Filter & Wrapper

1. What is Filter ?
   1)HTTP Request와 Response을 변경할 수 있는 재 사용 가능한 Code
   2)Request를 가로채서 요청된 리소스에 전달된 ServletRequest나 ServletResponse를 처리하는 객체
   3)객체의 형태로 존재
   4)Client로 부터 들어오는 Request나 최종 Resource(Servlet / JSP / 기타등등) 사이에 위치하여 Client의 Request정보를 알맞게 변경할 수 있게 한다.
   5)최종 Resource과 Client로 가는 Response사이에 위치하여 최종 Resource의 Request결과를 알맞게 변경할 수 있게 한다.

```
Client's Browser  -----(request)----> Filter  ------------> servlet/Jsp
                                                                |
Client's Browser  <---(response) --- Filter <--------------------
```

2. 기능
   1)Servlet / JSP등을 실행하기 이전에 Request가 올바른지 또는 Resource에 접근할 수 있는 권한을 가졌는 지의 여부를 미리 처리할 수 있다.
   2)Resource가 받게 되는 Request정보는 Client와 Resource사이에 존재하는 Filter에 의해 변경된 Request정보이다.
   3)Servlet / JSP이 생성한 Response데이터를 변경하거나 취소할 수 있는 기능을 구현할 수 있도록 해 준다.
   4)Client가 보게 되는 Response 정보는 Client와 Resource 사이에 존재하는 Filter에 의해 변경된 Response 정보가 된다.
   5)Filter는 Client와 Resource 사이에 한 개만 존재할 수 있는 것은 아니며, 여러 개의 Filter가 모여 하나의 Filter Chain을 형성할 수도 있다.
   6)만일 여러 개의 Filter가 모여서 하나의 체인을 형성할 때 첫 번째 Filter가 변경하는 Request 정보는 Client의 Request 정보가 되지만, Chain의 두 번째 Filter가 변경하는 Request 정보는 첫 번째 Filter를 통해서 변경된 Request 정보가 된다.
   7)즉, Request 정보는 변경에 변경을 거듭하게 되는 것이다.
   8)Filter는 변경된 정보를 변경하는 역할 뿐만 아니라 흐름을 변경하는 역할도 할 수 있다.
   9)즉, Filter는 Client의 요청을 Filter Chain의 다음 단계(결과적으로 Client 가 요청한 자원)에 보내는 것이 아니라 다른 자원의 결과를 Client에 전송할 수 있다.
   10)Filter 는 이러한 기능을 이용해서 사용자 인증이나 권한 체크와 같은 곳에서 사용할 수 있다.
   11)로그인 여부의 확인이나 요금 부과 처리와 같은 일은 Filter를 구현한 클래스 안에 구현하면 되지만, Web Browser와 Web Component 사이에 오가는 데이터를 변형하는 일은 할 수 없다.
   12)오고 가는 메시지를 암호화 하거나, 메시지에 포함된 데이터의 일부를 걸러서 전달되지 못하도록 막는 일은 할 수 없는데, 그런 일을 하기 위해서는 Filter 뿐만 아니라 Wrapper도 필요하다.
   13)Wrapper는 Web Browser와 Web Component 사이를 오가는 Request Message와 Response Message를 포장한다.

   [HelloServlet.java]

```
package com.example.servlets;

public class HelloServlet extends HttpServlet{
   public void doGet(HttpServletRequest req, HttpServletResponse res)
      throws IOException, ServletException{
      res.setContentType("text/html;charset=utf-8");
      PrintWriter out = res.getWriter();
      out.println("<html><body bgcolor='red'>");
      out.println("<font size='7' color='white'><b>안녕 Servlet</b></font>");
      out.println("</body></html>");
      out.close();
   }
}
```

   [web.xml]

```
<servlet>
   <servlet-name>HelloServlet</servlet-name>
   <servlet-class>com.example.servlets.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>HelloServlet</servlet-name>
   <url-pattern>/servlets/HelloServlet</url-pattern>
</servlet-mapping>
```

   14)http://localhost:8080/MyContext/servlets/HelloServlet 를 실행하여 HelloServlet 을 실행할 것이다.

15)여기서 아래와 같이 web.xml에 Filter를 추가하자.

[web.xml]

```
<filter>
   <filter-name>MyFilter</filter-name>
   <filter-class>com.example.filter.MyFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>MyFilter</filter-name>
    <!--Servlet Class 를 사용하고자 할 때 -->
    <servlet-name>HelloServlet</servlet-name>
    <!--JSP file 을 사용하고자 할 때 -->
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

16)HelloServlet 을 요청하면 MyFilter가 호출되고, MyFiler는 MyFilter.java 객체를 호출한다.

[MyFilter.java]

```
package com.example.filter;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyFilter implements Filter{
   @Override
   public void init(FilterConfig filterConfig) throws ServletException{}
   @Override
   public void destroy(){}
   @Override
   public void doFilter(ServletRequest request, ServletResponse response,
         FilterChain chain) throws java.io.IOException, ServletException{
      System.out.println("Filter is Starting...");
      chain.doFilter(request, response);
      System.out.println("Filter is Ending...");
   }
}
```

[HelloServlet.java]

```
package com.example.servlets;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.ServletException;
import java.io.IOException;
import java.io.PrintWriter;

public class HelloServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
   IOException{
      res.setContentType("text/html;charset=utf-8");
      PrintWriter out = res.getWriter();
      out.println("<html><body bgcolor='yellow'>");
      out.println("<font size='7' color='red'>안녕 Servlet</font>");
      out.println("</body></html>");
      out.close();
   }
}
```

[test.jsp]

```
<body bgcolor="yellow">
   <h1>Message Filter 적용 중</h1>
```

```
</body>


3. Filter 의 구현
   1)Filter를 구현하려면 핵심적인 역할을 하는 javax.servlet.Filter interface,
   javax.servlet.ServletRequestWrapper class, javax.servlet.ServletResponseWrapper class이다.
   2)Filter interface는 Client와 최종 Resource 사이에 위치하는 Filter를 나타내는 객체가 구현해야 하는 interface이다.
   3)ServletRequestWrapper class 와 ServletResponseWrapper class는 Filter가 요청한 변경한 결과 또는
   응답을 변경한 결과를 저장할 Wrapper class를 나타낸다.
   4)Methods
      a. void init(FilterConfig filterConfig) throws ServletException
         -Filter를 Web Container내에 생성한 후 초기화할 때 호출

      b. void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
      java.io.IOException, ServletException
         -Chain에 따라 다음에 존재하는 Filter로 이동한다.
         -Chain의 가장 마지막에는 Client가 요청한 최종 Resource이 위치한다.

      c. void destroy()
         -Filter가 Web Container에서 삭제될 때 호출된다.


4. Filter 설정하기
   1)web.xml에 <filter> tag와 <filter-mapping> tag를 설정한다.
   2)<filter>
      -Web Application에서 사용될 Filter를 지정하는 역할

   3)<filter-mapping>
      -특정 자원에 대해 어떤 Filter를 사용할지를 결정

   4)<url-pattern>
      -Client가 요청한 특정 URI에 대해서 Filtering할 때 사용
      -'/' 로 시작하고, '/*' 로 끝나는 경로 매핑
      -'*.' 로 시작하는 확장자에 대한 매핑
      -오직 '/'만 포함하는 경우 어플리케이션의 기본 서블릿으로 매핑
      -주의할 점은 계층적인 URL  경로명을 지정할 대는 와일드카드 문자와 파일 확장자를 함께 쓰면 안된다.
      -<url-pattern>/sub1/*.jsp</url-pattern>을 잘못된 것이다.

   5) 사용예
      a. <filter-mapping>
            <filter-name>Simple-filter</filter-name>
            <url-pattern>/*</url-pattern>
         <filter-mapping>

      b. <filter-mapping>
            <filter-name>Simple-filter</filter-name>
            <url-pattern>*.jsp</url-pattern>
         <filter-mapping>

      c. <filter-mapping>
            <filter-name>Simple-filter</filter-name>
            <url-pattern>/sub1/*</url-pattern>
         <filter-mapping>

      d. <filter-mapping>
            <filter-name>Simple-filter</filter-name>
            <url-pattern>/sub1/*</url-pattern>
            <url-pattern>/sub2/*</url-pattern>
         <filter-mapping>

      e. <filter-mapping>
            <filter-name>Simple-filter</filter-name>
            <servlet-name>hello-servlet</servlet-name>
         <filter-mapping>

      f. <filter-mapping>
            <filter-name>Simple-filter</filter-name>
```

```
            <url-pattern>/sub1/*</url-pattern>
            <url-pattern>/sub2/*</url-pattern>
            <servlet-name>hello-servlet</servlet-name>
        <filter-mapping>
```

5. <dispatcher> tag
   1)이 tag는 다음과 같이 <filter-mapping>태그의 자식 태그로 사용된다.

```
    <filter-mapping>
        <filter-name>Simple-filter</filter-name>
        <servlet-name>hello-servlet</servlet-name>
        <dispatcher>INCLUDE</dispatcher>
    <filter-mapping>
```

   2)이 태그는 실행되는 자원을 Client가 요청한 것인지, 아니면 RequestDispatcher의 forward()를 통해서 이동한 것인지 아니면, include()통해서 포함된 것인지에 따라서 필터를 적용하도록 지정가능하다.
   3)다음의 값을 가질 수 있다.
   a. REQUEST : Client 의 요청인 경우에 필터를 사용(기본값)
   b. FORWARD : forward() 를 통해서 제어를 이동하는 경우에 필터를 사용한다.
   c. INCLUDE : include() 를 통해서 포함하는 경우에 필터를 사용한다.
   d. ERROR : exception 이 발생했을 때 웹 컨테이너가 자동으로 호출하는 방법

6. FilterChain의 방향 바꾸기
   1)/member/welcome.jsp를 요청한다.
   2)session이 없기 때문에 Filter가 확인한 후, LoginForm.html로 이동시킨다.
   3)로그인이 성공하면 session 이 생성됐기 때문에, 다시 welcome.jsp로 요청하면 바로 들어간다.

[web.xml]

```
<filter>
    <filter-name>LoginCheckFilter</filter-name>
    <filter-class>com.example.filter.LoginCheckFilter</filter-class>
</filter>

<filter-mapping>
     <filter-name>LoginCheckFilter</filter-name>
     <url-pattern>/member/*</url-pattern>
</filter-mapping>
```

[member/welcome.jsp]

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<body>
    <h1>회원전용페이지</h1>
</body>
```

[LoginCheckFilter.java]

```
package com.example.filter;

public class LoginCheckFilter implements Filter{
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain)  throws java.io.IOException,  ServletException{
        HttpServletResponse res = (HttpServletResponse)response;
        HttpServletRequest req = (HttpServletRequest)request;
        HttpSession session = req.getSession();
        if(session == null)
            res.sendRedirect("/FilterDemo/LoginForm.html");
        String userid = (String)session.getAttribute("userid");
        if(userid == null)
            res.sendRedirect("/FilterDemo/LoginForm.html");
        chain.doFilter(request, response);
    }
    @Override
    public void destroy(){}
    @Override
```

```
    public void init(FilterConfig filterConfig) throws ServletException {}
}

[LoginForm.html]
<body>
    <div align="center">
        <form action="login.jsp" method="post">
            <table border="0">
                <tr>
                    <th>ID : </th>
                    <td><input type="text" name="userid" /></td>
                </tr>
                <tr>
                    <th>PWD : </th>
                    <td><input type="password" name="userpwd" /></td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                    <input type="submit" value="Login" /></td>
                </tr>
            </table>
        </form>
    </div>
</body>

[login.jsp]
<%
    String userid = request.getParameter("userid");
    String passwd = request.getParameter("userpwd");
    if(userid.equals("javasoft") && passwd.equals("123456")){
        session.setAttribute("userid", userid);
    }
%>
```

7. FilterConfig 사용하는 예제

```
[web.xml]
<filter>
    <filter-name>LogFilter</filter-name>
    <filter-class>com.example.filter.LogFilter</filter-class>
    <init-param>
        <param-name>FILE</param-name>
        <param-value>D:\\temp\\logfilter.txt</param-value>
    </init-param>
</filter>
<filter-mapping>
     <filter-name>LogFilter</filter-name>
     <servlet-name>HelloServlet</servlet-name>
</filter-mapping>

[LogFilter.java]
package com.example.filter;

public class LogFilter implements Filter{
    private PrintWriter writer;
    @Override
    public void init(FilterConfig filterConfig) throws ServletException{
        String filename = filterConfig.getInitParameter("FILE"); //D:\\temp\\logfilter.txt
        try{
            writer = new PrintWriter(new FileWriter(filename, true), true);
        }catch(IOException ex){
            throw new ServletException("Log file Error...");
        }
    }
    @Override
    public void destroy(){
```

```
        writer.close();
    }
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws java.io.IOException, ServletException{
        writer.println("Filter is Staring...");
        writer.flush();
        chain.doFilter(request, response);
        writer.println("Filter is Ending...");
        writer.flush();
    }
}


[web.xml]
<filter-mapping>
    <filter-name>LogMessageFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
<filter>
    <filter-name>LogMessageFilter</filter-name>
    <filter-class>com.example.filter.LogMessageFilter</filter-class>
    <init-param>
        <param-name>FILE1</param-name>
        <param-value>D:\\temp\\logmessage_filter.txt</param-value>
    </init-param>
</filter>

[LogMessageFilter.java]
package com.example.filter;

public class LogMessageFilter implements Filter {
    private PrintWriter writer;
    @Override
    public void destroy() {
        writer.close();
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse res,
            FilterChain chain) throws IOException, ServletException {
        Date now = new Date();
        this.writer.printf("방문일시 : %1$tF %1$tT %n", now);
        HttpServletRequest request = (HttpServletRequest)req;
        this.writer.printf("고객 IP 정보 : %s %n", request.getRemoteAddr());
        this.writer.flush();
        chain.doFilter(req, res);
        HttpServletResponse response = (HttpServletResponse)res;
        this.writer.printf("Content Type : %s %n", response.getContentType());
        this.writer.println("----------------------------------------------------------------");
        this.writer.flush();
    }

    @Override
    public void init(FilterConfig config) throws ServletException {
        String filename = config.getInitParameter("FILE1");
        try{
            this.writer = new PrintWriter(new FileWriter(filename, true), true);
        }catch(IOException ex){
            throw new ServletException("Error");
        }
    }
}

[test.jsp]
<body bgcolor="yellow">
    <h1>Message Filter 적용 중</h1>
```

```
</body>
```

8. Wrapper class의 작성, 설치, 사용

1)Web Browser와 Web Component사이를 오가는 데이터에 변형을 가하려면 Filter를 구현한 클래스와 더불어 Wrapper class를 작성해야 한다.

2)요청 객체를 포장하는 요청 래퍼 클래스와 응답 객체를 포장하는 응답 래퍼 클래스 두가지

3)javax.servlet.http.HttpServletRequestWrapper와 javax.servlet.http.HttpServletResponseWrapper 클래스를 상속받아야 한다.

9. RequestWrapper Demo

[ParamUpperCaseRequestWrapper.java]

```
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class ParamUpperCaseRequestWrapper extends HttpServletRequestWrapper {
   HttpServletRequest request;
   public ParamUpperCaseRequestWrapper(HttpServletRequest request) {
      super(request);
      this.request = request;
   }
   public String getParameter(String name) {
      String str = request.getParameter(name);
      if (str == null) return null;
      return str.toUpperCase();
   }
   public String[] getParameterValues(String name) {
      String str[] = request.getParameterValues(name);
      if (str == null) return null;
      for (int cnt = 0; cnt < str.length; cnt++)
         str[cnt] = str[cnt].toUpperCase();
      return str;
   }
}
```

[ParamUpperCaseFilter.java]

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class ParamUpperCaseFilter implements Filter {
   public void init(FilterConfig config) throws ServletException {}
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
                        throws IOException, ServletException {
      ParamUpperCaseRequestWrapper requestWrapper
                  = new ParamUpperCaseRequestWrapper((HttpServletRequest) request);
      chain.doFilter(requestWrapper, response);
   }
   public void destroy() {}
}
```

[welcome.jsp?NAME=Jessica]

```
<%@page contentType="text/html; charset=euc-kr"%>
<% String name = request.getParameter("NAME"); %>
<HTML>
   <HEAD><TITLE>환영 인사</TITLE></HEAD>
   <BODY>
      안녕하세요, <%= name %>님.
   </BODY>
</HTML>
```

[colors.jsp?COLOR=White&COLOR=Red&COLOR=Blue]

```
<%@page contentType="text/html; charset=euc-kr"%>
<% String color[] = request.getParameterValues("COLOR"); %>
<HTML>
   <HEAD><TITLE>색 고르기</TITLE></HEAD>
```

```
    <BODY>
      <H4>선택하신 색은 다음과 같습니다.</H4>
      <%
        if (color != null) {
          for (int cnt = 0; cnt < color.length; cnt++)
            out.println(color[cnt] + "<BR>");
        }
      %>
    </BODY>
  </HTML>


10. ResponseWrapper Demo

  [CookieLowerCaseResponseWrapper.java]
  import javax.servlet.http.*;
  import javax.servlet.*;
  import java.io.*;
  public class CookieLowerCaseResponseWrapper extends HttpServletResponseWrapper {
    private HttpServletResponse response;
    public CookieLowerCaseResponseWrapper(HttpServletResponse response) {
      super(response);
      this.response = response;
    }
    public void addCookie(Cookie cookie) {
      String value = cookie.getValue();
      String newValue = value.toLowerCase();
      cookie.setValue(newValue);
      response.addCookie(cookie);
    }
  }

  [CookieLowerCaseFiler.java]
  import javax.servlet.http.*;
  import javax.servlet.*;
  import java.io.*;
  public class CookieLowerCaseFilter implements Filter {
    public void init(FilterConfig config) throws ServletException {
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
                        throws IOException, ServletException {
      CookieLowerCaseResponseWrapper responseWrapper
                      = new CookieLowerCaseResponseWrapper((HttpServletResponse) response);
      chain.doFilter(request, responseWrapper);
    }
    public void destroy() {
    }
  }

  [cookieSaver.jsp]
  <%@page contentType="text/html; charset=euc-kr"%>
  <%
    Cookie cookie = new Cookie("CART", "Lemon, Apple, Chocolate");
    response.addCookie(cookie);
  %>
  <HTML>
    <HEAD><TITLE>쿠키 저장하기</TITLE></HEAD>
    <BODY>
      쿠키가 저장되었습니다.
    </BODY>
  </HTML>

  [cookieRetriever.jsp]
  <%@page contentType="text/html; charset=euc-kr"%>
  <HTML>
    <HEAD><TITLE>쿠키 가져오기</TITLE></HEAD>
    <BODY>
```

```
        CART = ${cookie.CART.value}
    </BODY>
</HTML>


11. WebFilter annotation을 이용한 Filter 설정
  1)@WebFilter를 사용할 경우에는 배포 기술서를 수정할 필요 없이 Filter class에 annotation만 붙여주면 되기 때문에 편리하다.
  2)하지만, 설정 정보를 바꿀 때마다 다시 컴파일해야 한다.
  3)반면 배포 기술서를 사용할 경우에는 컴파일을 할 필요 없이 텍스트 파일만 수정하면 된다.
  4)@WebFilter 속성
    a. asyncSupported : 비동기 동작 모드 사용 여부
    b. description
    c. dispatcherTypes
    d. displayName
    e. filterName
    f. initParams
    g. largeIcon
    h. servletNames
    i. smallIcon
    j. urlPatterns
    k. value

  5)@WebFiler(filterName="DataCompressionFilter", urlPattern={"/*"})
    <filter-mapping>
      <filter-name>DataCompressionFilter</filter-name>
      <url-pattern>/*</url-pattern>
    </filter-mapping>

  6)@WebFiler(filterName="Security Filter", urlPattern={"/*"},
              initParams = {
                 @WebInitParam(name="frequency", value="1909"),
                  @WebInitParam(name="resolution", value="1024")
            }
    )
    <filter>
      <filter-name>Security Filter</filter-name>
      <filter-class>filterClass</filter-class>
      <init-param>
        <param-name>frequency</param-name>
        <param-value>1909</param-value>
      </init-param>
      <init-param>
        <param-name>resolution</param-name>
        <param-value>1024</param-value>
      </init-param>
    </filter>
    <filter-mapping>
      <filter-name>Security Filter</filter-name>
      <url-pattern>/*</url-pattern>
    </filter-mapping>


12. @WebFilter Demo

  [test.jsp]
  <%@ page contentType="text/html;charset=utf-8" %>
  <!DOCTYPE html>
  <html lang="ko">
  <head>
    <meta charset="utf-8">
    <title>필터</title>
  </head>
  <body>
    필터 테스트
  </body>
  </html>
```

[LoggingFilter.java]

```
package com.example.filter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

import javax.servlet.Filter;
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.annotation.WebInitParam;
import javax.servlet.http.HttpServletRequest;

@WebFilter(filterName = "LoggingFilter", urlPatterns = { "/*" },
      initParams = {
            @WebInitParam(name = "logFileName",value = "log.txt"),
            @WebInitParam(name = "prefix", value = "URI: ") })
public class LoggingFilter implements Filter {
   private PrintWriter logger;
   private String prefix;
   @Override
   public void init(FilterConfig filterConfig) throws ServletException {
      prefix = filterConfig.getInitParameter("prefix");
      String logFileName = filterConfig.getInitParameter("logFileName");
      String appPath = filterConfig.getServletContext().getRealPath("/");
      // logFileName으로 경로가 전달되지 않았을 땐,
      // $TOMCAT_HOME/bin 에 로그파일을 생성함
      System.out.println("logFileName:" + logFileName);
      try {
         logger = new PrintWriter(new File(appPath,
               logFileName));
      } catch (FileNotFoundException e) {
         e.printStackTrace();
         throw new ServletException(e.getMessage());
      }
   }

   @Override
   public void destroy() {
      System.out.println("필터 제거 중");
      if (logger != null) {
         logger.close();
      }
   }

   @Override
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)
         throws IOException, ServletException {
      System.out.println("LoggingFilter.doFilter");
      HttpServletRequest httpServletRequest = (HttpServletRequest) request;
      logger.println(new Date() + " " + prefix + httpServletRequest.getRequestURI());
      logger.flush();
      filterChain.doFilter(request, response);
   }
}
```