

Java Web Programming 입문

(Servlet/JSP)

27일차

오늘의 키워드

- ▶ Cookie, Session
 - HTTP
 - URL Rewriting
 - Hidden Form Field
 - 쿠키 (Cookie)
 - 세션 (Session)

HTTP (Hypertext Transfer Protocol)

▶ Web Browser

- NCSA의 Marc Andreessen 제작
- 1993년에 이미 Hyperlink와 Image를 포함
- Web Browser에 사용하는 Protocol이 HTTP

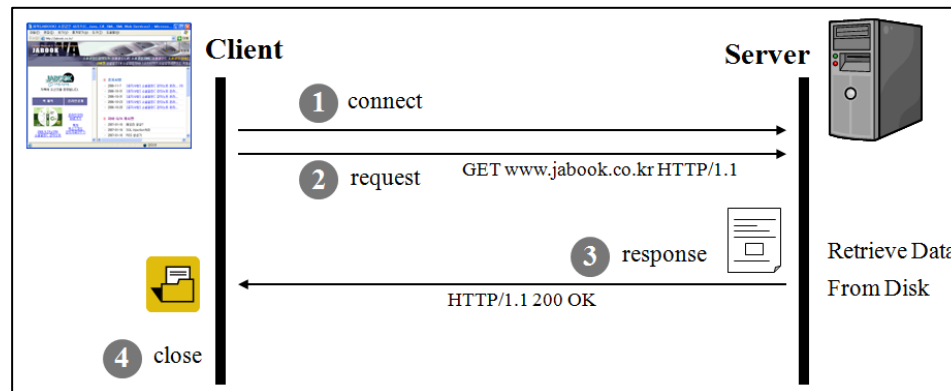
▶ Hypertext

- 다른 문서와 연결된 Text
 - 특정 Text를 Click하면 해당 Web Page로 이동
 - Hypertext를 전송하기 위한 Protocol이 HTTP

▶ Web에서 사용되는 문서 전송 Protocol

HTTP (Hypertext Transfer Protocol)

- ▶ 연결과 동작
 - Client와 Server 연결
 - Client가 Server로 요청
 - Server에서 Client로 응답 메시지 전송
 - 연결 해제



- ▶ HTTP는 Data 요청 후, Data를 받고 나면
 - 바로 Socket 연결 해제
- ▶ 지속적으로 연결을 유지하면서 Data를 주고 받지 않음
 - 필요할 때마다 Socket을 연결하고
 - Data를 받고 바로 연결이 끊어짐

HTTP (Hypertext Transfer Protocol)

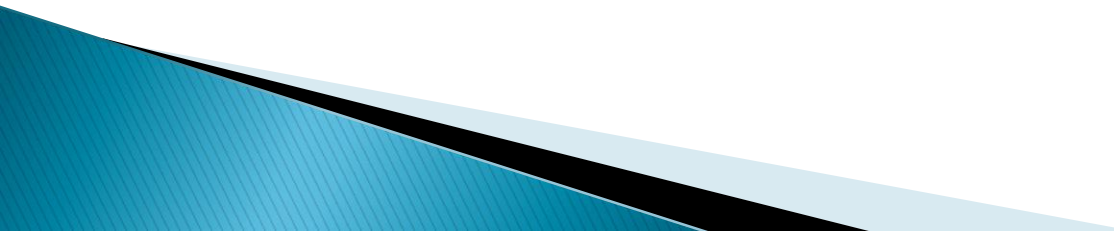
▶ 한계

- HTTP는 연결을 계속 유지하지 않음
 - 작업을 마치면 바로 연결을 해제
 - Connection의 지속성이 없음
 - 각각의 Client를 구분할 수 없음 (단점)
 - 보다 많은 사용자에게 보다 많은 서비스 제공이 가능 (장점)

▶ TCP Protocol의 구분

- HTTP Protocol은
 - TCP/IP (Transmission Control Protocol/Internet Protocol) 의
 - 상위에서 동작하는 Protocol
- TCP/IP는
 - 연결을 유지하지 않는(stateless) Protocol
 - HTTP
 - 한번 연결 후 필요한 작업 후 연결 해제
 - 연결을 유지하는(stateful) Protocol
 - FTP, Telnet
 - 한번의 연결로 계속적인 작업 서비스 가능

HTTP (Hypertext Transfer Protocol)

- ▶ HTTP에서 상태를 유지하기 위한 방법
 - URL Rewriting
 - Hidden Form Field
 - Cookie
 - Session
- 

URL Rewriting

- ▶ Client의 구별 / 정보를 넘기기 위해
 - 하나의 Page에서 다른 Page로 이동할 때
 - URL의 뒷부분에 정보를 포함시킴
 - **Get 방식**에서만 사용 가능
- ▶ 한계/문제점
 - Get 방식을 사용하므로
 - 보낼 수 있는 정보의 양이 제한적
 - 보낼 정보가 URL에 표시되므로 보안상의 문제
 - <http://localhost:8080/MySample/URLRewritingB.jsp?id=OnlyJiny>

URL Rewriting

▶ URLRewriting.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<html>
  <head>
    <title>URL Rewriting</title>
  </head>
  <body>
    <h3>URL Rewriting 로그인</h3>
    <form action="URLRewritingA.jsp" method="get">
      ID : <input type="text" name="id"/>
      <input type="submit" value="Commit"/>
    </form>
  </body>
</html>
```


URL Rewriting

▶ URLRewritingA.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<html>
  <head>
    <title>URL Rewriting</title>
  </head>
  <body>
    <h3>URL Rewriting A</h3>
    <%
      String id = request.getParameter("id");
    %>
    <h3>ID : <%=id%></h3>
    <a href="URLRewritingB.jsp?id=<%=id%>">다음 페이지로</a>
  </body>
</html>
```

▶ URLRewritingB.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<html>
  <head>
    <title>URL Rewriting</title>
  </head>
  <body>
    <h3>URL Rewriting B</h3>
    <h3>ID : <%=request.getParameter("id")%></h3>
  </body>
</html>
```

Hidden Form Field

- ▶ Post 방식의 Form 양식을 사용
 - Data를 넘길 때 Hidden Form 양식에
 - Client를 구분할 수 있는 구분자를 넣어서 전달
 - HTML의 <form>, <input> Tag의
 - 숨겨진 Text Field를 이용

```
<input type="hidden" name="id" value="OnlyJiny" />
```

- ▶ 한계 / 문제점
 - 연결을 유지하기 위해
 - 계속 Hidden Form 양식으로 값을 가지고 다녀야 함
 - Web Browser의 ‘소스보기’ 기능으로
 - Code를 다 볼 수 있으므로 보안에 취약

Hidden Form Field

▶ HiddenFF.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>

<html>
  <head>
    <title>Hidden Form Field</title>
  </head>
  <body>
    <h3>Hidden Form Field 로그인</h3>
    <form action="HiddenFFA.jsp" method="post">
      ID : <input type="text" name="id"/>
      <input type="submit" value="Commit"/>
    </form>
  </body>
</html>
```

Hidden Form Field

▶ HiddenFFA.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<html>
  <head>
    <title>Hidden Form Field</title>
  </head>
  <body>
    <h3>Hidden Form Field A</h3>
    <%
      String id = request.getParameter("id");
    %>
    <h3>ID : <%=id%></h3>
    <form action="HiddenFFB.jsp" method="post">
      <input type="hidden" name="id" value="<%=id%>" />
      <input type="submit" value="Next" />
    </form>
  </body>
</html>
```

▶ HiddenFFB.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<html>
  <head>
    <title>Hidden Form Field</title>
  </head>
  <body>
    <h3>Hidden Form Field B</h3>
    <h3>ID : <%=request.getParameter("id")%></h3>
  </body>
</html>
```

쿠키 (Cookie)

▶ HTTP

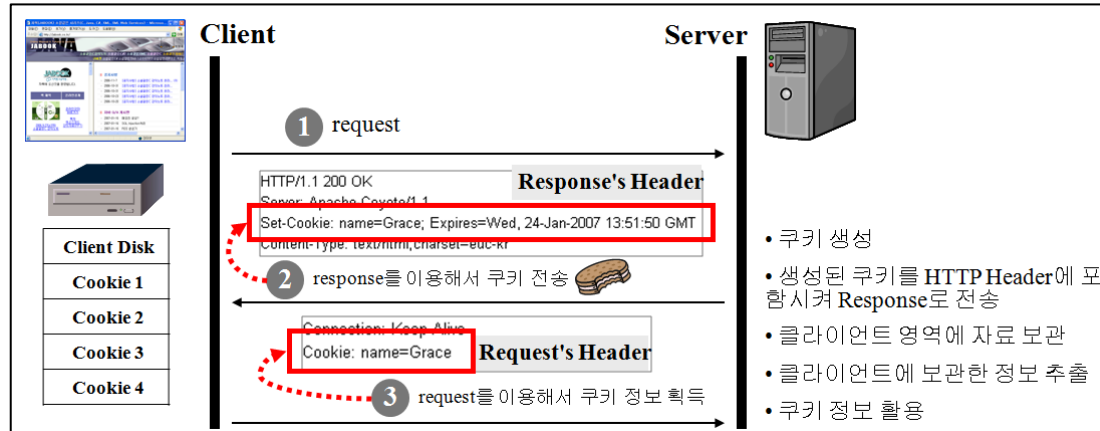
- Client가 정보 요청시 접속이 연결
- 요청한 정보 전송이 끝나면 접속이 끊어짐
 - 비 연결지향적 Protocol
- Server상의 부하나 Network 부하에 대해서는
 - 매우 효율적
 - 여러 개의 요청 처리에는 효율적
- Server에서
 - 사용자가 이전에 어떤 일을 했는지에 대한 정보를 유지 불가
 - 상태 유지 불가

▶ 쿠키 (Cookie)

- Netscape 사에서 정보의 지속성을 위해 고안한 방법
- HTTP Protocol 상에서 Client 상태 정보를
- Client의 **Hard Disk에 저장**한 후 재사용하는 방식
- Server가 Client에 전송하여 저장하는 Text 조각 (과자 부스러기?)
- Text 조각에 Client를 구분할 수 있는 정보를 담아서 전송

쿠키 (Cookie)

▶ 동작



- Client가 Server로 자원을 요청
- Client로 Cookie 설정
 - Web Server에서 JSP, CGI, ASP, Servlet 등을 통해 Client에 Cookie를 설정
- Web Browser는 전달받은 Cookie를 Client에 저장
 - Web Browser는 전달받은 Cookie 값을 Client의 메모리나 파일에 저장한 후 응답 Page 내용 출력
- 이후 Client가 다시 서비스를 요청하면 필요 시 Cookie 정보를 함께 Web Server에 전달
 - 요청 Page의 정보와 Cookie 정보를 함께 Web Server에 전달하고 Web Server는 이 Cookie 값을 받아 처리한 결과를 Client에 전달
 - 한번 Cookie 값이 설정되면, Cookie 값을 설정한 Server로 요청 시 무조건 Cookie 값을 Server로 전달
- Web Server가 전송한 Cookie는 응답 메시지의 HTTP Header에 포함되어 Client로 전송
 - Cookie는 Cookie의 이름과 값으로 되어있고, 필요 시 부가정보가 뒤에 붙음

쿠키 (Cookie)

- ▶ Server에서 Client로 전달되는 HTTP Header의 Cookie 설정 정보

```
Set-Cookie: name=Grace; Expires=Wed, 24-Jan-2007 13:51:50 GMT
```

- name은 Cookie의 이름과 Cookie의 값
- expire는 Cookie의 유효기간

- ▶ Set-Cookie의 부가 정보

쿠키 항목	정보
name = value	쿠키 이름과 쿠키 값 기술
expire = date	쿠키의 유효기간 기술
path = serverPath	쿠키가 전송 될 서버의 URL을 기술, URL이 맞는 경우에만 쿠키를 전송
domain = serverDomain	클라이언트에 저장된 쿠키를 요청 시에 전송될 서버의 도메인을 기술
secure	SSL과 같은 호스트와 보안성이 있는 채널을 사용하는 경우에만 쿠키를 전송

- ▶ Client에서 Server로 Cookie가 전송
 - HTTP Header 항목이 request 메시지에 포함되어 전송
 - Web Server는 request 메시지 내 HTTP Header 에서 Cookie Header 항목을 읽고, Cookie로부터 name 값을 가지고 각각의 Client를 식별

```
Cookie: name=Grace
```

쿠키 (Cookie)

▶ Cookie 설정

◦ 필요한 Class

- JSP에서 Cookie를 Client에 저장하려면?
 - Cookie Class를 사용
 - javax.servlet.http.Cookie Import
 - 하지만 필요 없다.
 - JSP 파일이 Servlet으로 변환될 때 자동으로 Import 되니까!

◦ 생성

- 이름이 name이고 값이 Grace인 Cookie object 생성

```
Cookie cookie = new Cookie("name", "Grace");
```

◦ 저장

- 생성한 Cookie Object를 response 내장 객체의
- addCookie() Method를 이용해서
- response 메시지의 HTTP Header에 Cookie 항목 설정

```
response.addCookie(cookie);
```

◦ 전송

- 이렇게 전송된 Cookie를 받은 Client는 Cookie를 Client에 저장
- 동일한 이름의 Cookie가 Client에 저장되어 있다면 덮어씀

쿠키 (Cookie)

▶ SetCookie.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>

<html>
  <head>
    <title>Set Cookie</title>
  </head>
  <body>
    <h3>쿠키 설정</h3>
    <%
      Cookie cookie = new Cookie("name", "Grace");
      cookie.setMaxAge(60 * 60 * 5);
      response.addCookie(cookie);
    %>
  </body>
</html>
```

- ▶ 실행 전에 Cookie를 삭제
 - 인터넷 익스플로러 > 도구 > 인터넷 옵션 > 쿠키 삭제
- ▶ 실행
 - <http://localhost:8080/MySample/SetCookie.jsp>
 - C:\Documents and Settings\사용자\Cookies

쿠키 (Cookie)

▶ Cookie 저장

- 위와 같이 저장되는 Cookie는 하나의 Domain당 총 20개까지 저장 가능
- 더 많은 Cookie를 저장하려면 배열 형태로 Cookie값을 저장
- 이름 + 값 < 4096bytes

▶ Cookie의 속성

- 값 저장 외 속성
- 아래 속성들은 생략이 가능
- 생략 시 기본 값 설정

속성	설명	형식	기본 값
expires	쿠키가 만료될 시점	Wed, 24-Jan-2007 13:51:50 GMT	브라우저 종료시
domain	쿠키가 사용할 수 있는 도메인	www.jabook.org	쿠키를 저장하는 서버의 주소
path	쿠키를 사용할 수 있는 도메인 내의 주소	/MySample	쿠키를 저장하는 페이지 경로
secure	SSL 상에서만 사용	true false	false

쿠키 (Cookie)

▶ expires

- Cookie가 만료되어 자동 소멸될 시점을 지정
 - 생략 시
 - Web Browser 종료 시 소멸
 - 값이 설정되었을 시
 - 만료 시점이 될 때까지 Client에 존재
 - 설정
 - Cookie의 expires 속성은 setMaxAge() Method를 이용
- ```
void setMaxAge(int expiry)
```
- setMaxAge() Method의 Parameter는 초가 기준
    - 5분 설정
- ```
cookie.setMaxAge(60 * 5);
```
- Browser 동작 시간으로 설정
- ```
cookie.setMaxAge(-1);
```
- 삭제
    - Cookie는 삭제 Method가 없음
    - 유효시간을 0으로 설정해서 삭제하는 효과

```
cookie.setMaxAge(0);
```

# 쿠키 (Cookie)

## ▶ domain

- Cookie가 사용될 Domain을 설정
- 생략 시
  - Cookie를 저장하는 Server의 주소가 설정
- 기본적으로 Cookie는
  - Cookie를 설정한 Web Server에 접속할 때만 Cookie가 전송
  - Cookie를 설정한 Web Server가 아닌 다른 Web Server에도 Cookie를 전송하고 싶을 경우 domain 설정
- setDomain() Method를 이용

```
cookie.setDomain("www.testURL.com");
```

# 쿠키 (Cookie)

## ▶ path

- Cookie를 사용할 수 있는 Domain 내 주소 지정
- 생략 시
  - Cookie를 설정한 Page 경로로 설정
- domain 지정과 마찬가지로 Cookie의 경로를 지정
  - domain, path 속성을 지정하여
    - Cookie가 사용될 구체적인 URL 지정
- setPath() Method 이용

```
cookie.setPath("/testPath1/testPath2");
```

## ▶ Cookie의 속성들은 모두 생략 가능하지만...

- domain, path 속성을 설정해주는 것이 안전
  - 설정하지 않았을 때 해당 Cookie 값을 인식 못하는 경우가 있음

# 쿠키 (Cookie)

## ▶ 제약 조건

- Client에 총 300개 까지의 Cookie 저장 가능
- 하나의 Domain 당 20개 값까지 저장 가능
- 하나의 Cookie 값의 크기는 4096byte 로 제한
- 하나의 Domain에서 설정한 Cookie 값이 20개가 넘어간다면
  - 최근에 가장 적게 사용된 Cookie 부터 삭제
- Cookie는 기존에 설정한 값이 있는 곳에 값을 저장하거나
- 배열 형태의 Cookie에 단일 값을 저장하려 할 때
  - 아무 경고 없이 덮어 쓰여지므로 사용에 주의가 필요함

# 쿠키 (Cookie)

## ▶ Cookie 읽기

- Client에 Cookie가 설정되어 있다면
  - Cookie는 request 메시지의 HTTP Header 항목에 포함되어 전송
  - Cookie의 설정 여부 체크

```
String cookieValue = request.getHeader("Cookie");
```

- request 메시지의 HTTP Header로부터 Cookie 항목을 읽어 String 문자열로 반환

Cookie 값을 읽어오려면

- request 내장 객체인 `getCookie()` Method 사용

```
Cookie[] get_cookies()
```

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

# 쿠키 (Cookie)

- ▶ Cookie 읽기
  - ReadCookie.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>

<html>
 <head>
 <title>Read Cookie</title>
 </head>
 <body>
 <h3>쿠키 읽기</h3>
 <%
 String cookieValue = request.getHeader("Cookie");
 if(cookieValue == null) {
 out.print("<h2>쿠키가 없습니다.</h2>");
 } else {
 Cookie[] cookies = request.getCookies();
 for(int i = 0; i < cookies.length; i++) {
 out.print("Name: " + cookies[i].getName() + "
");
 out.print("Value: " + cookies[i].getValue() + "
");
 out.print("Domain: " + cookies[i].getDomain() + "
");
 out.print("MaxAge: " + cookies[i].getMaxAge() + "
");
 out.print("Path: " + cookies[i].getPath() + "
");
 out.print("Comment: " + cookies[i].getComment() + "
");
 }
 }
 %>
 </body>
</html>
```



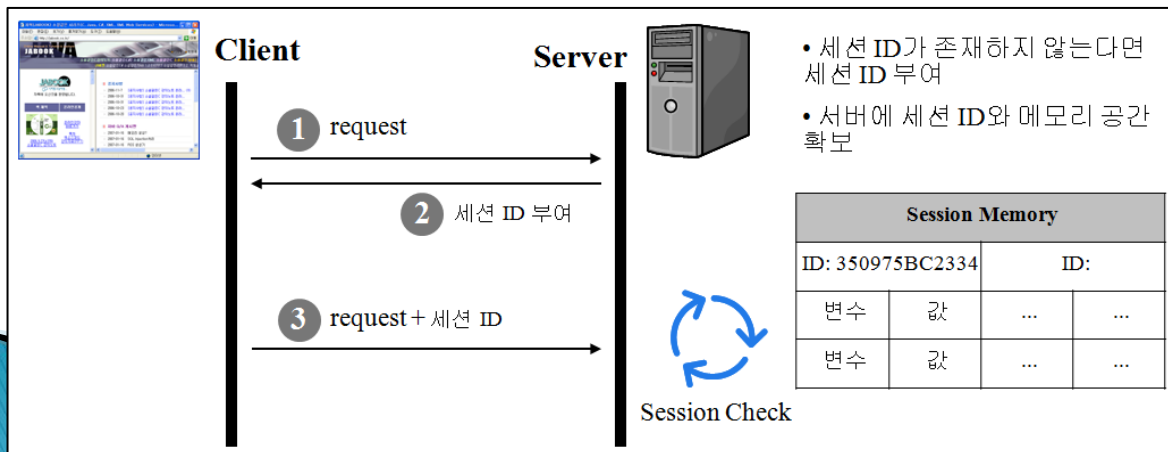
# 세션 (Session)

## ▶ Session

- Client와 Server간의 상태를 유지하기 위한 방법

## ▶ Session 과정

- Client가 처음 접속했을때 Session ID를 부여
- Client는 할당받은 Session ID를
  - Cookie에 임시로 저장하거나
  - Cookie가 지원되지 않을 경우 URL 끝에 붙임
- Server는 Server의 메모리에 Session ID 저장
  - Server에서도 Session ID를 보관하고 있어야 함
  - Client는 Server로부터 부여 받은 Session ID를 가지고 요청
    - Client 구분
  - Server는 Server에 보관된 Session ID 하나당 메모리를 할당
  - 이 Session 메모리에는 '변수 = 값' 형식으로 Data 보관
  - Session ID는 Client가 재접속했을 때 해당 Client를 구분할 수 있는 수단
- Server의 Session은 Time 한계를 두어 시간이 만료되면 스스로 Session을 소멸시킴



# 세션 (Session)

## ▶ Cookie VS Session

- 공통점
  - Client의 상태유지를 위한 기능 제공
- 차이점
  - 저장
    - Cookie
      - Client의 Web Browser가 지정하는 Memory나 Hard Disk에 저장
      - Client 측에서 Cookie 사용을 하지 않도록 Web Browser 설정 시
        - Cookie는 저장되지 않음
    - Session
      - Server의 Memory에 저장
  - 만료
    - Cookie
      - 저장 시 expires 속성을 정의하여 삭제될 시점을 정확히 지정
    - Session
      - Client측에서 Logout 할 경우
      - 설정한 시간 동안 Client 측에 반응이 없을 경우
      - 정확한 만료 시점을 알 수 없음

# 세션 (Session)

## ▶ Cookie VS Session

- 차이점
  - 리소스
    - Cookie
      - Client 측에 저장되고 Client의 Memory를 사용하므로
        - Server 상의 자원을 쓰지 않음
    - Session
      - Server에 저장되고, Server의 Memory로 Loading 되므로
        - Session이 생성될 때마다 그 만큼 Server의 Resource를 차지
  - 용량 제한
    - Cookie
      - Client측에서도 모르게 접속되는 Web Site에 의해 설정될 수도 있음
        - Cookie로 인해 문제가 발생하지 않도록
        - 한 Domain 당 20개, 총 300개, 하나의 Cookie당 4Kb로 용량 제한
    - Session
      - Client가 접속하면 Server에 의해서 생성되므로 개수, 용량에 제한이 없음
- 따라서...
  - Site의 특성에 따라 Session, Cookie의 장점을 살려 적절히 사용

# 세션 (Session)

## ▶ Session 생성

- JSP Page 내에서 Session 사용
  - page 지시문의 session 속성을 이용
    - Session은 사용할 수 있도록 설정

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

- 사실 session 속성의 기본값이 'true' 이므로 생략해도 됨
- request 내장 객체로부터 getSession() Method 이용

```
HttpSession session = request.getSession(true);
```

- getSession() method를 호출하면
    - Server에서는 Cookie나 URL로부터 Session ID를 추출
    - Server에 저장되어 있는 Session ID 목록과 비교
    - 일치하는 Session ID가 있으면
      - 해당 Session ID의 Session Object 를 전달
- 사실 Session Object는 원래 JSP 내장 객체로 제공됨
  - 위처럼 직접 선언하지 않아도 session 내장 객체를 이용
  - Session을 사용할 수 있음
  - session 내장 객체는
    - JSP Page의 pageContext 내장 객체로부터 getSession() Method를 이용하여 생성됨

```
HttpSession session = null;
session = pageContext.getSession();
```

# 세션 (Session)

## ▶ Session 사용

### ◦ 주요 Method

- isNew() - boolean
  - 새로운 Session 여부

```
<%=session.isNew()%>
```

- getId() - String
  - Session ID 출력

```
<%=session.getId()%>
```

- getCreationTime() - long
  - Session 생성시간 출력

```
<%=new java.util.Date(session.getCreationTime()).toString()%>
```

- getLastAccessedTime() - long
  - 마지막 접속시간 출력

```
<%=new java.util.Date(session.getLastAccessedTime()).toString()%>
```

- getMaxInactiveInterval() - int
  - Session Active 시간

```
<%=session.getMaxInactiveInterval()%>
```

# 세션 (Session)

- ▶ Session 사용
  - UseSession.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page session="true"%>

<html>
 <head>
 <title>Session</title>
 </head>
 <body>
 <h3>세션 정보</h3>
 isNew():
 <%=session.isNew()%>

 세션 ID:
 <%=session.getId()%>

 세션 생성 시간:
 <%=new java.util.Date(session.getCreationTime()).toString()%>

 세션 마지막 접속 시간:
 <%=new java.util.Date(session.getLastAccessedTime()).toString()%>

 세션 Active 시간:
 <%=session.getMaxInactiveInterval()%> sec
 </body>
</html>
```

# 세션 (Session)

- ▶ Session Data 저장
  - setAttribute() Method

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

- Parameter
  - String: Session의 이름
  - Object : Session의 값으로 사용
    - Session의 값으로 들어올 자료형을 예측할 수 없음
      - Polymorphism 다시 보세요!
    - 즉, 최상위 클래스인 Object 형을 Parameter로 받아서
      - Upcasting 후 모두 받아내겠다는 의미

```
session.setAttribute("userId", "Grace");
```

# 세션 (Session)

- ▶ Session Data 저장
  - SetSession.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page session="true"%>

<html>
 <head>
 <title>Session</title>
 </head>
 <body>
 <h3>세션 데이터 추가</h3>
 <%
 session.setAttribute("userId", "Grace");
 %>
 </body>
</html>
```



# 세션 (Session)

## ▶ Session Data 추출

- session 내장 객체의 `getAttribute()` Method

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

- `getAttribute()` Method 는
  - Session Object에 저장하였던 정보를 가져올 때 사용
  - Parameter로 가져오고자 하는 Session 이름 사용
  - 그 이름에 해당하는 값을 반환
  - Object 형으로 받아오므로 형 변환을 해주어야 한다
  - Polymorphism 다시 보세요!

```
String value = (String)session.getAttribute("userId");
```

# 세션 (Session)

- ▶ Session Data 추출
  - GetSession.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page session="true"%>

<html>
 <head>
 <title>Session</title>
 </head>
 <body>
 <h3>세션 데이터 출력추가</h3>
 ID: <%= (String) session.getAttribute("userId") %>
 </body>
</html>
```

- ▶ Session Data 삭제
  - session 내장객체의 removeAttribute() Method

```
void removeAttribute(java.lang.String name)
```

- parameter로 전달된 이름의 Session을 제거

```
session.removeAttribute("userId");
```

# 세션 (Session)

- ▶ Session 유지 시간 설정
  - Session 내장객체의 setMaxInactiveInterval() Method

```
void setMaxInactiveInterval(int interval)
```

- Session이 유지될 시간을 초단위로 지정

```
session.setMaxInactiveInterval(60 * 10);
```

- 이렇게 지정하면 10분후 Session 정보 삭제

- SessionTimeout.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page session="true"%>
<html>
 <head>
 <title>Session</title>
 </head>
 <body>
 <h3>세션 생명주기 설정</h3>
 <%
 session.setMaxInactiveInterval(60 * 10);
 %>
 </body>
</html>
```

# 세션 (Session)

## ▶ Session 종료

- Web Browser 종료하기
- 사용자가 아무일도 하지 않고 지정한 시간 만료 시 자동 종료
- session 내장객체의 invalidate() Method

```
void invalidate()
```

- 현재의 Session을 종료시킴

## ◦ InvalidateSession.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page session="true"%>
<html>
 <head>
 <title>Session</title>
 </head>
 <body>
 <h3>세션 종료</h3>
 <%
 if(session != null) {
 session.invalidate();
 }
 %>
 </body>
</html>
```

# 오늘 숙제

▶ 집에서 해보자!