第 5 章 Servlet 技术深入剖析

Servlet 是一个技术体系和规范，它提供了一组类、接口和协议，用于满足 Web 服务器所需功能。了解这些类和接口，有助于对 Servlet 技术的全面认识。在本章中，重点介绍过滤器及监听器技术，并通过实例让读者对这些常用类和接口的实际应用有全面了解。

5.1 Servlet 技术体系

前面介绍了 Servlet 技术的基础知识。为了全面了解 Servlet 技术体系，本节介绍 Servlet 的常用类和接口的用法。Servlet 的常用类和接口如图 5-1 所示。

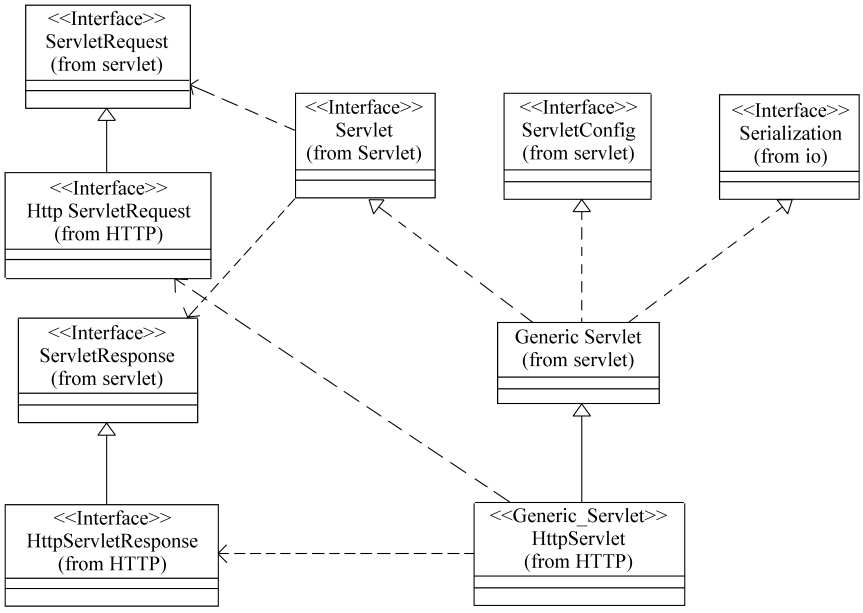

图 5-1 Servlet 的常用类和接口

5.1.1 常用类和接口

1. 与 Servlet 实现相关的类和接口

public interface Servlet 是所有 Servlet 必须直接或间接实现的接口。它定义了以下方法：

- init(ServletConfig config)方法：用于初始化 Servlet。
- destory()方法：用于销毁 Servlet。
- getServletInfo()方法：用于获得 Servlet 的信息。
- getServletConfig()方法：用于获得 Servlet 的配置信息。
- service(ServletRequest req, ServletResponse res)方法：是应用程序运行的逻辑入口点。

public abstract class GenericServlet 类提供了 Servlet 接口的基本实现。它是一个抽象类。GenericServlet 类的 service()方法是一个抽象的方法，GenericServlet 类的派生类必须直接或间接实现这个方法。

public abstract class HttpServlet 类是针对使用 HTTP 的 Web 服务器的 Servlet 类。

HttpServlet 类实现了抽象类 GenericServlet 的 service()方法，该方法的功能是根据请求类型调用合适的 do 方法。do 方法是由用户定义的 Servlet 根据特定的请求/响应情况具体实现的。也就是说，必须实现以下方法之一：

- doGet()。如果 Servlet 支持 HTTP GET 请求，则要实现该方法。
- doPost()。如果 Servlet 支持 HTTP POST 请求，则要实现该方法。
- 其他 do 方法。用于 HTTP 其他方式的请求。

2. 与请求、响应会话跟踪和 Servlet 上下文相关的接口

public interface HttpServletRequest 接口中最常用的方法就是获得请求中的参数。实际上，内置对象 request 就是实现该接口的类的一个实例，关于该接口的方法和功能在前面已讲述了，这里不再重复。

public interface HttpServletResponse 接口代表了对客户端的 HTTP 响应。实际上，内置对象 response 就是实现该接口的类的一个实例，关于该接口的方法和功能在前面已讲述了，这里不再重复。

会话跟踪接口(HttpSession)、Servlet 上下文接口(ServletContext)与 HttpServletRequest 接口类似，不再重复介绍。但有一点需要注意，JSP 与 Servlet 的内置对象相似，但二者获取内置对象的方法略有不同 JSP 与 Servlet 的比较如表 5-1 所示。

表 5-1　JSP 与 Servlet 的比较

技术	请求对象	响应对象	会话跟踪	上下文内容对象
JSP	request 由容器产生，直接使用	response 由容器产生，直接使用	session 由容器产生，直接使用	application 由容器产生，直接使用
Servlet	request 由容器产生，直接使用	response 由容器产生，直接使用	HttpSession session = request.getSession()	用 getServletContext() 方法获取

3. RequestDispatcher 接口

RequestDispatcher 接口代表 Servlet 协作，在前面已用到。它可以把一个请求转发到另一个 Servlet 或 JSP 页面。该接口主要有两个方法：

- forward(ServletRequest，ServletResponse response)方法。用于把请求转发到服务器上的另一个资源。
- include(ServletRequest，ServletResponse response)方法。用于把服务器上的另一个资源包含到响应中。

RequestDispatcher 接口的 forward()方法处理请求转发，在 Servlet 中是一个很有用的功能。由于这种请求转发属于 request 对象的范围，所以，应用程序往往用这种方法从 Servlet 向 JSP 页面或另一 Servlet 传输程序数据。其核心代码格式如下：

```
request.setAttribute("key", 任意对象数据);
RequestDispatcher dispatcher=null;
dispatcher= getServletContext().getRequestDispatcher("目的 JSP 页面或另一个
Servlet");
dispatcher.forward(request, response);
```

在以上代码中，RequestDispatcher 的实例化由上下文的.getRequestDispatcher()方法实现，在目的 JSP 页面或另一个 Servlet 中，用户程序可以用 request.setAttribute("key")方法获取传递的数据。另外，需要注意的是，利用 RequestDispatcher 接口的 forward()方法处理请求转发，其作用类似于 JSP 中的<jsp:forward>动作标签，属于服务器内部跳转。实际上，JSP 中的<jsp:forward>动作标签的底层实现就利用了 RequestDispatcher 技术。

4. Filter、FilterChain、FilterConfig 等过滤器接口

Filter、FilterChain、FilterConfig 等过滤器接口在 Web 应用中是比较有用的技术。例如，通过过滤器接口，可以完成统一编码(中文处理技术)、安全认证等工作。

5.1.2　全面了解 Servlet 配置

配置 Servlet 的目的除了前面所述的以外，还可以通过<init-param>、<load-on-startup>等元素的设置使容器(Tomcat)能够根据配置规则管理 Servlet，从而实现一些特定的目标。Servlet 配置工作可在 web.xml 中完成。

1. < servlet>元素及其子元素

在 web.xml 文件中，要注意各元素的顺序。<servlet>元素放在<servlet-mapping>元素之前，否则会导致 web.xml 移植困难。

以下为<servlet>元素及其子元素：

```
<servlet>
    <description>描述内容</description>
    <display-name>显示名</display-name>
    <servlet-name>Servlet 名</servlet-name>
    <servlet-class>Servlet 类名</servlet-class>
    <jsp-file>JSP 文件名</jsp-file>
    <init-param>
        <param-name>参数名</param-name>
        <param-value>参数值</param-value>
    </init-param>
    <load-on-startup>一个整型值</load-on-startup>
    <security-role-ref>
        <role-name>角色名</role-name>
        <role-link>角色的一个引用</role-link>
    </security-role-ref>
</servlet>
```

以上元素中，<description>、<display-name>和<security-role-ref>元素可以有 0 个或多个，<init-param>和<load-on-startup>元素可以有 0 个或 1 个。

各元素的作用及含义说明如下：

<description>元素为 Servlet 指定一个文本描述，无多大实际意义和作用。

<display-name>元素为 Servlet 指定一个简短的名字，这个名字可以被某些工具显示，无多大实际意义和作用。

<servlet-name>元素指定 Servlet 名，其作用相当于 Servlet 类的实例名，在程序中必须是唯一的。

<servlet-class>元素指定 Servlet 类名，且是完整限定名称，即包括包名(路径)。

<jsp-file>元素指定一个 JSP 文件名，且是完整限定名称(包括相对或绝对路径)。

<init-param>元素用来定义初始化参数，可以有多个<init-param>元素。在 Servlet 的类中通过 getInitParamenter(String name)方法访问初始化参数，这类似于 JSP 中的动作元素标签<jsp:param>。

<load-on-startup>元素指定当 Web 应用启动时装载 Servlet 的次序。当值为正数或零时，Servlet 容器先加载数值小的 Servlet，再依次加载其他数值大的 Servlet；当值为负或未定义时，Servlet 容器将延迟装载时间，也就是在 Web 客户端首次访问某个 Servlet 时才加载它。

<security-role-ref>元素用于声明在组件或部署组件的代码中安全角色的引用。

2. <servlet-mapping>元素及其子元素

<servlet-mapping>元素相对简单,只有两个子元素,如下所示:

```
<servlet-mapping>
    <servlet-name>Servlet 名</servlet-name>
    <url-pattern>URL 路径</url-pattern>
</servlet-mapping>
```

其中,<servlet-name>元素与前述一致。<url-pattern>元素在前面也已使用过,但该元素还有其他的作用。它主要通过通配符配置 URL 映射,对多个匹配的 URL 进行响应,而 JSP 页面只能通过一个具体的 URL 调用。这个特性可以使用户程序在请求进入某个具体的页面前被截获和处理。许多 Web 应用框架(如 Struts、Spring)都利用了 Servlet 的这个特性,并在此基础上创建构架。

3. 利用注解替换配置文件

其实,以上在 web.xml 中的所有配置内容都可以利用@WebServlet 注解的相关属性实现相同的功能。

5.2 过滤器技术

5.2.1 基本概念

Servlet 过滤器在 Java Servlet 2.3 规范中定义。

过滤器是小型的 Web 组件,若服务器(如 Tomcat)中部署了过滤器,则对于从客户端发送过来的请求,服务器首先让过滤器执行,这可能是安全、权限检查,也可能是字符集统一过滤处理;然后,或者让客户请求的目的页面或 Servlet 处理,或者直接进行页面转发(假如安全检查没有通过)。如果系统中设置了多个过滤器(一般情况下,一个过滤器完成一项特定任务),则一组过滤器会形成一个过滤链,客户请求会在过滤链中逐步过滤执行。

Java 中的过滤器并不是一个标准的 Servlet,它不能处理用户请求,也不能对客户端生成响应。它主要用于对 HttpServletRequest 进行预处理,也可以对 HttpServletResponse 进行后处理,是一个典型的处理链。

过滤器有如下几个用处:

- 在 HttpServletRequest 到达 Servlet 之前对其进行拦截。
- 根据需要检查 HttpServletRequest,可修改 HttpServletRequest 头和数据。
- 在 HttpServletResponse 到达客户端之前对其进行拦截。
- 根据需要检查 HttpServletResponse,可修改 HttpServletResponse 头和数据。

过滤器有如下几类：

- 用户授权的过滤器。负责检查用户请求，根据请求过滤用户的非法请求。
- 日志过滤器。详细记录某些特殊的用户请求。
- 负责解码的过滤器。包括对非标准编码的请求解码。

5.2.2 过滤器的主要方法、生命周期与部署

所有实现了 javax.servlet.Filter 接口的类都被称为过滤器类。这个接口含有 3 个过滤器类必须实现的方法：

(1) init(FileterConfig fileterconfig)：由 Servlet 容器调用，是 Servlet 过滤器的初始化方法，Servlet 容器创建 Servlet 过滤器实例后将立即调用这个方法，且该方法只被调用一次。在这个方法中可以读取 web.xml 文件中的 Servelt 过滤器的初始化参数。

(2) doFilter(ServletRequest request，ServletResponse response，FilterChain chain)：这个方法完成实际的过滤操作。当客户请求访问与过滤器相关联的 URL 时，Servlet 容器将首先调用过滤器的 doFilter()方法。FilterChain 参数用于访问后续过滤器，该参数中的 doFilter(Servletrequest request，Servletresponse response)方法真正决定了是否要继续访问后续的过滤器。

(3) destroy()：Servlet 容器在销毁过滤器实例前调用这个方法，在这个方法中可以释放 Servlet 过滤器占用的资源。

过滤器的生命周期如下：

(1) 启动服务器时加载过滤器的实例，并调用 init()方法初始化实例。

(2) 每一次请求时都只调用 doFilter()方法进行处理。

(3) 停止服务器时调用 destroy()方法销毁实例。

过滤器编写完成之后，需要配置与部署。这个工作有两种方法。

1. 通过 web.xml 进行配置与部署

通过 web.xml 进行配置与部署是一种传统的做法，主要涉及两个元素：<filter>和<filter-mapping>。

<filter>元素格式如下：

```
<filter>
    <filter-name>过滤器名称</filter-name>
    <filter-class>过滤器对应的类</filter-class>
    <!--初始化参数-->
    <init-param>
        <param-name>参数名称 1</param-name>
        <param-value>参数值 1</param-value>
    </init-param>
```

```
    <init-param>
        <param-name>参数名称 2</param-name>
        <param-value>参数值 2</param-value>
    </init-param>
    …
</filter>
```

＜filter-mapping＞元素的作用就是确定过滤器与特定 URL 的关联。只有指定 Servlet 过滤器和特定的 URL 关联，当客户请求访问此 URL 时，才会触发过滤器工作。过滤器的关联方式有 3 种：与一个 URL 关联，与一个 URL 目录下的所有资源关联，与一个 Servlet 关联。＜filter-mapping＞元素格式如下：

(1) 与一个 URL 关联时：

```
<filter-mapping>
    <filter-name>过滤器名称</filter-name>
    <url-pattern>xxx.jsp(或者 xxx.html)</url-pattern>
</filter-mapping>
```

(2) 与一个 URL 目录下的所有资源关联时：

```
<filter-mapping>
    <filter-name>过滤器名称</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

(3) 与一个 Servlet 关联时：

```
<filter-mapping>
    <filter-name>过滤器名称</filter-name>
    <url-pattern>Servlet 名称</url-pattern>
</filter-mapping>
```

2. 通过注解进行配置与部署

从 Servlet 3.0 标准开始，过滤器也提供了注解方法，与 Servlet 的注解类似。例如：

```
@WebFilter(filterName="SecurityCheck",value="/securitycheck/*")
public class SecurityCheck implements Filter {…}
```

在实际开发中，常用到的过滤器有身份验证过滤器(authentication filter)、字符集转换过滤器(encoding filter)、加密过滤器(encryption filter)和图像转换过滤器(image conversion filter)等。

5.2.3 过滤链

doFilter()方法的参数 chain 是接口 FilterChain 的实例，由服务器生成。若项目中有多

个过滤器，EncodingFilter 负责设置编码，SecurityFilter 负责控制权限，服务器会按照项目中过滤器定义的先后顺序将过滤器组装成一条链，然后依次执行其中的 doFilter()方法。假设过滤器的定义顺序是 EncodingFilter 在前、SecurityFilter 在后，则其执行的顺序如下：

(1) 执行第一个过滤器的 doFilter()方法中的 chain.doFilter()之前的代码，例如进行统一编码：

```
request.setCharacterEncoding(newCharSet);
```

(2) 执行第二个过滤器的 doFilter()方法中的 chain.doFilter()之前的代码。

(3) 执行请求的资源(如 Servlet、JSP 等)。

(4) 执行第二个过滤器的 chain.doFilter()之后的代码。

(5) 执行第一个过滤器的 chain.doFilter()之后的代码，最后将响应返回客户端。

以上执行顺序可用图 5-2 表示。

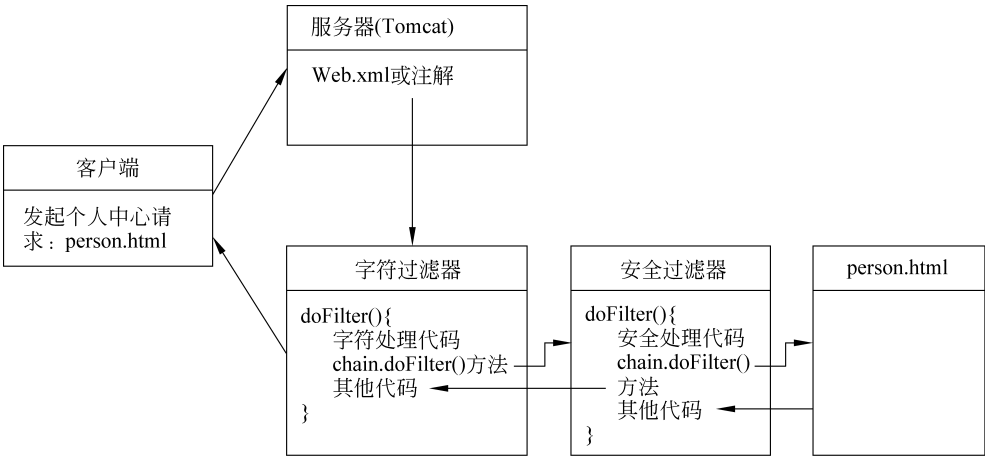

图 5-2 过滤器执行顺序

5.2.4 安全过滤器的开发

为了防止用户绕过登录(直接输入具体页面的 URL)或者登录失效时对页面进行非法操作，应对系统的所有请求进行安全过滤操作。前面章节的基本思路是对每个页面分别进行安全检查，这样显然有代码冗余。现在采用集中过滤检查方法，基本思路是把需要过滤的服务器资源集中放在一个目录中，如把个人中心、支付中心等页面放在 securitycheck/目录中，项目结构如图 5-3 所示。

当用户访问该目录下的资源时，则受到安全检查。过滤器核心代码如下：

```
package filter;
import javax.servlet.Filter;
```

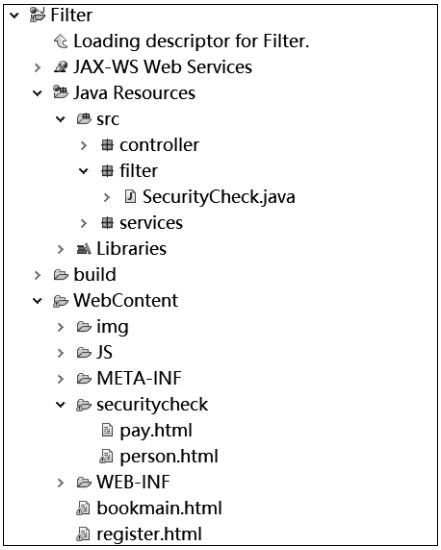

图 5-3　项目结构

```
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
@WebFilter(filterName="SecurityCheck",value="/securitycheck/*")
public class SecurityCheck implements Filter {
    …
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain
            chain) throws IOException, ServletException {
        HttpServletRequest request=(HttpServletRequest)req;
        HttpServletResponse response=(HttpServletResponse)res;
        HttpSession session=request.getSession();
        String username=(String)session.getAttribute("name");
        //条件成立时,不需要过滤,进入下一个过滤器或转到 Servlet
        if(username!=null) {
            //System.out.println("filter suc");
            chain.doFilter(req,res);
        }
        else response.sendRedirect("/Filter/bookmain.html");
            …
    }
}
```

代码说明：

HttpServletRequest 和 ServletRequest 都是接口，前者继承自后者，HttpServletRequest 比 ServletRequest 多了一些针对 HTTP 的方法，如 getHeader(String name)、getMethod()、getSession()、getRequestURI()等。如果业务需要用到 HttpServletRequest 的方法，则必须进行类型转换，代码如下：

```
HttpServletRequest request=(HttpServletRequest)req;
```

req 是 ServletRequest 类型的对象，这类似于以下情形：

```
session.setAttribute("name", "张三");
String name=(String)session.getAttibute("name");
```

过滤器配置采用注解方式：

```
@WebFilter(filterName="SecurityCheck",value="/securitycheck/*")
```

注解中的 filterName="SecurityCheck"相当于配置文件中的<filter>元素：

```
<filter>
    <filter-name>SecurityCheck</filter-name>
    <filter-class>filter.SecurityCheck</filter-class>
</filter>
```

注解中的 value="/securitycheck/*"相当于配置文件中的<filter-mapping>元素：

```
<filter-mapping>
    <filter-name>SecurityCheck</filter-name>
    <url-pattern>/securitycheck/*</url-pattern>
</filter-mapping>
```

5.3 监听器技术

5.3.1 基础知识

监听器(listener)用于监听 Java Web 程序中的各类事件，它是 Servlet 规范中的特殊类，也需要在 web.xml 文件中注册和配置才可以使用(除了两个例外)。它监听 ServletContext、HttpSession 和 ServletRequest 等对象的创建与销毁事件以及这些对象中属性发生改变的事件。当这些事件发生的时候，对应的监听器会立刻做出反应。目前一共有 8 个监听器接口和 6 个事件类别，如表 5-2 所示。