第3章 Servlet编程基础

学习目标：

通过本章的学习，你应该：

- 了解 HTTP 通信协议基本概念
- 了解 Servlet 的定义、功能和用途
- 掌握如何通过 NetBeans IDE 进行 Servlet 编程
- 掌握 Servlet 处理 HTTP 请求的流程
- 掌握设置 HTTP 响应头信息方法
- 掌握获取 HTTP 请求报头方法
- 掌握请求对象获取用户通过超级链接或表单提交的信息的方法
- 掌握在 Servlet 中处理表单通过 post 方式提交的信息中“中文乱码”问题
- 掌握响应对象对响应进行重定向的方法
- 掌握请求转发器转发请求的两个方法
- 理解 Servlet 的生命周期
- 了解 Servlet 的部署方式

一点提示：Servlet 的相关内容是非常重要的，掌握了 Servlet，学习 JSP 也会事半功倍，所以希望大家给予足够的重视。在本章中，并不深入探讨 Servlet 的理论知识，而是从应用入手，先通过 Servlet 技术完成一些简单而实用的操作，程序调试成功带来的小得意总好过一头雾水硬着头皮啃下去。唯一的要求就是：跟着书上的例子做一下。

Web 应用程序是基于 HTTP/HTTPS 的，所以先来了解一下 HTTP/HTTPS。

3.1 HTTP/HTTPS 通信协议基本概念

上网的时候，一般从用户的角度来说，其实并不关心某个网站是用什么技术实现的，采用了什么架构，大家更在乎的是网站的内容。用户操作的流程一般是这样的：打开浏览器，输入网址（或者通过某种快捷方式，例如：打开收藏夹），然后就开始了网上冲浪的旅程。一切都是那么的简单，例如：想访问百度，就输入 www. baidu. com ，想访问淘宝就输入 www. taobao. com 。但是实际操作时，注意观察一下就会发现，浏览器在这个过程中帮你做了一个补足的操作，地址栏里面真正显示出来的是 https://www. baidu. com/和 https://www.

taobao.com/ 。有共性没？有，前面多加了一个“https://”。（其实尾部还多了一个/，起什么作用请自行搜索 URL 格式）。

HTTPS(Hyper Text Transfer Protocol over Secure Socket Layer)是以安全为目标的 HTTP 通道，简单讲是 HTTP 的安全版。

HTTP(Hyper Text Transfer Protocol 超文本传输协议)是互联网上应用最为广泛的一种网络协议。所有的 WWW 文件都必须遵守这个标准。

设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法，浏览器就是通过这一协议与服务器交互。

HTTP 允许用户提出 HTTP“请求”(HTTP request)，然后由服务器根据实际处理的结果传回 HTTP“响应”(HTTP response)，它的基本运行方式为：

(1) 当用户向 Web 服务器发送请求时，Web 服务器将会开启一个新的连接，通过这个连接，用户可以将 HTTP 请求传送给 Web 服务器。

(2) 当 Web 服务器收到 HTTP 请求时，将进行解析与处理，并将处理结果包装成 HTTP 响应。

(3) 最后，Web 服务器会将 HTTP 响应传送给用户，只要用户接收到 HTTP 响应，Web 服务器就会关闭这个连接，用户的执行状态将不会被保存。

这个过程可以简化成“请求—处理—响应”的模型，简化模型如图 3.1 所示。本章学习的 Servlet 就是 Web 服务器端负责处理请求，并产生响应的一个组件。

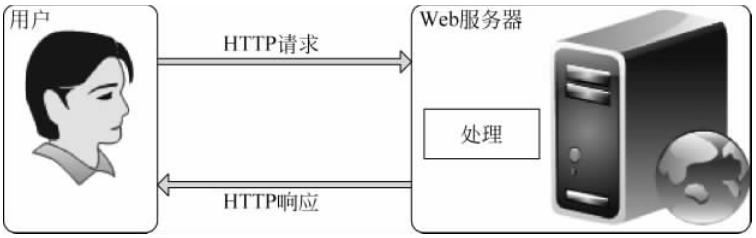

图 3.1 HTTP 运行方式简化模型“请求—处理—响应”

3.2 Servlet 的定义及作用

Servlet(Server Applet)是 Java Servlet 的简称，称为小服务程序或服务连接器，用 Java 编写的服务器端程序，主要功能在于交互式地浏览和修改数据，生成动态 Web 内容。

Servlet 的主要作用如下：

(1) 通过请求对象读取用户程序发送过来的显式数据(如表单数据)。

(2) 通过请求对象读取用户程序发送过来的隐式数据(如请求报头)。

(3) 处理数据并生成响应内容或设置响应报头。

在 Servlet 中，请求对象的类型为 HttpServletRequest，响应对象的类型为 HttpServletResponse。

3.3 使用 Servlet 生成一个网页

目标：创建一个 Java Web 应用程序，使用 Servlet 生成一个网页。

工程名：eg0301。

打开 NetBeans IDE，新建一个 Web 应用程序，具体过程可参考 1.2 节相关内容。在创建好的 eg0301 工程里，右击“源包”，在弹出的菜单中选择“新建→Servlet”命令，如图 3.2 所示。

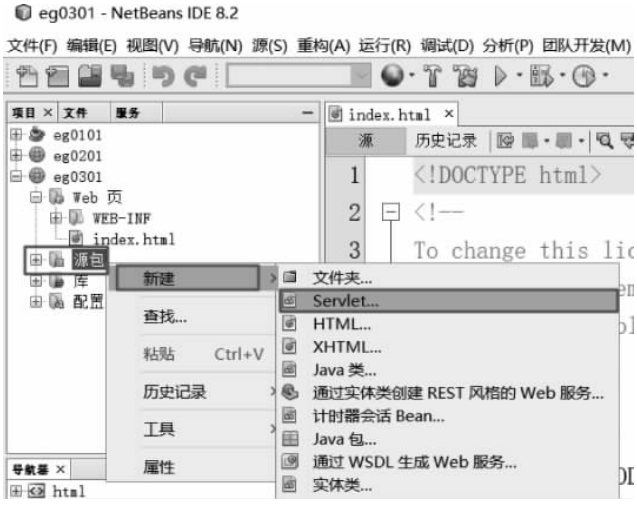

图 3.2 新建一个 Servlet

然后设置 Servlet 的名称和位置，此处类名暂时保持不变，包名设置为 cn.edu.djtu，然后单击“完成”按钮，如图 3.3 所示。

图 3.3 设置 Servlet 的名称和位置

完成后的效果如图 3.4 所示。

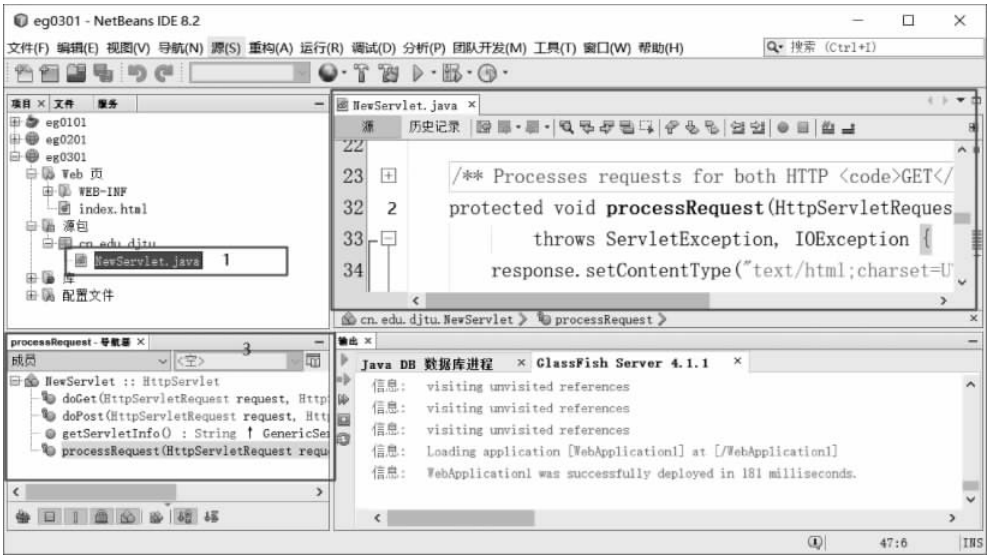

图 3.4　创建完 Servlet 后的状态

在图 3.4 区域 1 中右击 NewServlet.java 文档,选择“运行文件”命令,即可观察运行效果,运行会弹出如图 3.5 所示窗口,提示设置 Servlet 执行 URI,默认与 Servlet 名相同,此处暂时不设置,直接单击“确定”按钮。

图 3.5　设置 Servlet 执行 URI

运行效果如图 3.6 所示。

至此,就创建了一个名为 NewServlet.java 的 Servlet 文件,它的 urlPattern 是 NewServlet。当有用户请求该 Servlet 时,它生成一个网页返回给用户。

图 3.6　NewServlet 运行效果

3.4　doGet 方法与 doPost 方法

在 eg0301 中,NetBeans IDE 会根据代码模板自动生成一部分代码。图 3.4 所示区域 3 中可以直观便利地查看 NewServlet 类的结构。在图 3.4 中可以看到,创建 NewServlet 时

已经默认写好了4个方法，方法名分别为doGet、doPost、getServletInfo、processRequest。

其中getServletInfo方法可以返回对本Servlet的一个简短的描述，通常不需要实现具体业务逻辑，在实际应用中一般不必理会，默认其代码也是折叠的。

doGet方法响应用户通过get方式提交的请求，代码如下：

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
```

doPost方法响应用户通过post方式提交的请求，代码如下：

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
```

可见在NetBeans IDE中，doGet和doPost方法都是调用了processRequest方法。所以在NetBeans IDE中doGet和doPost方法代码也是折叠起来的。（其他开发环境未必如此）。

那么除了get方式和post方式，客户端是否有其他提交信息的方式呢？Servlet中又是否有对应的doXxx方法呢？答案是：有。

在NewServlet源码中的类代码空白处输入do，然后按下快捷键“Ctrl+\”。这时会发现还有doDelete、doHead、doTrace、doPut、doOptions五个方法，如图3.7所示。

图3.7 doGet/doPost以外的doXxx方法

这些方法虽然存在，但是很少有机会用到。在实际开发时，基本上只需要覆盖doGet或者doPost即可。在NetBeans IDE中，无论客户端是以get方式还是post方式提交请求，只要在processRequest方法中编写业务逻辑代码就可以了，都可以对客户端的请求做出正确响应。

3.5 使用 Servlet 生成服务器响应

在 eg0301 中,NewServlet 在服务器端生成了一个响应,形式是一个网页。具体代码如下:

NewServlet.java 代码片段

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code. */
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet NewServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet NewServlet at" + request.getContextPath() + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

这段代码是通过代码模板自动生成的,在编写自己的应用时,只要删除 try{}块中的代码,改为自己需要的业务逻辑代码即可。这段代码中,processRequest 方法参数列表中的形参 request 代表 HTTP 请求对象,response 代表 HTTP 响应对象。processRequest 方法体中,代码的第一条语句是:

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

通过响应对象 response 设置了响应的类型为网页形式,同时设置了编码格式为 UTF-8;接下来取得了一个 PrintWriter 类型的对象 out;最后通过 out 对象向客户端输出了一个网页。

除了生成类型为“text/html”形式的响应,也可以生成其他形式的响应,例如 Excel。这时需要设置响应的类型为"application/vnd.ms-excel" 。新建一个 Servlet 文件,文件名为 NewServlet1.java。其他参照 NewServlet 设置。运行 NewServlet1,会发现客户端接收到的是 Excel 文档。

NewServlet1.java 代码片段

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("application/vnd.ms-excel");
    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code. */
```

```
            out.println("1\t2");
            out.println("3\t4");
        }
    }
```

输出语句中的“\t”起的作用是在 Excel 中将 1 和 2 两个数字分隔在两个单元格里。这个样例不具有实用价值，实际生成 Office 文档时需要考虑的因素很多，所以通常都采用第三方的包，例如 Apache 的 POI 。

“text/html”和“application/vnd. ms-excel”都是 MIME 类型的一员。MIME (Multipurpose Internet Mail Extensions)是描述消息内容类型的因特网标准。MIME 消息能包含文本、图像、音频、视频以及其他应用程序专用的数据。

除了设置内容类型(就是 MIME 类型)，还可以生成其他类型的响应头，在这里就不一一举例了。可以尝试将 NewServlet.java 中增加一行代码。

```
response.sendError(404);
response.setContentType("text/html;charset = UTF - 8");
```

再次运行 NewServlet.Java 观察结果有什么变化。

3.6 使用 Servlet 读取请求报头

尽管在运行 NewServlet.java 时，看起来客户端并没有提交任何信息，其实还是有很多信息被提交了。这部分信息可以被服务器取得，一般不对其进行数据操作，主要就是请求报头的信息。

取得请求报头的信息需要用到请求对象，主要用到它的三个方法，如表 3.1 所示。

表 3.1 javax.servlet.http.HttpServletRequest 定义的方法

方法名	返回值类型	适 用 情 况
getHeader(String name)	String	取得特定请求报头的信息，返回值为字符串
getHeaders(String name)	Enumeration	取得特定请求报头的信息，返回值为枚举类型的对象
getHeaderNames()	Enumeration	取得 HTTP 请求内所有请求报头的信息，返回值为枚举类型的对象

下面通过示例演示如何通过 Servlet 来处理客户端提交的请求报头信息。

目标：学习使用 Servlet，读取客户端提交的请求报头信息。

工程名：eg0302。

用到的文件如表 3.2 所示。

表 3.2 eg0302 用到的文件及文件说明

文 件 名	说 明
ShowHeadServlet.java	手动创建的 Servlet，用来取得用户提交的请求报头信息，并以网页形式显示

编程思路：先取得全部的请求报头名，然后通过请求报头名取得对应的请求报头的值并输出。

打开 NetBeans IDE，新建一个名为 eg0302 的 Java Web 项目，具体过程略；新建名为 ShowHeadServlet.java 的 Servlet，具体过程略。编写各部分代码如下(仅列出核心代码，大部分自动生成的代码略)：

ShowHeadServlet.java 代码片段

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
        Enumeration<String> allHeaderNames = request.getHeaderNames();
            while (allHeaderNames.hasMoreElements()) {
                String headerName = allHeaderNames.nextElement();
                String headerValue = request.getHeader(headerName);
                out.print(headerName + " : " + headerValue);
                out.print("<hr />");
            }
        }
    }
```

运行 ShowHeadServlet，运行结果如图 3.8 所示。

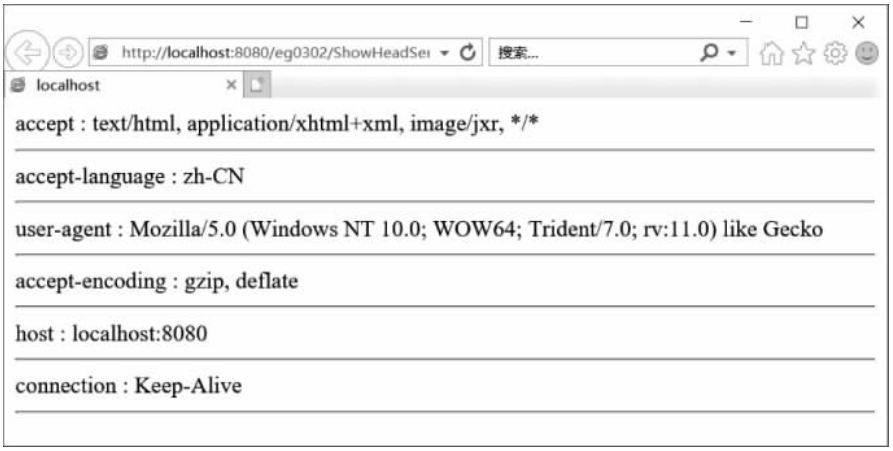

图 3.8 ShowHeadServlet 运行结果

3.7 使用 Servlet 读取用户通过超级链接传送的信息

在学习之前先看一个例子。打开百度首页，在右上角选择“更多产品”，里面有“糯米”，单击“糯米”即可到达“百度糯米”首页。观察地址栏可以看到 URL 中有“?cid=002540”，如图 3.9 所示。说明从百度首页跳转到百度糯米首页时，除了跳转这个动作以外，同时也传递了一个参数 cid，它传递的值是 002540。仿照这个例子，也传递一个值为 002540 的参数

cid，并且在 Servlet 里将参数传递的值输出到网页上。

图 3.9 百度首页跳转糯米网传递参数

目标：学习使用 Servlet，读取用户通过超级链接提交的信息。

工程名：eg0303。

用到的文件如表 3.3 所示。

表 3.3 eg0303 用到的文件及文件说明

文 件 名	说 明
index.html	创建工程时自动创建。为项目首页。页面上有一个超级链接，用于完成跳转及传递参数
ShowCidServlet.java	手动创建的 Servlet，用来取得用户通过超级链接提交的 cid 信息，并以网页形式显示

编程思路：先取得参数 cid 的值并赋给一个临时变量，将该变量的值输出。

打开 NetBeans IDE，新建一个名为 eg0303 的 Java Web 项目，具体过程略；新建名为 ShowCidServlet.java 的 Servlet，具体过程略。编写各部分代码如下（仅列出核心代码，大部分自动生成的代码略）：

index.html 代码片段

```
<!DOCTYPE html>
<html>
    <head>
        <title>通过超级链接提交 cid 的值</title>
        <meta charset = "UTF-8">
        <meta name = "viewport" content = "width = device-width, initial-scale = 1.0">
    </head>
    <body>
        <a href = "ShowCidServlet?cid = 002540">向 ShowCidServlet 提交 cid</a>
```

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

在 eg0302 中使用了请求对象的 3 个方法，getHeader(String name)、getHeaders(String name)、getHeaderNames()。在这个练习中将使用请求对象的 getParameter(String name)方法，如表 3.4 所示。这个方法适用于一个参数名对应一个参数值的情况。

表 3.4　javax.servlet.http.HttpServletRequest 定义的方法

方　法　名	返回值类型	适 用 情 况
getParameter(String name)	String	取得某个请求参数的参数值，假如该参数没有值则返回""；假如该参数不存在则返回 null

ShowCidServlet.java 代码片段

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            //为了区分参数 cid 和返回值，返回值刻意声明了新的引用名
            //通过请求对象获得变量 cid 对应的值
            String mycid = request.getParameter("cid");
            out.print(mycid);
    }
}
```

运行结果如图 3.10 所示。

图 3.10　eg0303 运行结果

3.8　使用 Servlet 读取用户通过表单传送的信息

尽管表单内的控件多种多样，但是实际上抽象为两类即可。一类是类似于超级链接 cid = 002540 这种，一个变量名对应一个变量值，例如：文本框、密码框、单选按钮等。另一类是复选框这种，用户的选择不是固定的，可能一个也没选，也可能全都选了。例如下面代码中，参数 fruits 对应的值在集合{banana,apple,orange}中选取。

```
<input type="checkbox" name="fruits" value="banana"/> 香蕉
<input type="checkbox" name="fruits" value="apple"/> 苹果
<input type="checkbox" name="fruits" value="orange"/> 橙子
```

对于第一类，用的方法依然是请求对象的 getParameter(String name)方法。而对于第二类，则要用到请求对象的另一个方法 getParameterValues(String name)，如表 3.5 所示。

表 3.5 javax.servlet.http.HttpServletRequest 定义的方法

方 法 名	返回值类型	适 用 情 况
getParameterValues(String name)	String[]	如果某个参数可能有多个值，可用此方法一次取得全部的值并封装在一个 String 数组中；假如该参数不存在则返回 null

目标：学习使用 Servlet，读取用户通过表单提交的信息。

工程名：eg0304。

用到的文件如表 3.6 所示。

表 3.6 eg0304 用到的文件及文件说明

文 件 名	说 明
index.html	创建工程时自动创建。为项目首页。页面上有一个表单。表单内有一个文本框，一组复选框
ShowFormServlet.java	手动创建的 Servlet，用来取得用户通过表单提交的信息，并以网页形式显示

编程思路：依次取得文本框和复选框的值，并分别输出，中间以水平线分隔。

打开 NetBeans IDE，新建一个名为 eg0304 的 Java Web 项目，具体过程略；新建名为 ShowFormServlet.java 的 Servlet，具体过程略。编写各部分代码如下(仅列出核心代码，大部分自动生成的代码略)：

index.html 代码片段

```
<!DOCTYPE html>
<html>
    <head>
        <title>通过表单提交</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <form action="ShowFormServlet" method="post">
          <input type="text" name="msg" value="" />
          <br />
          <input type="checkbox" name="fruits" value="banana" />香蕉
          <input type="checkbox" name="fruits" value="apple" />苹果
          <input type="checkbox" name="fruits" value="orange" />橙子
          <br />
```

```
            <input type="submit" value="提交" />
        </form>
    </body>
</html>
```

ShowFormServlet.java 代码片段

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            //获得 msg 的值并输出
            String msg = request.getParameter("msg");
            out.print(msg);
            out.print("<hr />");
            //获得 fruits 的值,如果用户未做选择,提示"未做选择"
            //否则遍历输出
            String[] fruits = request.getParameterValues("fruits");
            if (fruits == null) {
                out.print("未做选择");
            } else {
                for (String fruit : fruits) {
                    out.print(fruit + "<br />");
                }
            }
    }
}
```

运行结果如图 3.11 所示。

图 3.11 eg0304 运行结果

3.9 处理表单提交的中文乱码问题

重新运行 eg0304,在文本框中输入中文信息后提交,会发现尽管输出了信息,但是信息未能正确显示,出现了“乱码”,如图 3.12 所示。

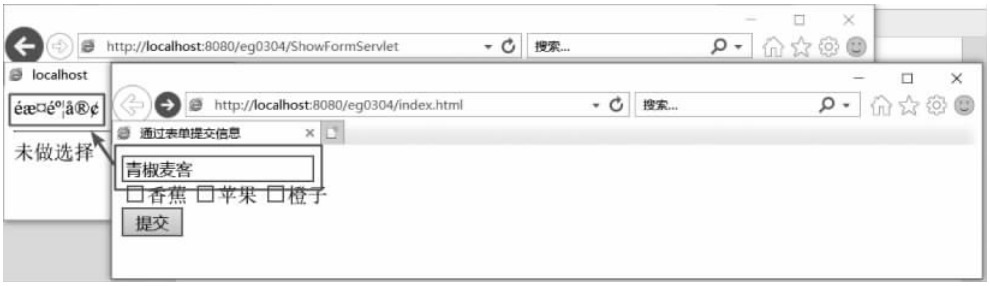

图 3.12 eg0304 运行出现乱码

追本溯源,乱码的产生主要是因为数据在传递过程中编码格式不完全一致。但是如果细分,中文乱码的产生原因很多,根据不同的成因也有不同的解决办法,本节只介绍如何处理经过 post 方式发送请求,提交的信息中含有中文的情况。对于乱码问题的更多内容,感兴趣的读者可以通过搜索引擎继续学习。

处理经过 post 方式提交中文,从而产生乱码的问题,解决方案并不复杂。只需要在调用请求对象,通过请求对象获取请求参数值之前,添加一条语句就可以了。

```
request.setCharacterEncoding("UTF-8");
```

这条语句将请求对象的编码格式设置为 UTF-8,与响应对象设置的字符集相同。(并不是永远都设置成 UTF-8,需要视当前项目的具体情况而定,保持统一即可)。在 NetBeans IDE 中,默认的各字符编码均为 UTF-8,例如响应类型默认的字符编码也是 UTF-8。

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

HTML 页面设置的编码也是 UTF-8。

```
<meta charset="UTF-8">
```

通常为了确保设置请求对象字符编码的语句绝对有效,需要把这条语句放在 Servlet 中的第一句,这样之后再通过请求对象获得参数值时,字符编码就肯定是 UTF-8 了。改写后的 ShowFormServlet 代码为:

ShowFormServlet.java 代码片段(处理中文乱码后)

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            //获得 msg 的值并输出
            String msg = request.getParameter("msg");
```

```
                out.print(msg);
                out.print("<hr />");
                //获得 fruits 的值,如果用户未做选择,提示"未做选择"
                //否则遍历输出
                String[] fruits = request.getParameterValues("fruits");
                if (fruits == null) {
                    out.print("未做选择");
                } else {
                    for (String fruit : fruits) {
                        out.print(fruit + "<br />");
                    }
                }
        }
    }
```

处理完乱码后再次输入数据测试程序,运行效果如图 3.13 所示。

图 3.13 乱码处理完成后运行效果

3.10 对响应进行重定向

在实际应用中,取得用户的信息只是第一步,之后往往还要根据输入的信息,对用户的请求做出响应,定位到其他的资源(Servlet 或者网页)。如何解决这一问题呢? 对响应进行重定向是一种解决方案。

对响应进行重定向,就是将响应重新定向到别的资源,相当于重新发送了一个 HTTP 请求。通过响应对象的 sendRedirect 方法可以实现响应的重定向,如表 3.7 所示。

表 3.7 javax.servlet.http.HttpServletResponse 定义的方法

方 法 名	返回值类型	适 用 情 况
sendRedirect(String location)	void	将用户重定向至其他页面或网站。假如 location 不是以"/"开头,容器解析为相对于当前 URI。假如 location 以"/"开头,容器认为相对于当前 Servlet 所在容器的根目录。假如响应已经被提交了,调用这个方法将抛出 IllegalStateException 类型的异常

下面通过一个用户登录的示例演示通过响应对象的 sendRedirect 方法实现响应的重定向。

目标：学习使用响应对象，完成对响应进行重定向。

工程名：eg0305。

用到的文件如表 3.8 所示。

表 3.8 eg0305 用到的文件及文件说明

文 件 名	说 明
login.html	手动创建的页面，包含一个表单，表单中有文本框用来输入用户名，密码框用来输入密码，提交按钮用来提交登录信息
error.html	手动创建的页面，页面显示“登录失败，返回登录页”，其中返回登录页为链接文字，链接至 login.html
HandleLoginServlet.java	手动创建的 Servlet，用来取得用户提交的登录信息，如果用户名为 admin，密码为 1234，则输出“欢迎光临!”，否则将响应重定向至 error.html

打开 NetBeans IDE，新建一个名为 eg0305 的 Java Web 项目，具体过程略；新建名为 login.html 和 error.html 的 HTML 文件；新建名为 HandleLoginServlet.java 的 Servlet，具体过程略。编写各部分代码如下(仅列出核心代码，大部分自动生成的代码略)：

login.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>用户登录</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <form method="post" action="HandleLoginServlet">
            用户名:<input type="text" name="userName" value="" /><br />
            密码:<input type="password" name="userPass" value="" /><br />
            <input type="submit" value="登录" />
        </form>
    </body>
</html>
```

error.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>登录失败</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
```

```
    <body>
        <div>登录失败<a href = "login.html">返回登录页</a></div>
    </body>
</html>
```

HandleLoginServlet.java 代码片段

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset = UTF - 8");
    try (PrintWriter out = response.getWriter()) {
        String userName = request.getParameter("userName");
        String userPass = request.getParameter("userPass");
        if ("admin".equalsIgnoreCase(userName) && "1234".equals(userPass)) {
            out.print("欢迎光临");
        } else {
            response.sendRedirect("error.html");
        }
    }
}
```

当用户名和密码输入错误时,效果如图 3.14 所示,注意地址栏,可以看到用户在 login.jsp 输入登录信息后,经过 Servlet 处理,被重新定向到了 error.jsp。

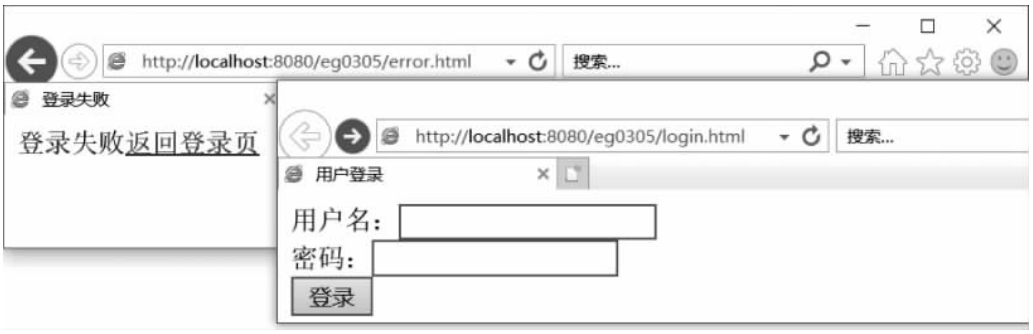

图 3.14 重新定向到了 error.html 效果

响应的重定向能够达到从一个资源跳转到另一个资源的目的,它类似于生活中拨打 114 的场景。例如说班级要搞团建,于是拨打 114 找某某饭店提前预订座位。这时 114 并不会帮你直接预订座位,而是把饭店的电话告诉你。你再拨打饭店电话才能由饭店帮你完成座位预定的需求。在这个过程中,你需要打两个电话,一个打给 114,一个打给饭店。

刚刚完成的用户登录的例子,用户最终提交了两次请求,一次是通过表单的 action 属性设置的,提交给 HandleLoginServlet;一次是由 Servlet 中的重定向语句提交的,提交给 error.html。

3.11 使用请求转发器(RequestDispatcher)转发请求

除了重定向响应以外,请求转发器的两个方法也可以实现在资源之间跳转,如表 3.9 所示。响应的重定向和请求转发器的方法适用场合不同。

请求转发器(javax.servlet.RequestDispatcher)是一个接口,可以通过请求对象得到一个 RequestDispatcher 对象,以下代码通过请求对象获得了一个转发器对象,该转发器可以将请求转发至 index.html。

```
request.getRequestDispatcher("index.html");
```

通过转发器对象的相应方法可以实现请求的转发。

表 3.9 javax.servlet.RequestDispatcher 定义的方法

方 法 名	返回值类型	适 用 情 况
forward(ServletRequest request, ServletResponse response)	void	将请求从一个 Servlet 转发给服务器上其他的 Servlet、JSP 或者 HTML
include(ServletRequest request, ServletResponse response)	void	在响应中包含其他资源的内容(如 Servlet、JSP 或者 HTML)

注意:include()方法与 forward()方法非常类似,唯一的不同在于:利用 include()方法将 HTTP 请求转送给其他 Servlet 后,被调用的 Servlet 虽然可以处理这个 HTTP 请求,但是最后的主导权仍然是在原来的 Servlet。换言之,被调用的 Servlet 如果产生任何 HTTP 响应,将会并入原来的 HttpResponse 对象。

下面通过用户登录样例的另一种实现来学习如何转发请求实现资源间的跳转。这个样例与 eg0305 有两个主要的区别:一是为了避免中文字符的乱码问题,网页文件不是 HTML 文档,而是 JSP 文档。二是实现的逻辑略有变化,输入的用户名密码正确时,显示 index.jsp 的内容;用户名和密码错误时,提示错误信息,并显示 login.jsp 页面内容。

目标:学习使用转发器对象(RequestDispatcher)的两个方法,实现对请求进行转发,并比较两个方法的异同。

工程名:eg0306。

用到的文件如表 3.10 所示。

表 3.10 eg0306 用到的文件及文件说明

文 件 名	说 明
login.jsp	手动创建的 JSP 页面,包含一个表单,表单中有文本框用来输入用户名,密码框用来输入密码,提交按钮用来提交登录信息
index.jsp	手动创建的 JSP 页面(为了避免引起混淆,请将 index.html 文件删除)。页面显示“欢迎光临!”
HandleLoginServlet.java	手动创建的 Servlet,用来取得用户提交的登录信息,如果用户名为 admin,密码为 1234,则转发至 index.jsp,否则输出“用户名或密码错误,请重新输入!”并显示登录页(login.jsp)内容

打开 NetBeans IDE,新建一个名为 eg0306 的 Java Web 项目,具体过程略;删除 index.html 文件;新建名为 login.jsp、index.jsp 的 JSP 文件;新建名为 HandleLoginServlet.java 的 Servlet,具体过程略。编写各部分代码如下(仅列出核心代码,大部分自动生成的代码略)。新建 JSP 文件的过程如图 3.15 所示。

图 3.15 新建 JSP 文件

login.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>用户登录</title>
    </head>
    <body>
            <h1>用户登录</h1>
            <form method="post" action="HandleLoginServlet">
                用户名:<input type="text" name="userName" value="" /><br />
                密码:<input type="password" name="userPass" value="" />
                <br />
                <input type="submit" value="登录" />
            </form>
    </body>
</html>
```

index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
        <title>首页</title>
    </head>
    <body>
            <h1>欢迎光临!</h1>
    </body>
</html>
```

HandleLoginServlet.java 代码片段

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            String userName = request.getParameter("userName");
            String userPass = request.getParameter("userPass");
            if ("admin".equalsIgnoreCase(userName) && "1234".equals(userPass)) {
                out.print("用户名、密码正确");              //本行输出不可见
              RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
                rd.forward(request, response);
            } else {
                out.print("用户名、密码错误,请重新输入");  //本行输出可见
              RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
                rd.include(request, response);
            }
    }
}
```

请注意代码中单行注释部分,通过两条输出语句效果的不同,比较 forward 方法和 include 方法的不同。

```
out.print("用户名、密码正确");                    //本行输出不可见
out.print("用户名、密码错误,请重新输入");        //本行输出可见
```

当用户名、密码输入正确时,效果如图 3.16 所示,注意地址栏,可以看到用户在 login.jsp 输入正确的登录信息后,在 Servlet 中处理时,请求被转发到了 index.jsp(因为显示出了 index.jsp 的内容),但是地址栏并没有变化,仍然是对 HandleLoginServlet 进行的请求。

当用户名、密码输入错误时,效果如图 3.17 所示,注意地址栏,可以看到用户在 login.jsp 输入登录信息后,在 Servlet 中处理时,请求被转发到了 login.jsp(因为显示出了 login.jsp 的内容),但是地址栏并没有变化,仍然是对 HandleLoginServlet 进行的请求。

使用请求转发器对请求进行转发,类似于生活中拨打 120 的场景。假设某一天,有人意外受伤了,或者得了急病需要治疗,这时需要拨打 120 急救电话。假如 120 急救中心接到电话后只是告诉你对不起,我处理不了,请找某某医院。那么这种情况就相当于响应的重定向,问题并没有得到最终的解决,你被重定向到另一个部门了,会由另一个部门解决。

好在现实生活中不是这样,120 急救中心通常会了解病人的具体情况、地址等信息,然

图 3.16 用户名和密码输入正确时的运行效果

图 3.17 用户名和密码输入错误时的运行效果

后根据就近、就急、病人家属意愿和专业对应等原则,指挥调度 120 急救车开展医疗救治工作。这样,从病人的角度看,好像只是呼叫了 120 急救中心就得到了服务,但实际处理中既涉及了急救中心的接警部门,也涉及了医院及医护人员,经过了多个部门,这种情况就相当于通过请求转发器对请求进行转发。尽管只是一次请求,但是该请求被转发到了好几个部门,多个部门协同解决了这一请求。

刚刚完成的改版之后的用户登录的例子,当用户提交的用户名和密码正确时,请求 HandleLoginServlet。在 HandleLoginServlet 中,请求转发器根据转发条件,将请求通过 forward 方法转发给 index.jsp 页面。由 index.jsp 页面负责生成响应,完成了资源间的跳转,但是地址栏没变,是一次请求而不是两次。如图 3.16 所示。

当用户提交的用户名和密码错误时,请求 HandleLoginServlet。在 HandleLoginServlet 中,

请求转发器根据转发条件，将请求通过 include 方法转发给了 login.jsp 页面。依然由 HandleLoginServlet 负责生成响应，login.jsp 页面的内容被包含到最终的响应中，也完成了资源间的跳转，地址栏没变，也是一次请求而不是两次。如图 3.17 所示。

注意：在本节不必太过于纠结到底该使用 forward 方法还是 include 方法。在第 10 章中可以看到，当可用的技术手段丰富起来时，选择方案也更加多样化。在 10.3 节 MIS 第 1 版实现中，因为结合使用了第 8 章的 EL 技术，样例中并没有使用 include 方法，只使用 forward 方法和 EL 就完成了 eg0306 类似的功能。

3.12 Servlet 的生命周期

在本章前面的练习中，已经使用了 Servlet，并且通过 Servlet 完成了一些基本的任务。那么，你是否想过这个问题：Servlet 并没有主方法，也从来没有见过用 new someServlet() 之类的代码去实例化一个对象出来，那么这个 Servlet 对象是由谁来实例化、初始化、使用和销毁的呢？

Servlet 是运行在服务器端的一种组件，它由 Servlet 容器来实例化、初始化、使用和销毁。每当有用户对该 Servlet 发出请求时，容器就实例化一个对象，然后启动一个线程为该用户服务，当另外有用户请求同一个 Servlet 时，就再启动另一个线程提供服务，所以有时候会说 Servlet 是“单实例，多线程”的。因为实例化以及与线程相关的任务都由 Servlet 容器来完成，所以程序员就可以专注于其他的业务逻辑部分了。回想一下前面的操作，是不是只要在 processRequest 方法中编程就可以了？

那么容器又是如何实例化、初始化、使用和销毁 Servlet 对象的呢？下面先看一个代码片段，在 NetBeans IDE 中打开 eg0306 这个工程，再打开 HandleLoginServlet.java 这个文件，可以看到这样的代码：

```
@WebServlet(name = "HandleLoginServlet", urlPatterns = {"/HandleLoginServlet"})
    public class HandleLoginServlet extends HttpServlet {
        //代码略
    }
```

（中间略去的代码是 processRequest、doGet、doPost 等几个方法，因为这一次重点不是这几个方法）。可以看到 HandleLoginServlet 是继承自 HttpServlet。按住 Ctrl 键，同时在 HttpServlet 上单击，可以看到 HttpServlet 的代码。

```
public abstract class HttpServlet extends GenericServlet {
    //代码略
}
```

可见 HttpServlet 又继承自 GenericServlet。按住 Ctrl 键，同时在 GenericServlet 上单击，还可以看到 GenericServlet 的代码。

```
public abstract class GenericServlet implements Servlet, ServletConfig, Serializable{
    //代码略
}
```

可见 GenericServlet 实现了 3 个接口。其中一个接口是 Servlet。按住 Ctrl 键,同时在 Servlet 上单击,可以看到 Servlet 的代码。

```
public interface Servlet {
    public void init(ServletConfig config) throws ServletException;
    public ServletConfig getServletConfig();
    public void service(ServletRequest req, ServletResponse res) throws ServletException,
IOException;
    public String getServletInfo();
    public void destroy();
}
```

在这段代码中就有答案了。Servlet 的生命周期分为四个阶段:实例化、初始化、对外服务、销毁。

(1) 实例化时调用的是 Servlet 的构造方法。

(2) 初始化时调用的是 init()方法,这个方法仅在 Servlet 首次载入时执行一次,并不是每次请求都要调用,这就是前面说的"单实例,多线程"。在 GenericServlet 中提供了两种初始化的方式:常规初始化和由初始化参数控制的初始化。

(3) 对外提供服务时调用的是 service()方法,这个方法在新线程中由服务器为每个请求而调用,但是并不直接对这个方法编程而是针对不同的请求方式调用相应的 doXxx()方法,例如前面提到过的 doGet()方法和 doPost()方法等。

(4) 销毁时调用 destroy()方法,该方法在服务器删除 Servlet 的实例时调用。并不是每次请求后都调用这个方法。

其实即使不知道这些,也不影响使用 Servlet 来编程,但是多了解一下也好,可以让我们对 Servlet 理解得更为透彻。下面通过一个例子来证明。

目标:理解 Servlet 的生命周期。

工程名:eg0307。

打开 NetBeans IDE,新建一个名为 eg0307 的 Java Web 项目,具体过程略;新建名为 NewServlet.java 的 Servlet,只设置包名为 cn.edu.djtu,其他设置取默认值。

在现有的 NewServlet.java 代码中,有四个方法 doGet、doPost、getServletInfo、processRequest,其中 doGet、doPost 均调用了 processRequest 方法,processRequest 方法是由 NetBeans 根据模板自动生成的。在 22 行代码的空白处,如图 3.18 所示,按快捷键"Ctrl+\"就可以根据提示选择要覆盖或者生成的方法。先生成一个构造方法 NewServlet(),然后再次在空白的地方按快捷键"Ctrl+\"——按 I 键,就可以看到提示覆盖 init 方法,init 方法有两个,一个是常规的初始化方法,一个是通过 ServletConfig 对象初始化的方法,选择无参的就好,如图 3.19 所示。

图 3.18 生成构造方法

图 3.19 覆盖 init 方法

按照类似的操作,可以覆盖 service()方法,service()方法也有两个,覆盖参数类型为 HttpServletRequest 和 HttpServletResponse 的就可以了,如图 3.20 所示。

图 3.20 覆盖 service()方法

最后用类似的操作覆盖 destroy()方法,图略。然后在每一个方法中添加一条控制台的输出语句,当该方法被调用时,会在控制台输出,这样就可以知道各个方法的执行顺序了。代码中省略了 doGet、doPost、getServletInfo 三个方法的代码,因为 doGet、doPost 方法调用的是 processRequest 方法,全部的代码如下:

NewServlet.java

```
@WebServlet(name = "NewServlet", urlPatterns = {"/NewServlet"})
public class NewServlet extends HttpServlet {
    public NewServlet() {
        System.out.println("调用构造方法");
    }
    @Override
    public void init() throws ServletException {
        System.out.println("调用 init 方法");
        super.init();
    }
    @Override
     protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("调用 service 方法");
        super.service(req, resp);
```

```
    }
    @Override
    public void destroy() {
        System.out.println("调用 destroy 方法");
        super.destroy();
    }
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            System.out.println("调用 processRequest 方法");
        }
    }
    //省略了 doGet、doPost、getServletInfo 三个方法的代码
    //因为 doGet、doPost 均调用 processRequest 方法
}
```

运行这个 Servlet,观察输出窗口的输出情况可以知道:先调用了构造方法,然后调用了初始化方法 init(),接着调用了提供服务的方法 service(),service()方法又调用了 processRequest 方法,但是没有看到"调用 destroy 方法"字样,说明 Servlet 对象目前还没有被销毁,如图 3.21 所示。

图 3.21 第 1 次访问时各方法执行情况

接着模拟另一个用户的访问,看看运行情况。另开一个浏览器窗口,将刚才访问的 URL"http://localhost:8080/eg0307/NewServlet"复制粘贴到地址栏并访问。运行结果见图 3.22。

可见又调用了一次 service()方法和 processRequest()方法,没有调用构造方法和初始化方法 init()。由此可见只创建了一个 Servlet 对象。这就证明了前面说过的:Servlet 是"单实例,多线程"的。也证明了 Servlet 的生命周期的阶段划分。

在上面操作的基础上,再做一点修订,将 service()方法中的 super.service(req, resp);语句注释掉然后保存,代码如下:

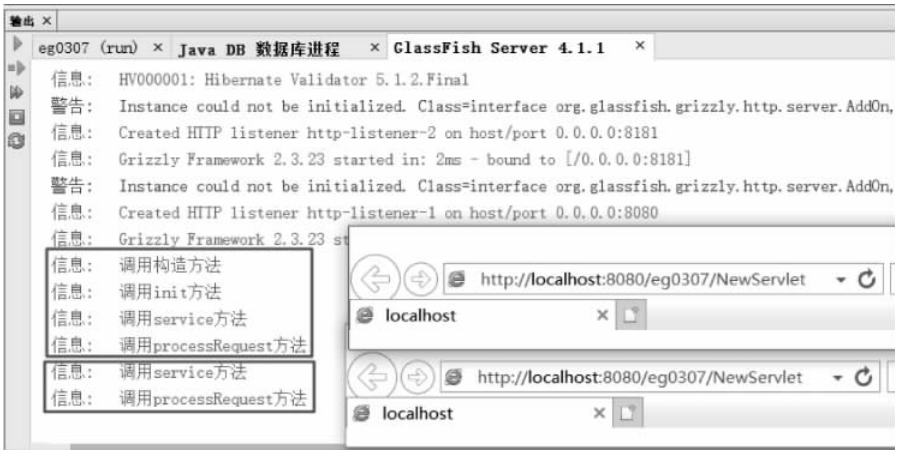

图 3.22 第 2 次访问时各方法执行情况

```
@Override
protected void service ( HttpServletRequest req, HttpServletResponse resp ) throws
ServletException, IOException {
    System.out.println("调用 service 方法");
    //super.service(req, resp);
}
```

由于代码改变了,所以 GlassFish Server 重新进行热部署,此时当前 Servlet 对象需要销毁,所以控制台能看到“调用 destroy 方法”字样,说明调用了 destroy 方法,如图 3.23 所示。

图 3.23 destroy 方法调用情况

此时,再一次运行 NewServlet,这时候服务器会重新启动,重新部署,也就意味着会重新实例化对象,初始化对象,对外提供服务,等待销毁,也许大家会觉得和上次一样,但是请看运行情况,如图 3.24 所示。

覆盖 service()方法后的运行结果。可以看到,调用了构造方法,调用了初始化方法 init(),调用了 service()方法。但是,没有调用 processRequest()方法。为什么呢,因为将下面这条

图 3.24 覆盖 service 方法后的运行结果

语句注释掉了。

```
//super.service(req, resp);
```

在父类的方法中，service 会根据请求的类型，自动调用 doXxx 方法，而注释掉之后，service 就不会自动去调用相应的 doXxx 方法了，当然 processRequest()方法也就没有了执行的机会了。

通过以上的讲解，相信大家对 Servlet 的生命周期应该有了一个清晰的认识。那么接下来看一下 Servlet 的部署，看看容器是如何找到 Servlet 的。

3.13 Servlet 的部署

在早期的开发中，部署 Servlet 时是一定要用到 web.xml 文件的。web.xml 文件中含有 Servlet 的部署描述符，一共是两组标记：< servlet > </servlet >和< servlet-mapping > </servlet-mapping >，如图 3.25 所示。

以图 3.25 中程序为例，对于外部的用户而言，只需要知道访问的时候按< url-pattern > </url-pattern >中设置的“/NewServlet”这种形式就可以访问资源了，至于这个资源是什么，在哪里，其实是一无所知的。容器在接到这种请求的时候就在配置文件的< servlet-mapping ></servlet-mapping >中查找，看看这个< url-pattern >/NewServlet </url-pattern >对应的< servlet-name > </servlet-name >是什么，于是找到了< servlet-name > NewServlet </servlet-name >；接下来就在< servlet > </servlet >中查找形如“< servlet-name > NewServlet </servlet-name >”的标记，于是找到了< servlet-class > cn.edu.djtu.servlet.NewServlet </servlet-class >，然后将其实例化。

为了进一步理解 Servlet 的部署，可以分别做如下几个尝试以作对比：

(1) 将< url-pattern >/NewServlet </url-pattern >改为

< url-pattern >/test.html </url-pattern >，然后运行 index.jsp 页面，在运行出来的浏览

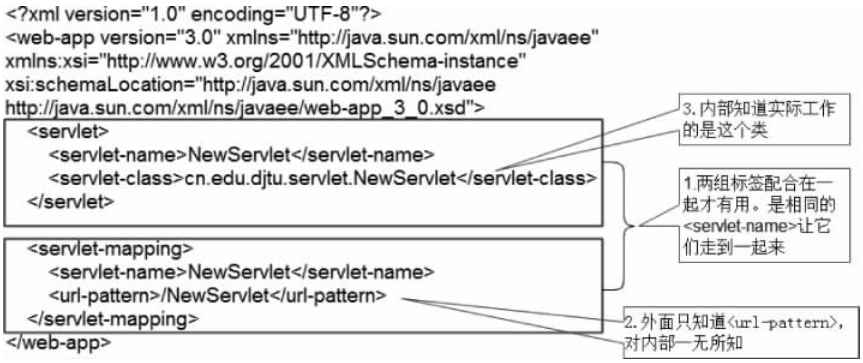

图 3.25 web.xml 中的 Servlet 配置

器地址栏中将 index.jsp 改成 test.html,会发现实际访问的是 NewServlet。

(2) 将两个< servlet-name > NewServlet </servlet-name >都改为

< servlet-name > test </servlet-name >,然后运行 NewServlet,仍然能运行,说明这个值也是可以单独配置的。

从维护的角度看,通常会取默认值,不做修订,有特殊需要时,再个别设置某个 Servlet 就行了。

在新的 Servlet 规范中,已经建议使用 Annotation(注解),而不是配置文件来完成 Servlet 部署工作。形式如下:

```
@WebServlet(name = "NewServlet", urlPatterns = {"/NewServlet"})
public class NewServlet extends HttpServlet {
    //代码略
}
```

不难理解其中 name = "NewServlet"作用等价于< servlet-name > test </servlet-name >; urlPatterns = {"/NewServlet" }作用等价于< url-pattern > /NewServlet </url-pattern >。

其实无论哪一种部署形式,都能顺利访问到 Servlet。对于 Servlet 的部署,由于篇幅所限,还有很多细节没有阐明,例如:一个 Servlet 可不可以有多个 urlPattern 呢? 可不可以在应用一启动时就实例化一个 Servlet 呢? 等等。如果想要深入了解,请查阅其他材料,对于大多数开发而言,部署形式取默认值就可以了。

3.14 本章回顾

本章简要介绍了 HTTP/HTTPS 的知识及其运行模型;介绍了 Servlet 的定义和作用;通过示例讲解了 Servlet 中需要覆写的方法;讲解了如何生成服务器响应;如何读取请求报头信息;如何读取用户提交的信息;如何实现资源间的跳转。最后讲解了 Servlet 的生命周期,如图 3.26 所示。

图 3.26 第 3 章内容结构图

3.15 课后习题

1. HTTP/HTTPS 的工作模型。
2. Servlet 的主要作用是什么？
3. Servlet 中一般需要覆写哪些方法？
4. 在 NetBeans IDE 中编写 Servlet 需要覆写哪个方法？
5. 如何设置响应类型？
6. 什么是 MIME？
7. 请求对象与获取请求报头相关的方法有哪些？
8. 读取用户提交的信息，一般要用到请求对象的哪些方法？对应何种场景？
9. 如何实现资源间的跳转？有几种方式？分别是什么？
10. 使用响应对象实现资源间的跳转，需要用到哪些/哪个方法？
11. 使用请求转发器对象实现资源间的跳转，需要用到哪些/哪个方法？
12. 比较响应的重定向与请求转发器的转发之间的不同。
13. 请求转发器的 forward 方法与 include 有什么区别与联系？