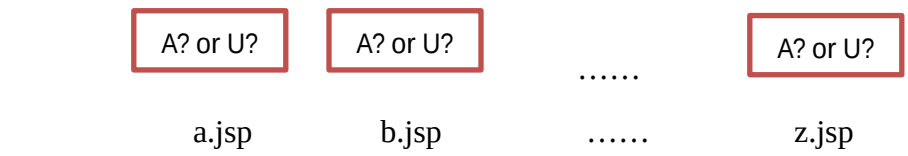
第 10 章 过滤器

10.1 引入过滤器原因

【引】飞机安检：喝水，北京天安门安检：喝水

【引】学校（高中）门口门卫：检查证件，检查带吃喝的没



1.情况一、为了解决中文乱码问题。

response.setContentType(“text/html;charset=utf-8”);
request.setCharacterEncoding(“utf-8”);
如果在 Servlet 的处理方法最前面没加入这两段代码，页面上就很有可能出现中文乱码问题。如果每个 Servlet 都添加这段代码，这些代码重复设置就会出现冗余。另外，一旦需求变了，要求换成另外的编码（utf-8 换成 gbk 或 gb2312 或 ISO-8859-1），对于编程人员来说就是一个天大的灾难。

2.情况二、解决 session 验证用户是否登录问题。

如果客户没有登录就访问网站的某一受限页面，在很多情况下会发生安全问题。可以使用 session 检查来完成。但在很多页面都添加 session 检查代码，会比较繁琐。

3.情况三、解决用户权限问题。

假设网站分为管理员和普通用户两种权限，登录过后，网页如何区分普通用户和管理员，如果每个页面都写一个判断用户类型的代码，也会比较繁琐。

10.2 创建过滤器

1.新建过滤器所在包和类名

（1）包：Java package:com.yp.filter

（2）类名：Class name:CharacterEncodingFilter

2.设置过滤器名、初始化参数和 URL 映射

（1）过滤器名：Name:CEF

（2）初始化参数：Initialization parameters:encoding == utf-8

（3）URL 映射：URL Pattern:/*

3.指定过滤器实现的接口和方法（直接跳过）

10.3 认识过滤器代码

1. 实现接口

Javax.servlet.Filter;

2. 实现方法

（1）初始化方法：执行过滤器初始化的动作

public void init(FilterConfig fConfig) throws ServletException {}

（2）销毁方法：执行过滤器销毁时的动作

public void destroy() {}

（3）过滤方法：执行过滤器过滤时的动作

public void doFilter(ServletRequest request,

ServletResponse response,

FilterChain chain)

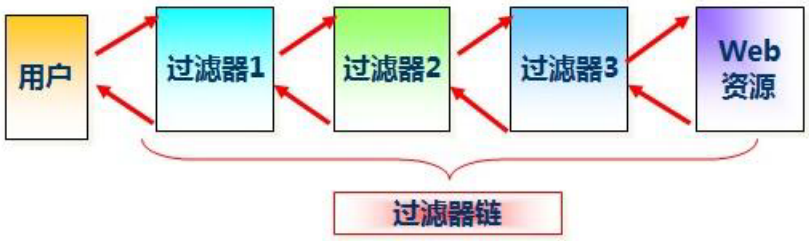
throws IOException, ServletException {chain.doFilter(request, response);}

关于 chain.doFilter(request,response)

它的作用是将请求转发给过滤器链上的下一个对象。这里的下一个对象指的是下一个 filter，如果没有 filter 那就是你请求的资源。一般 filter 都是一个链,web.xml 里面配置了几个就有几个。一个一个的连在一起 request -> filter1 -> filter2 ->filter3 -> .... -> request resource。

请求 -> 滤网 1 -> 滤网 2 -> 滤网 3 -> .... -> 请求资源

请求 -> 防火墙 1 -> 防火墙 2 -> 防火墙 3 -> .... -> 请求资源



3.FilterChain 接口

FilterChain 接口中也有一个 doFilter 方法，即 doFilter (ServletRequest request, ServletResponse response)。此方法是由 Servlet 容器提供给开发者的，用于对资源请求过滤链的依次调用，通过 FilterChain 调用过滤链中的下一个过滤器，如果是最后一个过滤器，则下一个就调用目标资源。

4.FilterConfig 接口

FilterConfig 接口可获取过滤器名、初始化参数以及活动的 Servlet 上下文：

（1）String getFilterName()

返回 web.xml 文件中定义的该过滤器的名称。

（2）ServletContext getServletContext()

返回调用者所处的 Servlet 上下文。

（3）String getInitParameter(String name)

返回过滤器初始化参数值的字符串形式，当参数不存在时，返回 null。

（4）public Enumeration getInitParameterNames()：

以 Enumeration 形式返回过滤器所有初始化参数值，如果没有初始化参数，返回 null。

10.4 认识配置文件代码

1. 配置文件中过滤器代码结构

```
<filter>
    <filter-name>CEF</filter-name>
    <filter-class>com.yp.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>
```

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

2.解释说明各标签意义

（1）<filter>：定义过滤器

（2）<filter-name>CEF</filter-name>：定义过滤器名字

（3）<filter-class>com.yp.filter.CharacterEncodingFilter</filter-class>定义过滤器的类路径

（4）<init-param>：定义初始化参数

（5）<param-name>encoding</param-name>：设置初始化参数名称

（6）<param-value>utf-8</param-value>：设置初始化参数值

（7）<filter-mapping>：配置过滤器的路径映射

（8）<url-pattern>/*</url-pattern>：设置过滤模式

3.过滤模式说明：

<url-pattern>标记中的过滤模式可分为 3 类：

（1）过滤所有文件（即所有请求都被过滤）

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

说明：/表示虚拟目录根目录

（2）过滤一个或多个 Servlet（JSP）

```
<filter-mapping>
    <filter-name>CEF</filter-name>
    <url-pattern>/PATH1/SERVLETNAME1(JSP1)</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>CEF</filter-name>
```

```
    <url-pattern>/PATH2/SERVLETNAME2(JSP2)</url-pattern>
</filter-mapping>
```

（3）过滤一个或多个文件目录

```
<filter-mapping>
    <filter-name>CEF</filter-name>
    <url-pattern>/PATH1/*</url-pattern>
</filter-mapping>
```

说明：对 PATH1 目录下的所有资源进行过滤

10.5 简单实例

1.实例：处理中文乱码。

2.实例：过滤非法 IP 地址。

（1）配置文件代码

```
<filter>
    <display-name>FilterIP</display-name>
    <filter-name>FilterIP</filter-name>
    <filter-class>com.yp.filter.FilterIP</filter-class>
    <init-param>
      <description></description>
      <param-name>filterIP</param-name>
      <param-value>0:0:0:0:0:0:0:1</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>FilterIP</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

（2）过滤器代码

```
package com.yp.filter;
```

```
public class FilterIP implements Filter {
    protected FilterConfig config;
    protected String filterIP = "";
    public FilterIP() {}
    public void destroy() {}
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        String userIP = request.getRemoteAddr();
        RequestDispatcher dis = request.getRequestDispatcher("IP_error.jsp");
        if (userIP.equals(this.filterIP)) {
            dis.forward(request, response);
        } else {
            chain.doFilter(request, response);
        }
    }
    public void init(FilterConfig fConfig) throws ServletException {
        this.config = fConfig;
        this.filterIP = fConfig.getInitParameter("filterIP");
    }
}
```

3.实例：计算请求页面处理时间。

（1）配置文件代码

```
<filter>
    <display-name>ResponseTimeFilter</display-name>
    <filter-name>ResponseTimeFilter</filter-name>
    <filter-class>com.yp.filter.ResponseTimeFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>ResponseTimeFilter</filter-name>
```

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

（2）过滤器代码

```
public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        long startTime = System.currentTimeMillis();
        chain.doFilter(request, response);
        long passedTime = System.currentTimeMillis() - startTime;
        String resourceName = "default";
        if (request instanceof HttpServletRequest) {
            resourceName = ((HttpServletRequest) request).getRequestURI();
        }
        config.getServletContext().log(
                "\"" + resourceName + "\"花费了" + passedTime + "毫秒（ms）");
}
```