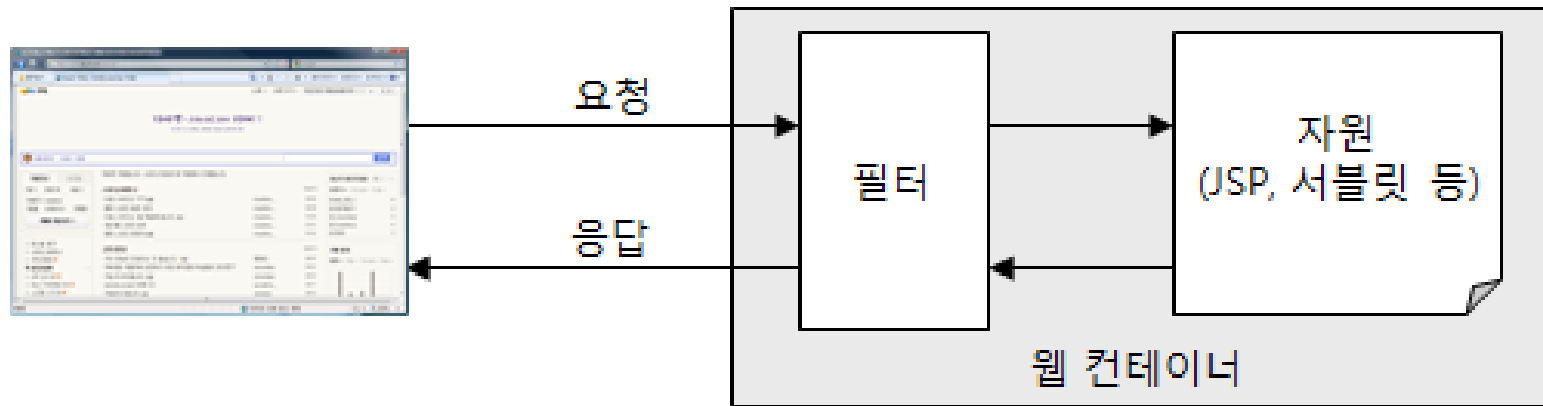


JSP (Java Sever Page)

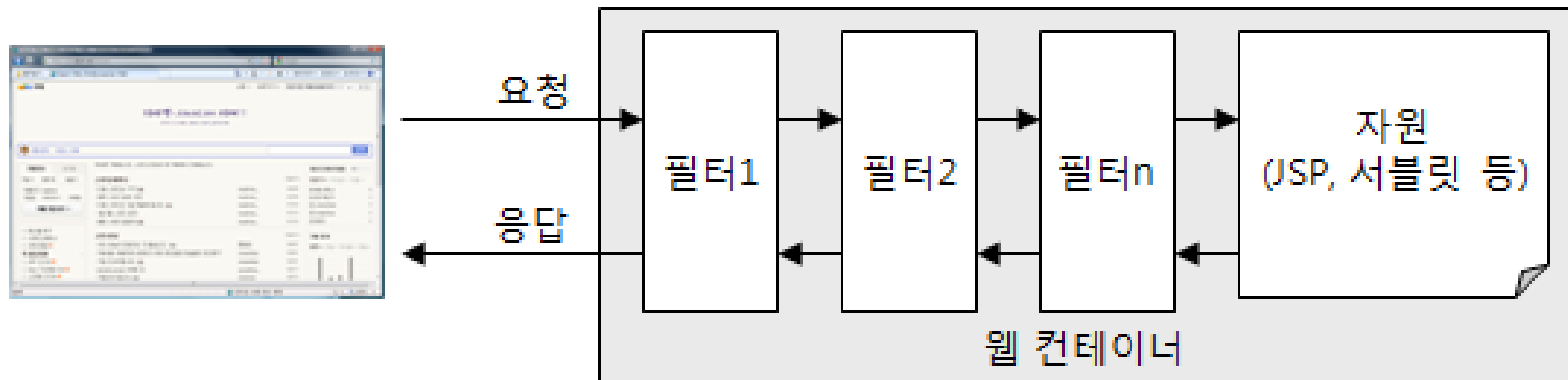
- 필터와 리스너

- 필터란
- 필터의 구현
- 필터 응용

- HTTP 요청과 응답을 변경할 수 있는 재사용 가능한 코드
- 필터의 기본 구조



- 요청의 내용을 변경하거나 응답의 내용을 변경 가능
- 1개 이상의 필터 연동 가능



■ 필터 구현

- Filter 인터페이스 사용
- Filter 인터페이스의 메서드
 - `public void init(FilterConfig filterConfig) throws ServletException`
필터를 초기화할 때 호출된다.
 - `public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws java.io.IOException, ServletException`
체인을 따라 다음에 존재하는 필터로 이동한다. 체인의 가장 마지막에는 클라이언트가 요청한 최종 자원이 위치한다.
 - `public void destroy()`
필터가 웹 컨테이너에서 삭제될 때 호출된다.

■ 필터 구현

- doFilter() 메서드에서 필터 기능 구현

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

    // 1. request 파라미터를 이용하여 요청의 필터 작업 수행

    ...

    // 2. 체인의 다음 필터 처리

    chain.doFilter(request, response);

    // 3. response를 이용하여 응답의 필터링 작업 수행

    ...
}
```

■ 필터 설정

- web.xml에 URL 별 매핑 설정 추가

```
<web-app ...>

  <filter>
    <filter-name>FilterName</filter-name>
    <filter-class>filter.FileClass</filter-class>
    <init-param>
      <param-name>paramName</param-name>
      <param-value>value</param-value>
    </init-param>
  </filter>

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

  ...
</web-app>
```

■ 필터 설정

- **<dispatcher>** 를 통한 필터 적용 대상 지정

```
<filter-mapping>  
  <filter-name>AuthCheckFilter</filter-name>  
  <servlet-name>AuthCheck</servlet-name>  
  <dispatcher>INCLUDE</dispatcher>  
</filter-mapping>
```

- **<dispatcher>**의 값

- REQUEST - 클라이언트의 요청인 경우에 필터를 사용 (기본값)
- FORWARD - forward()를 통해서 제어를 이동하는 경우에 필터를 사용
- INCLUDE - include()를 통해서 포함하는 경우에 필터를 사용

- **@WebFilter 애노테이션 이용**
 - **filterName**: Filter 이름
 - **value**: Filtering 대상 URL.
 - **urlPatterns**: Filtering 대상 URL 목록들
 - **servletNames**: Filtering 대상 서블릿 목록들.
 - **displayName**: Filter 표시 명
 - **dispatcherTypes** : **REQUEST, FORWARD, INCLUDE, ERROR, ASYNC**
 - **REQUEST** : Client로 부터 직접 전달되는 request, Default
 - **FORWARD** : urlPatterns와 servlerNames에 열거된 servlets와 jsp를 forward call 한다. 일반적으로 REQUEST와 같이 설정
 - **INCLUDE**: urlPatterns와 servlerNames에 열거된 servlets와 jsp를 include call한다. 일반적으로 REQUEST와 같이 설정
 - **ERROR** : urlPatterns에 열거된 error 처리자원 동작 할 시 호출된다.
 - **initParams** : 초기값 설정시 사용, 하위 Annotation으로 @WebInitParam를 가지고 있음

■ 필터의 응용

- 데이터 변환(다운로드 파일의 압축/데이터 암호화/이미지 변환 등)
- XSL/T를 이용한 XML 문서 변경
- 사용자 인증
- 캐싱 필터
- 자원 접근에 대한 로깅

■ 필터의 응용 : 로그인 검사 필터

filter.LoginCheckFilter.java

```
package filter;
```

```
import java.io.IOException;
```

```
import javax.servlet.Filter;
```

```
import javax.servlet.FilterChain;
```

```
import javax.servlet.FilterConfig;
```

```
import javax.servlet.RequestDispatcher;
```

```
import javax.servlet.ServletException;
```

```
import javax.servlet.ServletRequest;
```

```
import javax.servlet.ServletResponse;
```

```
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpSession;
```

```
public class LoginCheckFilter implements Filter {
```

```
    @Override
```

```
    public void init(FilterConfig config) throws ServletException {
```

```
}
```

■ 필터의 응용 : 로그인 검사 필터

filter.LoginCheckFilter.java

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
                    FilterChain chain) throws IOException, ServletException {
    HttpServletRequest httpRequest = (HttpServletRequest) request;
    HttpSession session = httpRequest.getSession(false);

    boolean login = false;
    if (session != null) {
        if (session.getAttribute("MEMBER") != null) {
            login = true;
        }
    }
    if (login) {
        chain.doFilter(request, response);
    } else {
        RequestDispatcher dispatcher = request.getRequestDispatcher("/loginForm.jsp");
        dispatcher.forward(request, response);
    }
}
```

■ 필터의 응용 : 로그인 검사 필터

filter.LoginCheckFilter.java

```
    @Override  
    public void destroy() {  
    }  
}
```

■ 필터의 응용 : 로그인 검사 필터

web.xml

```
<filter>
    <filter-name>LoginCheck</filter-name>
    <filter-class>filter.LoginCheckFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>LoginCheck</filter-name>
    <url-pattern>/board/*</url-pattern>
</filter-mapping>
```

■ 필터의 응용 : 로그인 검사 필터

login.jsp

```
<%@ page contentType= "text/html; charset=euc-kr" %>
<%
String memberId = request.getParameter("memberId");
session.setAttribute("MEMBER", memberId);
%>
<html>
<head> <title> 로그인 </title> </head>
<body>
로그인 처리
</body>
</html>
```

■ 필터의 응용 : 로그인 검사 필터

loginForm.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<html>
<head> <title>로그인 </title> </head>
<body>
<form action="<%= request.getContextPath() %>/login.jsp">
아이디 <input type="text" name="memberId">
암호 <input type="password" name="password">
<input type="submit" value="로그인">
</form>
</body>
</html>
```

■ 필터의 응용 : 캐릭터 인코딩 필터

filter.CharacterEncodingFilter.java

```
package filter;
```

```
import java.io.IOException;
```

```
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 CharacterEncodingFilter implements Filter {
```

```
    private String encoding;
```

```
    @Override
```

```
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)  
        throws IOException, ServletException {  
        req.setCharacterEncoding(encoding);
```


■ 필터의 응용 : 캐릭터 인코딩 필터

filter.CharacterEncodingFilter.java

```
        chain.doFilter(req, res);
    }

    @Override
    public void init(FilterConfig config) throws ServletException {
        encoding = config.getInitParameter("encoding");
        if (encoding == null) {
            encoding = "UTF-8";
        }
    }

    @Override
    public void destroy() {
    }

}
```

- ServletContextListener
- 리스너 실행 순서
- 예외 처리

- 웹 어플리케이션의 시작 이벤트나 종료 이벤트를 처리
- 웹 컨테이너는 ServletContextListener의 특정 메서드를 호출함
- ServletContextListener 인터페이스의 이벤트 처리 메서드
 - `public void contextInitialized(ServletContextEvent sce)`
웹 어플리케이션이 초기화될 때 호출된다.
 - `public void contextDestroyed(ServletContextEvent sce)`
웹 어플리케이션이 종료될 때 호출된다.

■ web.xml 파일에 리스너 등록

- **<listener>** 태그 이용

```
<web-app ...>
  <blistener>
    <blistener-class>jdbc.loader.DBCPInitListener</blistener-class>
  </blistener>

  <listener>
    <listener-class>CodeInitListener</listener-class>
  </listener>
  ...
</web-app>
```

■ ServletContextEvent로부터 필요 정보 조회

- contextInitialized()와 contextDestroyed() 메서드에 전달되는 객체
- ServletContext를 구해주는 getServletContext() 메서드 제공
 - ServletContext.getInitParameter() 메서드를 이용해서 web.xml 파일에 등록한 초기화 파라미터 값 조회

```
<web-app ...>
  <context-param>
    <param-name>jdbcdriver</param-name>
    <param-value>oracle.jdbc.OracleDriver</param-value>
  </context-param>
</web-app>
```

```
public class DBCPInitListener implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {
        try {
            ServletContext context = sce.getServletContext();
            String drivers = context.getInitParameter("jdbcdriver");

            ...
        }
    }
}
```

■ 실행 순서 & 예외 처리

- 한 개 이상의 리스너 등록 가능
 - `contextInitialized()` 메서드는 등록된 순서대로 실행
 - `contextDestroyed()` 메서드는 등록된 반대 순서대로 실행
 - `@WebListener`
- 리스너의 메서드에 `try - catch`로 예외를 잡은 뒤, `RuntimeException`을 발생시키도록 함
 - 리스너가 예외를 발생할 경우 웹 어플리케이션 시작에 실패함

```
public void contextInitialized(ServletContextEvent sce) {  
    try {  
        ...  
        ...  
    } catch (Exception ex) {  
        throw new RuntimeException(ex);  
    }  
}
```

■ 실행 순서 & 예외 처리

jdbc.loader.DBCPInitListener.java

```
package jdbc.loader;
```

```
import java.util.StringTokenizer;
```

```
import javax.servlet.ServletContext;
```

```
import javax.servlet.ServletContextEvent;
```

```
import javax.servlet.ServletContextListener;
```

```
public class DBCPInitListener implements ServletContextListener {
```

```
    @Override
```

```
    public void contextInitialized(ServletContextEvent sce) {
```

```
        try {
```

```
            ServletContext context = sce.getServletContext();
```

```
            String drivers = context.getInitParameter("jdbcdriver");
```

```
            StringTokenizer st = new StringTokenizer(drivers, ",");
```

```
            while (st.hasMoreTokens()) {
```

```
                String jdbcDriver = st.nextToken();
```

```
                Class.forName(jdbcDriver);
```

```
            }
```

■ 실행 순서 & 예외 처리

jdbc.loader.DBCPInitListener.java

```
        Class.forName("org.apache.commons.dbcp.PoolingDriver");  
    } catch (Exception ex) {  
        throw new RuntimeException(ex);  
    }  
    System.out.println("DBCP 초기화 완료");
```

```
}
```

```
@Override
```

```
public void contextDestroyed(ServletContextEvent sce) {  
    }  
}
```

```
}
```


■ 실행 순서 & 예외 처리

jdbc.loader.DBCPInitListener.java

```
<?xml version="1.0" encoding="euc-kr"?>
```

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0" >
```

```
  <listener>
```

```
    <listener-class>jdbc.loader.DBCPInitListener</listener-class>
```

```
  </listener>
```

```
  <context-param>
```

```
    <param-name>jdbcdriver</param-name>
```

```
    <param-value>oracle.jdbc.OracleDriver</param-value>
```

```
  </context-param>
```

```
</web-app>
```