第 19 章　Java Servlet开发

有关 Servlet 的介绍在第 1 章就已经有所涉及，它是使用 Java 技术的核心企业 Web 支持方法之一，即可以使用来实现网站的动态。应用程序的开发人员可以创建 J2EE 兼容 Java Servlet 组件，然后可以通过 HTTP 来处理 Web 业务特定请求和生成响应。Java Servlet 除了可以用来建立 Web 支持 Java 企业系统，还可以用来实现过滤器和监听器等功能，后面的功能将在下一章节重点介绍。这一章将重点介绍 Java Servlet 的体系结构以及基于 Web 的创建过程。

本章要点包括以下内容：

- Servlet 的介绍
- Servlet 的体系结构
- Servlet 的实现步骤
- Servlet 类的声明
- Servlet 的实例介绍

19.1　什么是Servlet

Servlet 是一种 Java 编程语言类，用来扩展通过响应－请求编程模型服务应用程序访问的服务器功能。尽管 Servlet 能够响应任何类型的请求，但是它们一般用来扩展由 Web 服务器支持的应用程序。对这样的应用程序，Java Servlet 技术定义了 HTTP 专用的 Servlet 类。

Javax.Servlet 和 Javax.servlet.http 包为编写 Servlet 提供了接口和类。所有的 Servlet 必须执行定义了生命周期方法的 Servlet 接口。执行一般服务时，可以使用或者扩展 Java Servlet API 提供的 GenericServlet 类。为了处理 HTTP 专有的服务，HttpServlet 类提供了一些方法，例如：doGet 和 doPost。

这一章将重点放在对 HTTP 请求生成响应而如何编写 Servlet 类上面，以及对 Servlet 的部署。下一章将对 Serlvet 的一些高级应用作重点介绍。

19.2　Servlet体系结构

Java Servlet API 提供的标准接口使开发人员可以处理客户端应用程序请求和生成这些请求的相应响应。因此，它提供了服务器端应用程序开发的基本建筑块。Java Servlet API 定义了一般请求与响应框架，但最常用于通过 Web 处理 HTTP 请求和生成 HTTP 响应。本小节将介绍这种 J2EE 兼容 Java Servlet 的逻辑、物理与动态体系结构。

19.2.1　Servlet逻辑与物理体系结构

由下面的图 19.1 可以看出，HTTP 请求与响应建立在一个更加一般的 Servlet 请求和响应之上。企业 Web 应用程序开发人员建立定制 Java Servlet，将 HTTP Servlet 抽象具体化，其又将通用的 Servlet 框

架具体化。通过 HTTP 会话管理和 cookie 还可以抽象同一用户多个请求间的数据管理。此外，可以用特殊 Servlet 过滤器截获和转换请求与响应（这部分内容将在下面章节介绍），在请求、响应与 Java Servlet 交互前后进行。

框图 19-1 的核心是 J2EE 兼容 Java Servlet 容器。容器通常由第三方厂家提供，像本书使用的 Tomcat 服务器一般都已经提供了对 Servlet 的兼容，这些服务器实现部署企业应用程序所需的许多核心服务。

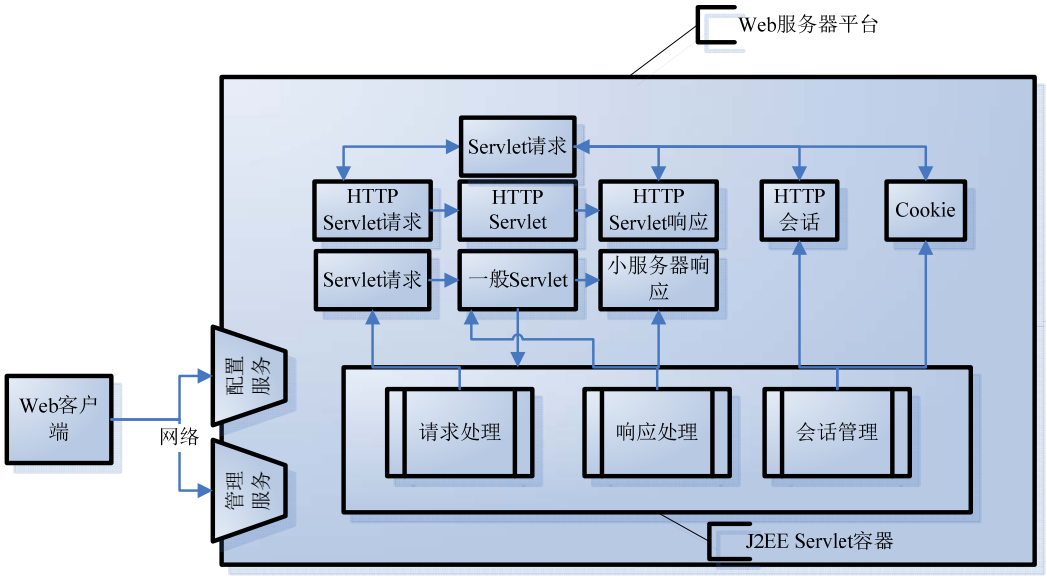

图 19.1　Servlet 体系结构

19.2.2　Servlet生命周期

Servlet 声明周期是由部署 Servlet 的容器控制。当请求被映射至 Servlet，容器执行下面步骤：

（1）Web 客户机请求 Servlet 服务时或者启动 Web 服务器时，容器将装载 Java.Servlet 类。

（2）容器根据客户机请求生成 Servlet 对象实例或者生成多个 Servlet 对象实例并将其加进 Servlet 实例池中。

（3）容器在 Servlet 实例化时调用 Servlet 初始化方法 HttpServlet.init()。

（4）容器构造 HttpServletRequest 与 HttpServletResponse 对象，包装 Web 客户机的待定 HTTP 请求和 Servlet 生成的响应。

（5）容器将 HttpServletRequest 与 HttpServletResponse 对象传入 HttpServlet.service()方法。定制 Java Servlet 可以访问这种 HTTP 请求与响应接口。

（6）定制 Java Servlet 从 HttpServletRequest 对象读取 HTTP 请求数据，从 HttpSession 或者 Cookie 对象访问任何状态信息，进行任何应用程序特定处理，并用 HttpServletResponse 对象生成 HTTP 响应数据返回客户端。

（7）Web 服务器关闭时，用 HttpServlet.destroy()方法关闭任何打开的资源。

19.3　Servlet实现

在编写 Servlet 程序时一般需要继承 Servlet 类接口（一般继承它的子类 GenericServlet，用来进行

Servlet 的初始化和关闭）和 HttpServlet 接口（用于处理 HTTP 请求和生成响应）。所以讲解 Servlet 编写，首先要了解 Servlet 和 HttpServlet 两接口，下面就从这个接口开始讲起。

19.3.1　Serlvet接口

Javax.servlet 包中的类与接口包装了一个抽象框架，建立接收请求和产生响应的组件（即 Servlet）。其中 Javax.servlet.Servlet 类是所有 Java Servlet 的基础接口类。Servlet.init()方法在 Servlet 实例化时由 Servlet 组件的容器调用一次。Servlet 容器要从服务器中删除 Servlet 时，容器会调用 Servlet.destroy()方法，使 Servlet 可以先清理任何资源之后再进入休眠状态。Init()方法取 javax.servlet.ServletConfig 对象作为参数，其中包括容器环境中的初始化信息。然后可以用 Servlet.getServletConfig()方法返回 ServletConfig 对象的句柄。可以用 getServletInfo()方法从 Servlet 实例取得 Servlet 的其他信息，如作者与版本信息。

Servlet 容器用 Servlet.service()方法在 Servlet 正常操作性使用期间处理请求和生成响应。Javax.servlet.ServletRequest 对象传入 serviced()方法，包含从输入流读取的所有 Servlet 请求信息。Javax.servlet.ServletResponse 对象也传入 service()方法，通过把信息写入输出流从 Servlet 中生成响应信息。另外要注意，Servlet 是线程安全的，Servlet 容器可以同时让多个线程调用同一个 Servlet 对象实例的 service()方法。

- ServletConfig 接口：Servlet 实现 ServletConfig 接口，以便从 Servlet 容器环境中接受初始化信息。
- GenericServlet 类：抽象 javax.servlet. GenericServlet 类提供大多数 Servlet 与 ServletConfig 接口的默认实现。
- ServletContext 接口：该接口类提供了 Servlet 容器情境的句柄。

19.3.2　HttpServlet类

应用程序特定 Servlet 类扩展了 HttpServlet 类，处理 HTTP 请求和生成 HTTP 响应。应用程序特定类需要覆盖掉一个或者几个 HttpServlet 方法及其基础 GenericServlet 类方法。GenericServlet 类派生的 init()、destroy()与 getServletInfo()方法是应用程序特定类通常会实现的方法。

HTTP 请求可以有多种形式，这些形式是由 HTTP 请求的请求方法字段名定义的。请求方法是所进行 HTTP 请求类型的标识符。容器接收相关 HTTP 请求方法类型时，小服务容器环境在 HttpServlet 实例中调用 HTTP 定义的 doXxx()方法之一。每一个 doXxx()方法接收一个 HttpServletRequest 与 HttpServletResponse 对象作为输入参数，处理 HTTP 请求和生成 HTTP 响应。指定 HTTP 请求方法情形中调用下列 HttpServlet 方法类型之一：

- doGet()方法：在 Web 服务器收到 HTTP GET 请求时调用。GET 请求取得 Web 服务器中特定资源的内容，入 HTML 页面。
- doPost()方法：在 Web 服务器收到 HTTP POST 请求时调用。POST 请求也取得 Web 服务器中的数据，但可以向 Web 服务器发信息。例如，POST 请求可以取得 HTML 表单中发出的数据。
- doPut()方法：在 Web 服务器收到 HTTP PUT 请求时调用。PUT 请求将数据上载到 Web 服务器中。
- doDelete()方法：在 Web 服务器收到 HTTP DELETE 请求时调用。DELETE 请求从 Web 服务器中删除数据，这个方法在实际开发过程中很少使用，让用户删除服务器中的资源风险太大。
- doHead()方法：在 Web 服务器收到 HTTP HEAD 请求时调用。HEAD 请求与 GET 请求相似，但只要求 HTTP 响应中的 HTTP 头。

- doOptions()方法：在 Web 服务器收到 HTTP OPTIONS 请求时调用。OPTIONS 请求取得 Web 服务器支持的配置选项。
- doTrace()方法：在 Web 服务器收到 HTTP TRACE 请求时调用。TRACE 请求让 Web 服务器返回客户机发送的 HTTP 头信息，用于调试。

在以上介绍的七个方法中，一般只需要在自己编写的 Servlet 类中覆盖其中的 doGet()和 doPost()方法即可。

19.3.3　初始化servlet类

将 servlet 生命周期的时候，已经介绍过，在 Web 容器加载及实例化 servlet 类之后或传送客户端请求之前，Web 容器会初始化 servlet 类。要自定义这个过程来使编写的 servlet 类读取持久配置数据，初始化资源，执行任何其他一次性活动，需要重载 servlet 接口中定义的 init()方法。例如下面的一段代码：

```
Public class CatalogServlet extends HttpServlet {
   Private BookDBAO bookDB;
   Public void init() throws ServletException {
      bookDB = (BookDBAO)getServletContext().getAttribute(“bookDB”);
      if(bookDB == null) throw new UnavailableException(“couldn’t get database.”);
   }
}
```

代码说明：在该 Servlet 执行之前会先调用其中的 init()方法进行相应资源的初始化。

19.3.4　编写服务方法

由 servlet 类提供的服务在 GenericServlet 的服务方法中实现，即通过覆盖 GenericServlet 父类中的相应方法。在 HttpServlet 对象的 doXxx()方法（其中 Xxx 可以取值为 Get、Post。Options 等）中或者任何其他由实现 servlet 接口的类定义的特定于协议的方法中实现。

服务方法的通常模式就是从请求中提取信息，访问外部资源，然后根据那些信息填充响应。

对于 HTTP Servlet 来说，填充响应的正确过程就是先从响应中获取输出流，然后填入响应标题，最后将任何主体内容写入输出流。响应标题总是必须在响应提交之前设定。任何在响应已经被提交之后设定和添加标题的尝试都会被 Web 容器忽略。紧接着下面讲解如何从请求中得到信息和生成响应。

19.3.4.1．从请求中获取信息

一个请求中包含了客户端和 servlet 之间传递的数据。所有的请求都实现 ServletRequest 接口，该接口中的方法用来接收数据。

例如下面一段获取传递信息的代码：

```
String bookId = request.getParameter(“add”);
If(bookId !=null)
      BookDetails book = bookDB.getBookDetails(bookId);
```

除了直接获取 HTTP 传递信息之后，还可以从请求中获取输入流并手动解析数据。要读取字符数据，请使用请求的 getReader()方法返回的 BufferReader 对象。要读取二进制数据，使用 getInputStream()方法返回的 ServletInputStream 对象。

19.3.4.2．构造响应

一个响应包括服务器和客户端之间传递的数据。所有的响应都实现 ServletResponse 接口。这个接口

定义的方法可以实现如下功能：

- 获取输出流来向客户端发送数据。
- 指出由带有 SetContentType（String）方法的响应返回的内容类型。
- 指出是否缓冲带有 setBufferSize(int)方法的输出。
- 设定本地化信息。
- 用来指出无法满足请求或者请求已经重定向的原因的状态码。
- 用来保存客户端的特定于应用程序的信息的 cookie。

下面通过覆盖 HttpServlet 接口类中的 doGet()方法来演示响应处理。详细代码如下：

```
public class BookDetailsServlet extends HttpServlet{
  public void doGet(HttpServletRequest request, HttpServletReponse response) throws ServletException, IOException{
    //设置响应标题
    response.setContentType(“text/html”);
    response.setBufferSize(8192);
    PrintWriter out = response.getWriter();
    out.println(“<html>”+”<head><title>”+message.getString(“titleBookDescription”)+”</title></head>”);
    RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(“/banner”);
    If(dispatcher != null)
      Dispatcher.include(request,response);
    String bookId = request.getParameter(“bookId”);
    If(bookId != null){
      Try {
        bookDetails bd = bookDB.getBookDetails(bookId);
        out.println(“<h2>”+bd.getTitle()+”</h2>”);
       }catch{Exception e}{
        Response.resetBuffer();
        Throw new ServletException(e);
      }
    }
    Out.println(“</body></html>”);
    Out.close();
  }
}
```

19.3.5　Servlet代码结构

下面的代码显示了一个简单 Servlet 的基本代码结构。该 Servlet 处理的是 GET 请求。如果读者不熟悉 GET，可以把它看成是当用户在浏览器地址栏输入 URL、点击 Web 页面中的链接、提交没有指定 METHOD 的表单时浏览器所发出的请求。Servlet 也可以很方便地处理 POST 请求。POST 请求是提交那些指定了 METHOD=“POST”的表单时所发出的请求。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SomeServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
```

```
    // 使用“request”读取和请求有关的信息（比如 Cookies）
    // 和表单数据

    // 使用“response”指定 HTTP 应答状态代码和应答头
    // （比如指定内容类型，设置 Cookie）

    PrintWriter out = response.getWriter();
    // 使用 "out"把应答内容发送到浏览器
}
}
```

代码说明：如果某个类要成为 Servlet，则它应该从 HttpServlet 继承，根据数据是通过 GET 还是 POST 发送，覆盖 doGet()、doPost()方法之一或全部。doGet()和 doPost()两个方法都有两个参数，分别为 HttpServletRequest 类型和 HttpServletResponse 类型。HttpServletRequest 提供访问有关请求的信息的方法，例如表单数据、HTTP 请求头等等。HttpServletResponse 除了提供用于指定 HTTP 应答状态（200，404 等）、应答头（Content-Type，Set-Cookie 等）的方法之外，最重要的是它提供了一个用于向客户端发送数据的 PrintWriter。对于简单的 Servlet 来说，它的大部分工作是通过 print 语句生成向客户端发送的页面。

注意 doGet 和 doPost 抛出两个异常，因此你必须在声明中包含它们。另外，你还必须导入 java.io 包（要用到 PrintWriter 等类）、javax.servlet 包（要用到 HttpServlet 等类）以及 javax.servlet.http 包（要用到 HttpServletRequest 类和 HttpServletResponse 类）。

最后，doGet 和 doPost 这两个方法是由 service 方法调用的，有时用户可能需要直接覆盖 service 方法，如 Servlet 要处理 GET 和 POST 两种请求时。

19.3.6　配置Servlet

要正常运行 Servlet 程序，还需要进行适当的配置，配置文件为 web.xml。其中使用到的部署描述项使用到几个元素，首先是<servlet>元素，它其中又包含两个子元素<servlet-name>与<servlet-class>。<servlet-name>元素指定 Servlet 名，它是对 Servlet 指定的任意名称，用作句柄，在部署描述项其他地方引用这个 Servlet。<servlet-class>元素指定 Servlet 类所在的完整位置。Servlet 声明语句如下：

```
<servlet>
  <servlet-name>BeeShirts</servlet-name>
  <servlet-class>cn.com.register.BeeShirtsServlet</servlet-class>
</servlet>
```

声明了 Servlet 之后，还需要使用<servlet-mapping>元素来将这个 Servlet 与输入请求相关联，其中该元素包含<servlet-name>元素，映射<servlet>元素中定义的<servlet-name>元素，还有一个<url-pattern>元素，指定要匹配的 URL 模式。例如：

```
<servlet-mapping>
  <servlet-name>BeeShirts</servlet-name>
  <url-pattern>/BeeShirtsServlet</url-patter>
</servlet-mapping>
```

这样配置完之后，就可以将 URL “/BeeShirtsServlet”映射到 cn.com.register.BeeShirtsServlet 类上。如果在本地调用该类（并且使用 Tomcat 服务器），则可以输入 URL 为：http://localhost:8080/BeeShirtsServlet。

19.3.7　一个简单的Servlet实例

下面是一个输出纯文本的简单 Servlet。

```
package example;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
   public void doPost(HttpServletRequest request,HttpServletResponse response)
   throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      out.println("Hello World");
   }
   public void doGet(HttpServletRequest request,HttpServletResponse response)
   throws ServletException, IOException {
      doPost(request,response);
   }
}
```

代码说明：该程序同时覆盖了 doPost()和 doGet()方法，只是调用 doGet()方法时，又定位到了 doPost()方法。如果需要同时服务这两种执行方法，则可以直接覆盖 service()方法。

在 Web.xml 配置文件中声明该 Servlet 文件：

```
<servlet>
   <servlet-name>helloworld</servlet-name>
   <servlet-class>example.helloworld</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>helloworld</servlet-name>
   <url-pattern>/helloworld</url-patter>
</servlet-mapping>
```

如果在本机运行该 Servlet，则可以直接在浏览器中输入 http://localhost:8080/helloworld；这时页面将打印出“helloworld”字符串信息。

19.4　Servlet处理表单数据

这一小节将介绍一个综合实例，来演示如何使用 Servlet 来处理用户提交的表单数据，然后将这些信息存储在数据库表中。

19.4.1　创建用户提交页面

首先创建表单提交页面 register.jsp，详细代码如下：

```
<html>
<head>
      <form method="post" action="/doUserRegist" name=form1">
      <table>
      <tr><td>用户姓名：</td><td><input name="username" size="15"/></tr>
```

```
    <tr><td>用户性别：</td>
    <td><select name="sex"><option value="name">男</option><option value="女">女</option></td>
    </tr>
    <tr><td>个人描述：</td><td><textArea cols=10 span=14></textArea></td></tr>
    <tr><td colsan=2><input type=”submit” value=”提交”></td></tr>
    </body>
</html>
```

代码说明：用户填写姓名、性别以及个人描述，然后单击“提交”按钮将数据提交给后台处理。为了简单起见，这里没有对输入框信息进行简单检验，读者可以使用 JavaScript 实现。

19.4.2　创建Servlet处理类

下面创建的 Servlet 类，是用来处理用户提交的数据，并进行数据库操作，这用户信息保存在数据库表 userinfo 中。Userinfo 数据库表只包含四个字段：id（用户 ID）、name（用户姓名）、sex（用户性别）以及 description（用户个人信息描述），读者在前面创建的 mysql 数据库中亲手创建该数据库表。这里创建的 Servlet 类文件如下：

```
package com.servlet;

import java.sql.Statement;
import java.sql.Connection;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class doUserRegist extends HttpServlet{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private String codeToString(String str){                    //对中文信息进行编码处理
        String s = str;
        try{
            byte tempB[] = s.getBytes("ISO-8859-1");
            s = new String(tempB);
            return s;
        }catch(Exception e){
            return s;
        }
    }
    public void init(ServletConfig config) throws ServletException{
        super.init(config);
    }
```

```
    public void destroy(){}
    public void doPost(HttpServletRequest request,HttpServletResponse response)
                                                    throws ServletException,IOException{
        response.setContentType("text/html;charset=GB2312");          //设置 out 输出所使用的编码规则
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        String username = codeToString(request.getParameter("username"));
        String sex = codeToString(request.getParameter("sex"));
        String description = codeToString(request.getParameter("description"));
        String str = "insert userinfo(name,sex,description) values('"+username+"','"+sex+"','"+description+"')";
        Connection con = null;
        Statement st = null;
        try{                                    //加载数据库驱动
            Class.forName("com.mysql.jdbc.Driver");
        }catch(ClassNotFoundException e){
            out.print("数据库驱动类找不到");
        }
        try{                                    //进行数据库连接，并进行数据库插入操作
            con = (Connection) DriverManager.getConnection("jdbc:mysql://localhost/mysql","root","12345");
            st =   con.createStatement();
            st.executeUpdate(str);
            out.print("用户注册成功");
        }catch(SQLException e){
            out.print("数据库操作异常");
        }
        out.print("</body></html>");
    }
}
```

代码说明：该 Servlet 类封装了数据库操作方法，将用户提交的信息保存到数据库表 userinfo 中。其中使用 PrintWriter 类中的 print()方法来动态地向客户端输入 html 静态页面。

19.4.3　Servlet类声明

Servlet 编写完之后，还需要在 Web 模块的 web.xml 配置文件中进行声明，该 Servlet 类的声明语句如下：

```
<servlet>
  <servlet-name> doUserRegist </servlet-name>
  <servlet-class>com.serlvet. doUserRegist </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name> doUserRegist </servlet-name>
  <url-pattern>/ doUserRegist </url-patter>
</servlet-mapping>
```

对 Servlet 声明完之后，就可以调用上面的 register.jsp 页面来进行演示。通过该实例，读者应该能够对 Servlet 来开发 Web 应用有了具体了解。

19.5　本章小结

Servlet是Java技术对CGI编程的回答。Servlet程序在服务器端运行，动态地生成Web页面。与传统的CGI以及许多其他类似CGI的技术相比，Java Servlet具有更高的效率，更容易使用，功能更强大，具有更好的可移植性，更节省投资。本章具体介绍了Servlet的概念和开发步骤。下一章将重点介绍Servlet的一些高级应用。