

# 예외처리, 세션, 쿠키, 필터

524730  
2020년 봄학기  
5/6/2020  
박경신

## 예외처리 개요

### ▣ 예외 처리

- 프로그램이 처리되는 동안 특정한 문제가 발생했을 때 처리를 중단하고 다른 처리를 하는 것으로 오류 처리라고도 함
- 웹 사이트를 이용하다가 주소를 잘못 입력하면 오류 페이지를 보게 됨
  - ▣ 웹 서버가 제공하는 오류 페이지로 해당 페이지에 발생한 오류, 디렉토리 구조, 톰캣 버전 등의 정보가 나타나 있기 때문에 웹 보안이 취약하여 쉽게 해킹 당할 수 있음.



## 예외처리 방법

### ▣ 예외 처리

- 웹 애플리케이션 실행 도중에 발생할 수 있는 오류에 대비한 예외 처리 코드를 작성하여 비정상적인 종료를 막을 수 있음
- 예외 처리 방법의 종류

예외처리 방법	설명
page 지시어 이용	errorPage와 isErrorPage 속성을 이용
web.xml 파일 이용	<error-code> 또는 <exception-type> 요소를 이용
try/catch/finally를 이용	자바 언어의 예외처리 구문을 이용

## page 지시어를 사용한 예외처리

### ▣ errorPage 속성으로 오류 페이지 호출하기

- 오류 페이지를 호출하는 page 디렉티브 태그의 속성

```
<%@ page errorPage="오류페이지.jsp" %>
```

- JSP 페이지가 실행되는 도중에 오류가 발생하면 웹 서버의 기본 오류 페이지를 대신하여 errorPage 속성에 설정한 페이지가 오류 페이지로 호출

## page 지시어를 사용한 예외처리

### ▫ errorPage 속성으로 오류 페이지 호출하기

```
<%-- welcome.jsp --%>
<%@ page errorPage="errorPage.jsp" %>
<html>
  <head>
    <title>isErrorPage and errorPage page directive example</title>
  </head>
  <body>
    <%= 0/0 %>
  </body>
</html>
```

## page 지시어를 사용한 예외처리

### ▫ errorPage 속성으로 오류 페이지 호출하기

```
<%-- errorPage.jsp --%>
<%@ page isErrorPage="true" %>
<html>
  <head>
    <title>Exception</title>
  </head>
  <body>
    오류가 발생하였습니다. <%= exception.toString() %>
  </body>
</html>
```

## page 지시어를 사용한 예외처리

### ▫ isErrorPage 속성으로 오류 페이지 만들기

- 현재 JSP 페이지를 오류 페이지로 호출하는 page 디렉티브 태그의 속성
  - 이때 오류 페이지에서 **exception** 내장 객체를 사용할 수 있음

```
<%@ page isErrorPage="true" %>
```

exception 내장 객체 메소드	설명
String getMessage()	오류 이벤트와 함께 들어오는 메시지를 출력
String toString()	오류 이벤트의 <code>toString()</code> 을 호출하여 간단한 오류 메시지를 확인
String printStackTrace()	오류 메시지의 발생 근원지를 찾아 단계별로 오류를 출력

## page 지시어를 사용한 예외처리

### ▫ isErrorPage 속성으로 오류 페이지 만들기

```
<%-- errorPage.jsp --%>
<%@ page contentType="text/html; charset=utf-8" %>
<%@ page isErrorPage="true" %>
<html>
  <head>
    <title>Exception</title>
  </head>
  <body>
    오류가 발생하였습니다.
    오류 유형: <%= exception.getClass().getName() %>
    오류 메시지: <%= exception.getMessage() %>
  </body>
</html>
```

## web.xml 파일을 사용한 예외처리

### □ web.xml 파일을 이용한 예외 처리

- <error-page>...</error-page> 요소 내에 처리할 오류 코드나 오류 유형 및 오류 페이지를 호출

요소	설명
<error-code>	오류 코드를 설정하는데 사용
<exception-type>	자바 예외 유형의 정규화된 클래스 이름을 설정하는데 사용
<location>	오류 페이지의 URL을 설정하는데 사용

## web.xml 파일을 사용한 예외처리

### □ 주요 오류 코드

코드	설명
200	OK. 요청이 정상적으로 처리
307	임시로 페이지가 리다이렉트
400	Bad Request, 요청 실패. 클라이언트의 요청이 잘못된 구문으로 구성
401	Unauthorized, 접근이 허용되지 않음
404	Not Found, 문서를 찾을 수 없음. 지정된 URL을 처리하기 위한 자원이 존재하지 않음(페이지가 없음)
405	Method not allowed, 요청된 메소드가 허용되지 않음
500	서버 내부의 에러 (JSP에서 예외가 발생하는 경우)
503	서버가 일시적으로 서비스를 제공할 수 없음(서버 과부하나 보수 중인 경우)
505	HTTP Version Not Supported

## web.xml 파일을 사용한 예외처리

### □ web.xml 오류 코드로 오류 페이지 호출하기

- 오류 코드는 웹 서버가 제공하는 기본 오류 페이지에 나타나는 404, 500과 같이 사용자의 요청이 올바르지 않을 때 출력되는 코드로 응답 상태 코드라고도 함
- JSP 페이지에서 발생하는 오류가 web.xml 파일에 설정된 오류 코드와 일치하는 경우 오류 코드와 오류 페이지를 보여줌

```
<web-app ...>
...
<error-page>
  <error-code>에러코드</error-code>
  <location>에러페이지의 URI</location>
</error-page>
...
</web-app>
```

## web.xml 파일을 사용한 예외처리

### □ web.xml 예외 유형별 오류 페이지 호출하기

- 예외 유형에 따른 오류 페이지 호출 방법은 JSP 페이지가 발생시키는 오류가 web.xml 파일에 설정된 예외 유형과 일치하는 경우 예외 유형과 오류 페이지를 보여줌

```
<web-app ...>
...
<error-page>
  <exception-type>예외클래스명</exception-type>
  <location>에러페이지의 URI</location>
</error-page>
...
</web-app>
```

## web.xml 파일을 사용한 예외처리

### ▣ 주요 예외 유형

예외유형	설명
ClassNotFoundException	클래스를 찾지 못했을 때 발생
NullPointerException	null 객체를 사용하려고 시도할 때 발생
ClassCastException	변환할 수 없는 클래스로 객체로 변환할 때 발생
OutOfMemoryException	메모리 부족으로 메모리를 확보하지 못했을 때
StackOverflowError	스택 오버플로일 때 발생
ArrayIndexOutOfBoundsException	배열의 범위를 벗어난 접근할 때 발생
IllegalArgumentException	메소드 인자 유형을 잘못 사용했을 때 발생
IOException	입출력 오류에 의해 발생
NumberFormatException	부적절한 문자열을 숫자로 변환하려 할 때 발생
ArithmaticException	산술 연산 오류에 의해 발생 (0을 정수로 나눔)

## 에러 페이지 우선 순위

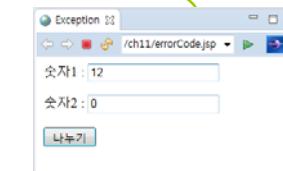
1. page 디렉티브의 errorPage 속성에서 지정한 에러 페이지를 보여줌
2. JSP 페이지에서 발생한 예외 타입이 web.xml 파일의 <exception-type>에서 지정한 예외 타입과 동일한 경우 지정한 에러 페이지를 보여줌
3. JSP 페이지에서 발생한 에러 코드가 web.xml 파일의 <error-code>에서 지정한 에러 코드와 동일한 경우 지정한 에러 페이지를 보여줌
4. 아무것도 해당되지 않을 경우 웹 컨테이너가 제공하는 기본 에러 페이지를 보여줌

## web.xml 파일을 사용한 예외처리

```
<web-app>
  <error-page>
    <error-code>404</error-code>
    <location>/errorCode404.jsp</location>
  </error-page>
  <error-page>
    <error-code>500</error-code>
    <location>/errorCode500.jsp</location>
  </error-page>
  <error-page>
    <exception-type>java.lang.ArithmaticException</exception-type>
    <location>/exceptionType.jsp</location>
  </error-page>
</web-app>
```

## web.xml 파일을 사용한 예외처리

```
<body>
  <form action="process.jsp" method="post">
    <p>숫자1 : <input type="text" name="num1">
    <p>숫자2 : <input type="text" name="num2">
    <p><input type="submit" value="나누기">
  </form>
</body>
```



```
<body>
<%
  String num1 = request.getParameter("num1");
  String num2 = request.getParameter("num2");
  int a = Integer.parseInt(num1);
  int b = Integer.parseInt(num2);
  int c = a / b;
  out.print(num1 + " / " + num2 + " = " + c);
%
</body>
```

## web.xml 파일을 사용한 예외처리

```
<body>
Error code 404 오류가 발생하였습니다.
요청한 페이지는 존재하지 않습니다!!
</body>
```

```
<body>
Error code 500 에러 Internal Server Error, 서버 내부 오류.
이 에러는 웹 서버가 요청사항을 수행할 수 없을 경우에 발생합니다.
</body>
```

```
<body>
<p>예외 : <%=exception%>
<p>toString() : <%=exception.toString()%>
<p>getClass().getName() : <%=exception.getClass().getName()%>
<p>getMessage() : <%=exception.getMessage()%>
</body>
```

## try-catch-finally를 이용한 예외 처리

- try-catch-finally를 이용한 예외 처리
  - 자바의 예외 처리 구문으로 스크립틀릿 태그에 작성

```
<%
try {
    // 예외가 발생할 수 있는 실행문
} catch (예외유형) {
    // 예외처리문
} finally {
    // 예외와 상관없이 무조건 실행되는 문 (생략가능)
}
%>
```

- try 구문에는 예외가 발생할 수 있는 코드를 작성하고, catch 구문에는 오류가 발생할 수 있는 예외 사항을 예측하여 오류를 처리하는 코드를 작성
- finally 구문에는 try 구문이 실행된 후 실행할 코드를 작성하는데 이는 생략 가능

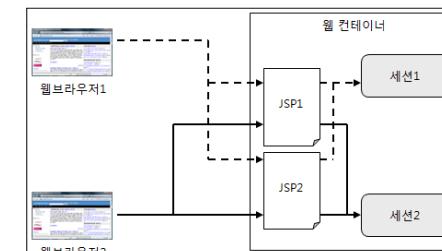
## try-catch-finally를 이용한 예외 처리

```
<body>
<%
try {
    int num = 20 / 0;
} catch (ArithmaticException e) {
    RequestDispatcher dispatcher
        = request.getRequestDispatcher("errorPage.jsp");
    dispatcher.forward(request, response);
}
%>
</body>
```

## 세션의 개요

### 세션(session)

- 클라이언트와 웹 서버 간의 상태를 지속적으로 유지하는 방법
  - 예를 들면 웹 쇼핑몰에서 장바구니나 주문 처리와 같은 회원 전용 페이지의 경우 로그인 인증을 통해 사용 권한을 부여. 그래서 다른 웹 페이지에 갔다가 되돌아와도 로그인 상태가 유지되므로 회원 전용 페이지를 계속 사용할 수 있음. 이렇게 사용자 인증을 통해 특정 페이지를 사용할 수 있도록 권한 상태를 유지하는 것.
- 오직 서버에서만 생성
- 클라이언트마다 세션이 생성



## 세션의 개요

### ▣ 세션(session)

- 웹 서버에서만 접근이 가능하므로 보안 유지에 유리하며 데이터를 저장하는 데 한계가 없음
- 오직 웹 서버에 존재하는 객체로 웹 브라우저마다 하나씩 존재하므로 웹 서버의 서비스를 제공받는 사용자를 구분하는 단위가 됨
- 웹 브라우저를 닫기 전까지 웹 페이지를 이동하더라도 사용자의 정보가 웹 서버에 보관되어 있어 사용자 정보를 잊지 않음.

## 세션의 동작 과정

### ▣ 세션 동작 과정

1. 클라이언트가 서버에 접속 시 세션 ID를 발급.
2. 서버에서는 클라이언트로 발급해준 세션 ID (JSESSIONID)를 저장. 즉, 세션을 구별하기 위해 ID가 필요하고 그 ID만 쿠키를 이용해서 저장해놓음. 쿠키는 자동으로 서버에 전송되니까 서버에서 세션아이디에 따른 처리를 할 수 있음
3. 클라이언트는 다시 접속할 때, 이 JSESSIONID를 이용해서 세션 ID값을 서버에 전달. 예를 들면, 게시판에 글을 작성할 때 작성 버튼을 누르면 세션에 있는 아이디를 참조해서 작성자를 지정하게 함

## 세션의 생성

### ▣ page 지시어의 session 속성 값을 true로 지정

- 세션이 존재하지 않을 경우 세션이 생성되고, 세션이 존재할 경우 이미 생성된 세션을 사용

### ▣ session 내장 객체를 이용해서 세션에 접근

- session의 기본 값은 true이므로 false로 하지 않는 이상 항상 세션 사용

```
<%@ page contentType = ... %>
<%@ page session = "true" %>
<%
...
    session.setAttribute("userInfo", userInfo);
...
%>
```

### ▣ 속성 이용해서 클라이언트 관련 정보 저장

## 세션의 생성

### ▣ 세션 생성

- session 내장 객체의 setAttribute() 메소드를 사용
- setAttribute() 메소드를 이용하여 세션의 속성을 설정하면 계속 세션 상태를 유지할 수 있음. 만약 동일한 세션의 속성 이름으로 세션을 생성하면 마지막에 설정한 것이 세션 속성 값이 됨.

#### void setAttribute(String name, Object value)

- 첫 번째 매개변수 name은 세션으로 사용할 세션 속성 이름을 나타내며, 세션에 저장된 특정 값을 찾아오기 위한 키로 사용.
- 두 번째 매개변수 value는 세션의 속성 값
- 세션 속성 값은 Object 객체 타입만 가능하기 때문에 int, double, char 등의 기본 타입은 사용할 수 없음

```
session.setAttribute("userName", "Park");
session.setAttribute("userAge", 10);
```

## 세션의 정보 얻기

### □ 단일 세션 정보 얻기

- 세션에 저장된 하나의 세션 속성 이름에 대한 속성 값을 얻어오려면 `getAttribute()` 메소드를 사용
- `getAttribute()` 메소드는 반환 유형이 `Object` 형이므로 반드시 형변환을 하여 사용해야 함

#### `Object getAttribute(String name)`

- 첫 번째 매개변수 `name`은 세션에 저장된 세션 속성 이름
- 해당 속성 이름이 없는 경우 `null`을 반환

```
String name = (String)session.getAttribute("userName");
int age = (Integer)session.getAttribute("userAge");
```

## 세션의 정보 얻기

### □ 다중 세션 정보 얻기

- `getAttributeNames()` 메소드를 사용하여 다중 세션 정보를 얻음

#### `Enumeration getAttributeNames()`

```
Enumeration items = session.getAttributeNames();
while(items.hasMoreElements()) {
    String name = items.nextElement().toString();
    String value = session.getAttribute(name).toString();
}
```

## 세션의 삭제

### □ 단일 세션 삭제하기

- 세션에 저장된 하나의 세션 속성 이름을 삭제하려면 `removeAttribute()` 메소드를 사용

#### `void removeAttribute(String name)`

```
session.removeAttribute("username");
session.removeAttribute("userAge");
```

### □ 다중 세션 삭제하기

- 세션에 저장된 모든 세션 속성 이름을 삭제하려면 `invalidate()` 메소드를 사용

#### `void invalidate()`

- 세션이 종료되면 기존에 생성된 세션이 삭제
- 이후 접근 시 새로운 세션 생성됨

```
session.invalidate(); // 세션 해제
```

## 세션 유효 시간 설정

### □ 세션 유효 시간

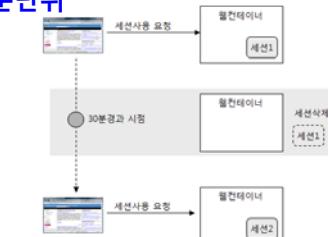
- 세션을 유지하기 위한 세션의 일정 시간
- 웹 브라우저에 마지막 접근한 시간부터 일정 시간 이내에 다시 웹 브라우저에 접근하지 않으면 자동으로 세션이 종료
- 세션 유효 시간을 설정하기 위해 `session` 내장 객체의 `setMaxInactiveInterval()` 메소드를 사용

#### `void setMaxInactiveInterval(int interval) // 초단위`

### □ 마지막 세션 사용 이후 유효 시간이 지나면 자동 종료

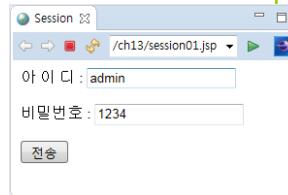
- WEB-INF/web.xml 파일에서 지정

```
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
```



## 세션의 생성

```
<body>
<%
String id = request.getParameter("userID");
String pw = request.getParameter("userPW");
if (id.equals("admin") && pw.equals("1234")) {
    session.setAttribute("userID", id);
    session.setAttribute("userPW", pw);
    out.println("Session Successful~");
    out.println("Welcome " + id);
    response.sendRedirect("loginSuccess.jsp");
}
else {
    out.println("Session Failed");
}
%>
</body>
```



## 세션의 정보

```
<body>
<%
String userID = (String)session.getAttribute("userID");
String userPW = (String)session.getAttribute("userPW");
if (userID != null && userPW != null) {
    out.println("Session Successful~ <br>");
    out.println("Welcome " + userID + " " + userPW + "<br>");
    String sessinID = session.getId();
    long lastTime = session.getLastAccessedTime();
    long startTime = session.getCreationTime();
    long elapsedTime = (lastTime - startTime) / 60000;
    out.println("세션 아이디 : " + sessinID + "<br>");
    out.println("요청 시작 시간 : " + startTime + "<br>");
    out.println("요청 마지막 시간 : " + lastTime + "<br>");
    out.println("웹 사이트에서 경과 시간 : " + elapsedTime + "<br>");
}
%>
</body>
```

## 세션의 정보

```
<body>
<%-- 모든 세션 속성 이름과 속성 값 출력 -->
<%
Enumeration e = session.getAttributeNames();
int i = 0;
while(e.hasMoreElements()) {
    i++;
    String name = e.nextElement().toString();
    String value = session.getAttribute(name).toString();
    out.println("session attribute name[" + i + "]=" + name + "<br>");
    out.println("session attribute value[" + i + "]=" + value + "<br>");
}
%>
</body>
```

## 세션의 삭제

```
<%
session.invalidate(); // 모든 세션 삭제
response.sendRedirect("index.jsp");
%>
```

## 쿠키의 개요

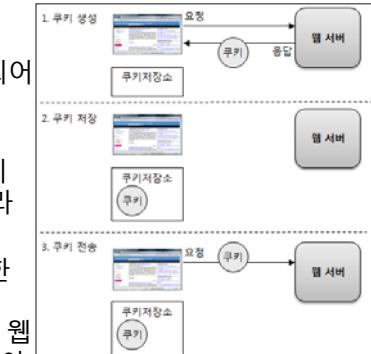
### ▣ 쿠키(cookie)

- 클라이언트와 웹 서버 간의 상태를 지속적으로 유지하는 방법
- 쿠키는 세션과 달리 상태 정보를 웹 서버가 아닌 클라이언트에 저장
  - ▣ 예를 들어 어떤 웹 사이트를 처음 방문한 사용자가 로그인 인증을 하고 나면 아이디와 비밀번호를 기록한 쿠키가 만들어지고 그 다음부터 사용자가 그 웹 사이트에 접속하면 별도의 절차를 거치지 않고 쉽게 접속할 수 있음
- 클라이언트의 일정 풀더에 정보를 저장하기 때문에 웹 서버의 부하를 줄일 수 있다는 것이 장점
- 반면에 웹 브라우저가 접속했던 웹 사이트에 관한 정보와 개인 정보가 기록되기 때문에 보안에 문제가 있음
- 쿠키의 제한
  - ▣ 클라이언트에 300개까지 쿠키저장 가능, 하나의 도메인당 20개 가질 수 있음. 하나의 쿠키값은 4KB 까지 저장

## 쿠키의 동작 과정

### ▣ 쿠키(cookie) 동작 과정

1. **쿠키 생성** 단계 - 쿠키를 사용하려면 먼저 쿠키를 생성해야 함. 쿠키는 주로 웹 서버 측에서 생성. 생성된 쿠키는 응답 데이터에 함께 저장되어 웹 브라우저에 전송됨.
2. **쿠키 저장** 단계 - 웹 브라우저는 응답 데이터에 포함된 쿠키를 쿠키 저장소에 보관. 쿠키는 종류에 따라 메모리나 파일로 저장.
3. **쿠키 전송** 단계 - 웹 브라우저는 한번 저장된 쿠키를 요청이 있을 때마다 웹 서버에 전송. 웹 서버는 웹 브라우저가 전송한 쿠키를 사용하여 필요한 작업을 수행할 수 있음.



## 쿠키의 구성

### ▣ 쿠키(cookie) 구성 요소

- 이름 - 각각의 쿠키를 구별하는 데 사용되는 이름
- 값 - 쿠키의 이름과 관련된 값
- 유효시간 - 쿠키의 유지 시간
- 도메인 - 쿠키를 전송할 도메인
- 경로 - 쿠키를 전송할 요청 경로

### ▣ 쿠키 이름의 제약

- 쿠키의 이름은 **아스키 코드의 알파벳과 숫자만**을 포함 가능
- **콤마(,), 세미콜론(;), 공백(' ')** 등의 문자는 불가능
- '\$'로 시작할 수 없음

## 쿠키와 세션의 차이

구분	쿠키	세션
사용클래스	Cookie 클래스	HttpSession 인터페이스
저장 형식	텍스트	Object형
저장 장소	클라이언트	서버(세션아이디만 클라이언트에 저장)
종료 시점	쿠키 저장 시 설정(설정하지 않을경우 웹브라우저 종료시 소멸)	정확한 시점을 알 수 없음
리소스	클라이언트의 리소스 사용	서버의 리소스 사용
보안	클라이언트에 저장되므로 사용자 변경이 가능하여 보안에 취약	서버에 저장되어 상대적으로 안정적

## 쿠키의 생성

### ▣ 쿠키 생성

- Cookie 클래스를 이용해서 쿠키를 생성한 후에 반드시 response 내장 객체의 addCookie() 메소드로 쿠키를 설정해야 함
  - 첫 번째 매개변수 name은 쿠키를 식별하기 위한 이름
  - 두 번째 매개변수 value는 쿠키 값

```
<%
    Cookie cookie = new Cookie("cookieName", "cookieValue");
    response.addCookie(cookie);
%>
```

## 쿠키의 정보 얻기

### ▣ 클라이언트가 보낸 쿠키 읽기

- 쿠키 객체가 여러 개일 때는 배열 형태로 가져옴

```
Cookie[] cookies = request.getCookies();
```

### ▣ 쿠키 객체의 정보 얻기

- 쿠키 객체를 얻어왔다면 이 쿠키 객체에 저장된 쿠키 이름과 값을 가져오기 위해 getName(), getValue() 메소드를 사용

메서드	설명
String getName()	쿠키의 이름을 구한다.
String getValue()	쿠키의 값을 구한다.

```
Cookie[] cookies = request.getCookies();

for (int i = 0; i < cookies.length; i++) {
    out.println(cookies[i].getName() + " : " + cookies[i].getValue() + "  
");
}
```

## 쿠키 값의 인코딩/디코딩 처리

### ▣ 쿠키는 값으로 한글과 같은 문자를 가질 수 없음

- 쿠키의 값을 인코딩해서 지정할 필요 있음

### ▣ 쿠키 값의 처리

- 값 설정시 : URLEncoder.encode("값", "euc-kr")
  - 예, new Cookie("name", URLEncoder.encode("값", "euc-kr"));
- 값 조회시 : URLDecoder.decode("값", "euc-kr")
  - Cookie cookie = ...;
  - String value = URLDecoder.decode(cookie.getValue(), "euc-kr");

## 쿠키 값 변경

### ▣ 기존에 존재하는지 확인 후, 쿠키 값 새로 설정

```
Cookie[] cookies = request.getCookies();
if (cookies != null && cookies.length > 0) {
    for (int i = 0; i < cookies.length ; i++) {
        if (cookies[i].getName().equals("name")) {
            Cookie cookie = new Cookie(name, value);
            response.addCookie(cookie);
        }
    }
}
```

## 쿠키의 도메인과 경로

### ▣ 도메인 지정시, 해당 도메인에 쿠키 전달

- Cookie.setDomain()으로 쿠키 설정
- 도메인 형식

- .madvirus.net - 점으로 시작하는 경우 관련 도메인에 모두 쿠키를 전송한다.
- www.madvirus.net - 특정 도메인에 대해서만 쿠키를 전송한다.

### ▣ 웹 브라우저는 도메인이 벗어난 쿠키는 저장하지 않음

### ▣ 쿠키 도메인에 따라 쿠키가 전달



## 쿠키의 경로 / 유효 시간

### ▣ 경로 설정시 해당 경로를 기준으로 쿠키 전달

- 경로 미 설정시, 요청 URL의 경로에 대해서만 쿠키 전달
- 경로 설정시, 설정한 경로 및 그 하위 경로에 대해서 쿠키 전달
- Cookie.setPath()로 경로 설정

### ▣ 유효 시간

- 유효 시간 미 지정시, 웹 브라우저 닫을 때 쿠키도 함께 삭제
- Cookie.setMaxAge()로 쿠키 유효 시간 설정
  - 유효 시간이 지나지 않을 경우 웹 브라우저를 닫더라도 쿠키가 삭제되지 않고, 이후 웹 브라우저를 열었을 때 해당 쿠키 전송됨
  - 유효 시간 : 초 단위로 설정

## 쿠키의 삭제

### ▣ 쿠키의 삭제

- 쿠키의 유효 기간을 결정하는 setMaxAge() 메소드에 유효 기간을 0으로 설정하여 쿠키를 삭제할 수 있음

```
Cookie cookie = new Cookie("userName", "park");
Cookie.setMaxAge(0);
response.addCookie(cookie);
```

## 쿠키 생성

```
<body>
<%
String id = request.getParameter("userID");
String pw = request.getParameter("userPW");
if (id.equals("admin") && pw.equals("1234")) {
    Cookie cookielD = new Cookie("userID", id);
    Cookie cookiePW = new Cookie("userPW", pw);
    response.addCookie(cookielD);
    response.addCookie(cookiePW);
    out.println("Cookie Creation Successful~");
    out.println("Welcome " + id);
    response.sendRedirect("loginSuccess.jsp");
} else {
    out.println("Cookie Creation Failed");
}
%>
</body>
```

## 쿠키 정보

```
<body>
<%
String userID = null; String userPW = null;
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    String name = cookies[i].getName();
    String value = cookies[i].getValue();
    out.println("cookie name["+i+"] : "+cookies[i].getName()+"<br>");
    out.println("cookie value["+i+"] : "+cookies[i].getValue()+"<br>");
    if (name.equals("userID")) userID = value;
    if (name.equals("userPW")) userPW = value;
}
if (userID != null && userPW != null) {
    out.println("Cookie Successful~ <br>");
    out.println("Welcome " + userID + " " + userPW + "<br>");
}
%>
</body>
```

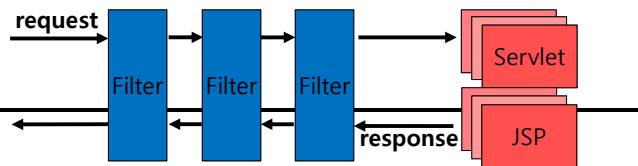
## 쿠키 삭제

```
<%
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    cookies[i].setMaxAge(0);
    response.addCookie(cookies[i]);
}
response.sendRedirect("index.jsp");
%>
```

## 필터

- ▣ **필터(filter)**
- 클라이언트와 서버 사이에서 request와 response 객체를 먼저 받아 **사전/사후 작업 등 공통적으로 필요한 부분**을 처리하는 것
  - 필터는 클라이언트의 **요청**이 웹 서버의 서블릿, JSP, HTML 페이지 같은 정적 리소스에 **도달하기 전**과, 반대로 정적 리소스에서 클라이언트로 **응답하기 전**에 필요한 전처리를 가능하게 함.
  - 필터는 HTTP 요청과 응답을 변경할 수 있는 코드로 재사용이 가능함. 한편 클라이언트와 정적 리소스 사이에 여러 개의 필터로 이루어진 **필터 체인**을 제공하기도 함

필터	기능
Request 필터	인증(사용자인증) 요청정보를 로그 파일로 작성 암호화 인코딩 작업
Response 필터	응답결과 데이터 압축 응답결과에 내용 추가/수정 총 서비스 시간 측정



## 필터 인터페이스

### ▣ Filter 인터페이스

- 필터 기능을 구현하는 데 핵심적인 역할을 하는 인터페이스.
- 클라이언트와 서버의 리소스 사이에 위치한 필터의 기능을 제공하기 위해 자바 클래스로 구현해야 함

```
Import javax.servlet.Filter;
public class AuthenFilter implements Filter {
    // 내부구현
}
```

메서드	설명
init(..)	필터 인스턴스 초기화 메소드
doFilter(..)	필터 기능을 작성하는 메소드
destroy()	필터 인스턴스의 종료 전에 호출되는 메소드

## 필터 인터페이스 구현클래스

### □ **public void init(FilterConfig config) throws ServletException**

- JSP 컨테이너가 필터를 초기화할 때 호출되는 메소드
- init 메소드는 JSP 컨테이너 내에서 초기화 작업을 수행할 필터 인스턴스를 생성한 후 한 번만 호출
- init 메소드는 JSP 컨테이너에 의해 호출되어 필터의 서비스가 시작되고 있음을 나타냄

### □ **public void destroy()**

- 필터 인스턴스를 종료하기 전에 호출하는 메소드
- JSP 컨테이너가 필터 인스턴스를 삭제하기 전에 청소 작업을 수행하는 데 사용되며, 이는 필터로 열린 리소스를 모두 닫을 수 있는 방법
- destroy 메소드는 필터의 수명 동안 한 번만 호출

## 필터 인터페이스 구현클래스

### □ **public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws ServletException**

- JSP 컨테이너가 필터를 리소스에 적용할 때마다 호출되는 메소드
- init 메소드 후에 호출되며, 필터가 어떤 기능을 수행할 필요가 있을 때마다 호출
- ServletRequest 객체는 체인을 따라 전달하는 요청
- ServletResponse 객체는 체인을 따라 전달할 응답
- **FilterChain** 객체는 체인에서 다음 필터를 호출하는 데 사용
  - 만약 호출 필터가 체인의 마지막 필터이면 체인의 끝에서 리소스를 호출

## 필터 인터페이스 구현클래스

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws ServletException {
    System.out.println("JSP 처리 전 필터 수행");

    String name = request.getParameter("name");
    if (name == null || name.equals("")) {
        PrintWriter out = response.getWriter();
        out.println("입력된 name 값은 null!");
        return;
    }
    filterChain.doFilter(request, response);
    System.out.println("JSP 처리 후 필터 수행");
}
```

## web.xml 파일의 필터 구성

### □ **web.xml 파일에 필터를 설정**

- 어떤 필터가 어떤 리소스에 대해 적용되는지 JSP 컨테이너에 알려줌
- **<filter>**와 **<filter-mapping>** 요소를 사용
- web.xml 파일에 여러 개의 필터가 설정되어 있으면 선언된 순서대로 실행

필터 요소	설명
<filter-name>	필터 이름
<filter-class>	자바 클래스 이름
<init-param>	매개 변수와 값

필터맵핑 요소	설명
<filter-name>	필터 이름
<url-pattern>	URL 패턴

## web.xml 파일의 필터 구성

- <filter> 요소
  - 웹 애플리케이션에서 자바 필터와 매개변수를 설정하는 데 사용
- <init-param> 요소
  - web.xml에 설정된 필터의 매개변수와 값을 자바 또는 JSP 코드에서 접근하려면 아래와 같이 작성

```
public void init(FilterConfig config) throws ServletException {  
    adminName = config.getInitParameter("adminName");  
}
```

```
<filter>  
    <filter-name>LogFilter</filter-name>  
    <filter-class>filter.AuthenFilter</filter-class>  
    <init-param>  
        <param-name>adminName</param-name>  
        <param-value>admin</param-value>  
    </init-param>  
</filter>
```

## 필터 예시

### □ login.jsp

```
<%@ page contentType="text/html; charset=utf-8"%>  
<html>  
    <head> <title>Filter</title> </head>  
    <body>  
        <form action="loginProcess.jsp" method="post">  
            <p>이름: <input type="text" name="name">  
            <input type="submit" value="전송">  
        </form>  
    </body>  
</html>
```

## web.xml 파일의 필터 구성

- <filter-mapping> 요소
  - 특정 리소스에 대해 어떤 필터를 사용할지 설정하는 데 사용
  - <filter-mapping> 요소 사용 예시: URL 패턴을 /\*로 설정

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

- <filter-mapping> 요소 사용 예시: URL 패턴을 /filter/loginProcess.jsp로 설정

```
<filter-mapping>  
    <filter-name>LogFilter</filter-name>  
    <url-pattern>/filter/loginProcess.jsp</url-pattern>  
</filter-mapping>
```

## 필터 예시

### □ loginProcess.jsp

```
<%@ page contentType="text/html; charset=utf-8"%>  
<html>  
    <head> <title>Filter</title> </head>  
    <body>  
        <%  
            String name = request.getParameter("name");  
        %>  
        <p>입력된 name 값: <%= name %>  
    </body>  
</html>
```