
```

这里，我们若在浏览器客户端通过请求路径 URL 串接我们的请求参数名和值的话，该 Servlet 就可以得到参数名 name 和 pwd 所对应的值，并且我们利用 HttpServletRequest 对象的 setAttribute(String key,String value)方法将这两个参数值存入 request 对象。最后我们通过转发方式，将请求转发给了 URI 为/getValues 所对应的 Servlet。接下来我们将在这个 Servlet 中得到之前存入进 request 对象中的值，并进行打印输出、验证。

2. 创建另一个 Servlet【4-9】GetValuesServlet.java。

```
package ch4_2.servlet;
......
public class GetValuesServlet extends HttpServlet {
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String value1=(String)request.getAttribute("key1");
    String value2=(String)request.getAttribute("key2");
    System.out.println("用户名称:"+value1+"  用户密码:"+value2);
}
}
```

3. 将【4-8】和【4-9】配置、部署在部署描述文件中，并定义好各自的 URI。

```
<servlet>
    <servlet-name>Store_Values</servlet-name>
    <servlet-class>ch4_2.servlet.StoreValuesServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Store_Values</servlet-name>
    <url-pattern>/req</url-pattern>
</servlet-mapping>
<servlet>
    <servlet-name>Get_Values</servlet-name>
    <servlet-class>ch4_1.servlet.GetValuesServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Get_Values</servlet-name>
    <url-pattern>/getValues</url-pattern>
</servlet-mapping>
```

4. 将项目【ch4_2】发布到指定 web 服务器的指定目录下，启动服务器。在客户端浏览器地址栏处输入如下请求路径及参数：

```
http://localhost:8181/ch4_2/req? name=yang&pwd=123456
```

当发出请求后，请求交给/req 路径所对应的 Servlet【4-8】，该类获取到我们的请求参数，并将它们存入到 request 对象中，通过转发动作，转到/getValues 路径所对应的 Servlet【4-9】，在该类中我们从 request 中获取到之前存入的参数及值，并打印输出。

5. 如图 4.13 所示为获得存储在 request 中的值。

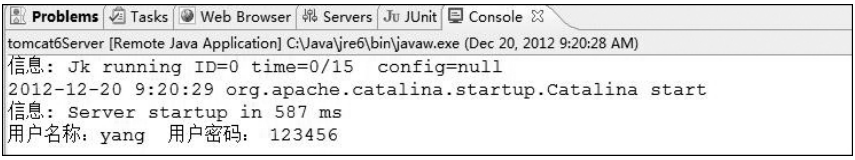

图 4.13 获得存储在 request 中的值

4.4.4.3 使用 HttpSession

对于 Web 应用程序来说,HTTP 会话至今最易于使用的和被推荐的方法。会话是用来标示一段指定的时间内源于同一个浏览器发来的页面请求,它可以很方便的被同一个客户端浏览器访问的应用程序中的所有 servlet 所共享。Javax.servlet.http.HttpSession 对象就是用来标识一个会话、并可以通过使用 HttpServletRequest.getSession( )方式得到。表 4.1 为 HttpSession 提供的方法。

表 4.1 HttpSession 提供的方法

方法名称	方法作用
getId( )	获得唯一的会话标示符 id 的值。
Invalidate( )	主动销毁服务器端的 session 对象,从而被垃圾回收机制回收。
getCreationTime( )	获得该会话对象的创建时间。
getMaxInactiveInterval( )	获得在客户端访问之间 servlet 容器保持的这个会话对象的最大时间间隔,以秒为单位。
getAttribute(key)	获得绑定在该 session 对象中指定的键所对应的值对象。
setAttribute(key,value)	向该 session 对象设置、绑定名值对。
getLastAccessdTime( )	获得客户发送与这个会话关联的请求的最晚时间。

【例如】获得一个 session 对象并为它绑定一对键值对。

```
HttpSession session=request.getSession();
session.setAttribute("attrName","attrValue");
```

我们已经了解 HTTP 协议是一个无状态的 web 通信协议,即它没有提供保持请求之间的数据机制,也就不能跟踪用户在不同请求之间的行为。当浏览器需要 web 服务器的一个网页时,它就重新打开一个连接,获取相应的网页,然后就关闭连接。该连接关闭后,web 服务器对浏览器的后续事件一无所知,因为没有主动的服务器到客户端的请求连接。

请思考购物车的原理:应用程序为了掌握该客户在结账的时候都买了什么,需要跟踪客户在各个请求中都添加了哪些商品。这意味着不仅要在每个请求中识别除相关的客户,而且还要存储多次请求中的购物数据,并且把它关联到相应的用户上,形成用户的累计的商品信息列表,最后进行结账。

要实现这样功能，一般都在服务器端创建一个会话(session)来关联客户端信息。一个会话就是一个 javax. Servlet. http. HttpSession 对象，对象都是存储在服务器端内存中的，也就是说不同的客户端通过在服务器端创建的 HttpSession 对象是不同的。

那么当客户端发送一个请求到服务器，比如向购物车中添加一个商品，我们当然不能把它添加到其他客户的购物车中，那么 web 服务器是如何区分特定的客户端与其会话之间的关联关系的呢？原来，对应每个 HttpSession 对象都有一个 id，服务器端根据此 id 与客户端关联，每个客户端在发送请求的时候都将此 id 发送给服务器端，服务器根据该 id 查找内存中匹配的 HttpSession 对象。即，session id 由服务器端生成，并在返回给客户端的信息中包含此 id 值。如图 4. 14 所示为 session id 工作原理示意。

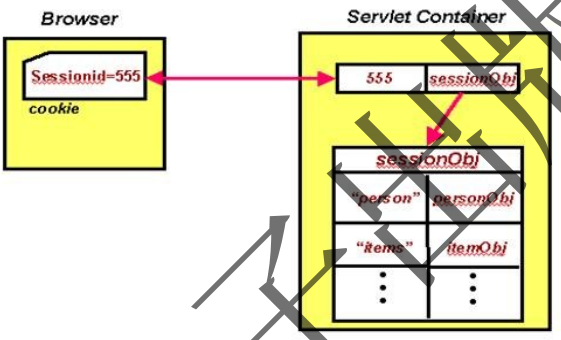

图 4. 14　session id 工作原理示意

服务器对客户端的 session id 是以 Cookie 形式写在客户端的，并且设置的 cookie 有效期为 -1，即：只在浏览器打开的时间段内有效。当浏览器关闭，它所对应的 session 对象将不再关联。对于没有与客户端关联的 session 对象，将很快被服务器端 java 虚拟机垃圾回收机制回收。

在之前关于 cookie 的描述中我们知道，客户端是可以禁止 cookie 的，那么如果浏览器禁用了 cookie，服务器端又如何与特定的客户端传递 session id 呢？

接下来我们根据上面提出的问题，给出一个比较好的解决办法，那就是应用 URL 重写技术。URL 重写技术思想是：即把唯一的 session id 附加在由服务器所发出、响应给客户端的每个相应的 URL 中，也就是说，在为第一个请求生成响应信息时，服务器在每个 URL 中附加了这个 session id，当客户端向这种 URL 之一提交一个请求时，由于此 URL 中附加了 session id，所以服务器可以区分每个请求所对应的 session 对象，从而得到了和存在 cookie 一样的功能。

我们需要使用 javax. servlet. http. HttpServletResponse 对象的如下两个方法来实现重写 URL：

✧ public String encodedURL(String url)；

✧ public String encodedRedirectURL(String url)；

encodedURL 方法主要是 servlet 生成 web 页面时,生成的连接地址,如:

```
String originalUrl="some url";
String encodedURL=response.encodedURL(originalUrl );
out.println("<a href=\" + encodedURL +\">link</a>");
```

如上代码将会在必要的时候添加 session id 到 encodedURL 上。如果需要重定向到本应用中的一个路径,则需要使用 encodedRedirectURL(String url)方法:

```
String originalUrl="some url";
String encodedURL=response.encodedRedirectURL(originalUrl );
response.sendRedirect(encodedURL );
```

由于我们不能确定不同的客户端 cookie 的禁用情况,所以好的编程方式是在每个生成的连接或是重定向时,使用以上方法对 URL 进行必要的重写。

【课堂实训 4-6】使用 HttpSession 技术及 URL 重写技术实现一个购物车系统

项目描述:用户根据注册的用户登录 ID 及登录密码登录系统,将展现所有数据库中的商品信息。并且用户信息被存入 session 内。用户在商品信息列表内可以通过点选所需的各个商品,提交后进入该用户的购物车页面。购物车页面将罗列所有该用户准备购买并放入购物车的商品,用户可以选择删除不准备购买的商品、也可以点击"继续购买"回到商品信息页面进行继续追加购买商品。通过点击"退出"链接,可实现用户的登录退出、并同时销毁服务器为该用户创建的 session 对象。

1. 如图 4.15 所示为项目【ch4_3】项目目录结构。所涉及的核心类列表如图 4.16 所示。

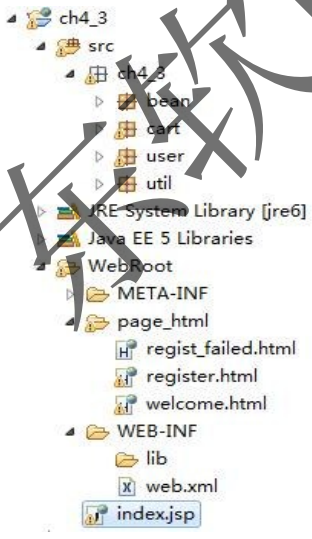

图 4.15 项目【ch4_3】项目目录结构

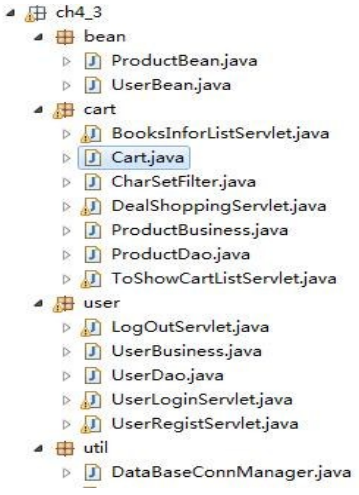

图 4.16 核心类列表

2. 创建购物车类【4-10】Cart.java。

```
package ch4_3.cart;
......
public class Cart {
    public Cart(){}
    private List<ProductBean> list_cart=null;
    public void addProToCart(ProductBean pro){
        if(list_cart == null){
            list_cart=new ArrayList<ProductBean>();
        }
        Iterator<ProductBean> it=list_cart.iterator();
        while(it.hasNext()){
            ProductBean prod=it.next();
            if(prod.getId().equals(pro.getId())){
                return;
            }
        }
        list_cart.add(pro);
    }
    public void removeProFromCart(String proId){
        ......
    }
    public int getAllProductPrice(){
        ......
    }
    public List<ProductBean> getAllProductFromCart(){
        return list_cart;
    }
}
```

3. 创建商品列表类【4-11】BooksInforListServlet.java。

```
package ch4_3.cart;
import java.io.IOException;
......
public class BooksInforListServlet  extends HttpServlet{
    private Connection con=null;
    public void init(ServletConfig config)throws ServletException{
        ServletContext context=config.getServletContext();
        String driver=context.getInitParameter("ORACLE_DRIVER");
        String url=context.getInitParameter("ORACLE_URL");
        String user=context.getInitParameter("ORACLE_USER");
```

```
        String pwd=context.getInitParameter("ORACLE_PWD");
        con=DataBaseConnManager.getConnection(driver,url,user,pwd);
    }
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
                throws ServletException,IOException{
        HttpSession session=request.getSession();
        System.out.println("session id is :"+session.getId());
        //取得 session 中出入的用户名
        String user_name=(String)session.getAttribute("user_name");
        //获得所有商品信息对象的集合
        ProductDao proDao=new ProductDao();
        ProductBusiness pb=new ProductBusiness(proDao);
        List<ProductBean> booksList=pb.getAllProduct(con);
        response.setContentType("text/html; charset=gb2312");
        PrintWriter out=response.getWriter();
        out.println("<html><head><title>信息列表页面</title></head>");
        out.println("<body>");
        out.println("<center>");
        out.println("<br>");
        out.println("<font color=blue size=3>"+user_name+" ," +
            "已经登陆到信息列表页面.</font><font size=3 color=green>" +
            "<a href=\""+response.encodeURL("logout")+"\">退出</font>");
        out.println("<br>");
        out.println("<br>");
        out.println("<form name=\"listProForm\" action=\""+
            response.encodeURL("doShopping")+"\" method=\"post\">");
        out.println("<table width=80% bgcolor=#ccffff>");
        out.println("<tr>");
        out.println("<th align=center bgcolor=red>商品 ID</th>");
        out.println("<th align=center bgcolor=#E6E6FA>所属类别</th>");
        out.println("<th align=center bgcolor=#E6E6FA>商品名称</th>");
        out.println("<th align=center bgcolor=#E6E6FA>生产商名</th>");
        out.println("<th align=center bgcolor=#E6E6FA>商品价格</th>");
        out.println("<th align=center bgcolor=#E6E6FA>商品描述</th>");
        out.println("<th align=center bgcolor=#E6E6FA>我要购买</th>");
        out.println("</tr>");
            Iterator<ProductBean> it=booksList.iterator();
            while(it.hasNext()){
                ProductBean p=it.next();
                out.println("<tr>");
```

```
                out.println("<td align=center>"+p.getId()+"</td>");
                out.println("<td align=center>"+p.getCateid()+"</td>");
                out.println("<td align=center>"+p.getName()+"</td>");
                out.println("<td align=center>"+p.getPublisher()+"</td>");
                out.println("<td align=center>"+p.getPrice()+"</td>");
                out.println("<td>"+p.getDesc()+"</td>");
                out.println("<td align=center><input type=\"checkbox\" " +
                    "name=\"proId\" value=\""+p.getId()+"\"></td>");
                out.println("<input type=\"hidden\" " +
                    "name=\"action\" value=\"add\"/>");
                out.println("</tr>");
            }
            //這裡僅是示例演示,實際開發時不要將大量數據存入 session 內。
            session.setAttribute("products",booksList);

        out.println("<tr><td colspan=7 align=center>");
        out.println("<input type=\"submit\" name=\"Submit\" value=\"购买\">");
        out.println("<input type=\"reset\" name=\"Reset\" value=\"取消\">");
        out.println("</td></tr>");
        out.println("</form>");
        out.println("</table>");
        out.println("<br>");
        out.println("<a href=page_html/welcome.html><b>返回首页</b></a>");
        out.println("<br><br>");
        out.println("</center>");
        out.println("</body>");
        out.println("</html>");
    }
    public void doPost(HttpServletRequest request,
            HttpServletResponse response)throws ServletException,IOException{
        this.doGet(request, response);

    }
}
```

4. 处理商品购买动作及从购物车中删除商品动作的 Servlet【4-12】DealShoppingServlet.java。

```
package ch4_3.cart;
......
public class DealShoppingServlet extends HttpServlet{
    private static final String CONTENT_TYPE="text/html;charset=gb2312";
    String[] proIds;
    List<ProductBean> products_list;
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
                    throws ServletException,IOException{
```

```
        response.setContentType(CONTENT_TYPE);
        HttpSession session=request.getSession();
        //从 session 中取出之前出入的所有商品对象的集合
        products_list=(List<ProductBean>)session.getAttribute("products");
        //第一次来购物时,该 session 里是没有这个 cart 对象的。
        Cart cart=(Cart)session.getAttribute("myCart");
        String action=request.getParameter("action");
        if("remove".equals(action)){
            String removeid=request.getParameter("removeid");
            cart.removeProFromCart(removeid);
        }else if("add".equals(action)){
            //该数组中,存放的都是一个同名的 id 对应的不同的值。(商品的 id 值)
            proIds=request.getParameterValues("proId");
            if(cart == null){
                cart=new Cart();
                session.setAttribute("myCart",cart);
            }
            if(proIds == null){
                proIds=new String[0];
            }
            Iterator<ProductBean> it=products_list.iterator();
            while(it.hasNext()){
                ProductBean products=it.next();
                for(int i=0;i<proIds.length;i++){
                    if(products.getId().equals(proIds[i])){
                        cart.addProToCart(products);
                    }
                }
            }
        }
RequestDispatcher rd=request.getRequestDispatcher("showCartList");
        rd.forward(request,response);
    }
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
            throws ServletException,IOException{
        doGet(request,response);
    }
}
```

5. 展现已经被选中并放入了购物车的商品信息界面的 Servlet【4-13】ToShowCartListServlet.java。

```
package ch4_3.cart;
......
import ch4_3.bean.ProductBean;
public class ToShowCartListServlet extends HttpServlet{
    private static final String CONTENT_TYPE="text/html;charset=gb2312";
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
                throws ServletException,IOException{
        response.setContentType(CONTENT_TYPE);
        PrintWriter out=response.getWriter();
        HttpSession session=request.getSession();
        Cart cart=(Cart)session.getAttribute("myCart");
        List<ProductBean> products=null;
        //这行代码是一定会执行的!! 注意!
        if(cart == null || (products=cart.getAllProductFromCart()) == null){
            out.println("<html>");
            out.println("<head><title>show cart Servlet</title></head>");
            out.println("<body bgcolor=\"orange\">");
            out.println("<center>");
            out.println("<p><h1>"+session.getAttribute("user_name")+
            "没有任何选择的商品</h1></p>");
            out.println("<p><a href=\""+response.encodeURL("booksInformationList")+
            "\">返回商品显示列表页面</a></p>");
            out.println("</center>");
            out.println("</body>");
            out.println("</html>");
            out.close();
        }else{
            Iterator<ProductBean> it=products.iterator();
            out.println("<html>");
            out.println("<head><title>show cart Servlet</title></head>");
            out.println("<body bgcolor=\"orange\">");
            out.println("<center>");
            out.println("<p><h1>"+session.getAttribute("user_name")+"," +
                "你目前购买的商品为:</h1></p>");
            out.println("<table border=\"1\" bgcolor=\"#99ffff\">");
            out.println("<tr bgcolor=\"#E6E6FA\"><th>产品名称</th><th>产品描述" +
```

```
                "</th><th>产品价格¥</th><th>操作</th></tr>");
            while(it.hasNext()){
                ProductBean pro=it.next();
                out.println("<tr><td>"+pro.getName()+"</td>");
                out.println("<td>"+pro.getDesc()+"</td>");
                out.println("<td>"+pro.getPrice()+"</td>");
                out.println("<td><a href=\""+response.encodeURL("doShopping?" +
                    "action=remove&removeid=")+pro.getId()+"\">删除该商品</a></td>");
                out.println("</tr>");
            }
            out.println("</table>");
            out.println("<p>目前你的购物车商品总价格是:"+cart.getAllProductPrice()
+"元人民币。</p>");
            out.println("<p></br><a href=\""+response.encodeURL("booksInformationList")+
"\">继续购买</a></p>");
            out.println("</center>");
            out.println("</body>");
            out.println("</html>");
            out.close();
        }
    }
public void doPost(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException{
        doGet(request,response);
    }
}
```

6. 接下来请参照提供的源代码完成处理用户的登录、注册、退出的 Servlet 类。并在部署描述文件 web.xml 中将这些 Servlet 进行正确的部署、配置。发布至 web 服务器后,通过访问路径。

```
http://localhost:8181/ch4_3/
```

即可登录默认欢迎页面,即如图 4.17 所示的购物模块默认登录页面。

图 4.17　购物模块默认登录页面

7. 当输入已经注册过的登录信息后，即可跳转到可购买的商品信息列表页面，如图 4.18 所示。

图 4.18　可购买的商品信息列表页面

8. 当在复选框中选择你的若干商品后，点击“购买”按钮，即进入“购物车”界面，其中罗列了所有你点选的、追加的商品信息。如图 4.19 所示为购物车商品信息页面。

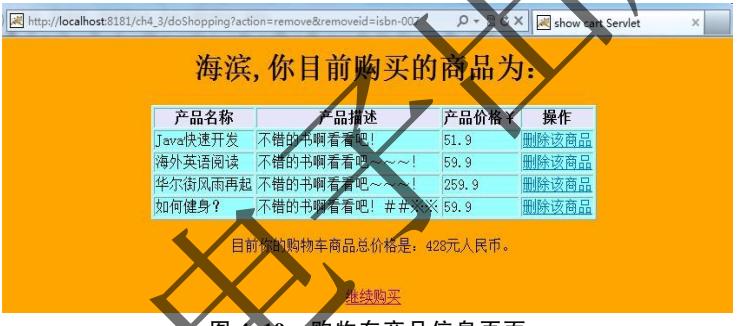

图 4.19　购物车商品信息页面

4.4.4.4　使用 ServletContext

在一个 Web 服务器中，每个 Web 应用程序都与一个上下文(context)环境关联，且不同的 Web 应用之间是彼此独立的。上下文环境与一个 ServletContext 对象相对应，也就是说每个部署在服务器中的 Web 应用，服务器都会为此应用创建一个独立的 ServletContext 对象，ServletContext 对象对于一个 Web 应用来说是唯一的。在同一个 Web 应用下的所有的 Web 资源都可以共享使用存储在 ServletContext 对象中的数据 。

在 Servlet 中我们可以通过多种方式获取 ServletContext 对象：

✧ 可以直接调用父类中的 getServletContext()方法获取；

✧ 通过 HttpSession 获取；

✧ 通过 FilterConfig 对象调用 getServletContext()方法获取；

✧ 通过 ServletConfig 对象获取。

在 web 应用中 ServletContext 对象更多的是用来提供给不同的 web 组件共享数据的，其存储数据的方法和 request 对象、session 对象中的相应方法一致。存储在 ServletContext 中的数据对整个应用程序是可见的。

```
public Object getAttribute(String name)
public void setAttribute (String name,Object attr)
public void removeAttribute (String name)
public java.util.Enumeration getArributeNames()
```

4.4.5 Servlet 请求资源的重定向

有时候我们在程序中需要把客户的请求跳转到服务器上的另一个资源或者其他服务器上的资源,这种情况下需要使用请求重定向技术,在 Servlet 中我们需要使用 HttpServletReponse 对象的 sendRedirect()方法,此方法定义为:

```
public void sendRedirect(String url);
```

参数 url 表示想要跳转资源的相对地址或者其他服务器上的绝对地址,例如:

```
response.sendRedirect("http://www.baidu.com");
```

请求重定向的基本原理是:把当前响应返回到浏览器,再通过浏览器发送一个新的资源请求到指定的地址。如图 4.20 所示为请求重定向示意图。

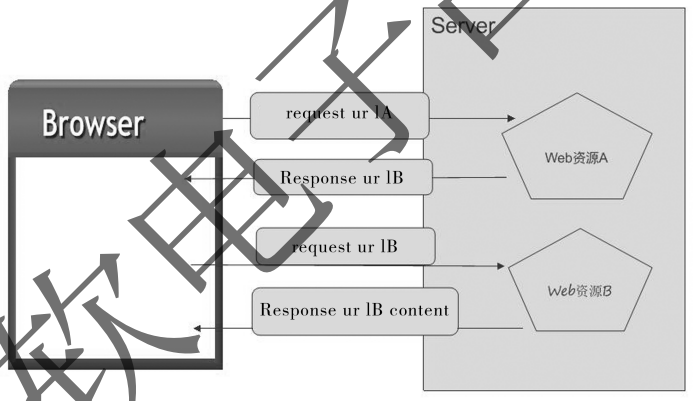

图 4.20 请求重定向示意图

请求重定向是发送一个新的请求到其他资源,所以请求资源 A 和请求资源 B 分别接收到了不同的 request 对象。

4.4.6 Servlet 请求资源的转发

Servlet 请求转发与 Servlet 请求重定向较为相似,是把当前的请求转发到另外一个资源,在 Servlet 中是使用一个 RequestDispatcher 对象的 forward 方法来转发请求资源的,具体的方法定义为:

```
public void forward(HttpServletRequest  request, HttpServletResponse response)
```

在这个方法中我们没有看到需要转发的资源地址,其实它的转发地址是在创建 RequestDispatcher 对象时设置的。

在 Servlet 中可以通过两种方式获取 RequestDispatcher 对象：

✧ HttpServletRequest. getRequestDispatcher(String url)；

✧ ServletContext. getRequestDispatcher(String url)；

对于第一种方式转向的地址 url，可以是针对此 Servlet 的相对路径，它的 url 可以不用加上表示路径的上下文信息符号“/”。例如：

```
//通过 request 获取 RequestDispatcher 对象，没有“/”表示相对于请求上下文的路径
RequestDispatcher rs= request.getRequestDispatcher(“index.htm”);
//转向特定资源
rs.forward( request,response);
```

对于第二种方式则必须是针对 Web 应用的绝对路径，所以在使用时，必须加上“/”。例如：

```
//在 Servlet 中获取 ServletContext 对象
ServletContext servletContext=getServletContext() ;
//通过 servletContext 获取 RequestDispatcher 对象，含有“/”表示 Web 应用上下文路径
RequestDispatcher rs=servletContext.getRequestDispatcher(“/index.htm”);
rs.forward( request,response);
```

这两种形式转向的页面都是 index. html，但是系统寻找的路径不一样。对于第一种，通过 request 转向的 index. html，系统会在该 Servlet 所在的上下文路径中去读取，如同相对路径一样的效果。对于第二种，转向的特定资源一定加上“/”，表示从绝对路径开始找这个文件，因为对于 ServletContext 而言，没有相对路径可获取，它所表示的，就是整个应用，这个应用中的所有资源，都是相对于上下文的根路径的！

另外要注意的一点是：request 对象和 ServletContext 对象都可以从 Servlet 程序中获取。但是，请求转发 url 只能是本应用程序类中的资源，不能请求到其他应用程序上的资源。

请求转发的基本原理是：在当前的请求基础上直接的转向到本应用中的另一个资源，中间不再创建新的请求对象，使用的是相同的 request 对象。所以当我们需要保存请求的相关信息的时候，我们一般使用请求转发。转发过程完全在服务器上进行，浏览器也无从知道转发的过程，所以在浏览器地址栏中，地址保持不变。另外，转发过程与客户端浏览器没有通信，所以转发的性能优于重定向。如图 4.21 所示为请求转发示意图。

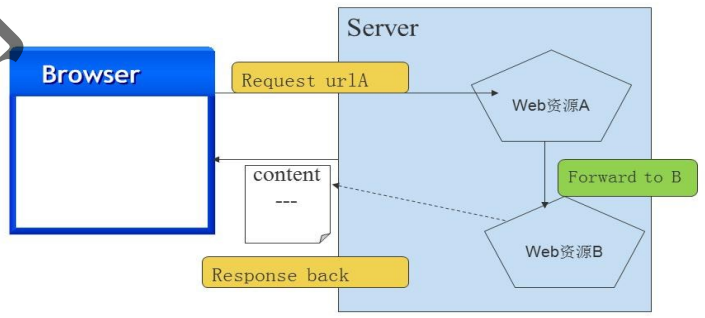

图 4.21　请求转发示意图

4.4.7 Servlet 请求资源的包含

在 Servlet 中除了可以将请求转发到其他的 Web 资源以外,还可以在一个资源中引用另一资源。

在 Servlet 中可以使用如下方式包含一个资源:

```
RequestDispatcher rd= request.getRequestDispatcher("包含的资源路径");
rd.include(request,response);
```

和转发一样,我们都是通过 RequestDispatcher 对象的相关方法执行的。该对象可以通过 request 对象或是 servletContext 对象获取,它们之间的区别与请求转发中分析的区别是相同的。

4.5 Servlet 过滤器

Servlet 过滤器(Filter)技术是从 Servlet 2.3 规范开始引入的,与 Servlet 技术一样,Servlet 过滤器也是一种 Web 应用程序组件,可以部署在 Web 应用程序中,但与其他的 web 组件不同的是,过滤器是"链"在容器的处理过程中的。这意味着过滤器能够在 servlet 或其他 web 资源被调用之前访问 request 对象,调用之后访问 response 对象,所以它能够对 servlet 容器的请求和响应对象进行检查或是修改其内容。

当用户的请求到达指定的网页之前,可以借助过滤去来改变这些请求的内容;同样的,当执行结果要响应到用户之前,若先经过过滤器,就可以修改输出的内容。

4.5.1 Servlet 过滤器的作用

Servlet 过滤器的应用比较广泛,目前主要应用在如下方面:

✧ 可以进行请求的权限判断。

✧ 可以处理文本乱码问题。页面表单通常提交的数据编码是"ISO8859-1",而对于中文系统来说我们需要接收页面的中文输入,为了能够正确获取页面的数据,需要在接收请求的资源中做编码设置与转换,在多个请求资源中都需要相同的操作,而使用过滤器可以只需要一次设置,整个 web 都可用。

✧ 过滤器还有很多的其他用途,例如 XML 转换过滤、数据压缩过滤、图像转换过滤、加密过滤 、请求与响应封装等。

4.5.2 Servlet 过滤器的创建及部署

要创建一个 Servleet 过滤器,就需要我们实现一个过滤器接口 javax.servlet.Filter,并且实现其中的三个抽象方法:

✧ public void init(FilterConfig config) throws ServletException 方法;

✧ public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)　　throws ServletException,IOException 方法；

✧ public void destroy()方法；

init 方法与 Servlet 中的 init 方法类似，只在此过滤器第一次初始化时执行。对于简单的过滤器，我们一般都把此方法置空，但是如果需要从 web.xml 文件中获取初始化参数，就必须要使用 init 方法。从该方法参数中可以获取一个 FilterConfig 对象，该对象由容器创建，它具有一个 getInitParameter()方法，能够访问部署描述符文件(web.xml)中分配给过滤器的初始化参数。见代码示例【4-14】CharSetFilter.java 实现一个字符集过滤器：

```
public class CharSetFilter implements Filter{
    String charSetName;
    public void init(FilterConfig config)throws ServletException{
            charSetName=config.getInitParameter("charset");
    }
    public void doFilter(ServletRequest request, ServletResponse response,FilterChain chain)
throws ServletException,IOException{
    HttpServletRequest httpRequest=(HttpServletRequest)request;
    httpRequest.setCharacterEncoding(charSetName);
    chain.doFilter(request, response);
    }
    public void destroy(){}
}
```

在每个 Servlet 的 init()方法中可以使用 ServletConfig 对象来获取 Servlet 的初始化参数及其 web 应用上下文环境。同样对于过滤器也有对应的 FilterConfig 对象，该对象与 ServletConfig 对象功能相类似，通过 init()方法由容器创建并传递给 Filter 实现类使用。

FilterConfig 对象与 ServletConfig 对象功能类似，通过 init 方法由容器创建并传递给 Filter 实现类使用。此接口中主要有以下几个方法：

✧ public String getFilterName()：返回 web.xml 中配置的过滤器的名字。

✧ public String getInitParameter(String name)：获取给定参数 name 的所对应的值。

✧ public Enumeration getInitParameterNames()：获得所有参数的名字集合。

✧ ServletContext getServletContext()：获得 ServletContext 对象用来获取有关 Servlet 上下文应用信息。

Filter 接口的 doFilter 方法以一个 FilterChain 对象作为它的第三个参数。在调用该对象的 doFilter 方法时，调用下一个相关的过滤器。这个过程一般持续到链中最后一个过滤器为止。在最后一个过滤器调用其 FilterChain 对象的 doFilter 方法时，调用请求的 web 资源本身。

```
FilterChain.doFilter(ServletRequest request,ServletResponse response);
```

当一个过滤器编写好之后，我们需要将它部署到容器里。Servlet 过滤器必须配置在 web.xml 文件中。主要配置两个元素，分别是：<filter>元素系统向服务器注册一个过滤器对象；<filter-mapping>元素指定该过滤器所匹配的 URL。

<fiter>元素类似于前面学到的 Servlet 配置的 servlet 元素，是用来定义并注册 Filter 对象的。

```
<filter>
    <filter-name> characterFilter </filter-name>
    <filter-class>myfilter.CharactEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>GB2312</param-value>
    </init-param>
</filter>
```

<filter-mapping>元素与 servlet 配置中的<servlet-mapping>一样是用来匹配客户端请求路径的。

```
<filter-mapping>
    <filter-name>encodingfilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

见过滤器的另一个应用,示例代码【4-15】AuthorizationFilter.java 展示了一个局域网访问权限控制过滤器:

```
public class AuthorizationFilter implements Filter {
    private FilterConfig filterConfig;
    public void init(FilterConfig filterConfig) throws ServletException{
        this.filterConfig=filterConfig;
    }
public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) {
    try{
        HttpServletRequest httpRequest=(HttpServletRequest)request;
        HttpServletResponse httpResponse=( HttpServletResponse)response;
        String hostAddress=httprequest. getRemoteAddr ();
        if (hostAddress.startsWith("192.168.0")){
            filterChain.doFilter(request, response);
        } else {
            String errorPage=filterConfig.getInitParameter("errorPage");
                httpResponse.sendRedirect(errorPage);
            }
        } catch (ServletException sx){
            filterConfig.getServletContext().log(sx.getMessage());
        } catch (IOException iox) {
            filterConfig.getServletContext().log(iox.getMessage());
        }
    }
    public void destroy(){}
}
```

该过滤器的部署文件 web.xml。

```
<filter>
    <filter-name>authorizationfilter</filter-name>
    <filter-class>myfilter.AuthorizationFilter </filter-class>
    <init-param>
        <param-name>errorPage</param-name>
        <param-value>error.htm</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name> authorizationfilter </filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

在同一个 web 应用中我们若配置了两个或多个过滤器，且匹配的都是相同的路径("/")，这是完全可以的。但是这在 Servlet 配置是不允许的，不同的 Servlet 匹配的路径不能相同。但在过滤器中是允许的。每一个配置的过滤器都将被服务器所调用。在相同的请求到来时，服务器将会按照 web.xml 中定义的<filter-mapping>的顺序依次调用相关的过滤器。

4.6　Servlet 事件监听

4.6.1　事件监听器的作用

Servlet 监听器用于监听一些重要事件的发生，监听器对象可以在事情发生前、发生后可以做一些必要的处理。Web 应用程序员可以利用 Servlet 时间监听 Listener 接口监听在容器中某一个执行的程序，并且根据其应用程序的需求做出适当的响应。在 Servlet 技术中已经定义了一些事件，并且我们可以针对这些事件来编写相关的事件监听器，从而对事件作出相应处理。

4.6.2　事件监听的分类和接口

Servlet 事件主要有 3 类：Servlet 上下文事件、会话事件与请求事件。与之对应的事件监听器是上下文事件监听器、会话事件监听器、请求事件监听器。表 4.2 为监听接口与对应事件类列表。

表 4.2　　监听接口与对应事件类列表

Servlet 版本	监听接口	事件类
Servlet2.2、Jsp1.1	HttpSessionBindingListener	HttpSessionBindingEvent
Servlet2.3、Jsp1.2	ServletContextListener	ServletContextEvent
	ServletContextAttributeListener	ServletContextAttributeEvent
	HttpSessionListener	HttpSessionEvent

(续表)

Servlet 版本	监听接口	事件类
	HttpSessionActivationListener	
	HttpSessionAttributeListener	
Servlet2.4、Jsp2.0	ServletRequestListener	ServletRequestEvent
	ServletRequestAttributeListener	ServletRequestAttributeEvent

1.对 Servlet 上下文进行监听

可以监听 ServletContext 对象的创建和删除以及属性的添加、删除和修改等操作。该监听器需要使用到如下两个接口类:

(1) ServletContextAttributeListener:监听对 ServletContext 属性的操作,如增加、删除、修改操作。

(2) ServletContextListener:监听 ServletContext 对象,当创建 ServletContext 时,激发 contextInitialized (ServletContextEvent sConEvent)方法;当销毁 ServletContext 时,激发 contextDestroyed(ServletContext-Event sConEvent)方法。

2.对 HttpSession 会话进行监听

可以监听 Http 会话活动情况、Http 会话中属性设置情况,也可以监听 Http 会话的 active、paasivate 情况等。该监听器需要使用到如下多个接口类:

(1) HttpSessionListener:监听 HttpSession 的操作。当创建一个 Session 时,触发 sessionCreated(SessionEvent sEvent)方法;当销毁一个 Session 时,触发 sessionDestroyed (HttpSessionEvent sEvent)方法。

(2) HttpSessionActivationListener:用于监听 Http 会话 active、passivate 情况。

(3) HttpSessionAttributeListener:监听 HttpSession 中属性的操作。当在 Session 增加一个属性时,激发 attributeAdded(HttpSessionBindingEvent se) 方法;当在 Session 删除一个属性时,激发 attributeRemoved(HttpSessionBindingEvent se)方法;在 Session 属性被重新设置时,激发 attributeReplaced(HttpSessionBindingEvent se) 方法。

3.对客户端请求进行监听

对客户端的请求进行监听是在 Servlet 2.4 规范中新添加的一项技术,使用的接口类如下:

(1)ServletRequestListener。ServletRequestListener 接口与 ServletContextListener 接口很类似,当有请求产生或是消亡时,会自动调用 requestInitialized( )方法和 requestDestroyed( )方法。

(2)ServletRequestAttrubuteListener。ServletRequestAttrubuteListener 接口监听 request 范围的变化,它有三个方法,见表 4.3。

表 4.3　　ServletRequestAttrubuteListener 的方法

方法	说明
attributeAdded(ServletRequesAttributeEvent　event)	如果有对象加入到 request 范围时，通知正在监听的对象
attributeReplaced(ServletRequestAttributeEvent event)	如果在 request 范围有对象取代另一个对象时，通知正在监听的对象
attributeRemoved(ServletRequestAttributeEvent event)	如果对象从 request 的范围被清除掉时，通知正在监听的对象

4.6.3　事件监听实现

1. 实现一个 Application 监听器，实际上就是对 ServletContext(ServletContext 上下文)的监听，主要使用 ServletContextListener 和 ServletContextAttributeListener 两个接口。在 ServletContext 监听操作中，一旦触发了 ServletContextListener 接口中定义的事件后，可以通过 ServletContextEvent 进行事件的处理。我们通过项目【ch4_4】实现一个简单的上下文监听器。

创建上下文监听器【4-16】ContextListenerTest.java：

```
package ch4_4.contextListener;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
public class ContextListenerTest implements ServletContextListener{
    //加载时，触发
    public void contextInitialized ( ServletContextEvent sce ){
        System.out.println("容器初始了...."+sce.getServletContext().getContextPath());
    }
    //销毁时，触发
    public void contextDestroyed ( ServletContextEvent sce ){
        System.out.println("容器销毁时触发了..."+sce.getServletContext().getContextPath());
    }
}
```

并为该监听器进行部署和配置，为 web.xml 文件增加如下代码：

```
<listener>
        <listener-class>ch4_4.contextListener.ContextListenerTest</listener-class>
  </listener>
```

发布该项目，启动服务器的同时，注意观察控制台的信息输出，会有如图 4.22 所示信息：

```
2013-1-13 11:39:32 org.apache.catalina.core.StandardEngine start
信息: Starting Servlet Engine: Apache Tomcat/6.0.14
容器初始化了..../ch4_4
2013-1-13 11:39:33 org.apache.catalina.core.ApplicationContext log
```

图 4.22　上下文监听器初始化应用信息监听

当我们在服务器上卸载该应用时,注意观察控制台的信息输出,会有如图 4.23 所示信息:

```
2013-1-13 11:47:40 org.apache.catalina.startup.HostConfig checkResources
信息: Undeploying context [/ch4_4]
容器销毁时触发了.../ch4_4
2013-1-13 11:47:40 org.apache.catalina.core.StandardContext listenerStop
```

图 4.23　上下文监听器卸载应用信息监听

2.在监听中,针对 session 的监听操作主要使用 HttpSessionListener 和 HttpSessionAttributeListener 和 HttpSessionBindingListener 接口。当需要对创建和销毁 session 操作进行监听时,我们会用到 HttpSessionListener 接口的 sessionCreated(HttpSessionEvent e)方法和 sessionDestroyed(HttpSessionEvent e)方法。当创建和销毁 session 后,将会产生 HttpSessionEvent 事件,用到的方法为 getSession( )。接下来我们通过我们项目【ch4_4】实现一个简单的 session 监听器。

创建上下文监听器【4-17】ContextListenerTest.java。

```
package ch4_4.sessionListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;
public class SessionListenerTest implements HttpSessionListener{
    //session 被创建时触发...
    public void sessionCreated ( HttpSessionEvent se ){
        System.out.println("Session 已经被创建,session id 为:"+se.getSession().getId());
    }
    //session 被销毁时触发...
    public void sessionDestroyed ( HttpSessionEvent se ){
        System.out.println("Session 已经被销毁,session id 为:"+se.getSession().getId());
    }
}
```

并为该监听器进行部署和配置,为 web.xml 文件增加如下代码:

```
<listener>
    <listener-class>ch4_4.sessionListener.SessionListenerTest</listener-class>
</listener>
```

发布该项目,启动服务器,访问我们本项目的某个动态页面的时候,如 index.jsp,注意观察控制台的信息输出,会有如图 4.24 所示信息:

```
2013-1-13 12:00:11 org.apache.catalina.startup.Catalina start
信息: Server startup in 1873 ms
Session已经被创建, session id 为: C6831521BB8648E31D183A78170B9B62
```

图 4.24　session 监听器监听会话创建信息