Web应用开发
之
高级应用

本章内容

- 8.1 Web监听器
- 8.2 Web过滤器
- 8.3 Servlet的多线程问题
- 8.4 Servlet的异步处理

本章内容

- Web应用程序运行过程中可能发生各种事件，如ServletContext事件、会话事件及请求有关的事件等，Web容器采用监听器模型处理这些事件。
- 过滤器用于拦截传入的请求或传出的响应，并监视、修改或以某种方式处理这些通过的数据流。
- 本章主要介绍这两个高级应用，它们是Web事件处理模型和Servlet过滤器。此外，本章还将讨论Servlet多线程问题以及Servlet 3.0的异步处理问题。

8.1 Web监听器

- Web应用程序中的事件主要发生在三个对象上：ServletContext、HttpSession和ServletRequest对象。
- 事件的类型主要包括对象的生命周期事件和属性改变事件。
- 例如，对于ServletContext对象，当它初始化和销毁时会发生ServletContextEvent事件，当在该对象上添加属性、删除属性或替换属性时会发生ServletContextAttributeEvent事件。
- 对于会话对象和请求对象也有类似的事件。为了处理这些事件，Servlet容器采用了监听器模型，即需要实现有关的监听器接口。

8.1 Web监听器

- 在Servlet 3.0 API中定义了7个事件类和9个监听器接口，根据监听器所监听事件的类型和范围，可以把它们分为三类：
  - ServletContext事件监听器
  - HttpSession事件监听器
  - ServletRequest事件监听器

8.1.1 监听ServletContext事件

- 在ServletContext对象上可能发生两种事件，对这些事件可使用两个事件监听器接口处理

监听对象	事件	监听器接口
ServletContext	ServletContextEvent	ServletContextListener
	ServletContextAttributeEvent	ServletContextAttributeListener

1. 处理ServletContextEvent事件

- 该事件是Web应用程序生命周期事件，当容器对ServletContext对象进行初始化或销毁操作时，将发生ServletContextEvent事件。

- 处理这类事件，需实现ServletContextListener接口，该接口定义如下两方法

①void contextInitialized (ServletContextEvent sce)
当ServletContext对象初始化时调用。

②void contextDestroyed (ServletContextEvent sce)
当ServletContext对象销毁时调用。

1.处理ServletContextEvent事件

- 上述方法的参数是一个ServletContextEvent事件类对象，该类只定义了一个方法，如下所示。

```
public ServletContext
            getServletContext()
```

- 该方法返回状态发生改变的ServletContext对象

2. 处理ServletContextAttributeEvent事件

- 当ServletContext对象上属性发生改变时，如添加属性、删除属性、替换属性等，将发生ServletContextAttributeEvent事件，处理该类事件，需要实现ServletContextAttributeListener接口。

2. 处理ServletContextAttributeEvent事件

- 该接口定义了如下三个方法。

① public void attributeAdded(ServletContextAttributeEvent sre)：当在ServletContext对象中添加属性时调用该方法。

② public void attributeRemoved(ServletContextAttributeEvent sre)：当从ServletContext对象中删除属性时调用该方法。

③ public void attributeReplaced(ServletContextAttributeEvent sre)：当在ServletContext对象中替换属性时调用该方法。

2. 处理ServletContextAttributeEvent事件

- 上述方法的参数是ServletContextAttributeEvent类的对象，它是ServletContextEvent类的子类，它定义了下面三个方法。

① public ServletContext getServletContext()：返回属性发生改变的ServletContext对象。

② public String getName()：返回发生改变的属性名。

③ public Object getValue()：返回发生改变的属性值对象。

- 注意，当替换属性时，该方法返回的是替换之前的属性值。

2. 处理ServletContextAttributeEvent事件

- 程序8.1 MyContextListener.java实现当Web应用启动时就创建一个数据源对象并将它保存在ServletContext对象上，当应用程序销毁时将数据源对象从ServletContext对象上清除，当ServletContext上属性发生改变时登记日志。
- 该程序在ServletContextListener接口的contextInitialized()中首先从InitialContext对象中查找数据源对象dataSource并将其存储在ServletContext对象中。
- 在ServletContext的属性修改方法中先通过事件对象的getServletContext()获得上下文对象，然后调用它的log()向日志中写一条消息。

2. 处理ServletContextAttributeEvent事件

- 程序8.2　listenerTest.jsp页面是对监听器的测试，这里使用了监听器对象创建的数据源对象。

- 在该页面中首先通过隐含对象application的getAttribute()得到数据源对象，然后创建ResultSet对象访问数据库。

8.1.2 监听请求事件

- 在ServletRequest对象上可能发生两种事件，对这些事件使用两个事件监听器处理

监听对象	事件	监听器接口
ServletRequest	**ServletRequestEvent**	**ServletRequestListener**
	ServletRequestAttributeEvent	**ServletRequestAttributeListener**

1. 处理ServletRequestEvent事件

- ServletRequestEvent是请求对象生命周期事件，当一个请求对象初始化或销毁时将发生该事件，处理该类事件需要使用ServletRequestListener接口如下两个方法：

- public void requestInitialized (ServletRequestEvent sce)

  当请求对象初始化时调用。

- public void requestDestroyed (ServletRequestEvent sce)

  当请求对象销毁时调用。

1. 处理ServletRequestEvent事件

- 上述方法的参数是ServletRequestEvent类对象，该类定义了下面两个方法：
- public ServletContext getServletContext()：返回发生该事件的ServletContext对象。
- public ServletRequest getServletRequest()：返回发生该事件的ServletRequest对象。

2. 处理ServletRequestAttributeEvent事件

- 在请求对象上添加、删除和替换属性时将发生ServletRequestAttributeEvent事件，处理该类事件需要使用ServletRequestAttributeListener接口，它定义了如下三个方法。
- public void attributeAdded(ServletRequestAttributeEvent src)：当在请求对象中添加属性时调用该方法。
- public void attributeRemoved (ServletRequestAttributeEvent src)：当从请求对象中删除属性时调用该方法。
- public void attributeReplaced (ServletRequestAttributeEvent src)：当在请求对象中替换属性时调用该方法。

2. 处理ServletRequestAttributeEvent事件

- 在上述方法中传递的参数为ServletRequestAttributeEvent类的对象，该类定义了下面两个方法。
- public String getName()：返回在请求对象上添加、删除或替换的属性名。
- public Object getValue()：返回在请求对象上添加、删除或替换的属性值。
- 注意，当替换属性时，该方法返回的是替换之前的属性值。

2. 处理ServletRequestAttributeEvent事件

- 下面的MyRequestListener监听器类监听对某个页面的请求并记录自应用程序启动以来被访问的次数。
- 程序8.3 MyRequestListener.java
- 测试JSP页面：程序8.4 onlineCount.jsp

8.1.3 监听会话事件

- 在HttpSession对象上可能发生两种事件，对这些事件可使用四个事件监听器处理

监听对象	事件	监听器接口
HttpSession	HttpSessionEvent	HttpSessionListener
		HttpSessionActivationListener
	HttpSessionBindingEvent	HttpSessionAttributeListener
		HttpSessionBindingListener

1. 处理HttpSessionEvent事件

- HttpSessionEvent事件是会话对象生命周期事件，当一个会话对象被创建和销毁时发生该事件，处理该事件应该使用HttpSessionListener接口，该接口定义了两个方法：
- public void sessionCreated(HttpSessionEvent se)：当会话创建时调用该方法。
- public void sessionDestroyed(HttpSessionEvent se)：当会话销毁时调用该方法。
- 上述方法的参数是一个HttpSessionEvent类对象，该类中只定义了一个getSession()，它返回状态发生改变的会话对象，格式如下。

  public HttpSession getSession()

2. 处理会话属性事件

- 当在会话对象上添加属性、删除属性、替换属性时将发生HttpSessionBindingEvent事件，处理该事件需使用HttpSessionAttributeListener接口，该接口定义了下面三个方法：
- public void attributeAdded(HttpSessionBindingEvent se)：当在会话对象上添加属性时调用该方法。
- public void attributeRemoved(HttpSessionBindingEvent se)：当从会话对象上删除属性时调用该方法。
- public void attributeReplaced(HttpSessionBindingEvent se)：当替换会话对象上的属性时调用该方法。
- 注意：上述方法的参数是HttpSessionBindingEvent，没有HttpSessionAttributeEvent这个类。

2. 处理会话属性事件

- HttpSessionBindingEvent类中定义了下面三个方法。
- public HttpSession getSession()：返回发生改变的会话对象。
- public String getName()：返回绑定到会话对象或从会话对象解除绑定的属性名。
- public Object getValue()：返回在会话对象上添加、删除或替换的属性值。

2. 处理会话属性事件

- 下面定义的监听器类实现了HttpSessionListener接口，它用来监视当前所有会话对象。当一个会话对象创建时，将其添加到一个ArrayList对象中并将其设置为ServletContext作用域的属性以便其他资源可以访问。当销毁一个会话对象时，从ArrayList中删除会话。
- 程序8.5 MySessionListener.java
- 程序8.6 sessionDisplay.jsp

3. 处理会话属性绑定事件

- 当一个对象绑定到会话对象或从会话对象中解除绑定时发生HttpSessionBindingEvent事件，应该使用HttpSessionBindingListener接口来处理这类事件，该接口定义的方法有：
- public void valueBound(HttpSessionBindingEvent event)：当对象绑定到一个会话上时调用该方法。
- public void valueUnbound (HttpSessionBindingEvent event)：当对象从一个会话上解除绑定时调用该方法。

3. 处理会话属性绑定事件

- 下面定义的User类实现了HttpSessionBindingListener接口。当将该类的一个对象绑定到会话对象上时，容器将调用valueBound()，当从会话对象上删除该类的对象时，容器将调用valueUnbound()，这里向日志文件写入有关信息。

  程序8.7 User.java

- 程序从HttpSessionBindingEvent对象中检索会话对象，从会话对象中得到ServletContext对象并使用log()登录消息。

3. 处理会话属性绑定事件

- 程序8.8 LoginServlet.java：接受登录用户的用户名和口令，然后创建一个User对象并将其绑定到会话对象上。

8.1.4 事件监听器的注册

- 从前面的例子中可看到，我们使用@WebListener注解来注册监听器，这是Servlet 3.0规范增加的功能。

- 事件监听器也可以在DD文件中使用<listener>元素注册。该元素只包含一个<listener-class>元素，用来指定实现了监听器接口的完整的类名。
- 下面代码给出了如何注册MyContextListener和MySessionListener两个监听器。

```
<listener>
        <listener-class>com.listener.MyContextListener</listener-class>
</listener>

<listener>
        <listener-class>com.listener.MySessionListener</listener-class>
</listener>
```

8.1.4　事件监听器的注册

- 在web.xml文件中并没有指定哪个监听器类处理哪个事件，这是因为当容器需要处理某种事件时，它能够找到有关的类和方法。
- 容器按照DD文件中指定的类的顺序将事件传递给监听器。这些类必须存放在WEB-INF\classes目录中或者与其他Servlet类一起打包在JAR文件中。
- 提示：可以在一个类中实现多个监听器接口。这样，在部署描述文件中就只需要一个<listener>元素。容器就仅创建该类的一个实例并把所有的事件都发送给该实例。

8.2 Web过滤器

- 8.2.1 什么是过滤器
- 8.2.2 过滤器API
- 8.2.3 一个简单的过滤器
- 8.2.4 @WebFilter注解
- 8.2.5 在DD中配置过滤器

8.2.1 什么是过滤器

- 过滤器（Filter）是Web服务器上的组件，它拦截客户对某个资源的请求和响应，对其进行过滤。
- 图8-3说明了过滤器的一般概念，其中F1是一个过滤器。它显示了请求经过滤器F1到达Servlet，Servlet产生响应再经过滤器F1到达客户。这样，过滤器就可以在请求和响应到达目的地之前对它们进行监视。

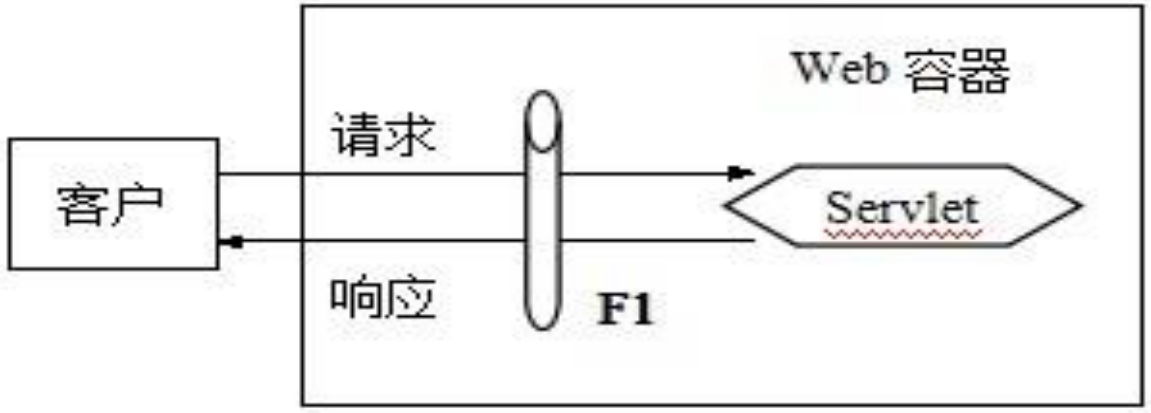


图 8-3 单个的过滤器

8.2.1 什么是过滤器

- 可以在客户和资源之间建立多个过滤器，从而形成过滤器链（filter chain）。在过滤器链中每个过滤器都对请求处理，然后将请求发送给链中的下一个过滤器。类似地，在响应到达客户之前，每个过滤器以相反的顺序对响应处理。
- 图8-4中，请求是按下列顺序处理的：过滤器F1、过滤器F2、过滤器F3，而响应的处理顺序是过滤器F3、过滤器F2、过滤器F1。

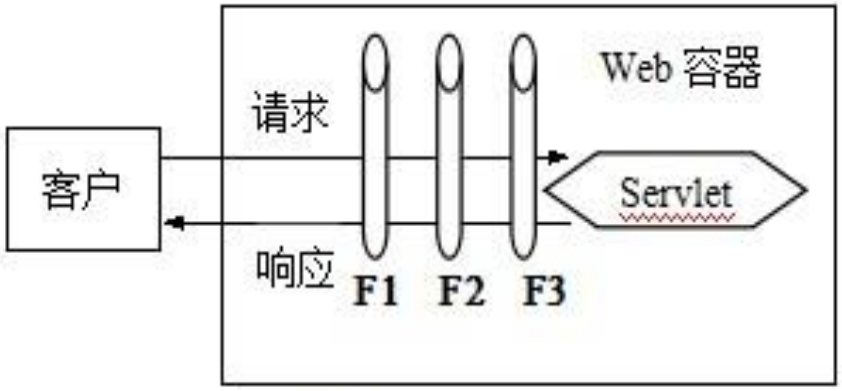
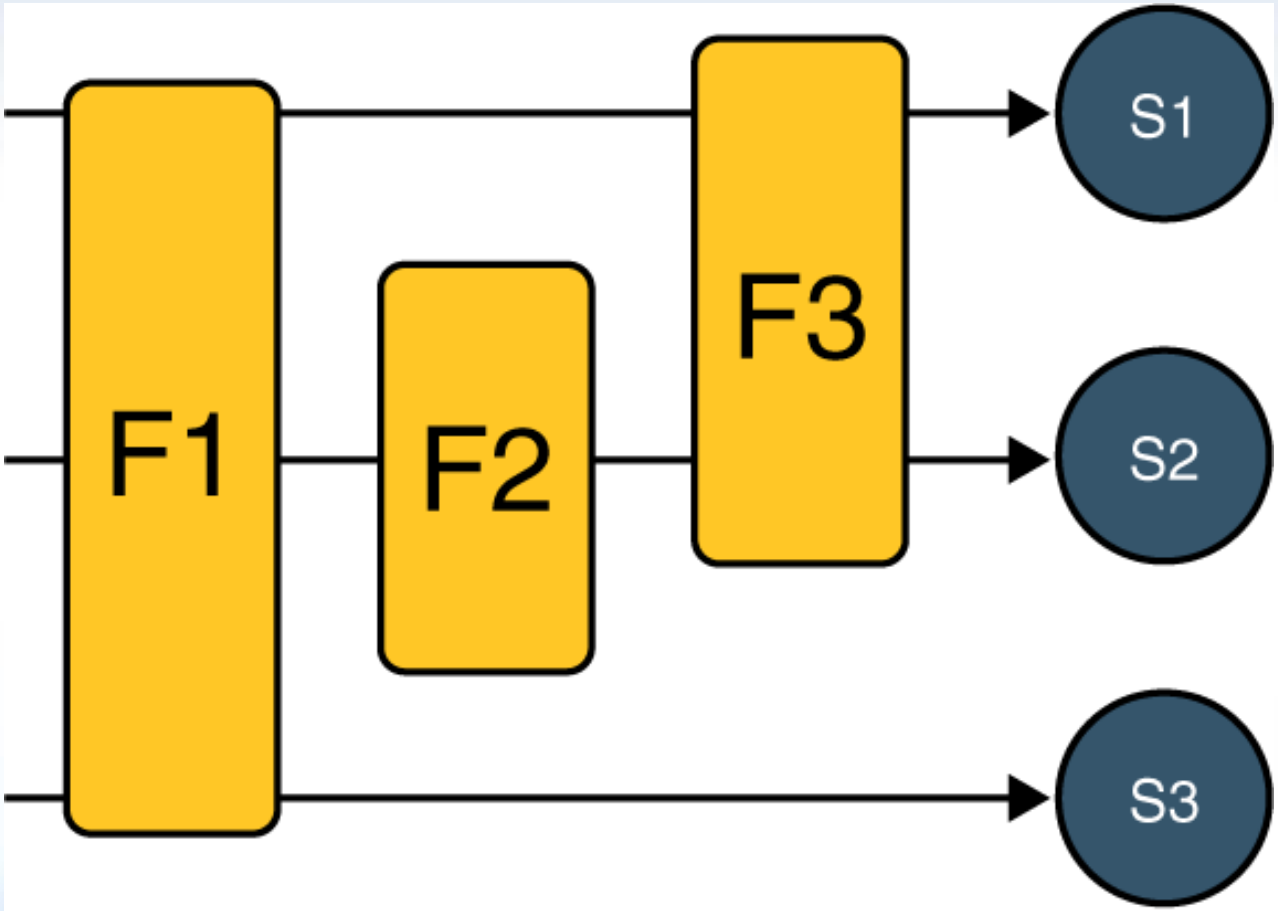


图 8-4 使用多个过滤器

Servlet的过滤器模型



1. 过滤器是如何工作的

- 当容器接收到对某个资源的请求时，它首先检查是否有过滤器与该资源关联。如果有过滤器与该资源关联，容器先把该请求发送给过滤器，而不是直接发送给资源。

- 在过滤器处理完请求后，它将做下面三件事：
  （1）将请求发送到目标资源。
  （2）如果有过滤器链，它将把请求（修改过或没有修改过）发送给下一个过滤器。
  （3）直接产生响应并将其返回给客户。
- 当请求返回到客户时，它将以相反的方向经过同一组过滤器。过滤器链中的每个过滤器都可能修改响应。

2. 过滤器的用途

- Servlet规范中提到的过滤器的一些常见应用包括：

  验证过滤器

  登录和审计过滤器

  数据压缩过滤器

  加密过滤器

  XSLT过滤器（eXtensible Stylesheet Language Transformation，可扩展样式表语言转换）

8.2.2 过滤器API

- 过滤器的类和接口定义在javax.servlet和javax.servlet.http包中。

- javax.servlet包中的3个接口
  - Filter：所有的过滤器都需要实现该接口
  - FilterConfig：过滤器配置对象。容器提供了该对象，其中包含了该过滤器的初始化参数
  - FilterChain：过滤器链对象

1. Filter接口

- Filter接口是过滤器API的核心，所有的过滤器都必须实现该接口。该接口声明了三个方法，分别是init()、doFilter()和destroy()，它们是过滤器的生命周期方法。
- init()是过滤器初始化方法。在过滤器的生命周期中，init()仅被调用一次。在该方法结束之前，容器并不向过滤器转发请求。该方法的声明格式为：

```
public void init(FilterConfig filterConfig)
```

1. Filter接口

- doFilter()是实现过滤的方法。如果客户请求的资源与该过滤器关联，容器将调用该方法，格式如下：

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

- 该方法执行过滤功能，对请求进行处理或者将请求转发到下一个组件或者直接向客户返回响应。注意，request和response参数被分别声明为ServletRequest和ServletResponse的类型。

1. Filter接口

- destroy()是容器在过滤器对象上调用的最后一个方法，声明格式为：

```
public void destroy();
```

- 该方法给过滤器对象一个释放其所获得资源的机会，在结束服务之前执行一些清理工作。

2. FilterConfig接口

- FilterConfig对象是过滤器配置对象，通过该对象可以获得过滤器名、过滤器运行的上下文对象以及过滤器的初始化参数。它声明了如下4个方法：
- public String getFilterName()
- public ServletContext getServletContext()
- public String getInitParameter(String name)
- public Enumeration getInitParameterNames()
- 容器提供了FilterConfig接口的一个具体实现类，容器创建该类的一个实例、使用初始化参数值对它初始化，然后将它作为一个参数传递给过滤器的init()。

3. FilterChain接口

- FilterChain接口只有一个方法，如下所示。

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

- 在Filter对象的doFilter()中调用该方法使过滤器继续执行，它将控制转到过滤器链的下一个过滤器或实际的资源。
- 容器提供了该接口的一个实现并将它的一个实例作为参数传递给Filter接口的doFilter()。在doFilter()内，可以使用该接口将请求传递给链中的下一个组件，它可能是另一个过滤器或实际的资源。该方法的两个参数将被链中下一个过滤器的doFilter()或Servlet的service()接收。

8.2.3　一个简单的过滤器

- 程序8.9 LogFilter.java是一个简单的日志过滤器，这个过滤器拦截所有的请求并将请求有关信息记录到日志文件中。程序声明的LogFilter类实现了Filter接口，覆盖了其中的init()、doFilter()和destroy()。

- 程序在doFilter()中首先将请求对象（request）转换成HttpServletRequest的类型，然后获得当前时间、客户请求的URI和客户地址，并将其写到日志文件中。之后将请求转发到资源，当请求返回到过滤器后再得到当前时间，计算请求资源的时间并写到日志文件中。

8.2.4 @WebFilter注解

- @WebFilter注解用于将一个类声明为过滤器，该注解在部署时被容器处理，容器根据具体的配置将相应的类部署为过滤器。表8-5给出该注解包含的常用元素。

8.2.4 @WebFilter注解

表 8-5 @WebFilter 注解的常用元素

元素名	类 型	说 明
filterName	String	指定过滤器的名称，等价于 web.xml 中的<filter-name>元素。如果没有显式指定，则使用 Filter 的完全限定名作为名称
urlPatterns	String[]	指定一组过滤器的 URL 匹配模式，该元素等价于 web.xml 文件中的<url-pattern>元素
value	String[]	该元素等价于 urlPatterns 元素。两个元素不能同时使用
servletNames	String[]	指定过滤器应用于哪些 Servlet。取值是@WebServlet 中 name 属性值，或者是 web.xml 中<servlet-name>的取值
dispatcherTypes	DispatcherType	指定过滤器的转发类型。具体取值包括：ASYNC、ERROR、FORWARD、INCLUDE 和 REQUEST
initParams	WebInitParam[]	指定一组过滤器初始化参数，等价于<init-param>元素
asyncSupported	boolean	声明过滤器是否支持异步调用，等价于<async-supported>元素
description	String	指定该过滤器的描述信息，等价于<description>元素
dispalyName	String	指定该过滤器的显示名称，等价于<display-name>元素

8.2.4 @WebFilter注解

- @WebFilter中所有属性均为可选属性，但是 value、urlPatterns、servletNames 三者必须至少包含一个，且 value 和 urlPatterns 不能共存，如果同时指定，通常忽略 value 的取值。
- 过滤器接口Filter与Servlet非常相似，它们具有类似的生命周期行为，区别只是Filter的doFilter()中多了一个FilterChain的参数，通过该参数可以控制是否放行用户请求。
- 像Servlet一样，Filter也可以具有初始化参数，这些参数可以通过@WebFilter注解或部署描述文件定义。在过滤器中获得初始化参数使用FilterConfig实例的getInitParameter()。

8.2.4 @WebFilter注解

- 在实际应用中，使用Filter可以更好实现代码复用。

① 一个系统可能包含多个Servlet，这些Servlet都需要进行一些通用处理，比如权限控制、记录日志等，这将导致多个Servlet的service()中包含部分相同代码。

② 为解决这种代码重复问题，就可以考虑把这些通用处理提取到Filter中完成，这样在Servlet中就只剩下针对特定请求相关的处理代码。

8.2.4 @WebFilter注解

- 程序8.10 AuthorityFilter.java定义一个较为实用的Filter

① 对用户请求进行过滤，为请求设置编码字符集，从而可以避免为每个JSP页面、Servlet都设置字符集

② 实现验证用户是否登录，如果用户没有登录，系统直接跳转到登录页面。

8.2.4 @WebFilter注解

- 该过滤器通过@WebFilter注解的initParams元素指定了三个初始化参数，参数使用@WebInitParam注解指定，每个@WebInitParam指定一个初始化参数。在Filter的doFilter()中通过FilterConfig对象取出参数的值。
- 程序中设置了请求的字符编码，还通过Session对象验证用户是否登录，若没登录将请求直接转发到登录页面，若已登录则转发到请求的资源。