



10장 서블릿의 필터와 리스너 기능

10.1 서블릿 속성과 스코프

10.2 서블릿의 여러 가지 URL 패턴

10.3 Filter API

10.4 여러 가지 서블릿 관련 Listener API



10.1 서블릿 속성과 스코프

서블릿 속성(attribute)

- ServletContext, HttpServlet, HttpServletRequest 객체에 바인딩되어 저장된 객체(정보)
- 각 서블릿 API의 `setAttribute(String name, Object value)`로 바인딩함
- 각 서블릿 API의 `getAttribute(String name)`으로 접근함
- 각 서블릿 API의 `removeAttribute(String name)`으로 속성을 제거함

서블릿 스코프(scope)

- 서블릿 API에 바인딩된 속성에 대한 접근 범위
- ServletContext 속성은 애플리케이션 전체에서 접근 가능
- HttpSession 속성은 사용자만 접근 가능
- HttpServletRequest 속성은 해당 요청/응답에 대해서만 접근 가능
- 각 스코프를 이용해서 로그인 상태 유지, 장바구니, MVC의 Model과 View의 데이터 전달 기능을 구현



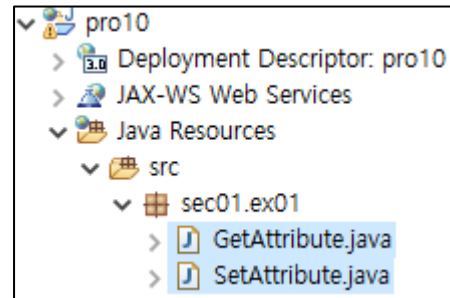
10.1 서블릿 속성과 스코프

스코프 종류와 특징

스코프 종류	해당 서블릿 API	속성의 스코프
애플리케이션 스코프	ServletContext	속성은 애플리케이션 전체에 대해 접근할 수 있습니다.
세션 스코프	HttpSession	속성은 브라우저에서만 접근할 수 있습니다.
리퀘스트 스코프	HttpServletRequest	속성은 해당 요청/응답 사이클에서만 접근할 수 있습니다.

각 서블릿 속성의 스코프 실습

1. 다음과 같이 GetAttribute, SetAttribute 클래스 파일을 준비합니다.





10.1 서블릿 속성과 스코프

2. SetAttribute 클래스를 다음과 같이 작성합니다.

코드 10-1 pro10/src/sec01/ex01/SetAttribute.java

```
package sec01.ex01;

...
@WebServlet("/set")
public class SetAttribute extends HttpServlet{
    protected public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        String ctxMesg = "context에 바인딩됩니다.";
        String sesMesg = "session에 바인딩됩니다.";
        String reqMesg = "request에 바인딩됩니다.";

        ServletContext ctx = getServletContext();
        HttpSession session = request.getSession();
        ctx.setAttribute("context", ctxMesg);
        session.setAttribute("session", sesMesg);
        request.setAttribute("request", reqMesg);
        out.print("바인딩을 수행합니다.");
    }
}
```

HttpServletContext 객체, HttpSession 객체, HttpServletRequest 객체를 얻은 후 속성을 바인딩합니다.



10.1 서블릿 속성과 스코프

3. 두 번째 서블릿인 GetAttribute 클래스를 다음과 같이 작성합니다.

코드 10-2 pro10/src/sec01/ex01/GetAttribute.java

```
package sec01.ex01;

...
@WebServlet("/get")
public class GetAttribute extends HttpServlet{
    protected public void doGet(HttpServletRequest request , HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        ServletContext ctx = getServletContext();
        HttpSession sess = request.getSession();

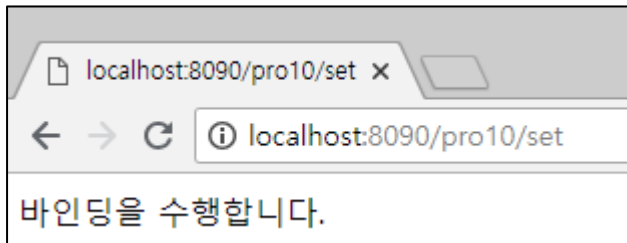
        String ctxMesg = (String)ctx.getAttribute("context");
        String sesMesg = (String)sess.getAttribute("session");
        String reqMesg = (String)request.getAttribute("request");

        out.print("context값 : " + ctxMesg + "<br>");
        out.print("session값 : " + sesMesg + "<br>");
        out.print("request값 : " + reqMesg + "<br>");
    }
}
```

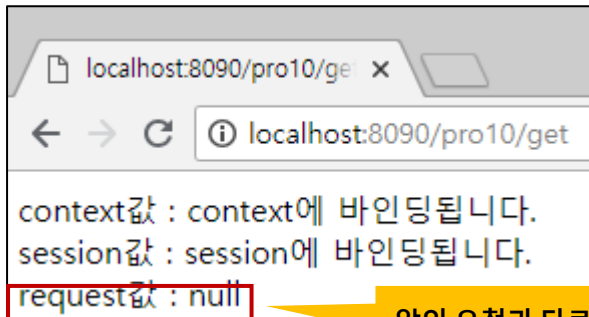
각 서블릿 API에서
바인딩된 속성의 값
을 가져옵니다.

10.1 서블릿 속성과 스코프

4. 브라우저에서 /set으로 요청해 속성을 바인딩합니다.



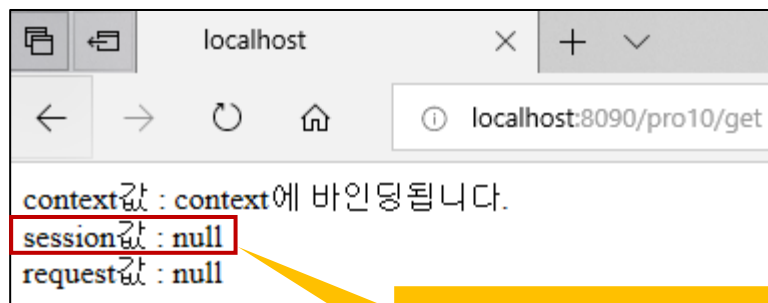
5. Context와 Session 객체에 바인딩된 속성은 같은 브라우저에서 접근할 수 있으므로 값을 출력합니다.



앞의 요청과 다르므로 바인딩
된 속성이 유지되지 않음

10.1 서블릿 속성과 스코프

6. 인터넷 익스플로러에서 /get으로 요청해 볼까요? 익스플로러에서 요청했기 때문에 이번에는 크롬의 세션 객체에는 접근할 수 없어 null을 출력합니다.



다른 브라우저라서
세션도 다르므로 속성이 유지되지 않
습니다.



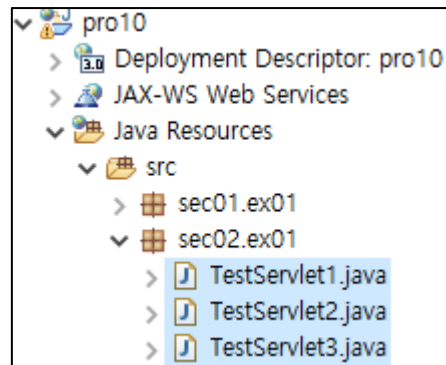
10.2 서블릿의 여러 가지 URL 패턴

URL 패턴

- 서블릿 매핑 시 사용되는 가상의 이름
- 클라이언트가 브라우저에서 요청할 때 사용되며 반드시 /(슬래시)로 시작해야함
- 이름까지 일치, 디렉터리까지 일치, 확장자만 일치하는 세가지로 나뉘어짐

• 10.2.1 서블릿에 여러 가지 URL 패턴 적용 실습

1. 다음과 같이 TestServlet1~3 클래스 파일을 준비합니다.





10.2 서블릿의 여러 가지 URL 패턴

2. 첫 번째 서블릿인 TestServlet1 클래스를 다음과 같이 작성합니다.

코드 10-3 pro10/src /sec02/ex01/TestServlet1.java

```
package sec02.ex01;
```

```
...
```

```
@WebServlet("/first/test")
```

정확히 이름까지 일치하는 URL 패턴

```
public class TestServlet1 extends HttpServlet {
```

```
public protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
```

```
    request.setCharacterEncoding("utf-8");
```

```
    response.setContentType("text/html;charset=utf-8");
```

```
    PrintWriter out = response.getWriter();
```

```
    String context = request.getContextPath();
```

컨텍스트 이름만 가져옵니다.

```
    String url = request.getRequestURL().toString();
```

전체 URL을 가져옵니다.

```
    String mapping = request.getServletPath();
```

서블릿 매핑 이름만 가져옵니다.

```
    String uri = request.getRequestURI();
```

URI를 가져옵니다.

```
    out.println("<html>");
```

```
    out.println("<head>");
```

```
    out.println("<title>Test Servlet1</title>");
```

```
    out.println("</head>");
```

```
    out.println("<body bgcolor='green'>");
```

```
    out.println("<b>TestServlet1입니다.</b><br>");
```



10.2 서블릿의 여러 가지 URL 패턴

3. 두 번째 서블릿인 TestServlet2 클래스를 다음과 같이 작성합니다.

코드 10-4 pro10/src /sec02/ex01/TestServlet2.java

```
package sec02.ex01;
...
@WebServlet("/first/*") ← 디렉터리 이름만 일치하는 URL 패턴
public class TestServlet2 extends HttpServlet {
protected public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html;charset=utf-8");
    PrintWriter out = response.getWriter();
    String context = request.getContextPath();
    String url = request.getRequestURL().toString();
    String mapping = request.getServletPath();
    String uri = request.getRequestURI();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Test Servlet2</title>");
    out.println("</head>");
    out.println("<body bgcolor='yellow'>");
    out.println("<b>TestServlet2입니다.</b><br>");
    out.println("<b>컨텍스트 이름 : " + context + "</b><br>");
    out.println("<b>전체 경로 : " + url + "<b><br>");
    out.println("<b>매핑 이름 : " + mapping + "<b><br>");
    out.println("<b>URI : " + uri + "<b>");
}
```



10.2 서블릿의 여러 가지 URL 패턴

4. 세 번째 서블릿인 TestServlet3 클래스를 다음과 같이 작성합니다.

코드 10-5 pro10/src /sec02/ex01/TestServlet3.java

```
package sec02.ex01;
```

```
...
```

```
@WebServlet("*.do")
```

확장자만 일치하는 패턴

```
/*@WebServlet("/*")*/
```

모든 요청 URL 패턴

```
public class TestServlet3 extends HttpServlet {
```

```
public protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
```

```
    request.setCharacterEncoding("utf-8");
```

```
    response.setContentType("text/html;charset=utf-8");
```

```
    PrintWriter out = response.getWriter();
```

```
    String context = request.getContextPath();
```

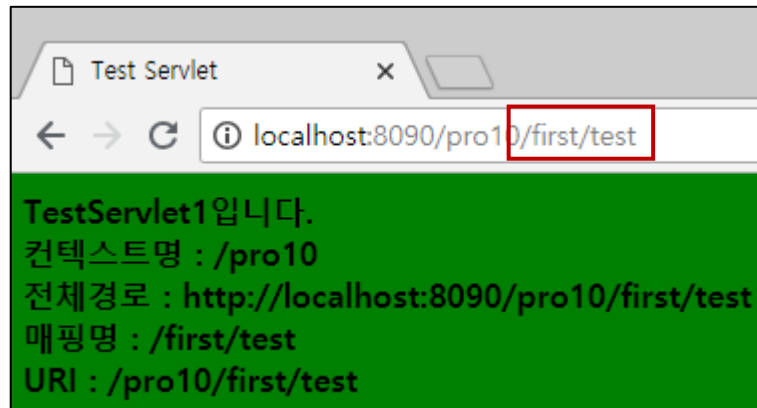
```
    String url = request.getRequestURL().toString();
```

```
    String mapping = request.getServletPath();
```

```
    ...
```

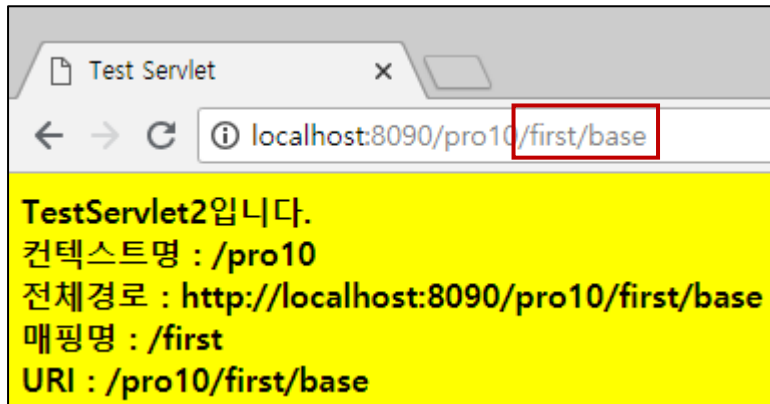
10.2 서블릿의 여러 가지 URL 패턴

5. 각각의 매핑 이름으로 요청해 보겠습니다. 우선 정확한 매핑 이름(/first/test)으로 요청한 경우에는 다음과 같이 출력됩니다.

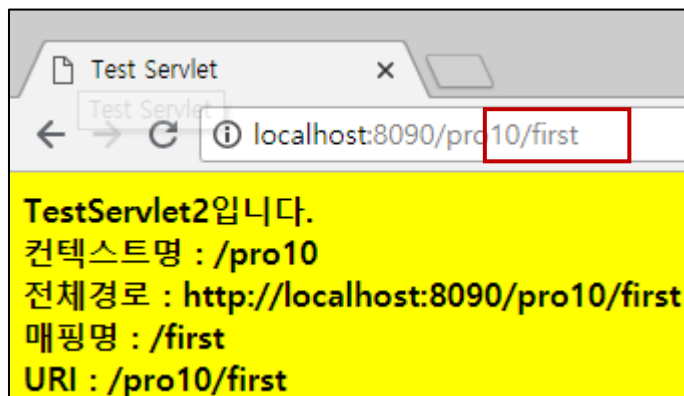


10.2 서블릿의 여러 가지 URL 패턴

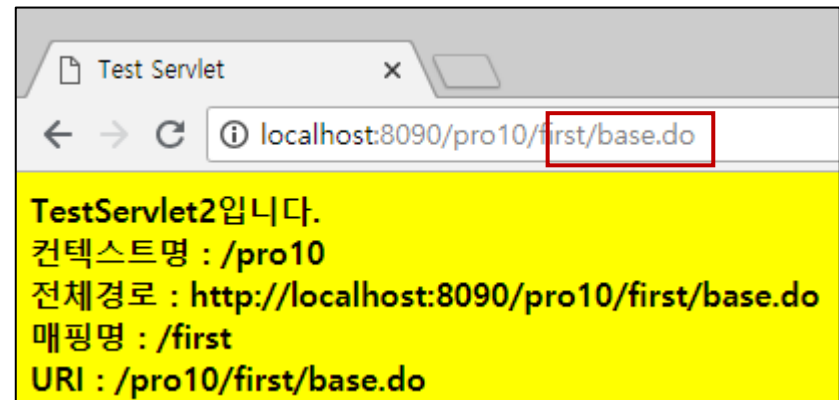
6. 디렉터리 이름만 일치하는 경우에는 각각 다음과 같이 출력됩니다.



디렉토리명('/first/')만 일치하는 경우



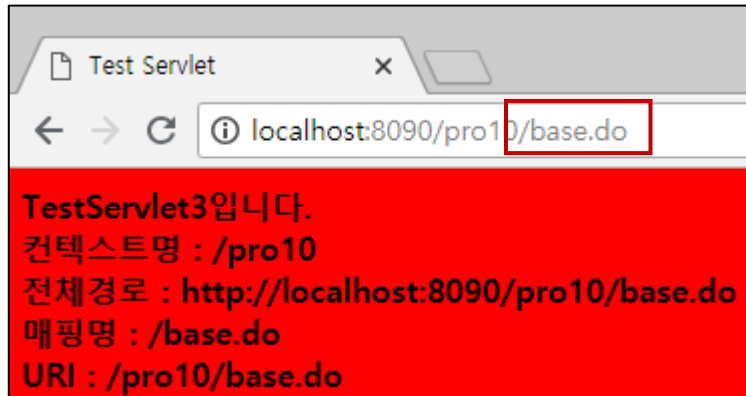
디렉토리명('/first/base.do')만 일치하는 경우



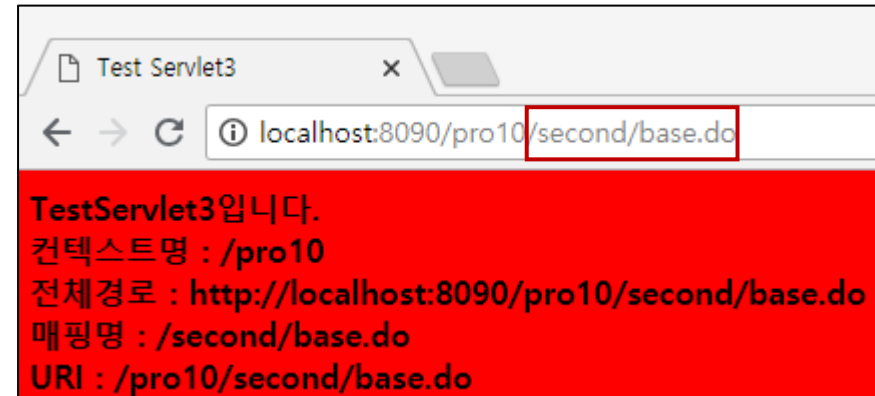
10.2 서블릿의 여러 가지 URL 패턴

7. 다음은 확장자가 일치했을 경우의 출력 결과로, 각각 /base.do와 /second/base.do로 요청했을 때의 출력 결과입니다.

확장자 .do('/base.do')로 요청 시



/second/base.do'로 요청 시





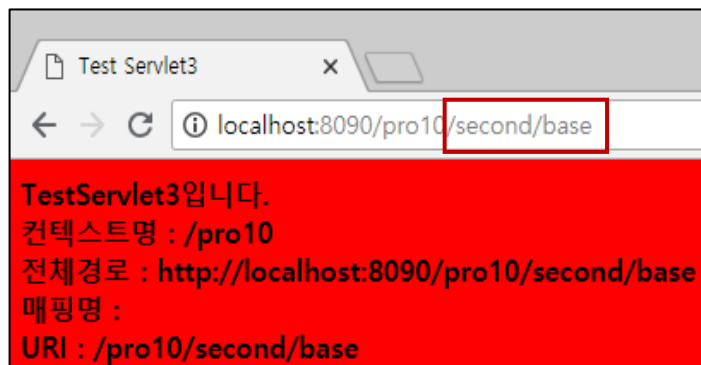
10.2 서블릿의 여러 가지 URL 패턴

8. 다음은 TestServlet3 클래스의 URL 패턴을 /*로 설정한 후 요청한 결과입니다.
@WebServlet("*.do")를 주석 처리하고, @WebServlet("/")을 입력하여 실행합니다.

```
14  */
15  /*@WebServlet("*.do")*/
16  @WebServlet("/")
17  public class TestServlet3 extends HttpServlet {
18      private static final long serialVersionUID = 1L;
19
20      /**
```

9. 톰캣을 다시 실행한 후 /second/base로 요청하여 결과를 출력합니다.

확장명 없이 요청 시 출력



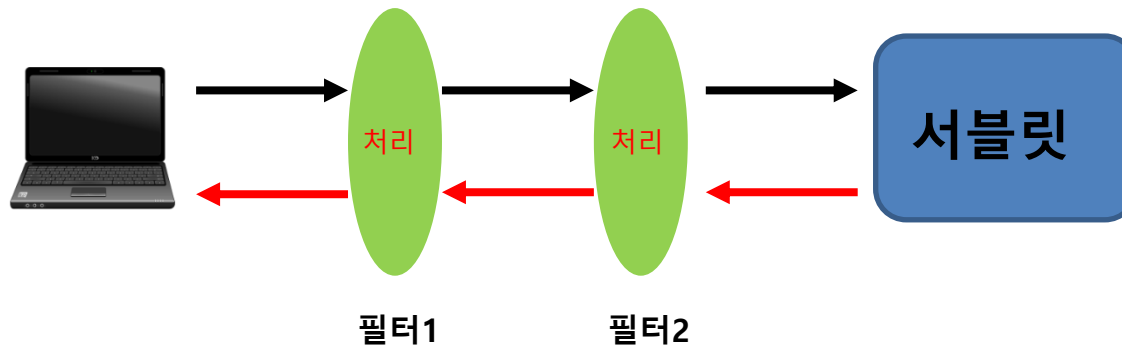
- 확장명은 지정하지 않을 수도 있고, do 대신 자신이 원하는 이름으로 지정해서 사용할 수 있음
- 확장명을 do로 요청하는 경우는 일반적으로 MVC나 프레임워크에서 많이 쓰는 확장명임

10.3 Filter API

필터(Filter)

- 브라우저에서 서블릿에 요청하거나 응답할 때 미리 요청이나 응답과 관련해 여러 가지 작업을 처리하는 기능
- 요청이나 응답 시 공통적인 작업을 처리하는데 이용됨

필터 기능 수행 과정





10.3 Filter API

request에 한글 인코딩 설정

```
36 public void doHandle(HttpServletRequest request, HttpServletResponse response)
37     request.setCharacterEncoding("utf-8");
38     response.setContentType("text/html;charset=utf-8");
39     PrintWriter out = response.getWriter();
40     String user_id = request.getParameter("user_id");
41     String user_pwd = request.getParameter("user_pwd");
42
43     MemberVO memberVO = new MemberVO();
44     memberVO.setId(user_id);
45     memberVO.setPwd(user_pwd);
46     MemberDAO dao = new MemberDAO();
47     boolean result = dao.isExisted(memberVO);
48
49
50     if (result) {
51         HttpSession session = request.getSession();
52         session.setAttribute("isLogon", true);
53         session.setAttribute("login.id", user_id);
54         session.setAttribute("login.pwd", user_pwd);
```



서블릿에서 일일이 한글 인코딩을 구현하는 것이 아니라 필터에서 먼저 처리하면 편리함.



10.3 Filter API

필터 용도

➤ 요청 필터

- 사용자 인증 및 권한 검사
- 요청 시 요청 관련 로그 작업
- 인코딩 기능

➤ 응답 필터

- 응답 결과에 대한 암호화 작업
- 서비스 시간 측정

필터 관련 API

- javax.servlet.Filter
- javax.servlet.FilterChain
- javax.servlet.FilterConfig



10.3 Filter API

Filter 인터페이스에 선언된 메서드

메서드	기능
destroy()	필터 소멸 시 컨테이너에 의해 호출되어 종료 작업을 수행합니다.
doFilter()	요청/응답 시 컨테이너에 의해 호출되어 기능을 수행합니다
init()	필터 생성 시 컨테이너에 의해 호출되어 초기화 작업을 수행합니다.

FilterConfig의 메서드

메서드	기능
getFilterName()	필터 이름을 반환합니다.
getInitParameter(String name)	매개변수 name에 대한 값을 반환합니다.
getServletContext()	서블릿 컨텍스트 객체를 반환합니다.



10.3 Filter API

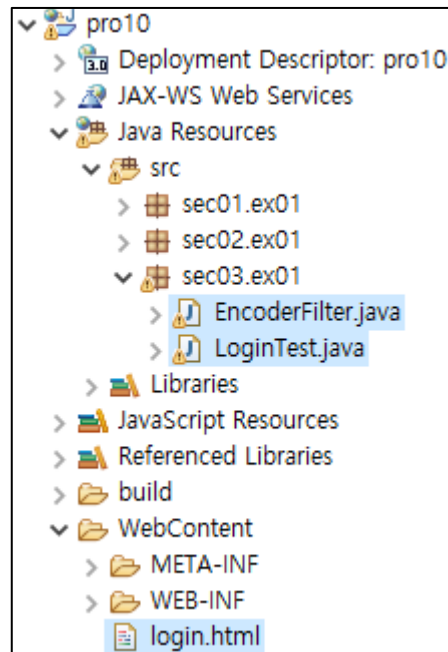
- 10.3.1 사용자 정의 Filter 만들기

Filter 매핑 방법

- 애너테이션을 이용하는 방법
- web.xml에 설정하는 방법

- 10.3.2 Filter를 이용한 한글 인코딩 실습

1. 다음과 같이 LoginTest, EncoderFilter 클래스 파일을 준비합니다.





10.3 Filter API

2. 로그인창에서 ID 대신 이름을 입력한 후 서블릿으로 전송하도록 login.html을 작성합니다.

코드 10-6 pro10/WebContent/login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>로그인창</title>
</head>
<body>
  <form name="frmLogin" method="post" action="login" encType="utf-8">
    이름 :<input type="text" name="user_name"><br>
    비밀번호:<input type="password" name="user_pw"><br>
    <input type="submit" value="로그인">
    <input type="reset" value="다시입력">
  </form>
</body>
</html>
```



10.3 Filter API

3. LoginTest 클래스를 다음과 같이 작성합니다.

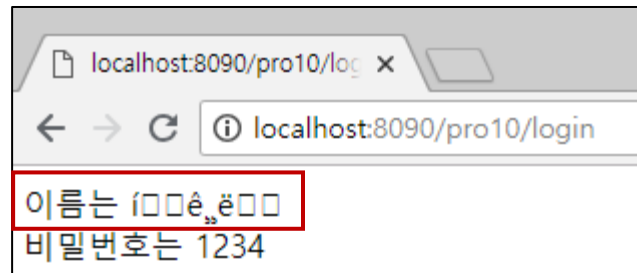
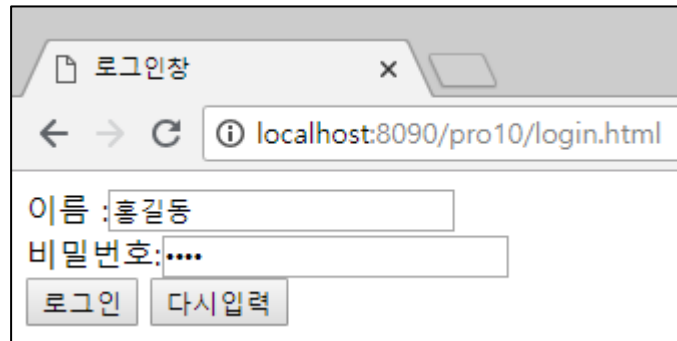
코드 10-7 prro10/src /sec03/ex01/LoginTest.java

```
package sec03.ex01;
...
@WebServlet("/login")
public class LoginTest extends HttpServlet {
protected public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //request.setCharacterEncoding( "utf-8" );
    response.setContentType("text/html;charset=utf-8");
    PrintWriter out = response.getWriter();
    String user_name = request.getParameter("user_name");
    String user_pw = request.getParameter("user_pw");
    out.println("<html><body>");
    out.println("이름는 " + user_name + "<br>");
    out.println("비밀번호는 "+user_pw + "<br>");
    out.println("</body></html>");
}
}
```

post 방식으로 한글 전송 시
인코딩 작업을 생략합니다.

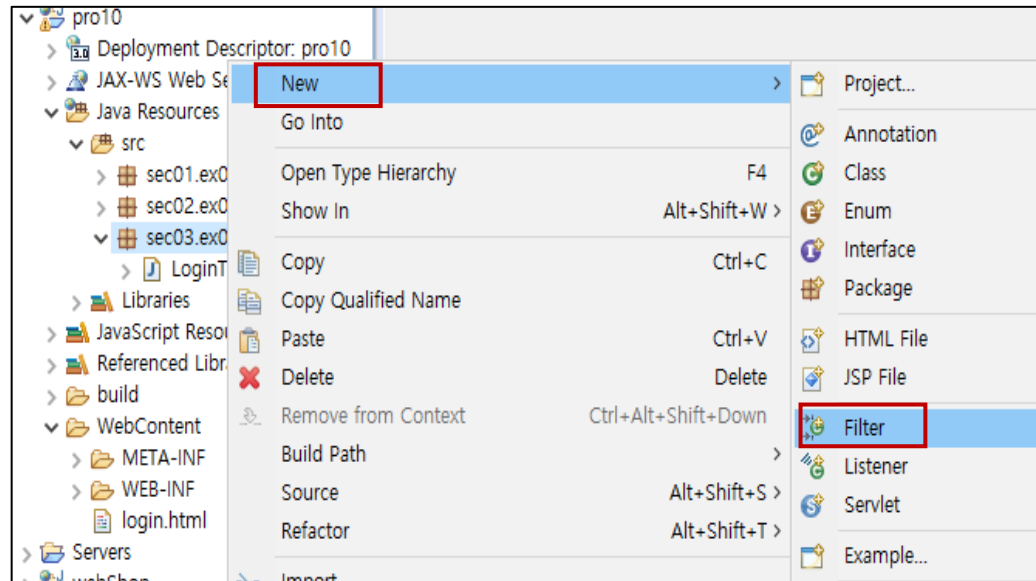
10.3 Filter API

4. 다음은 인코딩 처리를 하지 않았을 때의 출력 결과입니다. 한글이 깨져서 표시되는 것을 볼 수 있죠?



10.3 Filter API

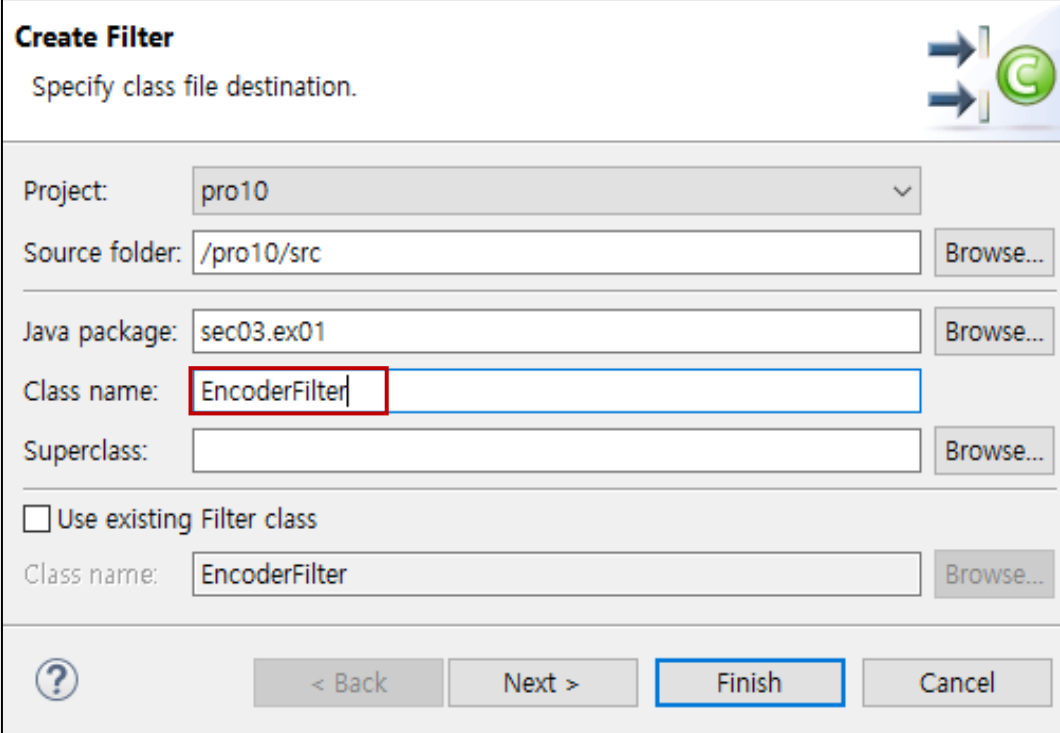
5. 이번에는 필터를 이용해 한글 인코딩 기능을 구현해 보겠습니다. sec03.ex01 패키지를 선택하고 마우스 오른쪽 버튼을 클릭한 후 New > Filter를 선택합니다.





10.3 Filter API

6. Class name으로 EncoderFilter를 입력하고 Next를 클릭합니다.



Create Filter
Specify class file destination.

Project: pro10

Source folder: /pro10/src Browse...

Java package: sec03.ex01 Browse...

Class name: EncoderFilter

Superclass: Browse...

☐ Use existing Filter class

Class name: EncoderFilter Browse...

? < Back Next > Finish Cancel



10.3 Filter API

7. Filter mappings에서 /EncoderFilter를 선택한 후 Edit를 클릭합니다.

Create Filter
Enter servlet filter deployment descriptor specific information.

Name: EncoderFilter

Description:

Initialization parameters:


Name	Value	Description
------	-------	-------------

Add...

Edit...

Remove

Filter mappings:


URL Pattern / Servlet Name	Dispatchers
 /EncoderFilter	

Add...

Edit...

Remove

☐ Asynchronous Support



< Back

Next >

Finish

Cancel



10.3 Filter API

8. 모든 요청에 대해 필터 기능을 수행하도록 Pattern을 /*로 수정합니다.

☐ Servlet ☒ URL pattern

Pattern:
/*


Select dispatchers

☐ REQUEST ☐ FORWARD
☐ INCLUDE ☐ ERROR

OK Cancel

10.3 Filter API

9. URL Pattern에서 /*을 확인하고 Next를 클릭합니다.

Create Filter

Enter servlet filter deployment descriptor specific information.

Name:

Description:

Initialization parameters:


Name	Value	Description
------	-------	-------------

Add...

Edit...

Remove

Filter mappings:


	URL Pattern / Servlet Name	Dispatchers
	<input type="text" value="/*"/>	

Add...

Edit...

Remove

☐ Asynchronous Support



< Back

Next >

Finish

Cancel



10.3 Filter API

10. Finish를 클릭하여 필터 클래스가 생성된 것을 확인합니다.

Create Filter
Specify modifiers, interfaces to implement, and method stubs to generate.

Modifiers: ☒ public ☐ abstract ☐ final

Interfaces: ☒ javax.servlet.Filter Add... Remove

Which method stubs would you like to create?

☐ Constructors from superclass
☒ Inherited abstract methods
☒ init ☒ destroy ☒ doFilter

? < Back Next > Finish Cancel

```
EncoderFilter.java
11
12 /**
13  * Servlet Filter implementation class EncoderFilter
14  */
15 @WebFilter("/*")
16 public class EncoderFilter implements Filter {
17
18     /**
19      * Default constructor.
20      */
21     public EncoderFilter() {
22         // TODO Auto-generated constructor stub
23     }
24
25     /**
26      * @see Filter#destroy()
27      */
28     public void destroy() {
29         // TODO Auto-generated method stub
30     }
```



10.3 Filter API

11. 이제 다음과 같이 EncoderFilter 클래스를 작성합니다.

코드 10-8 pro10/src /sec03/ex01/EncoderFilter.java

```
package sec03.ex01;
...

@WebFilter("/*")
public class EncoderFilter implements Filter{
    ServletContext context;

    public void init(FilterConfig fConfig) throws ServletException{
        System.out.println("utf-8 인코딩.....");
        context = fConfig.getServletContext();
    }

    public void doFilter(ServletRequest request, ServletResponse response,
                        FilterChain chain) throws ServletException, IOException {
        System.out.println("doFilter 호출");
        request.setCharacterEncoding( "utf-8" );
        String context= ((HttpServletRequest)request).getContextPath();
        String pathinfo = ((HttpServletRequest)request).getRequestURI();
        String realPath = request.getRealPath( pathinfo);
    }
}
```

@WebFilter("/*") WebFilter 애너테이션을 이용해 모든 요청이 필터를 거치게 합니다.

implements Filter{ 사용자 정의 필터는 반드시 Filter 인터페이스를 구현해야 합니다.

doFilter() 안에서 실제 필터 기능을 구현합니다.

System.out.println("doFilter 호출"); 한글 인코딩 설정 작업을 합니다.

request.setCharacterEncoding("utf-8"); 웹 애플리케이션의 컨텍스트 이름을 가져옵니다.

String context= ((HttpServletRequest)request).getContextPath();

String pathinfo = ((HttpServletRequest)request).getRequestURI();

String realPath = request.getRealPath(pathinfo); 웹 브라우저에서 요청한 요청 URI를 가져옵니다.

요청 URI의 실제 경로를 가져옵니다.



10.3 Filter API

```
String mesg = " Context  정보:" + context
              + "\n URI 정보 : " + pathinfo
              + "\n 물리적 경로: " + realPath;
System.out.println(mesg);
chain.doFilter( request, response );
}

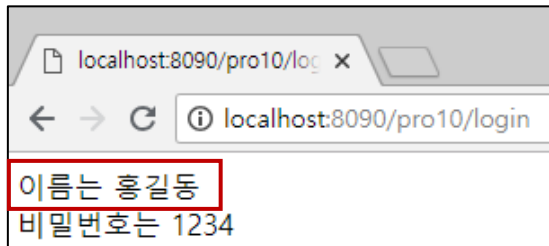
public void destroy(){
    System.out.println("destroy 호출");
}
}
```

다음 필터로 넘기는 작업을 수행합니다.



10.3 Filter API

12. 톰캣을 재실행하고 로그인창에서 한글을 입력합니다.



```
9월 03, 2018 2:58:10 오후 org.apache.catalina.core.StandardContext reload
경보: Reloading Context with name [/pro10] is completed
doFilter 호출
Context 정보:/pro10
URI 정보 : /pro10/login
물리적 경로: C:\myJSP\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpw
```




10.3 Filter API

• 10.3.3 응답 필터 사용

```
15 public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws ServletException, IOException {
16     System.out.println("doFilter 호출");
17
18     request.setCharacterEncoding( "utf-8" );
19     long begin = System.currentTimeMillis();
20     String path = ((HttpServletRequest)request).getContextPath();
21     String pathinfo = ((HttpServletRequest)request).getRequestURI();
22     String path = request.getRealPath( pathinfo );
23     String msg = "Context 정보:" + path + " URI 정보 : " + pathinfo + "
24
25     chain.doFilter( request, response );
26     long end = System.currentTimeMillis();
27     System.out.println("작업 시간:"+(end-begin)+"ms");
28 }
```

요청 필터 기능

응답 필터 기능



➤ doFilter() 메서드를 기준으로 위쪽에 위치한 코드는 요청 필터 기능을 수행하고, 아래에 위치한 코드는 응답 필터 기능을 수행함



10.3 Filter API

• 10.3.4 응답 필터 기능으로 작업 시간 구하기

1. 앞 절의 EncoderFilter 클래스를 그대로 사용합니다. chain.doFilter() 메서드 위아래에 요청 전과 후의 시각을 구하는 코드를 각각 추가합니다.

코드 10-9 pro10/src/sec03/ex01/EncoderFilter.java

```
package sec03.ex01;

...

@WebFilter("/*")
public class EncoderFilter implements Filter{
    ServletContext context;
    public void init(FilterConfig fc) throws ServletException{
        System.out.println("utf-8 인코딩.....");
        context = fc.getServletContext();
    }
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)throws ServletException, IOException {
        System.out.println("doFilter 호출");
        request.setCharacterEncoding( "utf-8" );
        String path = ((HttpServletRequest)request).getContextPath();
        String pathinfo = ((HttpServletRequest)request).getRequestURI();
        String realPath = request.getRealPath( pathinfo);
        String mesg = " Context  정보:" + context
            + "\n URI 정보 : " + pathinfo
            + "\n 물리적 경로: " + realPath;
        System.out.println(mesg);
        long begin = System.currentTimeMillis();
```

요청 필터에서 요청 처리 전의 시각을 구합니다.



10.3 Filter API

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

    long end = System.currentTimeMillis();
    System.out.println("작업 시간: "+(end-begin)+"ms");
}
public void destroy(){
    System.out.println("destroy 호출");
}
}
```

응답 필터에서 요청 처리 후의 시각을 구합니다.

작업 요청 전과 후의 시각 차를 구해 작업 수행 시간을 구합니다.



10.3 Filter API

2. 실행하면 다음과 같이 로그인 요청 작업에 걸린 시간을 콘솔로 출력합니다. 로컬 PC에서의 실행이므로 너무 빨라 0ms를 표시합니다.

```
doFilter 호출  
Context 정보:/pro10  
URI 정보 : /pro10/login  
물리적 경로: C:\myJSP\workspace\.metadata\.plugins\org.ec1  
작업 시간:0ms
```



10.3 Filter API

web.xml에 필터 설정하기

```
2= <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
3=   <filter>  
4     <filter-name>encode</filter-name>  
5     <filter-class>sec03.ex01.EncoderFilter</filter-class>  
6   </filter>  
7=   <filter-mapping>  
8     <filter-name>encode</filter-name>  
9     <url-pattern>/*</url-pattern>  
10  </filter-mapping>  
11 </web-app>
```



10.4 여러 가지 서블릿 관련 Listener API

Listener API

➤ 서블릿에서 발생하는 이벤트에 대해서 처리를 할 수 있는 기능

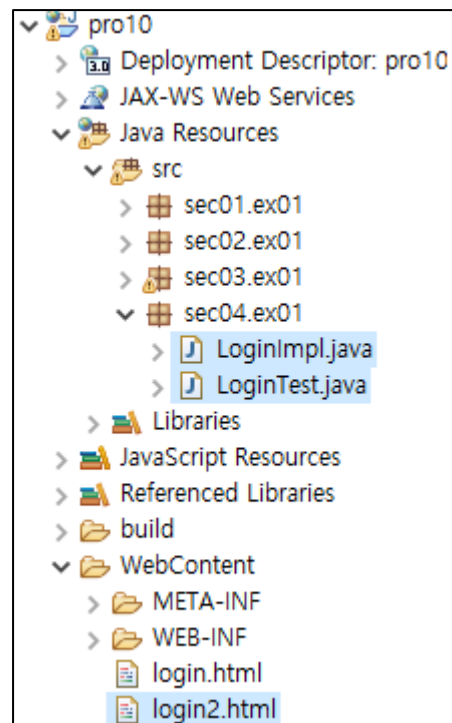
서블릿 관련 여러가지 리스너들

서블릿 관련 리스너	추상 메서드	기능
ServletContextAttributeListener	attributeAdded() attributeRemoved() attributeReplaced()	Context 객체에 속성 추가/제거/수정 이벤트 발생 시 처리합니다.
HttpSessionListener	sessionCreated() sessionDestroyed()	세션 객체의 생성/소멸 이벤트 발생 시 처리합니다
ServletRequestListener	requestInitialized() requestDestroyed()	클라이언트의 요청 이벤트 발생 시 처리합니다
ServletRequestAttributeListener	attributedAdded() attributedRemoved() attributeReplaced()	요청 객체에 속성 추가/제거/수정 이벤트 발생 시 처리합니다
HttpSessionBindingListener	valueBound() valueUnbound()	세션에 바인딩/언바인딩된 객체를 알려 주는 이벤트 발생 시 처리합니다
HttpSessionAttributeListener	attributedAdded() attributedRemoved() attributeReplaced()	세션에 속성 추가/제거/수정 이벤트 발생 시 처리합니다
ServletContextListener	contextInitialized() contextDestroyed()	컨텍스트 객체의 생성/소멸 이벤트 발생 시 처리합니다
HttpSessionActivationListener	sessionDidActivate() sessionWillPassivate()	세션의 활성화/비활성화 이벤트 발생 시 처리합니다

10.4 여러 가지 서블릿 관련 Listener API

- 10.4.1 HttpSessionBindingListener 이용해 로그인 접속자수 표시

1. 다음과 같이 실습 파일을 새로 준비합니다.





10.4 여러 가지 서블릿 관련 Listener API

2. ID와 비밀번호를 입력하여 전송하는 로그인창을 작성합니다.

코드 10-10 pro10/WebContent/login2.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>로그인창</title>
</head>
<body>
  <form name="frmLogin" method="post" action="login" encType="utf-8">
    아이디 :<input type="text" name="user_id"><br>
    비밀번호:<input type="password" name="user_pw"><br>
    <input type="submit" value="로그인">
    <input type="reset" value="다시 입력">
  </form>
</body>
</html>
```




10.4 여러 가지 서블릿 관련 Listener API

3. LoginTest 클래스를 다음과 같이 작성합니다.

코드 10-11 pro10/src/sec04/ex01/LoginTest.java

```
package sec04.ex02;

...

@WebServlet("/login")
public class LoginTest extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding( "utf-8" );
        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        HttpSession session=request.getSession();
        String user_id = request.getParameter("user_id");
        String user_pw = request.getParameter("user_pw");
        LoginImpl loginUser=new LoginImpl(user_id,user_pw);
        if(session.isNew()){
            session.setAttribute("loginUser", loginUser);
        }
        out.println("<head>");
        out.println("<script  type='text/javascript'>");
```

이벤트 핸들러를 생성한 후
세션에 저장합니다.

세션에 바인딩 시
LoginImpl의 valueBound()
메서드를 호출합니다.



10.4 여러 가지 서블릿 관련 Listener API

```
out.println("<head>");
out.println("<script  type='text/javascript'>");
out.println("setTimeout('history.go(0);', 5000)");
out.println("</script>");
out.println("</head>");
out.println("<html><body>");
out.println("아이디는 " + loginUser.user_id + "<br>");
out.println("총 접속자수는"+LoginImpl.total_user + "<br>");
out.println("</body></html>");
}
}
```

메서드를 호출합니다.

자바스크립트의 `setTimeout()` 함수를
이용해 5초마다 서블릿에 재요청하여
현재 접속자수를 표시합니다.

접속자수를 브라우저로
출력합니다.



10.4 여러 가지 서블릿 관련 Listener API

4. LoginImpl 클래스를 다음과 같이 작성합니다.

코드 10-12 pro10/src/sec04/ex01/LoginImpl.java

```
package sec04.ex01;
```

```
import javax.servlet.http.HttpSessionBindingEvent;
```

```
public class LoginImpl implements HttpSessionBindingListener {
```

```
    String user_id;
```

```
    String user_pw;
```

```
    static int total_user=0;
```

세션에 바인딩 시 1씩 증가시킵니다.

```
    public LoginImpl() {
```

```
    }
```

```
    public LoginImpl(String user_id, String user_pw) {
```

```
        this.user_id = user_id;
```

```
        this.user_pw = user_pw;
```

```
    }
```

```
    @Override
```

```
    public void valueBound(HttpSessionBindingEvent arg0) {
```

```
        System.out.println("사용자 접속");
```

```
        ++total_user;
```

```
    }
```

세션에 저장 시 접속자수를
증가시킵니다.

```
    @Override
```

```
    public void valueUnbound(HttpSessionBindingEvent arg0) {
```

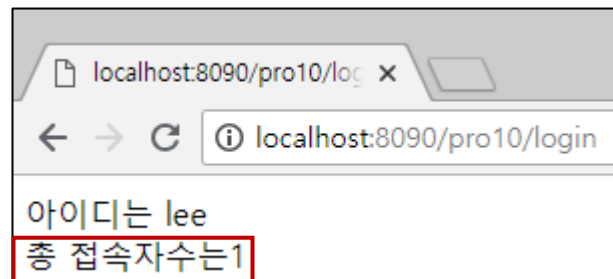
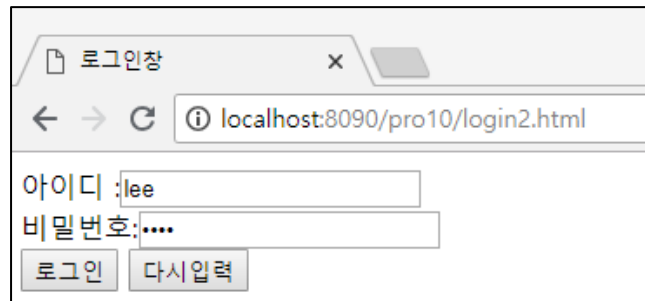
```
        System.out.println("사용자 접속 해제");
```

```
        total_user--;
```

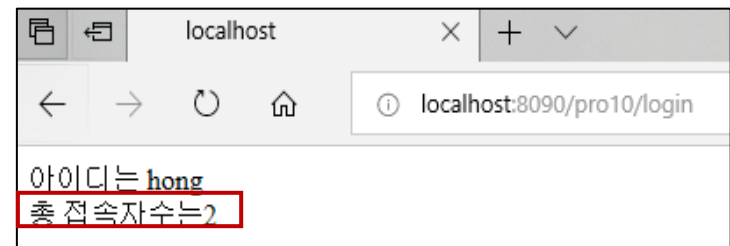
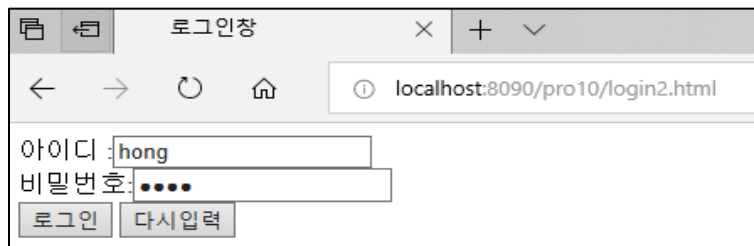
세션에서 소멸 시 접속자
수를 감소시킵니다.

10.4 여러 가지 서블릿 관련 Listener API

5. 서로 다른 종류의 브라우저에서 접속하여 실행 결과를 확인해 보겠습니다. 우선 크롬에서 로그인하면 접속자 ID와 접속자수가 표시됩니다.

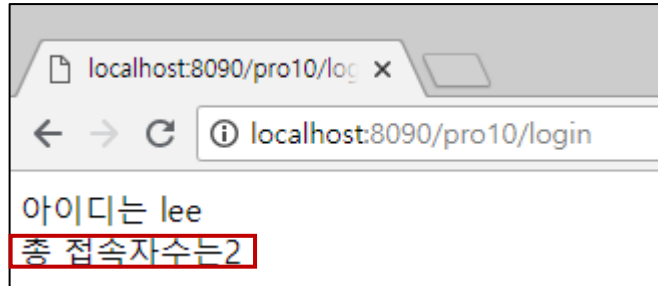


6. 이번에는 익스플로러에서 로그인하면 다음과 같이 접속자 ID와 접속자수가 표시됩니다.



10.4 여러 가지 서블릿 관련 Listener API

7. 5초 후 크롬에서는 접속자수가 갱신되어 표시됩니다.

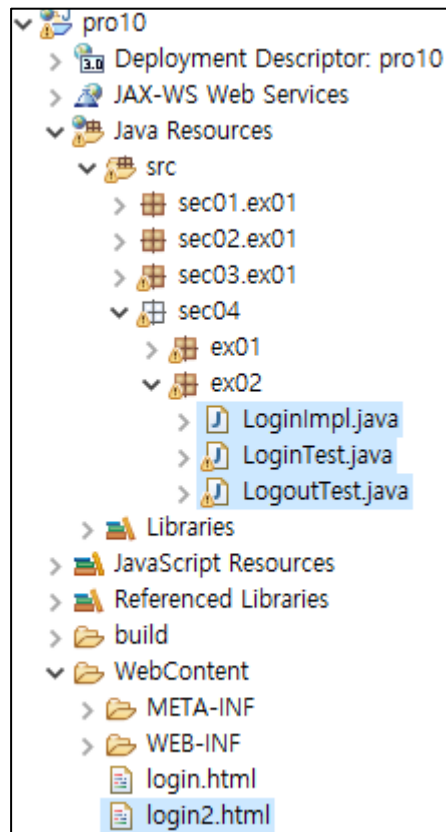


❖ HttpSessionBindingListener를 구현한 LoginImpl 클래스는 리스너를 따로 등록할 필요가 없습니다.

10.4 여러 가지 서블릿 관련 Listener API

- 10.4.2 HttpSessionListener 이용해 로그인 접속자수 표시

1. 다음과 같이 실습 파일을 준비합니다.





10.4 여러 가지 서블릿 관련 Listener API

2. 첫 번째 서블릿인 LoginTest 클래스 파일을 다음과 같이 수정합니다.

코드 10-13 pro10/src/sec04/ex02/LoginTest.java

```
package sec04.ex02;  
  
...  
@WebServlet("/login")  
public class LoginTest extends HttpServlet {  
    ServletContext context=null;  
    List user_list=new ArrayList();
```

로그인한 접속자 ID를 저장하는 ArrayList입니다.

```
public protected void doPost(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
    request.setCharacterEncoding( "utf-8" );  
    response.setContentType("text/html;charset=utf-8");  
    context=getServletContext();  
    PrintWriter out = response.getWriter();  
    HttpSession session=request.getSession();  
    String user_id = request.getParameter("user_id");  
    String user_pw = request.getParameter("user_pw");
```



10.4 여러 가지 서블릿 관련 Listener API

```
LoginImpl loginUser=new LoginImpl(user_id,user_pw);
```

LoginImpl 객체를 생성한 후
전송된 ID와 비밀번호를 저장합니다.

```
if(session.isNew()){
    session.setAttribute("loginUser", loginUser);
    user_list.add(user_id);
    context.setAttribute("user_list",user_list);
}
```

최초 로그인 시 접속자 ID를 ArrayList에
차례로 저장한 후 다시 context 객체에
속성으로 저장합니다.

```
out.println("<html><body>");
```

```
out.println("아이디는 " + loginUser.user_id + "<br>");
```

세션에 바인딩 이벤트 처리 후
총 접속자수를 표시합니다.

```
out.println("총 접속자 수는"+LoginImpl.total_user + "<br><br>");
```

```
out.println("접속 아이디:<br>");
```

```
List list=(ArrayList)context.getAttribute("user_list");
```

context 객체의 ArrayList를 가
져와 접속자 ID를 차례로 브라
우저로 출력합니다.

```
for(int i=0; i<list.size();i++){
    out.println(list.get(i)+"<br>");
}
```

```
out.println("<a href='logout?user_id="+user_id+"'>로그아웃 </a>");
```

```
out.println("</body></html>");
```

로그아웃 클릭 시 서블릿
logout으로 접속자 ID를 전송
해 로그아웃합니다.

```
}
```

```
}
```




10.4 여러 가지 서블릿 관련 Listener API

3. LogoutTest 클래스를 다음과 같이 작성합니다.

코드 10-14 pro10/src/sec04/ex02/LogoutTest.java

```
package sec04.ex02;
```

```
...
```

```
@WebServlet("/logout")
```

```
public class LogoutTest extends HttpServlet {
```

```
    ServletContext context;
```

```
protected public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    doHandle(request,response);
}
```

```
protected public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    doHandle(request,response);
}
```

```
private public void doHandle(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    request.setCharacterEncoding( "utf-8" );
```



10.4 여러 가지 서블릿 관련 Listener API

```
response.setContentType("text/html;charset=utf-8");
context=getServletContext();
PrintWriter out = response.getWriter();
HttpSession session=request.getSession();
String user_id = request.getParameter("user_id");
session.invalidate();
List user_list=(ArrayList)context.getAttribute("user_list");
user_list.remove(user_id);
context.removeAttribute("user_list");
context.setAttribute("user_list", user_list);
out.println("<br>로그아웃했습니다.");
}
}
```

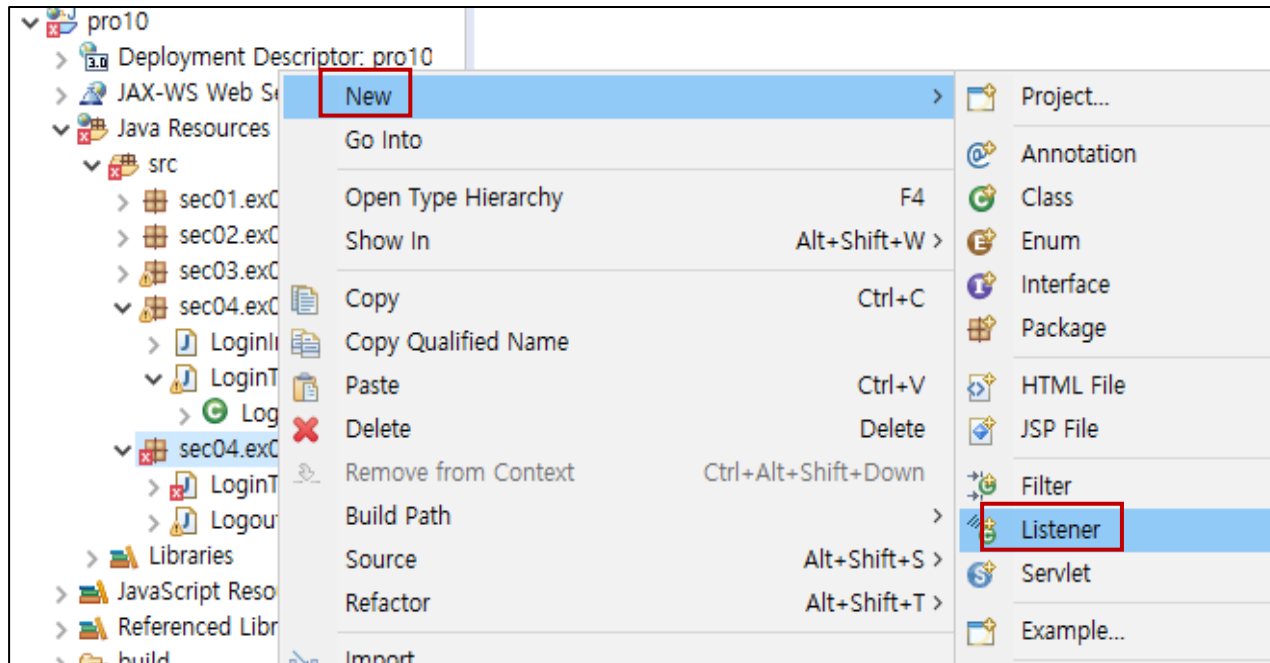
user_list에서 삭제할 ID를 가져옵니다.

로그아웃 시 세션을 소멸시킵니다.

user_list에서 로그아웃한 접속자 ID를 삭제한 후 다시 user_list를 컨텍스트에 저장합니다.

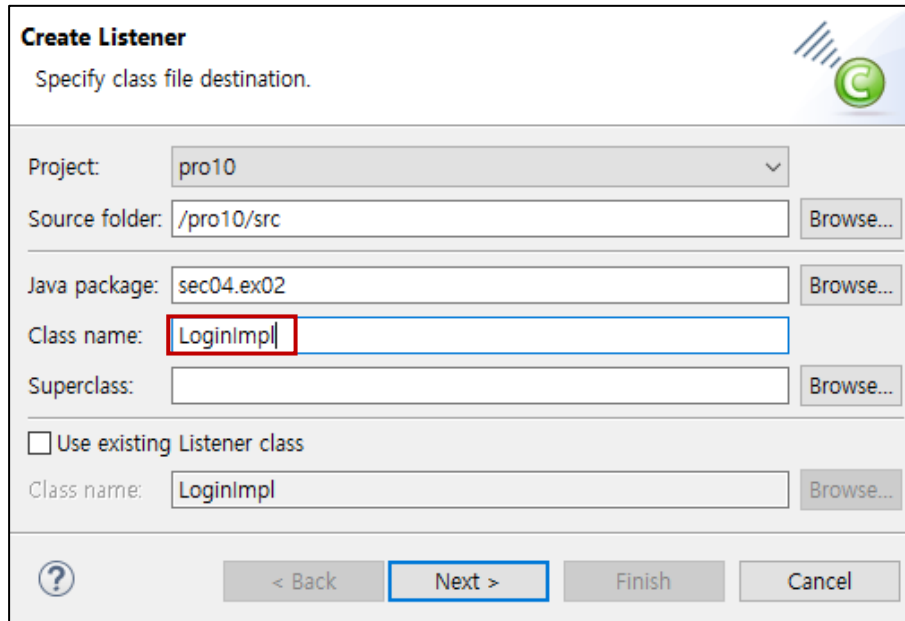
10.4 여러 가지 서블릿 관련 Listener API

1. sec04.ex02 패키지를 선택하고 마우스 오른쪽 버튼을 클릭한 후 New > Listener를 선택합니다.



10.4 여러 가지 서블릿 관련 Listener API

2. Class name으로 LoginImpl을 입력하고 Next를 클릭합니다.



Create Listener
Specify class file destination.

Project: pro10

Source folder: /pro10/src Browse...

Java package: sec04.ex02 Browse...

Class name: LoginImpl

Superclass: Browse...

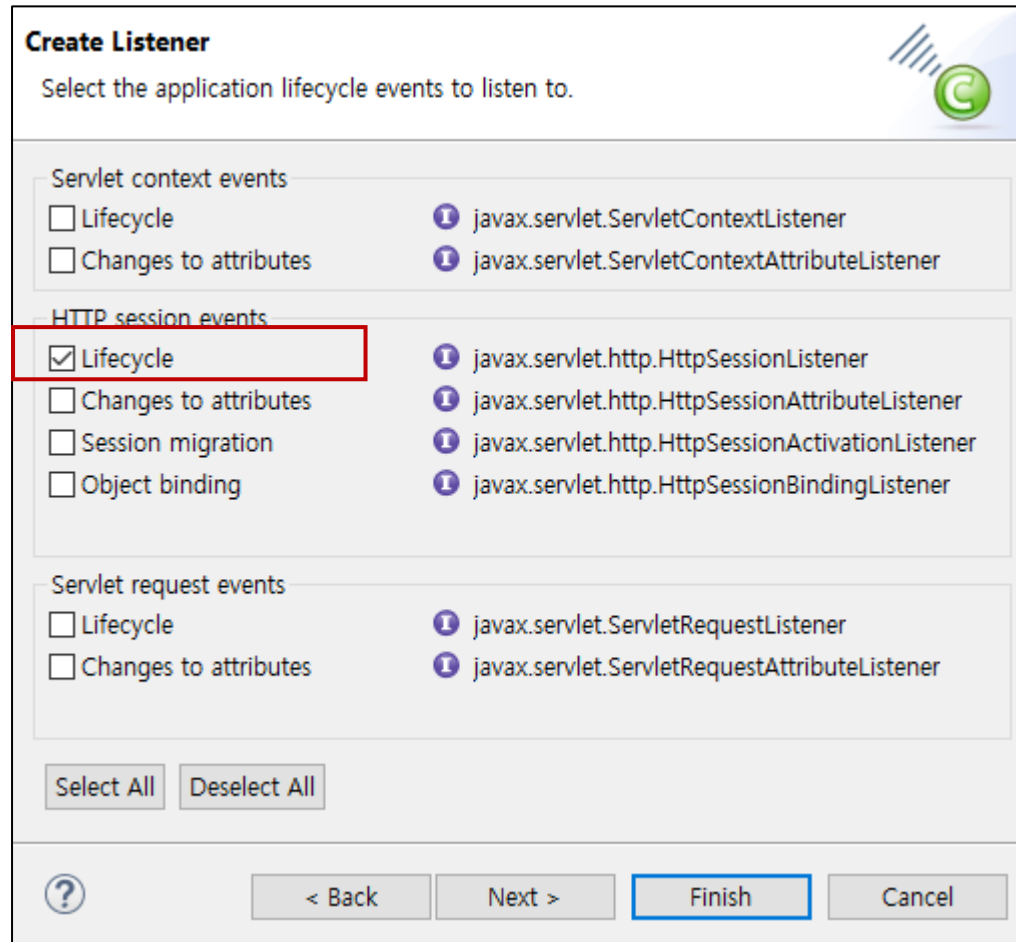
☐ Use existing Listener class

Class name: LoginImpl Browse...

? < Back Next > Finish Cancel

10.4 여러 가지 서블릿 관련 Listener API

3. HttpSessionListener에 체크하고 Next를 클릭합니다.



The image shows a 'Create Listener' dialog box with a title bar and a green 'C' icon. The main text says 'Select the application lifecycle events to listen to.' The dialog is divided into three sections: 'Servlet context events', 'HTTP session events', and 'Servlet request events'. Each section has a list of events with checkboxes and corresponding listener classes. In the 'HTTP session events' section, the 'Lifecycle' checkbox is checked and highlighted with a red rectangle. At the bottom, there are 'Select All' and 'Deselect All' buttons, and a footer with a help icon, '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel' buttons.

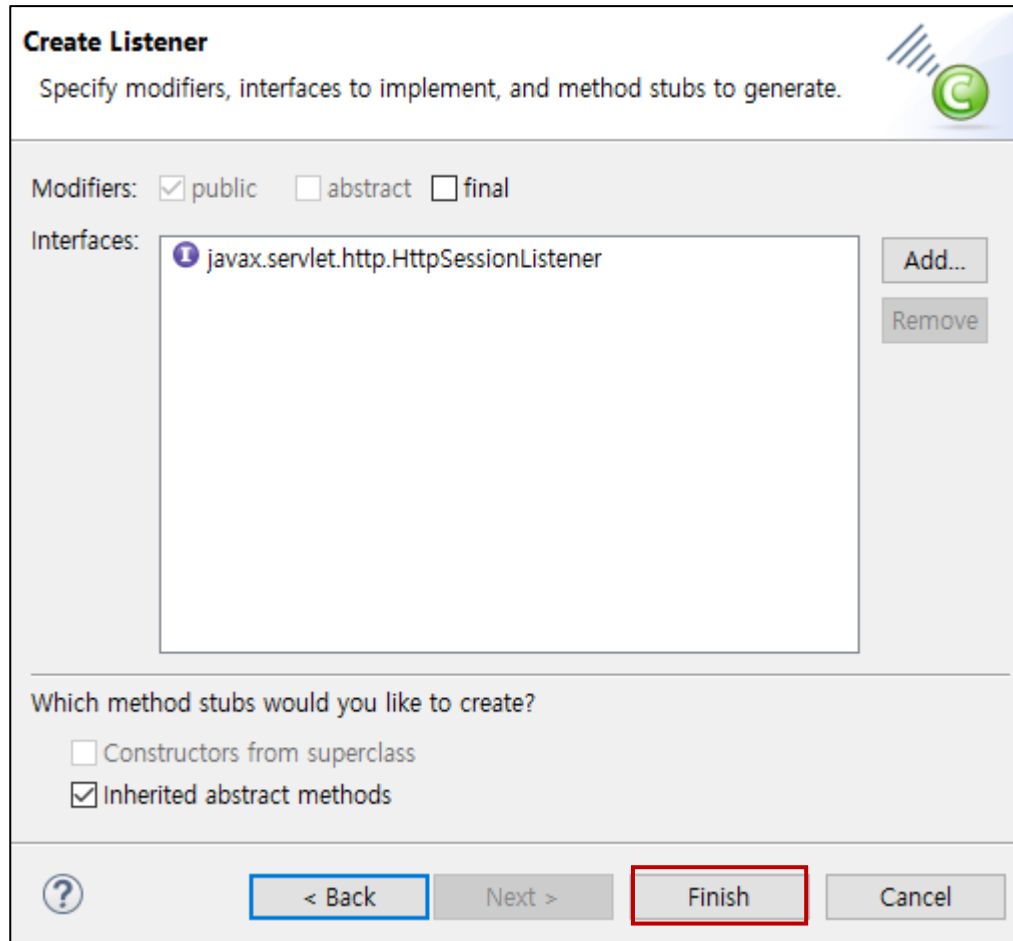
Event	Listener Class
Servlet context events	
<input type="checkbox"/> Lifecycle	<code>javax.servlet.ServletContextListener</code>
<input type="checkbox"/> Changes to attributes	<code>javax.servlet.ServletContextAttributeListener</code>
HTTP session events	
<input checked="" type="checkbox"/> Lifecycle	<code>javax.servlet.http.HttpSessionListener</code>
<input type="checkbox"/> Changes to attributes	<code>javax.servlet.http.HttpSessionAttributeListener</code>
<input type="checkbox"/> Session migration	<code>javax.servlet.http.HttpSessionActivationListener</code>
<input type="checkbox"/> Object binding	<code>javax.servlet.http.HttpSessionBindingListener</code>
Servlet request events	
<input type="checkbox"/> Lifecycle	<code>javax.servlet.ServletRequestListener</code>
<input type="checkbox"/> Changes to attributes	<code>javax.servlet.ServletRequestAttributeListener</code>

Select All Deselect All

? < Back Next > Finish Cancel

10.4 여러 가지 서블릿 관련 Listener API

4. Finish를 클릭합니다.



Create Listener
Specify modifiers, interfaces to implement, and method stubs to generate.

Modifiers: ☒ public ☐ abstract ☐ final

Interfaces: ☒ javax.servlet.http.HttpSessionListener

Add...
Remove

Which method stubs would you like to create?

☐ Constructors from superclass
☒ Inherited abstract methods

? < Back Next > **Finish** Cancel



10.4 여러 가지 서블릿 관련 Listener API

5. @WebListener 애너테이션으로 리스너가 등록된 것을 확인할 수 있습니다.

```
10  */
11  @WebListener
12  public class LoginImpl implements HttpSessionListener {
13
14      /**
15       * Default constructor.
16       */
17      public LoginImpl() {
18          // TODO Auto-generated constructor stub
19      }
20
21      /**
```



10.4 여러 가지 서블릿 관련 Listener API

6. 리스너를 등록한 이벤트 핸들러를 이용해서 세션을 생성할 때는 `sessionCreated()` 메서드로 이벤트를 처리하고, 세션을 삭제할 때는 `sessionDestroyed()` 메서드로 이벤트를 처리합니다.

코드 10-15 pro10/src/sec04/ex02/LoginImpl.java

```
package sec04.ex02;
```

```
import javax.servlet.annotation.WebListener;  
import javax.servlet.http.HttpSessionEvent;  
import javax.servlet.http.HttpSessionListener;
```

@WebListener

HttpSessionBindingListener를 제외한 Listener를 구현한 모든 이벤트
핸들러는 반드시 애너테이션을 이용해서 Listener로 등록해야 합니다.

```
public class LoginImpl implements HttpSessionListener {  
    String user_id;  
    String user_pw;  
    static int total_user=0;  
  
    public LoginImpl() {  
    }  
  
    public LoginImpl(String user_id, String user_pw) {  
        this.user_id = user_id;  
        this.user_pw = user_pw;  
    }  
}
```


10.4 여러 가지 서블릿 관련 Listener API

@Override

```
public void sessionCreated(HttpSessionEvent arg0) {  
    System.out.println("세션 생성");  
    ++total_user;  
}
```

세션 생성 시 이벤트를 처리합니다.

세션 생성 시 접속자수를 1 증가시킵니다.

@Override

```
public void sessionDestroyed(HttpSessionEvent arg0) {  
    System.out.println("세션 소멸");  
    --total_user;  
}
```

세션 소멸 시 이벤트를 처리합니다.

세션 소멸 시 접속자수를 1 감소시킵니다.

}

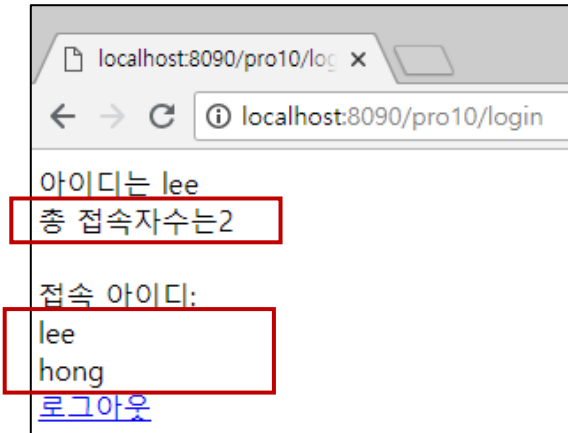
10.4 여러 가지 서블릿 관련 Listener API

7. 실행하면 사용자마다 로그인/로그아웃 시 접속자수와 접속자 ID를 표시해 줍니다. 다음은 첫 번째 아이디로 로그인한 결과입니다.

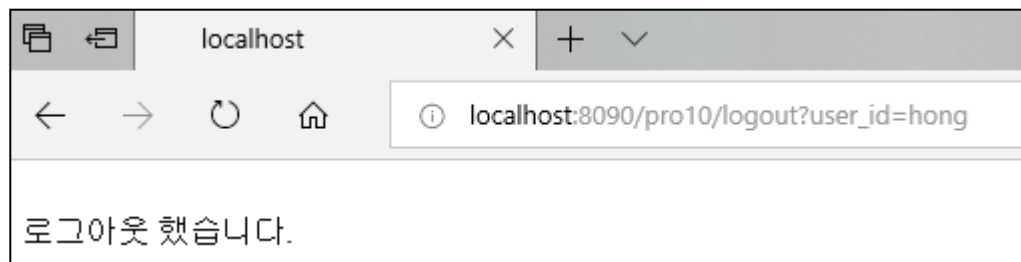
8. 이번에는 인터넷 익스플로러에서 두 번째 ID로 로그인하면 다음과 같이 현재 접속자수와 접속자 ID가 출력됩니다.

10.4 여러 가지 서블릿 관련 Listener API

9. 다시 크롬에서 화면을 갱신하면 다음과 같이 현재 접속자수와 접속자 ID가 표시됩니다.

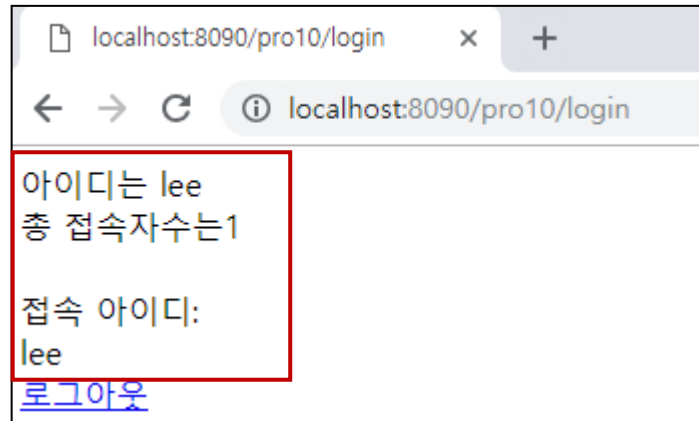


10. 익스플로러에서 로그아웃을 클릭합니다.



10.4 여러 가지 서블릿 관련 Listener API

11. 크롬에서 화면을 재요청하면 다음과 같이 현재 접속자수와 접속자 ID가 표시됩니다.



10.4 여러 가지 서블릿 관련 Listener API

실습 예제

1. 사용자는 자신의 아이디로 모든 기기에서 한 번만 로그인 가능하게 구현하라.
2. 하나의 아이디로 모든 기기에서 동시에 접속 가능 횟수를 5회로 구현하라.

방법

- ① 로그인 시 기존 로그인 정보를 ServletContext에서 얻습니다.
- ② 로그인 아이디와 로그인 정보의 아이디가 같은 지 체크합니다.
- ③ 로그인 정보에 아이디가 없으면 로그인 아이디를 ServletContext에 바인딩 후 로그인 합니다.
- ④ 로그인 정보가 이미 존재하면 “이미 로그인 중입니다.”라는 메시지를 표시합니다.