

Chapter

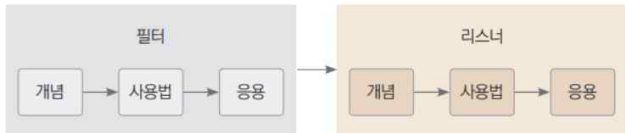
15

필터(Filter)와 리스너 (Listener)

■ 학습 목표

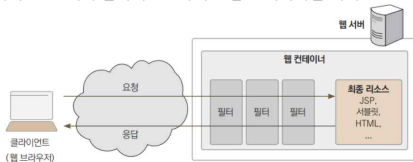
- 필터는 클라이언트의 요청을 가장 먼저 받아 사전 처리하는 역할을 합니다.
- 리스너는 웹 컨테이너에서 발생하는 이벤트를 감지하고 처리합니다.
- 이번 장에서는 필터와 리스너를 학습합니다.

■ 학습 순서



■ 필터란..??

- 필터는 웹 컨테이너의 '전면'에서 클라이언트와 주고받는 메시지를 처리



- 클라이언트가 요청을 보내거나, 웹서버가 응답할때는 필터를 거치게 됨
- 요청과 응답에 대해 전처리 혹은 후처리를 할 수 있음
- 또한 2개 이상을 연결할 수 있어 이를 "필터체인"이라 함

■ Filter 인터페이스

- 필터 기능 개발을 위해 Filter 인터페이스를 구현(implements)해야 함
- Filter 인터페이스에 정의된 메서드

메서드명	설명	필수 여부
init()	필터를 초기화할 때 호출	X
doFilter()	필터를 리소스에 적용할 때마다 호출	O
destroy()	필터가 소멸될 때 호출	X

■ init() 메서드

- 웹 컨테이너가 필터를 초기화할때 딱 한번 호출됨
- 디폴트 메서드이므로 오버라이딩은 필수사항 아님

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

■ **init() 메서드**

- 매개변수로 선언된 FilterConfig는 web.xml에 정의한 초기화 매개변수를 읽을 수 있음

메서드명	설명
getFilterName()	필터 매핑 시 지정한 필터명을 반환합니다. <filter-name> 요소로 지정합니다.
getInitParameter()	해당 필터에 지정한 초기화 매개변수의 값을 읽어옵니다. <init-param> 요소로 지정합니다.
getInitParameterNames()	해당 필터에 지정한 모든 초기화 매개변수의 이름을 읽어옵니다. 반환 타입은 Enumeration<String>입니다.
getServletContext()	application 내장 객체의 타입인 ServletContext를 반환합니다.

■ doFilter() 메서드

- 클라이언트의 요청을 리소스에 적용할 때마다 호출

```
public void doFilter(ServletRequest req, ServletResponse resp,  
                    FilterChain filter)
```

- 매개변수로 선언된 객체의 역할
 - ServletRequest : 요청 정보를 저장한 객체
 - ServletResponse : 응답 정보를 저장한 객체
 - FilterChain : 필터 체인에서 다음 필터를 호출하거나 최종 리소스를 호출할 때 사용
- 가장 먼저 요청에 대해 전처리를 진행
- 그런 다음 doFilter()를 호출하여 필터 체인에서 다음 필터를 호출하거나, 최종 리소스를 호출

[Note] chain.doFilter() 메서드를 호출하지 않으면 전처리 결과를 다음 단계로 넘기지 않으므로 요청 처리가 여기서 멈추게 되어 웹 브라우저에서는 아무런 응답도 받을 수 없음

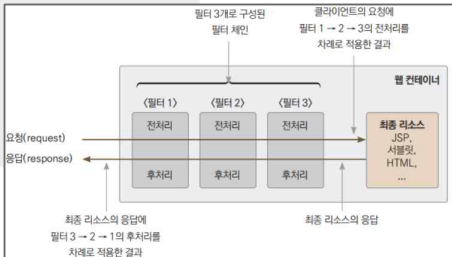
■ doFilter() 메서드

- doFilter() 일반적인 오버라이딩 형태 및 전체 처리 과정

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain)
    throws IOException, ServletException {
    // 전처리(request 매개변수 이용)

    // 다음 필터(혹은 최종 리소스) 호출
    chain.doFilter(request, response);

    // 후처리(response 매개변수 이용)
}
```



■ destroy() 메서드

- 필터가 종료될 때 딱 한 번 호출되어 필터를 통해 열었던 리소스를 닫음.
- init() 메서드와 마찬가지로 디폴트 메서드므로 필요할 때만 오버라이딩

```
public void destroy()
```

■ web.xml에서 필터 매핑하기

- 필터 사용을 위해서는 요청명과 해당 필터를 매핑해야 함
- web.xml은 다음과 같이 사용

필터로 사용할 클래스

```
public class 필터클래스명 implements Filter { ①
```

```
<filter>
  <filter-name>필터명</filter-name> ②
  <filter-class>패키지를 포함한 필터 클래스명</filter-class>
  <init-param> ④
    <param-name>초기화 매개변수명</param-name>
    <param-value>초기화 매개변수값</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>필터명</filter-name> ③
  <url-pattern>필터를 적용할 요청명</url-pattern> ⑤
</filter-mapping>
```




15.1 필터

■ 필터 클래스 작성

```
public class BasicFilter implements Filter { ❶
    FilterConfig config;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException { ❷
        config = filterConfig; ❸
        String filterName = filterConfig.getFilterName(); ❹

        System.out.println("BasicFilter -> init() 호출됨 : " + filterName);
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        String filterInitParam = config.getInitParameter("FILTER_INIT_PARAM"); ❺
        System.out.println("BasicFilter -> 초기화 매개변수 : " + filterInitParam);
    }
}
```

BasicFilter.java

- ❶ 필터 클래스이므로 Filter 인터페이스를 구현
- ❷ init() 메서드 오버라이딩
- ❸ doFilter() 메서드에서 사용하기 위해 값 설정
- ❹ <filter-name> 읽어옴
- ❺ 초기화 매개변수 읽어옴. ❸에서 저장해 둔 FilterConfig 객체사용

■ 필터 클래스 작성

```
//      String method = request.getMethod(); 에러 발생(형변환 후 호출할 수 있음)
String method = ((HttpServletRequest)request).getMethod(); ⑥
System.out.println("BasicFilter -> 전송 방식 : " + method);

chain.doFilter(request, response); ⑦
}

@Override
public void destroy() { ⑧
    System.out.println("BasicFilter -> destroy() 호출됨");
}
}
```

BasicFilter.java(계속)

⑥ getMethod()를 통해 전송 방식을 출력

⑦ 필터에서 작업이 끝나면 동적 자원인 JSP로 제어권을 넘김

⑧ 웹 컨테이너가 중지될때 호출되어 필터를 소멸시킴

[Note] ⑥에서 매개변수인 request는 ServletRequest 타입이므로 먼저 HttpServletRequest로 형변환해야만 request 내장객체의 메서드를 호출할 수 있음

■ 필터 클래스 작성

```
<filter>
  <filter-name>BasicFilter</filter-name> ①
  <filter-class>filter.BasicFilter</filter-class> ②
  <init-param> ③
    <param-name>FILTER_INIT_PARAM</param-name>
    <param-value>필터 초기화 매개변수</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>BasicFilter</filter-name> ④
  <url-pattern>/15FilterListener/BasicFilter.jsp</url-pattern> ⑤
</filter-mapping>
```

web.xml

- ⑤ 해당 요청명으로 요청이 들어오면 ④ 필터명을 찾는다. ① 해당 필터명으로 등록된 ② 클래스를 필터로 사용하게 된다.
- ③ 해당 필터에서 사용할 초기화 매개변수를 등록

■ 필터를 적용할 JSP 작성

```
<script>
function formSubmit(form, methodType) { ❶
    if (methodType == 1) { ❷
        form.method = "get";
    }
    else if (methodType == 2) {
        form.method = "post";
    }
    form.submit(); ❸
}
</script>
```

```
<h2>web.xml에서 매핑하기</h2>
<form> ❹
    <input type="button" value="Get 방식 전송" onclick="formSubmit(this.form,
1);" />
    <input type="button" value="Post 방식 전송" onclick="formSubmit(this.form,
2);" />
</form>
```

BasicFilter.jsp

❶ Javascript 함수에서 get / post 두 가지 방식으로 submit 할수있도록 정의

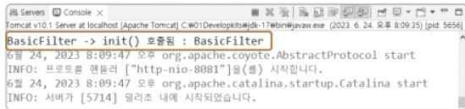
❷ 1이면 get방식, 2면 post방식 전송



15.1 필터

동작 확인

이클립스의 Server 탭에서 톰캣 서버를 시작 후 콘솔 확인



```
Tomcat v10.1 Server at localhost [Apache Tomcat] C:\01Developkits\jdk-17\bin\java.exe (2023. 6. 24. 오후 8:09:35) (pid: 5656)
BasicFilter -> init() 호출됨 : BasicFilter
6월 24, 2023 8:09:47 오후 org.apache.coyote.AbstractProtocol start
INFO: 프로토콜 핸들러 ["http-nio-8081"]을(를) 시작합니다.
6월 24, 2023 8:09:47 오후 org.apache.catalina.startup.Catalina start
INFO: 서버가 [5714] 밀리초 내에 시작되었습니다.
```

JSP 실행. 콘솔을 보면 doFilter() 메서드가 실행된것을 알 수 있음



■ 동작 확인

- 웹 브라우저에서 get방식, post방식 버튼을 순서대로 클릭



```
Tomcat v10.1 Server at localhost [Apache Tomcat] C:\01Developkits\jdk-17\bin\java.exe (2023. 6. 24. 오후)
INFO: 서버가 [2439] 밀리초 내에 시작되었습니다.
BasicFilter -> 초기화 매개변수 : 필터 초기화 매개변수
BasicFilter -> 전송방식 : GET
BasicFilter -> 초기화 매개변수 : 필터 초기화 매개변수
BasicFilter -> 전송방식 : GET
BasicFilter -> 초기화 매개변수 : 필터 초기화 매개변수
BasicFilter -> 전송방식 : POST
```

- Servers 탭에서 톰캣 서버 종료



```
<terminated> Tomcat v10.1 Server at localhost [Apache Tomcat] C:\01Developkits\jdk-17\bin\java.exe (2023. 6. 24)
INFO: 서비스 [Catalina]을(를) 중지시킵니다.
BasicFilter -> destroy() 호출됨
6월 24, 2023 8:15:49 오후 org.apache.coyote.AbstractProtocol stop
INFO: 프로토콜 핸들러 ["http-nio-8081"]을(를) 중지시킵니다.
```

■ 애너테이션으로 필터 매핑하기

필터 매핑의 2번째 방법은 애너테이션을 사용하는 것

매핑할 요청명이 1개인 경우

```
@WebFilter(filterName="필터명", urlPatterns="요청명")
```

```
public class 필터클래스명 implements Filter {
```

매핑할 요청명이 2개 이상인 경우

```
@WebFilter(filterName="필터명", urlPatterns={"요청명1", "요청명2"})
```

```
public class 필터클래스명 implements Filter {
```

■ 필터 클래스 작성

```
@WebFilter(filterName="AnnoFilter", urlPatterns="/15FilterListener/
```

```
AnnoFilter.jsp") ❶
```

```
public class AnnoFilter implements Filter {
```

filter/AnnoFilter.java

❶ 애너테이션으로 필터명과 요청명을 매핑. 즉 해당 요청이 들어오면 필터를 적용하겠다는 뜻.

■ 필터 클래스 작성

- init(), destroy() 메서드는 필요없는 경우 오버라이딩 하지 않아도 됨

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException {
    String searchField = request.getParameter("searchField");
    String searchWord = request.getParameter("searchWord");
    System.out.println("검색 필드 : " + searchField);
    System.out.println("검색어 : " + searchWord);
    chain.doFilter(request, response);
}
```

AnnoFilter.java(계속)

③ 매개변수로 전달된 값을 읽어서 콘솔에 출력 후 필터 체인의 다음 노드로 제어권을 넘김

■ 필터를 적용할 JSP 작성

```
<body>
  <form> ①
    <select name="searchField">
      <option value="title">제목</option>
      <option value="content">내용</option>
    </select>
    <input type="text" name="searchWord" value="애니메이션" />
    <input type="submit" value="검색하기" />
  </form>
</body>
```

AnnoFilter.jsp

② 일반적인 검색폼으로 구성



■ 로그인/로그아웃 구현

- 6장에서 이미 구현했던 로그인/로그아웃 코드를 최대한 활용해서 필터를 적용

```
<form method="post" name="loginFrm" onsubmit="return validateForm(this);"> ④  
<input type="hidden" name="backUrl" value="${ param.backUrl }" /> ⑤  
아이디 : <input type="text" name="user_id" /><br />  
패스워드 : <input type="password" name="user_pw" /><br />  
<input type="submit" value="로그인하기" />  
</form>  
<%  
} else { ⑥  
%>  
    <%= session.getAttribute("UserName") %> 회원님, 로그인하셨습니다.<br />  
    <a href="?mode=logout">[로그아웃]</a> ⑦  
    <%  
    }  
    %>
```

LoginFilter.jsp

6장의 LoginForm.jsp와 거의 비슷

⑤ 매개변수로 backUrl이 전달된다면 hidden박스에 담아둠. 로그인 성공시 이 경로로 이동하게 됨.

■ 로그인 처리용 필터 클래스

```
@WebFilter(filterName="LoginFilter",  
          urlPatterns="/15FilterListener/LoginFilter.jsp") ❶  
public class LoginFilter implements Filter {  
    // 회원 정보를 얻어오기 위해 필요한 데이터베이스 접속 정보  
    String oracleDriver, oracleURL, oracleId, oraclePwd;  
  
    @Override  
    public void init(FilterConfig filterConfig) throws ServletException {  
        ServletContext application = filterConfig.getServletContext(); ❷  
  
        oracleDriver = application.getInitParameter("OracleDriver");  
        oracleURL = application.getInitParameter("OracleURL");  
        oracleId = application.getInitParameter("OracleId");  
        oraclePwd = application.getInitParameter("OraclePwd");  
    }  
}
```

LoginFilter.java

❶ 필터 매핑

❷ FilterConfig 객체로 web.xml의 컨텍스트 초기화 매개변수를 읽어옴

❸ 오라클 접속 정보 저장

■ 로그인 처리용 필터 클래스

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException {
    HttpServletRequest req = (HttpServletRequest)request;
    HttpServletResponse resp = (HttpServletResponse)response;

    HttpSession session = req.getSession();
    String method = req.getMethod();

    if (method.equals("POST")) { // 로그인 처리
        // 로그인 정보와 일치하는 회원 확인
        MemberDAO dao = new MemberDAO(oracleDriver, oracleURL, oracleId,
            oraclePwd);
        MemberDTO memberDTO = dao.getMemberDTO(user_id, user_pw);
        dao.close();
    }
```

LoginFilter.java(계속)

doFilter()에서는 로그인/로그아웃 처리. post 방식이라면 로그인, get 방식이라면 로그아웃

④ request와 response를 다음과 같이 HTTP용 타입으로 형변환

⑤ session 객체와 전송방식 얻어옴

⑥ post 방식이라면 로그인 처리

⑦ 오라클에 접속한 후 회원정보 확인

■ 로그인 처리용 필터 클래스

```
if (memberDTO.getId() != null) { // 일치하는 회원 존재 ①
    // 세션에 로그인 정보 저장 ②
    session.setAttribute("UserId", memberDTO.getId());
    session.setAttribute("UserName", memberDTO.getName());

    // 다음 페이지로 이동 ⑩
    String backUrl = request.getParameter("backUrl");
    if (backUrl != null && !backUrl.equals("")) {
        JSFunction.alertLocation(resp, "로그인 전 요청한 페이지로 이동합니다.", backUrl);
        return;
    }
    else {
        resp.sendRedirect("../15FilterListener/LoginFilter.jsp");
    }
}
```

LoginFilter.java(계속)

⑧ 회원인증에 성공하면 세션영역에 로그인 정보를 저장

⑩ 만약 backUrl 매개변수가 있다면 해당 페이지로 이동. 없다면 기존 로그인 페이지로 이동.

■ 로그인 처리용 필터 클래스

```
    else { // 일치하는 회원 없음 ⑪
        req.setAttribute("LoginErrMsg", "로그인에 실패했습니다.");
        req.getRequestDispatcher("../15FilterListener/LoginFilter.jsp")
            .forward(req, resp);
    }
}
else if (method.equals("GET")) { // 로그아웃 처리 ⑫
    String mode = request.getParameter("mode");
    if ("mode != null && mode.equals(\"logout\")") {
        session.invalidate();
    }
}

chain.doFilter(request, response);
}
```

LoginFilter.java(계속)

⑪

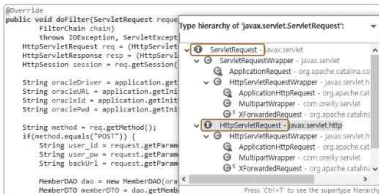
⑫

⑬

로그인 처리용 필터 클래스

이클립스에서 클래스 상속 구조 확인하기

이클립스에서 특정 클래스의 상속 구조를 보고 싶다면 해당 코드에 커서를 둔 상태에서 **Ctrl + t**를 눌러줍니다. 다음 그림은 ServletRequest의 상속 구조를 확인한 결과입니다. ServletRequest는 HttpServletRequest의 부모 인터페이스임을 알 수 있습니다.



■ 모델1 방식 회원제 게시판과 연동

- 회원제 게시판에서는 로그인 확인용 IsLoggedIn.jsp 파일을 로그인에 필요한 모든 페이지 상단에 인클루드 해야만 구현 가능
- 하지만 필터를 활용하면 매핑 정보만 변경하면 구현 가능

```
@WebFilter(urlPatterns={"/09PagingBoard/Write.jsp",  
    "/09PagingBoard/Edit.jsp", "/09PagingBoard/DeleteProcess.jsp"}) ❶  
public class IsSessionFilter implements Filter {  
    @Override  
    public void doFilter(ServletRequest request, ServletResponse response,  
        FilterChain chain) throws IOException, ServletException { ❷  
        HttpServletRequest req = (HttpServletRequest)request; ❸  
        HttpServletResponse resp = (HttpServletResponse)response;  
  
        HttpSession session = req.getSession();
```

session 객체를 얻어와서 세션영역에 로그인 정보가 있는지 확인한다.

IsSessionFilter.java

페이징 처리가 되어있는 게시판의 글 쓰기, 수정하기, 삭제처리하기 페이지를 매핑해서 로그인 확인 처리를 구현

init()와 destroy() 메서드에서는 처리할 내용이 없어 생략

■ 모델1 방식 회원제 게시판과 연동

96	페이지 처리-91	musthave	0	2021-08-17
1 2 3 4 5 [다음 블록] [마지막 페이지]				<input type="button" value="글쓰기"/>

리소스인 Write.jsp 파일이 실행 되기 전에 필터가 먼저 요청을 확인한 후 경고창을 띄움

```
/15FilterListener/LoginFilter.jsp?backUrl=/MustHaveJSP/09PagingBoard/Write.jsp
```

로그인에 성공하면 로그인 전에 요청했던 글쓰기 페이지로 이동

처음에는 로그아웃 상태이므로 글쓰기를 누르면 경고창이 뜬다



주소표시줄을 보면 다음과 같이 backUrl이 매개변수로 추가되어 있는 것을 볼 수 있음



■ 모델1 방식 회원제 게시판과 연동

- 로그인 페이지를 직접 실행하는 경우
 - backUrl 없음
 - 로그인 후 로그인 페이지로 이동
- IsSessionFilter 클래스에 의해 로그인 페이지로 이동한 경우
 - getRequestURI() 메서드로 최초 요청된 페이지의 URL을 가져옴
 - backUrl 매개변수에 값으로 할당
 - 로그인 후 backUrl에 지정된 페이지로 이동

■ 리스너란..??

- 리스너는 사전적 의미로 청취자, 즉 소리를 듣는 사람을 의미
- 웹 애플리케이션에서 발생하는 다양한 이벤트(event)를 맡아 처리해주는 역할을 함
- 마우스 클릭(click), 키보드 입력(keydown), 웹 애플리케이션의 시작 및 종료 등의 이벤트 발생을 감지하는 인터페이스를 가리켜 리스너(Listener)라고 한다

■ 리스너의 종류

이벤트 소스	이벤트 리스너	설명
ServletContext	ServletContextListener	웹 애플리케이션의 시작 및 종료 시 발생하는 이벤트 감지
	ServletContextAttributeListener	application 내장 객체를 통해 속성을 추가, 수정, 삭제할 때 발생하는 이벤트 감지
HttpSession	HttpSessionListener	세션의 시작, 종료 시 발생하는 이벤트 감지
	HttpSessionAttributeListener	session 내장 객체를 통해 속성을 추가, 수정, 삭제할 때 발생하는 이벤트 감지

ServletRequest	ServletRequestListener	클라이언트의 요청 및 서버의 응답 시 ServletRequest 객체의 생성 및 제거 이벤트 감지
	ServletRequestAttributeListener	ServletRequest 객체에 속성을 추가, 수정, 제거할 때 발생하는 이벤트 감지

■ web.xml에서 리스너 등록하기

```
# 리스너로 사용할 클래스
public class 리스너클래스명 implements XxxListener {
    // 실행할 코드;
}

# web.xml
<listener>
    <listener-class>패키지를 포함한 리스너 클래스명</listener-class>
</listener>
```

■ 리스너 클래스 작성

- 웹 애플리케이션의 시작과 종료 이벤트를 받아 간단한 메시지를 출력하는 리스너를 작성

```
public class ContextListener implements ServletContextListener { ❶
    @Override
    public void contextInitialized(ServletContextEvent sce) { ❷
        Enumeration<String> apps = sce.getServletContext().getInitParameterNames(); ❸
        while (apps.hasMoreElements()) { ❹
            System.out.println("[리스너] 컨텍스트 초기화 매개변수 생성 : "
                + apps.nextElement());
        }
    }
    @Override
    public void contextDestroyed(ServletContextEvent sce) { ❺
        Enumeration<String> apps = sce.getServletContext().getInitParameterNames();
        while (apps.hasMoreElements()) {
            System.out.println("[리스너] 컨텍스트 초기화 매개변수 소멸 : "
                + apps.nextElement());
        }
    }
}
```

ContextListener.java

- ❶ ServletContextListener 리스너를 구현하여 클래스 정의
- ❷ 웹 애플리케이션 시작 이벤트를 감지
- ❸ web.xml에 정의된 컨텍스트 초기화 매개변수 목록을 얻어와서 출력
- ❹ 웹 애플리케이션 종료 이벤트를 감지

■ web.xml 작성(리스너 등록)

예제 15-10] webapp/WEB-INF/web.xml

```
<listener>
  <listener-class>listener.ContextListener</listener-class> ❶
</listener>
</>
```

<listener-class> 요소에 패키지를 포함한 클래스명을 입력하기만 하면 리스너가 등록됨

Servers 뷰에서 톰캣을 재시작하면 콘솔에는 다음과 같은 내용이 출력됨

```
Tomcat v10.1 Server at localhost (Apache Tomcat/Catalina Core Development Work-17)
[리스너]컨텍스트 초기화 매개변수 생성:OracleURL
[리스너]컨텍스트 초기화 매개변수 생성:POSTS_PER_PAGE
[리스너]컨텍스트 초기화 매개변수 생성:PAGES_PER_BLOCK
[리스너]컨텍스트 초기화 매개변수 생성:OracleId
[리스너]컨텍스트 초기화 매개변수 생성:CHAT_ADDR
[리스너]컨텍스트 초기화 매개변수 생성:OracleDriver
[리스너]컨텍스트 초기화 매개변수 생성:OraclePwd
BasicFilter -> init() 호출됨 : BasicFilter
6월 24, 2023 8:22:10 오후 org.apache.catalina.startup.Catalina start
INFO: 서버가 [2385] 밀리초 내에 시작되었습니다.
```

web.xml에 등록해둔 컨텍스트 초기화 매개 변수 전체가 출력

■ 애너테이션으로 리스너 등록하기

- 리스너 등록에는 @WebListener 애너테이션을 이용

```
@WebListener
public class 리스너클래스명 implements XxxListener, YyyListener {
    // 실행할 코드;
}
```

```
@WebListener ❶
public class SessionAttrListener implements HttpSessionAttributeListener { ❷
    @Override
    public void attributeAdded(HttpSessionBindingEvent se) { ❸
        System.out.println("[리스너] 세션 속성 추가 : "
            + se.getName() + " = " + se.getValue());
    }
}
```

SessionAttrListener.java

- ❶ @WebListener 애너테이션만 붙여주면 리스너 등록 완료
- ❷ HttpSessionAttributeListener 인터페이스는 세션 영역의 속성이 변경될 때의 이벤트를 감지하는 리스너

■ 애너테이션으로 리스너 등록하기

```
@Override
public void attributeRemoved(HttpSessionBindingEvent se) { ④
    System.out.println("[리스너] 세션 속성 제거 : "
        + se.getName() + " = " + se.getValue());
}

@Override
public void attributeReplaced(HttpSessionBindingEvent se) { ⑤
    System.out.println("[리스너] 세션 속성 변경 : "
        + se.getName() + " = " + se.getValue());
}
}
```

SessionAttrListener.java

③④⑤ 메서드는 각각 세션에 속성이 추가, 제거, 변경되는 이벤트를 감지

■ 리스너 동작 확인용 JSP 작성

```
<%  
String mode = request.getParameter("mode"); ❶  
if (mode != null && mode.equals("1")) { ❷  
    session.setAttribute("mySession", "세션 영역");  
}  
else if (mode != null && mode.equals("2")) {  
    session.removeAttribute("mySession");  
}  
else if (mode != null && mode.equals("3")) {  
    session.invalidate();  
}  
%>  
<script>  
function formSubmit(form, modeValue) { ❸  
    form.mode.value = modeValue; ❹  
    form.submit(); ❺  
}  
</script>
```

MyListener.jsp

❷ 매개변수로 전달된 mode의 값에 따라 세션을 추가, 삭제, 무효화 하는 코드 실행

❸ Javascript 함수로 mode값을 변경한 후 전송

■ 동작 확인

리스너 활용하기

세션 속성 저장

세션 속성 삭제

세션 전체 삭제

실행 화면에서 [세션 속성 저장] → [세션 속성 삭제] → [세션 속성 저장] → [세션 전체 삭제] 순서로 클릭



■ HttpSessionListener 인터페이스

- 세션의 생성과 소멸 이벤트를 감지
- 이런 특성을 이용해서 접속자 수를 확인하는 세션 카운터 제작

@WebListener

```
public class SessionListener implements HttpSessionListener { ❶
    private int sessionCount; ❷

    @Override
    public void sessionCreated(HttpSessionEvent se) { ❸
        sessionCount++; ❹
        System.out.println("[리스너] 세션 생성 : " + se.getSession().getId()); ❺
        System.out.println("[리스너] 세션 카운트 : " + this.sessionCount);
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        sessionCount--; ❻
        System.out.println("[리스너] 세션 소멸 : " + se.getSession().getId());
        System.out.println("[리스너] 세션 카운트 : " + this.sessionCount);
    }
}
```

SessionListener.java

❶ 리스너 생성

❸ 새로운 세션이 생성되면 카운트 1 증가

❹ 세션 객체가 소멸되면 카운트 1 감소

동작 확인

Tomcat v10.1 Server at localhost [Apache Tomcat] C:\01\Development\jdk-17\bin\java.exe (2023. 6. 24 오후 8:30:27)

INFO: 서버가 [2829] 밀리초 내에 시작되었습니다.

[리스너] 세션 생성: 82D6A3BEA67E8C242B2664E7E180851E

[리스너] 세션카운트: 1

[리스너] 세션 속성 추가: mySession=세션 영역

[리스너] 세션 속성 변경: mySession=세션 영역

[리스너] 세션 속성 제거: mySession=세션 영역

[리스너] 세션 소멸: 82D6A3BEA67E8C242B2664E7E180851E

[리스너] 세션카운트: 0

[리스너] 세션 생성: F8D48216F1495FCBB42DAE9B5EFB845A

[리스너] 세션카운트: 1

[리스너] 세션 소멸: F8D48216F1495FCBB42DAE9B5EFB845A

[리스너] 세션카운트: 0

초초 실행

1 세션 속성 저장

2 세션 속성 저장

3 세션 속성 삭제

4 세션 전체 삭제

5 세션 전체 삭제

실행 화면에서 [세션 속성 저장] → [세션 속성 삭제] → [세션 전체 삭제] 순서로 눌러보며 콘솔의 출력 결과를 확인

웹 브라우저는 탭을 여러 개 열어도 세션을 공유하므로 세션 카운트는 1을 넘지 못함.
따라서 두번째 웹브라우저로 접속한 후 테스트

Tomcat v10.1 Server at localhost [Apache Tomcat] C:\01\Development\jdk-17\bin\java.exe (2023. 6. 24. 오후 8:30:27)

[리스너] 세션 생성: 86C1ED6A4CD1B08E17A8CB4D43AF0ED4

[리스너] 세션카운트: 1

[리스너] 세션 생성: 043205283D6D8E1D10DDE3CA59ED38EB

[리스너] 세션카운트: 2

크롬에서 접속

엣지에서 접속

■ 핵심요약

- 웹 애플리케이션은 보통 수많은 JSP/서블릿으로 구성됩니다. 이때 필터를 활용하면 한글 인 코딩 처리, 로그인, 로깅 등과 같은 공통 처리를 하나의 파일에서 관리할 수 있습니다.
- 필터는 `jakarta.servlet.Filter` 인터페이스를 구현해 작성하며, `web.xml`이나 애너테이션 으로 요청명과 매핑해 사용합니다.
- 리스너는 웹 컨테이너에서 발생하는 다양한 이벤트를 감지할 수 있습니다. 리스너를 활용하 면 웹 애플리케이션에 필요한 데이터 초기화나 속성값 변경 추적 등을 손쉽게 처리할 수 있습니다.
- 리스너는 `jakarta.servlet` 패키지의 다양한 `XxxListener` 인터페이스를 구현해 작성하며, `web.xml`이나 애너테이션으로 등록해 사용합니다.

