

5 장

서블릿 이해하기

- 5.1 서블릿이란?
- 5.2 서블릿 API 계층 구조와 기능
- 5.3 서블릿의 생명주기 메서드
- 5.4 FirstServlet을 이용한 실습
- 5.5 서블릿 동작 과정
- 5.6 애너테이션을 이용한 서블릿 매팅

정적 웹 페이지의 문제점을 보완하여 나온 것이 동적 웹 페이지를 구현하는 JSP라고 했습니다. 그러나 사실 동적 웹 페이지를 처음으로 구현한 방법은 JSP가 아니었습니다. 초기 동적 웹 페이지들은 이 장에서 배울 서블릿(Servlet, 자바로 만든 CGI 프로그램)을 이용해서 구현했습니다.

그리고 이 서블릿의 문제점을 보완하여 나온 것이 JSP입니다. JSP의 많은 기능은 서블릿의 기능을 따르므로 서블릿을 먼저 이해하고 나면 JSP를 좀 더 수월하게 이해할 수 있을 것입니다. 실제 웹 애플리케이션을 개발할 때도 JSP와 서블릿이 각자의 고유한 역할을 나누어 기능을 구현하고 있습니다. 이 장에서는 서블릿에 대한 기본적인 내용을 실습을 통해 알아보겠습니다.

5.1 서블릿이란?

JAVA WEB

서블릿은 서버 쪽에서 실행되면서 클라이언트의 요청에 따라 동적으로 서비스를 제공하는 자바 클래스입니다. 서블릿은 자바로 작성되어 있으므로 자바의 일반적인 특징을 모두 가집니다. 하지만 서블릿은 일반 자바 프로그램과 다르게 독자적으로 실행되지 못하고 톰캣과 같은 JSP/Servlet 컨테이너에서 실행된다는 점에서 차이가 있습니다.

▼ 그림 5-1 자바 서블릿



Note 이 책에서는 JSP/Servlet 컨테이너로 자바 기반 오픈 소스로 제공되는 톰캣을 사용합니다. 그 외 자바 기반의 JEUS와 Web Logic, 웹스피어(WebShpere)는 IBM 등 특정 소프트웨어 회사가 개발해서 유료로 제공하는 JSP/Servlet 컨테이너도 있습니다. 또 다른 자바 기반 오픈 소스 JSP/Servlet 컨테이너로 JBOSS는 지금은 거의 사용하지 않는 EJB 컨테이너 기능도 제공합니다.

서블릿은 서버에서 실행되다가 웹 브라우저에서 요청을 하면 해당 기능을 수행한 후 웹 브라우저에 결과를 전송합니다. 서버에서 실행되므로 보안과 관련된 기능도 훨씬 안전하게 수행할 수 있습니다.

그림 5-2에 서블릿의 동작 과정을 나타내었습니다.

▼ 그림 5-2 서블릿 동작 과정



클라이언트가 웹 서버에 요청하면 웹 서버는 그 요청을 톰캣과 같은 웹 애플리케이션 서버(WAS)에 위임합니다. 그러면 WAS는 각 요청에 해당하는 서블릿을 실행합니다. 그리고 서블릿은 요청에 대한 기능을 수행한 후 결과를 반환하여 클라이언트에 전송하죠.

그럼 서블릿은 어떤 기능이 있는지 알아볼까요? 그림 5-2를 보면 단순히 고정된 정보를 브라우저에 보여주는 용도는 웹 서버로도 충분합니다. 그러나 쇼핑몰 화면에 실시간으로 변하는 상품의 할인 가격을 보여주려면 상품의 할인 가격을 데이터베이스에서 가져오는 기능이나 직접 계산하는 기능이 필요합니다. 따라서 그런 기능을 서버 쪽에서 서블릿이 처리해 주면 상품 할인 가격 표시처럼 웹 페이지에서 동적으로 변하는 정보를 효과적으로 다룰 수 있습니다.

그 외 서블릿은 다음과 같은 특징이 있습니다.

- 서버 쪽에서 실행되면서 기능을 수행합니다.
- 기존의 정적인 웹 프로그램의 문제점을 보완하여 동적인 여러 가지 기능을 제공합니다.
- 스레드 방식으로 실행됩니다.
- 자바로 만들어져 자바의 특징(객체 지향)을 가집니다.
- 컨테이너에서 실행됩니다.
- 컨테이너 종류에 상관없이 실행됩니다(플랫폼 독립적).
- 보안 기능을 적용하기 쉽습니다.
- 웹 브라우저에서 요청 시 기능을 수행합니다.

5.2

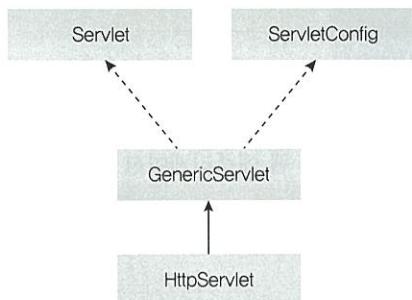
서블릿 API 계층 구조와 기능

서블릿은 자바로 만들어졌으므로 당연히 클래스들 간의 계층 구조를 가집니다. 그림 5-3에 서블릿 관련 클래스들의 계층 구조를 나타내었습니다.

Tip

인터페이스와 추상 클래스 개념이 어려운 사람은 자바 입문서를 다시 보기 바랍니다.

▼ 그림 5-3 서블릿 클래스 계층 구조



서블릿 API는 `Servlet`과 `ServletConfig` 인터페이스를 구현해 제공하며 `GenericServlet` 추상 클래스가 이 두 인터페이스의 추상 메서드를 구현합니다. 그리고 이 `GenericServlet`을 다시 `HttpServlet`이 상속받습니다.

5.2.1 서블릿 API 기능

표 5-1에 서블릿 API를 구성하는 여러 구성 요소들의 특징을 정리했습니다.

▼ 표 5-1 서블릿 API 구성 요소 특징

서블릿 구성 요소	기능
Servlet 인터페이스	<ul style="list-style-type: none"> <code>javax.servlet</code> 패키지에 선언되어 있습니다. Servlet 관련 추상 메서드를 선언합니다. <code>init()</code>, <code>service()</code>, <code>destroy()</code>, <code>getServletInfo()</code>, <code>getServletConfig()</code>를 선언합니다.

서블릿 구성 요소	기능
ServletConfig 인터페이스	<ul style="list-style-type: none"> javax.servlet 패키지에 선언되어 있습니다. Servlet 기능 관련 추상 메서드가 선언되어 있습니다. getInitParameter(), getInitParameterNames(), getServletContext(), getServletName()이 선언되어 있습니다.
GenericServlet 클래스	<ul style="list-style-type: none"> javax.servlet 패키지에 선언되어 있습니다. 상위 두 인터페이스를 구현하여 일반적인 서블릿 기능을 구현한 클래스입니다. GenericServlet을 상속받아 구현한 사용자 서블릿은 사용되는 프로토콜에 따라 각각 service()를 오버라이딩해서 구현합니다.
HttpServlet 클래스	<ul style="list-style-type: none"> javax.servlet.http 패키지에 선언되어 있습니다. GenericServlet을 상속받아 HTTP 프로토콜을 사용하는 웹 브라우저에서 서블릿 기능을 수행합니다. 웹 브라우저 기본 서비스를 제공하는 서블릿을 만들 때 상속받아 사용합니다. 요청 시 service()가 호출되면서 요청 방식에 따라 doGet()이나 doPost()가 차례대로 호출됩니다.

GenericServlet은 일반적인 여러 통신 프로토콜에 대한 클라이언트/서버 프로그램에서 서블릿 기능을 구현하는 클래스입니다. HttpServlet은 이 GenericServlet을 상속받습니다. HttpServlet은 이름에서 알 수 있듯이 HTTP 프로토콜을 사용하는 서블릿 기능을 구현하는 클래스입니다. 바로 이 HttpServlet을 상속받아 HTTP 프로토콜로 동작하는 웹 브라우저의 요청을 처리하는 서블릿이 바로 이 책에서 만들어 사용할 서블릿입니다. 그 외 다른 서블릿 구성 요소들의 기능은 API 문서를 참고하기 바랍니다.

표 5-2에 HttpServlet의 주요 메서드와 그 기능을 정리했습니다.

▼ 표 5-2 HttpServlet의 여러 가지 메서드 기능

메서드	기능
protected doDelete(HttpServletRequest req, HttpServletResponse resp)	서블릿이 DELETE request를 수행하기 위해 service()를 통해서 호출됩니다.
protected doGet(HttpServletRequest req, HttpServletResponse resp)	서블릿이 GET request를 수행하기 위해 service()를 통해서 호출됩니다.
protected doHead(HttpServletRequest req, HttpServletResponse resp)	서블릿이 HEAD request를 수행하기 위해 service()를 통해서 호출됩니다.
protected doPost(HttpServletRequest req, HttpServletResponse resp)	서블릿이 POST request를 수행하기 위해 service()를 통해서 호출됩니다.

메서드	기능
<code>protected service (ServletRequest req, ServletResponse resp)</code>	request를 public service()에서 전달받아 doXXX() 메서드를 호출합니다.
<code>public service (ServletRequest req, ServletResponse resp)</code>	클라이언트의 request를 protected service()에게 전달합니다.

표를 자세히 보면 클라이언트 요청 시 `public service()` 메서드를 먼저 호출한 후 다시 `protected service()` 메서드를 호출합니다. 그런 다음 다시 `request` 종류에 따라 `doXXX()` 메서드를 호출하는 과정으로 실행됩니다. 자세한 것은 다음 절에서 알아보겠습니다.

5.3 서블릿의 생명주기 메서드

5.1절에서는 서블릿의 동작 과정을 살펴보았습니다. 서블릿도 자바 클래스이므로 실행하면 당연히 초기화 과정 그리고 메모리에 인스턴스를 생성하여 서비스를 수행한 후 다시 소멸하는 과정을 거칩니다. 이런 단계를 거칠 때마다 서블릿 클래스의 메서드가 호출되어 초기화, 데이터베이스 연동, 마무리 작업을 수행합니다. 각 과정에서 호출되어 기능을 수행하는 메서드들이 서블릿 생명주기 메서드입니다.

따라서 서블릿 생명주기(Life Cycle) 메서드란 서블릿 실행 단계마다 호출되어 기능을 수행하는 콜백 메서드를 말합니다. 표 5-3에 서블릿의 생명주기 메서드들과 그 특징을 정리했습니다.

▼ 표 5-3 서블릿의 생명주기 메서드 기능

생명주기 단계	호출 메서드	특징
초기화	<code>init()</code>	<ul style="list-style-type: none"> 서블릿 요청 시 맨 처음 한 번만 호출됩니다. 서블릿 생성 시 초기화 작업을 주로 수행합니다.
작업 수행	<code>doGet()</code> <code>doPost()</code>	<ul style="list-style-type: none"> 서블릿 요청 시 매번 호출됩니다. 실제로 클라이언트가 요청하는 작업을 수행합니다.
종료	<code>destroy()</code>	<ul style="list-style-type: none"> 서블릿이 기능을 수행하고 메모리에서 소멸될 때 호출됩니다. 서블릿의 마무리 작업을 주로 수행합니다.

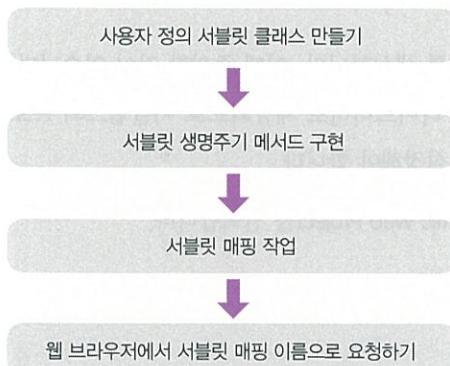
`int()` 메서드는 실행 초기에 서블릿 기능 수행과 관련된 기능을 설정하는 용도로 많이 사용됩니다. 그리고 `destroy()`는 서블릿이 메모리에서 소멸될 때 여러 가지 종료 작업을 수행합니다. 따라서 만약 이런 기능이 필요 없으면 생략해도 상관없습니다. 반면에 `doGet()`이나 `doPost()`와 같이 `do`로 시작하는 메서드는 서블릿의 핵심 기능을 처리하므로 반드시 구현해야 합니다.

5.4 FirstServlet을 이용한 실습

JAVA WEB

앞 절에서는 서블릿 기능과 각 생명주기 메서드 기능을 알아보았습니다. 이번에는 사용자 정의 서블릿을 실제로 만들어서 서블릿의 동작 과정을 실습해 보겠습니다.

다음은 이클립스에서 서블릿을 만들고 실행하는 과정입니다.



5.4.1 사용자 정의 서블릿 만들기

실제 웹 프로그래밍에서 사용되는 사용자 정의 서블릿은 `HttpServlet` 클래스를 상속받아서 만듭니다.

그리고 3개의 생명주기 메서드, 즉 `init()`, `doGet()`, `destory()` 메서드를 오버라이딩해서 기능을 구현합니다.

코드 사용자 정의 서블릿 형식

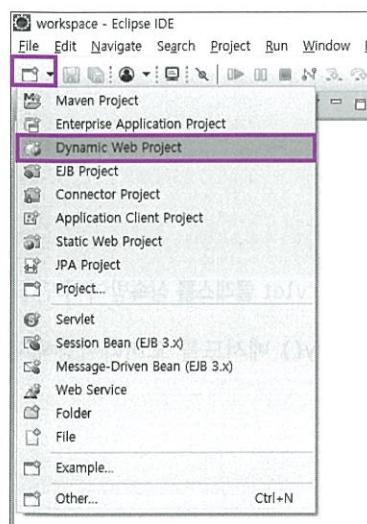
```
public class FirstServlet extends HttpServlet {  
    @Override  
    public void init() {  
        ...  
    }  
    @Override  
    public void doGet(HttpServletRequest req, HttpServletResponse resp) {  
        ...  
    }  
  
    @Override  
    public void destroy() {  
        ...  
    }  
}
```

5.4.2 톰캣의 servlet-api.jar 클래스 패스 설정하기

사용자 정의 서블릿을 실습하기 위해 새로운 프로젝트를 생성합니다. 이때 주의할 것이 있습니다. 앞에서 설명한 서블릿 API들은 톰캣의 servlet-api.jar 라이브러리로 제공되므로 이클립스의 프로젝트에서 서블릿을 사용하려면 반드시 클래스 패스를 설정해야 합니다.

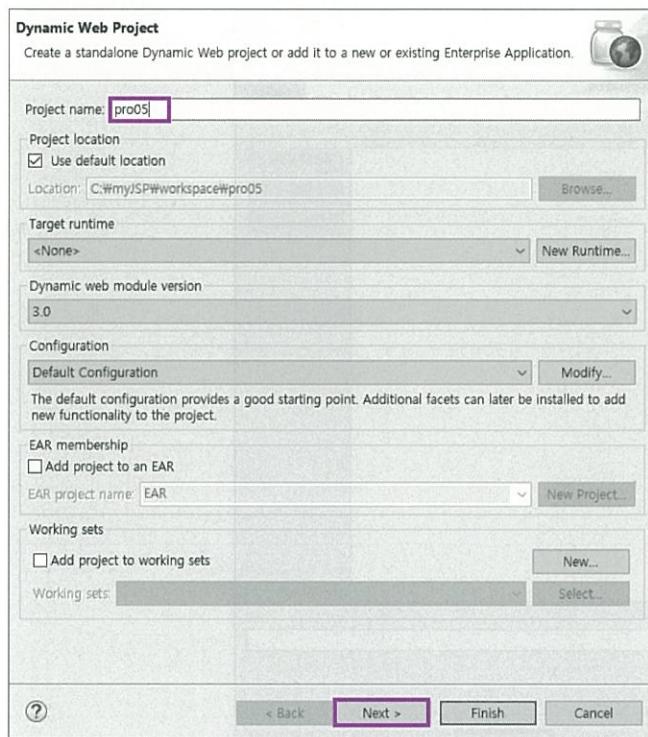
1. 이클립스 상단의 New 아이콘을 클릭한 후 Dynamic Web Project를 선택합니다.

▼ 그림 5-4 New(□) 아이콘 클릭 후 Dynamic Web Project 선택



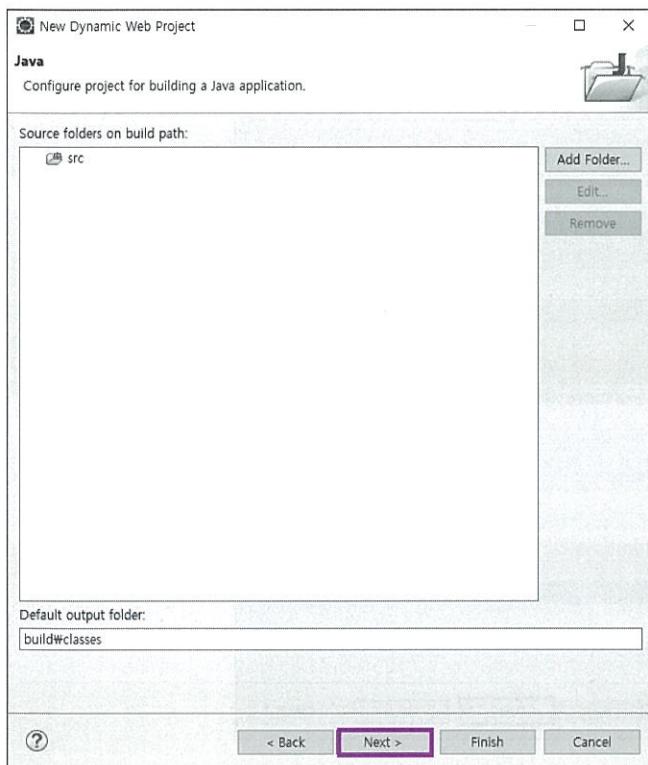
2. 프로젝트 이름을 pro05로 입력한 후 Next를 클릭합니다.

▼ 그림 5-5 프로젝트 이름 입력 후 Next 클릭



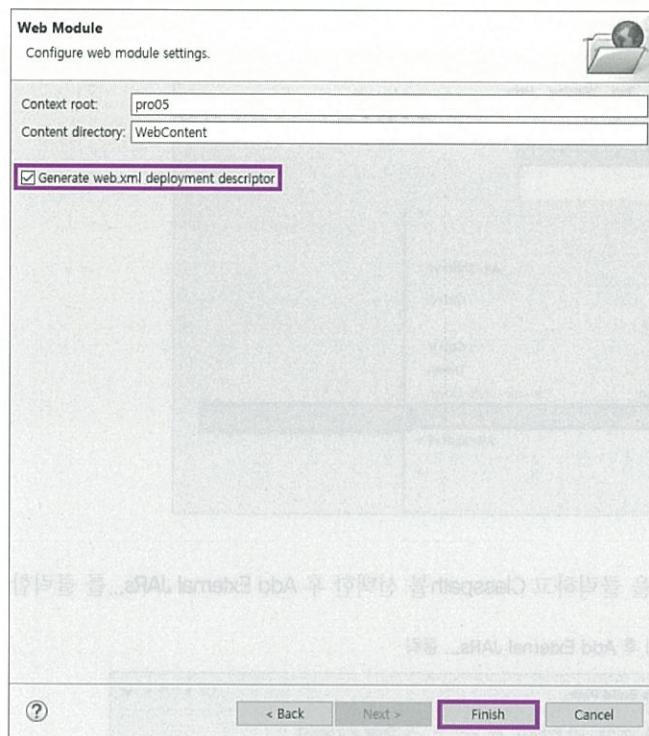
3. 경로를 확인한 후 Next를 클릭합니다.

▼ 그림 5-6 경로 확인 후 Next 클릭



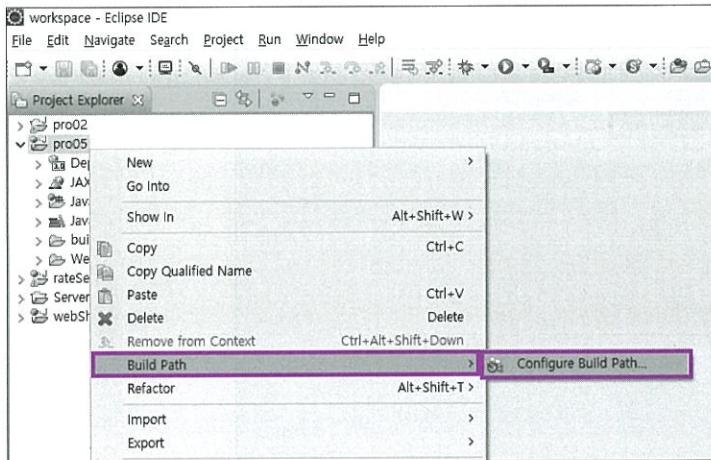
4. Generate web.xml deployment descriptor 옵션의 체크박스에 체크한 후 Finish를 클릭합니다.

▼ 그림 5-7 옵션 체크 후 Finish 클릭



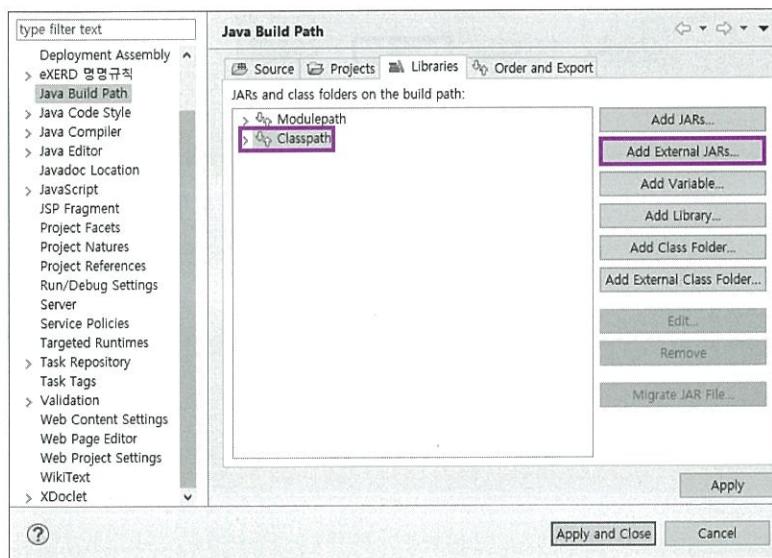
5. 프로젝트 이름을 선택하고 마우스 오른쪽 버튼을 클릭한 후 Build Path > Configure Build Path...를 선택합니다.

▼ 그림 5-8 Build Path > Configure Build Path... 선택



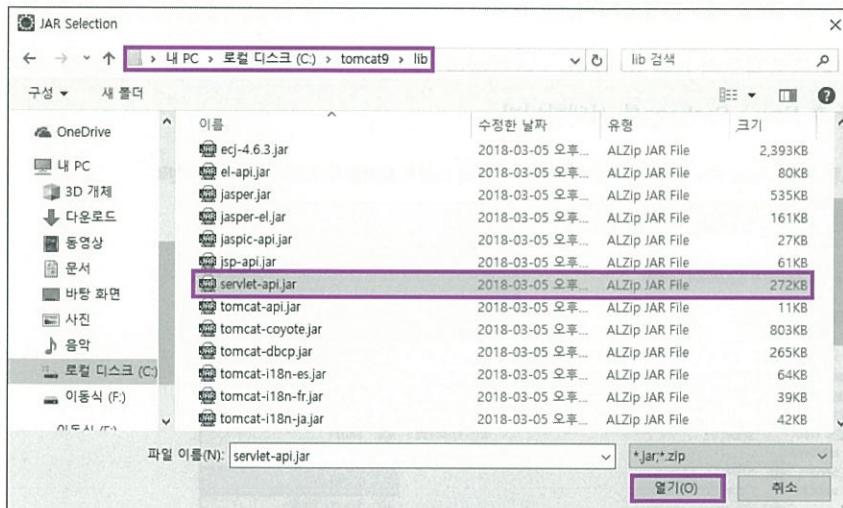
6. 설정창에서 Libraries 탭을 클릭하고 Classpath를 선택한 후 Add External JARs...를 클릭합니다.

▼ 그림 5-9 Libraries 탭 클릭 후 Add External JARs... 클릭



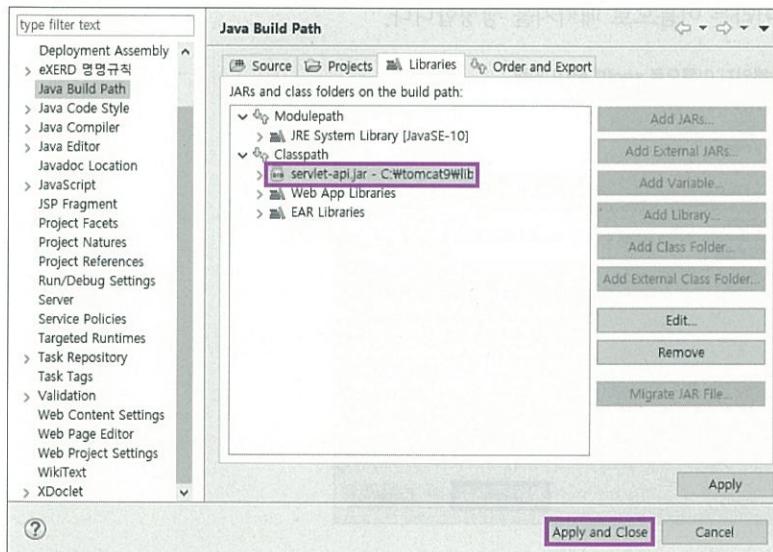
7. CATALINA_HOME(톰캣 루트 디렉터리)의 lib 디렉터리에 있는 servlet-api.jar을 선택한 후 열기를 클릭합니다.

▼ 그림 5-10 servlet-api.jar 선택 후 열기 클릭



8. servlet-api.jar 클래스의 패스 설정을 확인한 후 Apply and Close를 클릭해 종료합니다.

▼ 그림 5-11 servlet-api.jar 클래스 패스 설정 확인 후 Apply and Close 클릭

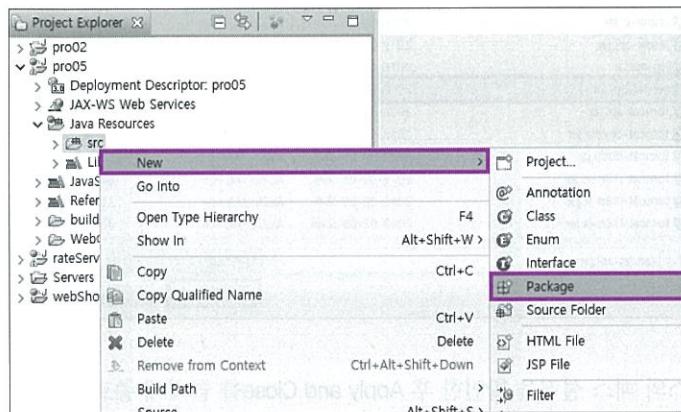


5.4.3 첫 번째 서블릿 만들기

이번에는 실제로 브라우저의 요청을 처리하는 첫 번째 서블릿을 만들어 보겠습니다. 우선 FirstServlet 클래스를 생성합니다.

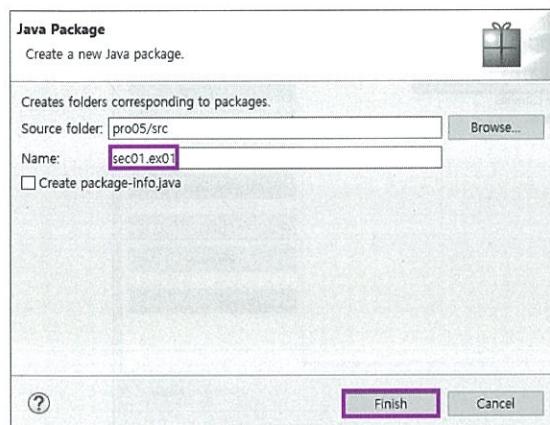
1. pro05 프로젝트의 Java Resources 디렉터리 하위의 src를 선택하고 마우스 오른쪽 버튼을 클릭한 후 New > Package를 선택합니다.

▼ 그림 5-12 Java Resource > src에서 마우스 오른쪽 버튼을 클릭한 후 New > Package 선택



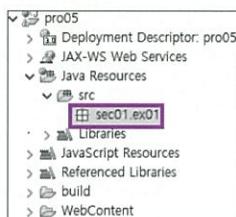
2. sec01.ex01]라는 이름으로 패키지를 생성합니다.

▼ 그림 5-13 패키지 이름으로 sec01.ex01 입력



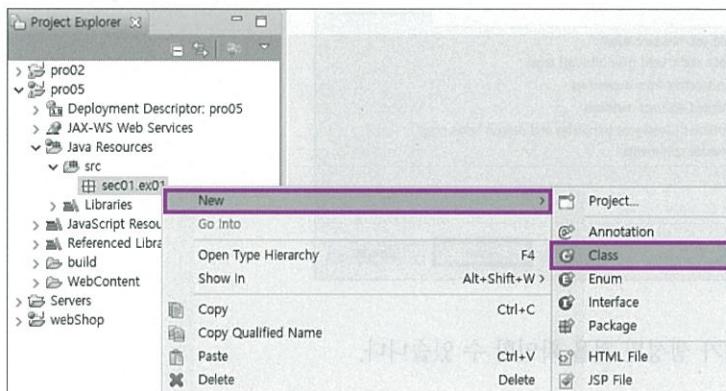
3. Project Explorer에서 src 하위에 sec01.ex01이라는 패키지가 생긴 것을 확인할 수 있습니다.

▼ 그림 5-14 sec01.ex01 패키지 생성 확인



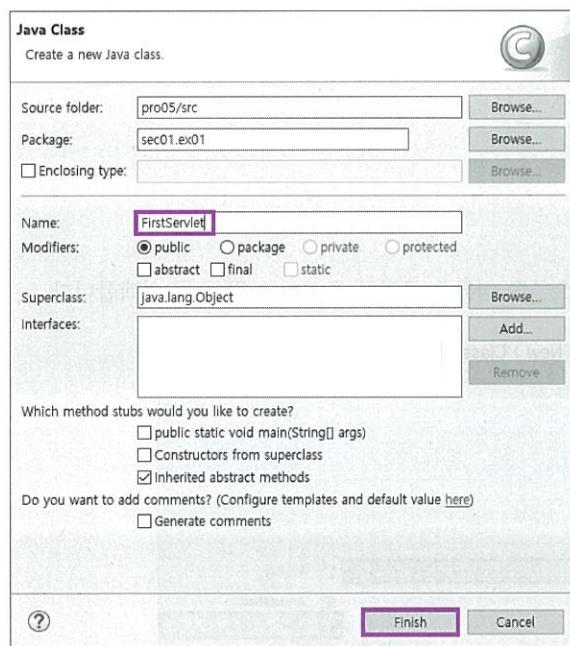
4. 이 패키지 이름 위에서 마우스 오른쪽 버튼을 클릭한 후 New > Class를 선택합니다.

▼ 그림 5-15 마우스 오른쪽 버튼 클릭 후 New > Class 선택



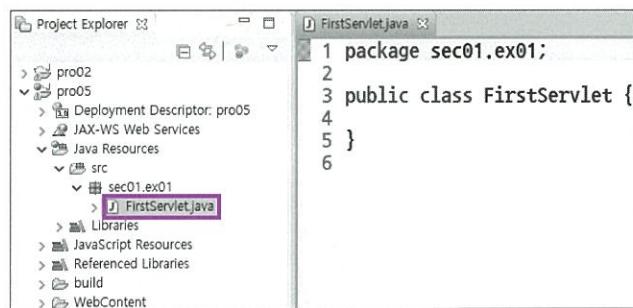
5. 클래스 이름으로 FirstServlet을 입력한 후 Finish를 클릭합니다.

▼ 그림 5-16 클래스 이름으로 FirstServlet 입력 후 Finish 클릭



6. FirstServlet.java가 생성된 것을 확인할 수 있습니다.

▼ 그림 5-17 FirstServlet.java 생성 확인



7. 자, 이제 이클립스에서 생성한 FirstServlet.java에 다음과 같이 자바 코드를 작성합니다.

코드 5-1 pro05/src/sec01/ex01/FirstServlet.java

```
package sec01.ex01;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FirstServlet extends HttpServlet{ ← HttpServlet를 상속받습니다.

    @Override
    public void init() throws ServletException {
        System.out.println("init 메서드 호출"); ← 브라우저의 요청을 처리합니다.
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        System.out.println("doGet 메서드 호출");
    }

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

우리가 만든 서블릿은 HttpServlet을 상속받고 3개의 생명주기 메서드를 차례로 구현합니다. 각 메서드는 호출 시 메시지만 출력합니다.

소스 코드를 작성했으니 이제 첫 번째 서블릿을 어떻게 실행하여 동작시키는지 알아보겠습니다.

5.4.4 서블릿 매핑하기

브라우저에서 서블릿 이름으로 요청하는 방법은 다음과 같습니다. 프로젝트 이름 뒤에 패키지 이름이 포함된 클래스 이름 전부를 입력합니다.

▼ 그림 5-18 브라우저에서 서블릿 요청 방법

http://IP주소:포트번호/프로젝트이름/패키지이름이 포함된 클래스이름



http://127.0.0.1:8090/pro05/sec01.ex01.FirstServlet

pro05 프로젝트를 톰캣에 추가한 후 패키지 이름까지 포함된 서블릿 클래스 이름인 sec01.ex01.FirstServlet으로 요청해야 합니다. 그런데 클래스 이름이 길어지면 입력하기가 불편하겠죠. 그리고 일반적으로 클래스 이름을 보면 그 클래스가 어떤 기능을 하는지 짐작할 수 있는데, 브라우저에서 버젓이 클래스 이름으로 입력하면 보안에도 좋지 않습니다. 따라서 지금은 이런 식으로 사용하지 않고 서블릿 클래스 이름에 대응되는 서블릿 매핑 이름으로 실제 서블릿을 요청합니다.

그럼 지금부터는 서블릿 매핑 방법에 대해서 알아보겠습니다. 서블릿 매핑 과정은 다음과 같습니다.

① 각 프로젝트에 있는 web.xml에서 설정합니다.

① <servlet> 태그와 <servlet-mapping> 태그를 이용합니다.

② 여러 개의 서블릿 매핑 시에는 <servlet> 태그를 먼저 정의하고 <servlet-mapping> 태그를 정의합니다.

실제 서블릿 매핑을 보면 <servlet> 태그와 <servlet-mapping> 태그의 하위 태그에 <servlet-name> 태그가 공통으로 있습니다. <servlet-name> 태그의 값 aaa가 <servlet>과 <servlet-mapping> 태그를 연결시켜 줍니다.

그러면 웹 브라우저에서 <url-pattern> 태그의 /first로 요청할 경우 aaa 값을 가지는 <servlet> 태그를 찾아 실제 서블릿인 sec01.ex01.FirstServlet을 실행합니다.

코드 5-2

```
<servlet>
    <servlet-name>aaa</servlet-name>
    <servlet-class>sec01.ex01.FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>aaa</servlet-name>
    <url-pattern>/first</url-pattern>
</servlet-mapping>
```

〈servlet〉 태그와 〈servlet-mapping〉 태그를 연결시켜 줍니다.

웹 브라우저에서 요청하는 매핑 이름

그럼 지금부터 서블릿 매핑 형식을 실제 프로젝트에 적용해 보겠습니다.

1. pro05 프로젝트의 WebContent > WEB-INF 폴더를 클릭한 후 web.xml을 선택하여 엽니다.

▼ 그림 5-19 web.xml 선택



2. web.xml에 <web-app> 태그의 하위 태그를 지우고 다음과 같이 서블릿 매핑을 작성합니다.

코드 5-3 pro05/WebContent/WEB-INF/web.xml

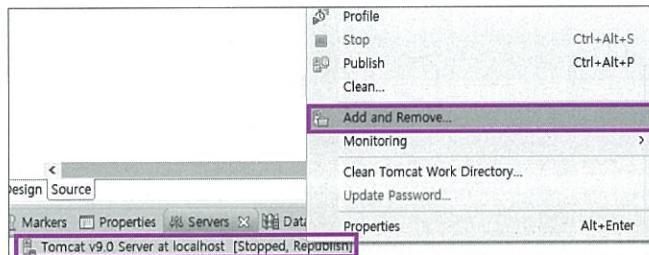
```
<servlet> ━━━━ 브라우저에서 요청하는 매핑 이름에 대해 실제로 실행하는 서블릿 클래스를 설정하는 태그입니다.  
    <servlet-name>aaa</servlet-name> ━━━━ <servlet-mapping> 태그의 <servlet-name> 태그와 값이 동일합니다.  
    <servlet-class>sec01.ex01.FirstServlet</servlet-class> ━━━━ 브라우저에서 요청하는 이름에  
  </servlet>                                매핑 이름으로 요청 시 대해 실제로 기능을 수행하는 서블릿  
  <servlet-mapping>                         클래스를 설정합니다.  
      <servlet-name>aaa</servlet-name>          ━━━━ 값이 같은 <servlet> 태그  
      <url-pattern>/first</url-pattern>          안의 <servlet-name> 태  
  </servlet-mapping>                          그와 연결됩니다.  
                                              ━━━━ 브라우저에서 요청하는 논리적인 서블릿을 설정합니다.  
                                              ━━━━ 브라우저에서 sec01.ex01.FirstServlet을 요청하는 논리적인  
                                              서블릿 이름입니다.
```

5.4.5 톰캣에 프로젝트 실행

이제 새로 만든 프로젝트를 다시 톰캣 서버에 등록한 후 톰캣을 다시 실행하여 웹 브라우저에서 서블릿 매핑 이름인 /first로 요청하겠습니다. 이때 콘솔로 메시지가 출력되는 것을 확인하면 됩니다.

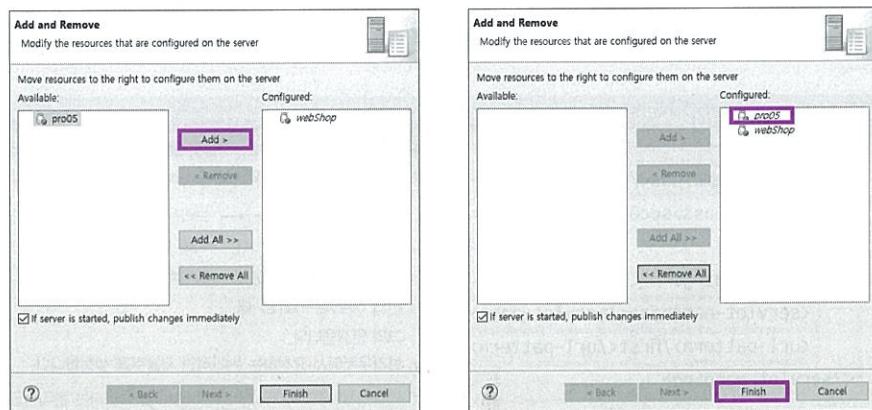
1. 톰캣 서버를 선택하고 마우스 오른쪽 버튼을 클릭한 후 Add and Remove...를 선택합니다.

▼ 그림 5-20 톰캣 서버 선택 후 Add and Remove... 선택



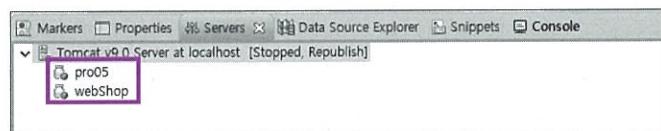
2. pro05 프로젝트를 선택한 후 Add를 클릭하여 추가하고 Finish를 클릭합니다.

▼ 그림 5-21 프로젝트 추가



3. 톰캣에 정상적으로 새 프로젝트가 등록된 것을 확인할 수 있습니다.

▼ 그림 5-22 톰캣에 프로젝트 등록 확인



5.4.6 브라우저에서 서블릿 요청하기

이번에는 실제 웹 브라우저에서 서블릿 매핑 이름으로 서블릿을 요청하는 방법을 알아보겠습니다.

웹 브라우저에서 서블릿을 요청하려면 다음과 같이 웹 브라우저 주소창에 프로젝트 이름까지 입력하고 web.xml에 매핑한 매핑 이름을 슬래시(/) 다음에 입력한 후 요청하면 됩니다.

- `http://IP주소:포트번호/프로젝트이름(컨텍스트이름)/서블릿매핑이름`
- 요청 예: `http://127.0.0.1:8090/pro05/first`

Note 톰캣이 로컬 PC에 설치된 경우에는 다음과 같이 입력해도 됩니다.

`http://localhost:8090/pro05/first`

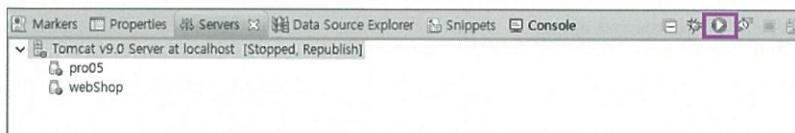


자신의 IP 주소를 확인하려면 명령 프롬프트창에서 `ipconfig` 명령을 입력하면 됩니다.

그럼 실제로 실행해 보겠습니다.

1. 이클립스에서 톰캣을 다시 실행합니다.

▼ 그림 5-23 톰캣 재실행



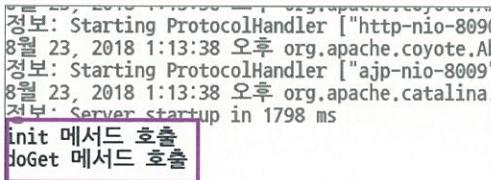
2. 브라우저에서 서블릿 매핑 이름으로 요청합니다.

▼ 그림 5-24 브라우저에 요청



3. /first로 웹 브라우저에서 요청하면 이클립스 콘솔에 각각의 메서드가 호출되면서 메시지가 출력됩니다.

▼ 그림 5-25 브라우저 요청 결과



Note 三 서블릿 매핑을 잘못한 경우 오류 메시지 출력

다음은 web.xml에 서블릿 매핑을 잘못한 상태에서 톰캣을 실행했을 때 나타난 오류 메시지입니다. 오류가 발생한 상태에서 웹 브라우저에 요청하면 정상적으로 실행되지 않습니다.

web.xml에 서블릿 매핑을 할 경우에는 문법이나 태그의 철자가 틀리지 않도록 대소문자까지 주의해서 입력해야 합니다. 톰캣을 실행할 때는 어떤 오류도 발생하면 안 됩니다. 오류가 발생하면 그 원인을 찾아 바로 수정한 후 다시 실행해야 합니다.

▼ 그림 5-26 web.xml에서 < servlet-name>을 < sevlet-name>으로 잘못 입력한 경우

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   <servlet>
4     <sevlet-name>aaa</sevlet-name>
5     <sevlet-class>sec01.ex01.FirstServlet</sevlet-class>
6   </servlet>
7   <servlet-mapping>
8     <sevlet-name>aaa</sevlet-name>
9     <url-pattern>/first</url-pattern>
10   </servlet-mapping>
11 </web-app>
```

▼ 그림 5-27 톰캣 실행 시 출력되는 오류 메시지

```
Markers Properties Servers Data Source Explorer Snippets Problems Console
tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\Java-9.0.4\bin\javaw.exe (2018. 4. 18. 오후 5:13:27)
정보: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a detailed report.
    2018-04-18 5:13:32 오후 org.apache.tomcat.util.digester.Digester endElement
    디테일: End event threw exception
java.lang.IllegalArgumentException: Can't convert argument: null
    at org.apache.tomcat.util.IntrospectionUtils.convert(IntrospectionUtils.java:450)
    at org.apache.tomcat.util.descriptor.web.CallMethodMultiRule.end(WebRuleSet.java:992)
    at org.apache.tomcat.util.digester.Digester.endElement(Digester.java:944)
    at java.xml/com.sun.org.apache.xerces.internal.parsers.AbstractSAXParser.endElement(Unknown Source)
    at java.xml/com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl.scanEndElement(Unknown Source)
    at java.xml/com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl$FragmentContentDispatcher.dispatch(Unknown Source)
    at java.xml/com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl.next(Unknown Source)
    at java.xml/com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl.scanDocument(Unknown Source)
    at java.xml/com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)
```

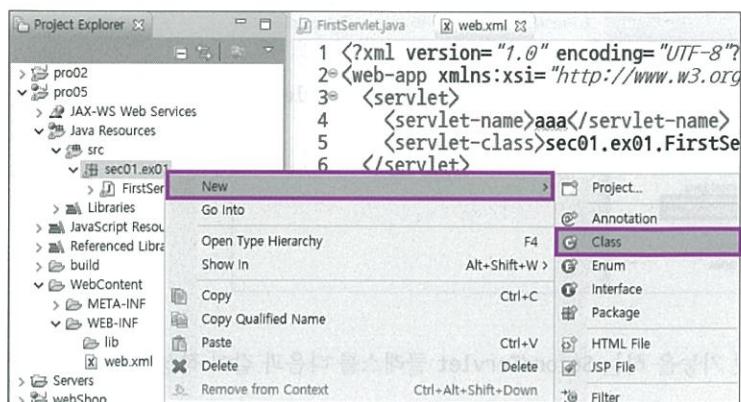
5.4.7 다수의 서블릿 매핑하기

온라인 쇼핑몰 같은 경우 대부분은 상품 조회, 주문, 회원 관리 등의 기능으로 이루어져 있습니다. 만약 이런 기능을 모두 서블릿 하나에 만들어서 제공한다면 소스가 복잡해져 관리하기 불편할 것입니다. 따라서 일반적인 웹 애플리케이션은 각 기능에 대한 서블릿을 따로 만들어서 서비스를 제공합니다. 즉, 프로젝트에서 여러 개의 서블릿을 만들어 사용합니다.

그럼 이번에는 다른 서블릿을 SecondServlet.java로 추가해 보겠습니다.

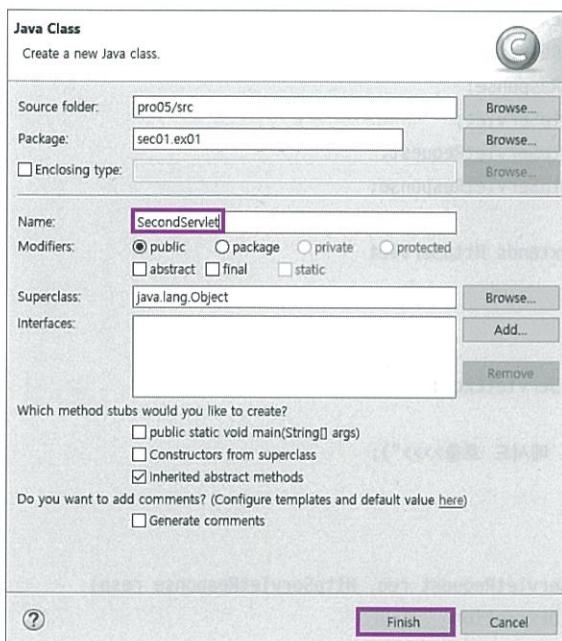
1. 패키지 sec01.ex01을 선택하고 마우스 오른쪽 버튼을 클릭한 후 New > Class를 선택합니다.

▼ 그림 5-28 New > Class 선택



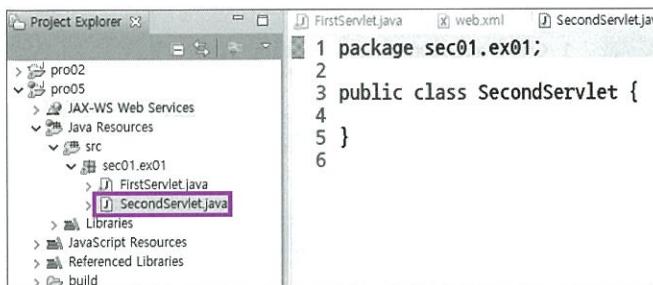
2. 클래스 이름으로 SecondServlet을 입력하고 Finish를 클릭합니다.

▼ 그림 5-29 클래스 이름으로 SecondServlet 입력 후 Finish 클릭



3. Project Explorer에 SecondServlet.java가 생성된 것을 확인할 수 있습니다.

❖ 그림 5-30 SecondServlet.java 생성 확인



4. 또 다른 서블릿 기능을 하는 SecondServlet 클래스를 다음과 같이 작성합니다.

코드 5-4 pro05/src/sec01/ex01/SecondServlet.java

```
package sec01.ex01;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SecondServlet extends HttpServlet
{
    @Override
    public void init() throws ServletException
    {
        System.out.println("init 메서드 호출>>>");
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        System.out.println("doGet 메서드 호출>>>");
    }

    @Override
    public void destroy()
    {
        System.out.println("destroy 메서드 호출>>>");
    }
}
```

```

    }
}

```

5. 다시 SeocndServlet.java를 web.xml에 매핑해 보겠습니다. 이때 주의할 할 것은 여러 개의 서블릿을 매핑할 때는 <servlet> 태그와 <servlet-mapping> 태그를 각각 분리해서 작성해야 한다는 것입니다.

코드 5-5 pro05/WebContent/WEB_INF/web.xml

```

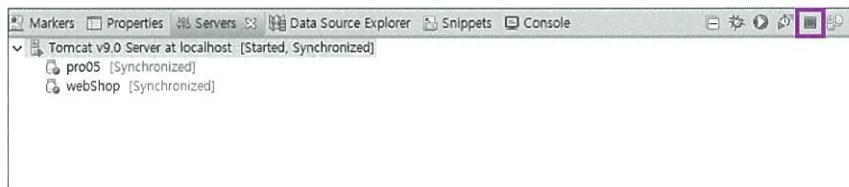
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <servlet>
        <servlet-name>aaa</servlet-name>
        <servlet-class>sec01.ex01.FirstServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>bbb</servlet-name> •
        <servlet-class>sec01.ex01.SecondServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>aaa</servlet-name>
        <url-pattern>/first</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>bbb</servlet-name>
        <url-pattern>/second</url-pattern>
    </servlet-mapping>
</web-app>

```

6. 프로젝트의 web.xml 변경 사항을 반영하려면 톰캣을 재실행해야 합니다. Servers의 빨간색 버튼을 클릭해 톰캣을 종료한 후 다시 녹색 버튼을 클릭해 톰캣을 실행합니다.

▼ 그림 5-31 톰캣 종료 후 재실행



7. 다음은 브라우저에서 /second라는 매핑 이름으로 요청했을 때의 결과입니다. 이번에는 SecondServlet 클래스들의 메서드가 호출되어 메시지를 출력합니다.

▼ 그림 5-32 브라우저 요청 결과 / 이클립스 콘솔창 결과



이처럼 여러 개의 서블릿을 web.xml에 매핑하려면 <servlet> 태그와 <servlet-mapping> 태그를 분리한 후 <servlet-name> 태그의 값을 다른 값으로 설정해야 합니다.

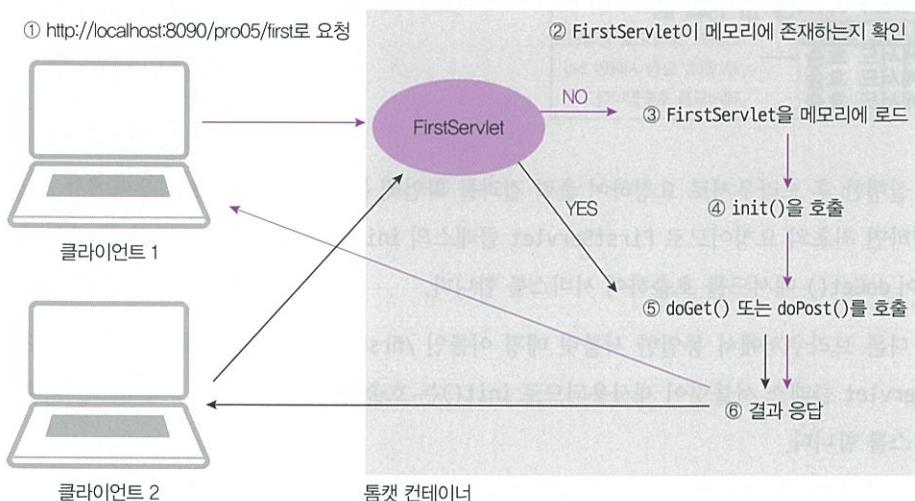
5.5

서블릿 동작 과정

이번에는 앞 절에서 실습한 서블릿의 상세 동작 과정을 살펴보겠습니다.

여러 클라이언트가 서블릿을 요청했을 때의 서블릿 처리 과정을 그림 5-33에 나타내었습니다.

▼ 그림 5-33 서블릿 실행 구조

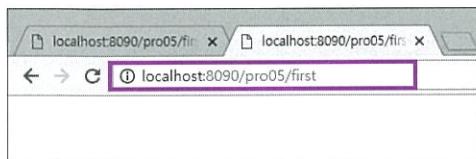


클라이언트 1이 요청하면 톰캣은 FirstServlet이 메모리에 로드되어 있는지 확인합니다. 최초의 요청이므로 init() 메서드를 호출하여 FirstServlet 인스턴스를 메모리에 로드합니다. 그런 다음 doGet()이나 doPost() 메서드를 호출하여 서비스를 합니다.

클라이언트 2가 다시 동일한 서블릿을 요청하면 톰캣은 다시 FirstServlet이 메모리에 로드되어 있는지 확인합니다. 이번에는 메모리에 있는 것이 확인되므로 바로 doGet()이나 doPost() 메서드를 호출하여 서비스를 합니다.

다음은 브라우저의 다른 탭에서 동일한 요청을 했을 때의 결과입니다.

▼ 그림 5-34 브라우저의 다른 탭에서 새로 창을 열고 요청



▼ 그림 5-35 클라이언트에서 재요청 시 아틀리스 콘솔 결과

```
정보: Starting ProtocolHandler ["ajp-nio"]
8월 23, 2018 1:24:41 오후 org.apache.cat
서버: Server startup in 1544 ms
init 메서드 호출      스레드 방식으로 동작하므로
doGet 메서드 호출      최초 요청 시에만 init()
doGet 메서드 호출
doGet 메서드 호출      메서드를 호출합니다.
```

톰캣을 실행한 후 브라우저로 요청하여 출력 결과를 확인해 봅시다. 맨 처음 브라우저에서 /first로 요청하면 최초의 요청이므로 FirstServlet 클래스의 init()를 호출해 초기화한 후 메모리에 로드되어 doGet() 메서드를 호출하여 서비스를 합니다.

그러나 다른 브라우저에서 동일한 서블릿 매핑 이름인 /first로 요청하면 미리 메모리에 로드된 FirstServlet 클래스 서블릿이 재사용되므로 init()는 호출하지 않고 doGet() 메서드만 호출하여 서비스를 합니다.

이제 서블릿이 스레드 방식으로 동작하는 원리를 이해할 수 있나요? 이처럼 동일한 작업의 경우 서블릿은 메모리에 존재하는 서블릿을 재사용함으로써 훨씬 빠르고 효율적으로 동작합니다.

5.6

애너테이션을 이용한 서블릿 매핑

앞 절에서 실습했듯이 여러 서블릿을 web.xml에 설정할 경우 복잡해진다는 단점이 있습니다. 따라서 각 서블릿 클래스에 기호(@)를 이용해서 서블릿 표시를 해주면 훨씬 가독성이 좋아집니다. 이처럼 소스 코드에 직접 기능을 설정하는 방법을 **애너테이션(annotation)**이라고 합니다.

톰캣 7 버전부터는 서블릿 매핑을 web.xml 외에 애너테이션을 이용해 서블릿 클래스에 직접 설정 할 수 있는 기능이 추가되었습니다. 두 가지 방법 다 많이 사용되지만 필자는 애너테이션을 이용하는 방법을 선호합니다. 따라서 이후에 실습하는 모든 서블릿들은 애너테이션을 이용하는 방법으로 매핑하겠습니다.

5.6.1 애너테이션을 이용한 서블릿 매핑

애너테이션을 이용해 서블릿 매핑을 하려면 `@WebServlet`을 이용하면 됩니다. 그리고 애너테이션 이 적용되는 클래스는 반드시 `HttpServlet` 클래스를 상속받아야 합니다.

코드 5-6 @WebServlet 사용 방법

```
(서블릿 클래스 위에 선언)
@WebServlet("/서블릿매핑이름");
</>
```

<코드> 애너테이션을 이용한 서블릿 매핑 예

```
@WebServlet("/third")
public class ThirdServlet extends HttpServlet {
    ...
}
```

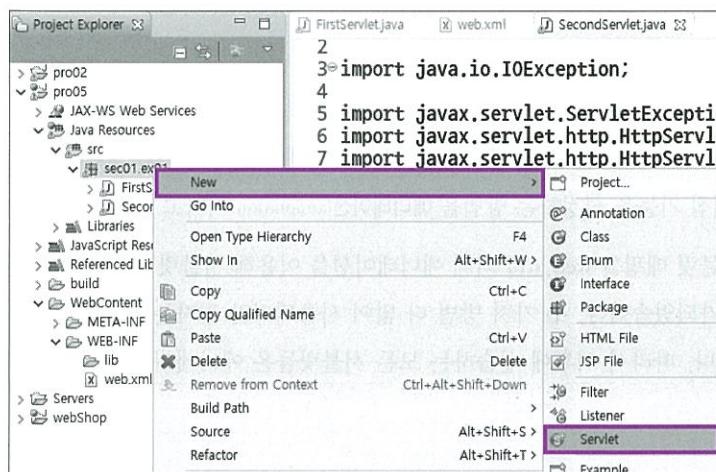
애너테이션을 이용해서 만드는 서
블릿 클래스는 반드시 `HttpServlet`
을 상속받아야 합니다.

5.6.2 애너테이션을 이용한 서블릿 매핑 실습

그럼 지금부터 이클립스에서 애너테이션을 이용하여 직접 서블릿을 생성해 보겠습니다. 앞 절에 서와 마찬가지로 프로그래머가 서블릿 클래스를 직접 만든 후 자바 코드에 애너테이션을 붙여서 매핑할 수도 있지만 이클립스에서는 서블릿을 만들면서 애너테이션을 바로 적용할 수 있습니다.

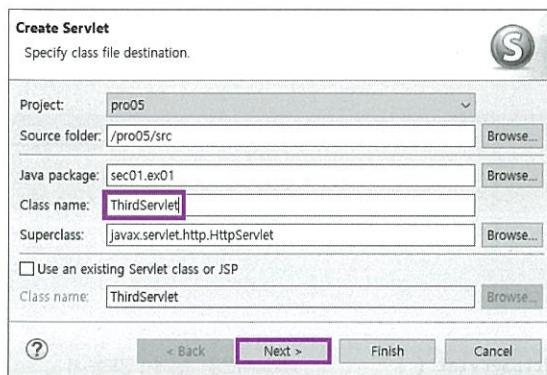
1. sec01.ex01 패키지를 선택하고 마우스 오른쪽 버튼을 클릭한 후 New > Servlet을 선택합니다.

▼ 그림 5-36 New > Servlet 선택



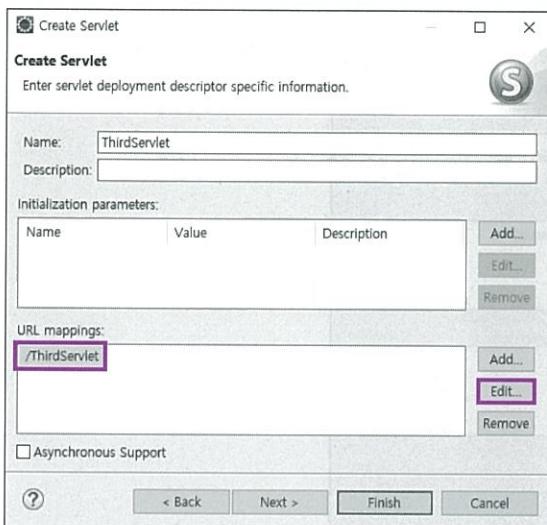
2. 클래스 이름으로 ThirdServlet을 입력하고 Next를 클릭합니다.

▼ 그림 5-37 클래스 이름으로 ThirdServlet 입력 후 Next 클릭



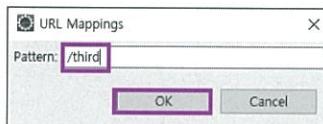
3. 우선 기본 URL mapping 이름을 선택한 후 매핑 이름을 수정하기 위해 Edit...를 클릭합니다.

▼ 그림 5-38 매핑 이름을 수정하기 위해 Edit... 클릭



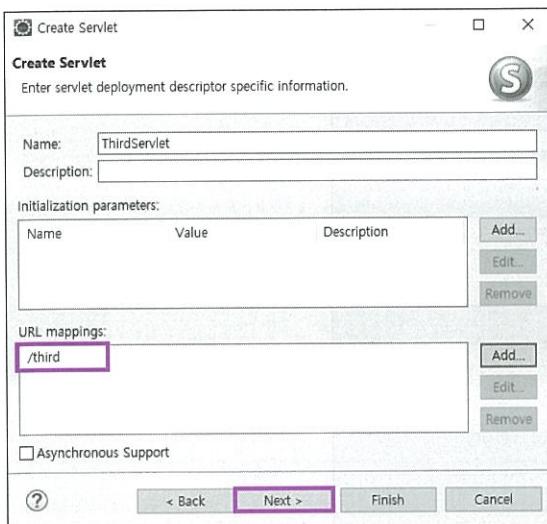
4. 서블릿 매핑 이름을 /third로 수정하고 OK를 클릭합니다.

▼ 그림 5-39 매핑 이름을 /third로 수정



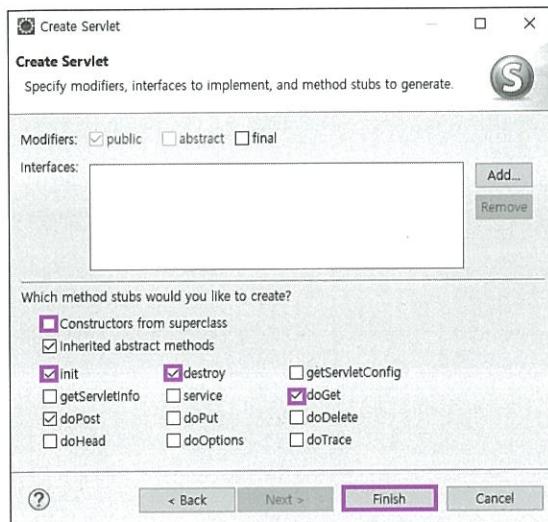
5. 매핑 이름이 수정된 것을 확인한 후 Next를 클릭합니다.

▼ 그림 5-40 매핑 이름 설정 확인 후 Next 클릭



6. Constructors from superclass 옵션 체크박스의 체크를 해제한 후 오버라이딩할 생명주기 메서드를 추가합니다. 기본값으로 설정된 상태에서 init과 destroy, doGet, doPost에 체크하고 Finish를 클릭합니다.

▼ 그림 5-41 생명주기 메서드 오버라이딩



7. 애너테이션에 수정한 맵핑 이름이 추가된 것을 확인할 수 있습니다.

◀ 그림 5-42 추가된 매팅 이름 /third 확인

```
1 package sec01.ex01;
2
3 import java.io.IOException;
4
5 /**
6  * Servlet implementation class ThirdServlet
7  */
8
9 @WebServlet("/third")
10 public class ThirdServlet extends HttpServlet {
11     private static final long serialVersionUID = 1L;
12
13     /**
14      * @see Servlet#init(ServletConfig)
15      */
16     public void init(ServletConfig config) throws ServletException {
17         // TODO Auto-generated method stub
18     }
19
20     /**
21      * @see Servlet#destroy()
22      */
23     public void destroy() {
24         // TODO Auto-generated method stub
25     }
26 }
```

8. 이클립스에서 생성한 `ThirdServlet` 클래스를 봅시다. `serialVersionUID`는 서블릿 클래스의 직렬화를 위해 이클립스에서 자동으로 생성된 상수인데 삭제해도 됩니다. 그리고 메서드 안에 자동으로 생성된 주석이나 기능은 삭제한 후 메시지 출력 기능을 추가합니다.

코드 5-7 pro05/src/sec01/ex01/ThirdServlet.java

```
package sec04.ex01;  
...  
  
/**  
 * Servlet implementation class ThirdServlet  
 */  
@WebServlet("/third")  
public class ThirdServlet extends HttpServlet {  
    private static final long serialVersionUID = 1L; •———— 서블릿 클래스 직렬화를 위해 이클리п스에서 자동으로 지정한 상수입니다.  
    /**  
     * @see Servlet#init(ServletConfig)  
     */  
    public void init(ServletConfig config) throws ServletException {  
        System.out.println("ThirdServlet init 메서드 호출");  
    }  
  
    /**  
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse  
     *      response)  
     */
```

```

protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    System.out.println("ThirdServlet doGet 메서드 호출");
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
 *      response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    System.out.println("ThirdServlet destroy 메서드 호출");
}
}

```

9. 이클립스에서 애너테이션을 이용하여 서블릿 매핑을 설정했으니 톰캣을 중지했다가 재실행한 다음 웹 브라우저에서 서블릿 매핑 이름으로 요청해 보겠습니다.

- http://localhost:8090/pro05/third

▼ 그림 5-43 웹 브라우저 서블릿 요청 결과

```

8월 23, 2018 1:31:36 오후 org.apache.cat
정보: Server startup in 1696 ms
ThirdServlet init 메소드 호출
ThirdServlet doGet 메소드 호출

```

지금까지 이클립스에서 서블릿을 생성하고 애너테이션을 어떻게 사용하는지 알아보았습니다. 애너테이션을 사용할 때는 매핑 이름이 이미 사용된 다른 매핑 이름과 충복되지 않도록 주의해야 합니다.

다음은 ThirdServlet.java의 /third라는 매핑 이름을 이미 사용 중인 /first로 직접 수정한 후 톰캣을 재실행했을 때 오류 메시지가 출력되는 경우입니다.

▼ 그림 5-44 애너테이션으로 설정한 매핑 이름이 web.xml의 매핑 이름과 충복되는 경우

```

3@ import java.io.IOException;
10
11 /**
12  * Servlet implementation class ThirdServlet
13 */
14 @WebServlet("/first")
15 public class ThirdServlet extends HttpServlet {
16     private static final long serialVersionUID = 1L;
17
18     /**
19      * @see HttpServlet#init(ServletConfig)
20     */

```

▼ 그림 5-45 중복된 매핑 이름을 설정한 경우 발생하는 오류

```
<terminated> Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\Jre-9.0.4\bin\javaw.exe (2018-4-18 오전 7:37:26)
严重: At least one JAR was scanned for TLDs yet contained NO TLDs. Enable debug logging for this logger for a complete list or
4월 18, 2018 7:37:32 오후 org.apache.catalina.core.ContainerBase startInternal
심각: A child container failed during start
java.util.concurrent.ExecutionException: org.apache.catalina.LifecycleException: Failed to start component [StandardEngine[Catalin
at java.base/java.util.concurrent.FutureTask.report(Unknown Source)
at java.base/java.util.concurrent.FutureTask.get(Unknown Source)
at org.apache.catalina.core.ContainerBase.startInternal(ContainerBase.java:949)
at org.apache.catalina.core.StandardHost.startInternal(StandardHost.java:839)
at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:183)
at org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1427)
at org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1417)
```

이후부터 실제 서블릿 클래스는 직접 클래스를 만들어서 생성하고 서블릿 매핑은 애너테이션을 이용해서 실습하겠습니다. 앞에서 web.xml로 실습한 서블릿도 애너테이션을 적용해서 실습해 보세요.



Tip ☆ 이 책을 실습하다 보면 login, first, member 등 자주 사용하는 매핑 이름들이 있습니다. 이 책과 함께 제공하는 예제 소스 파일을 보면 중복된 매핑 이름의 경우 주석 처리를 해 두었습니다. 따라서 실습 파일을 이용할 때는 매핑 이름을 반드시 한 번 더 확인하고 실습하기 바랍니다. 필요에 따라 매핑 이름을 주석 처리하거나 주석 처리를 해제하면서 사용하면 됩니다.

6 장

서블릿 기초

6.1 서블릿의 세 가지 기본 기능

6.2 <form> 태그 이용해 서블릿에 요청하기

6.3 서블릿에서 클라이언트의 요청을 얻는 방법

6.4 서블릿의 응답 처리 방법

6.5 웹 브라우저에서 서블릿으로 데이터 전송하기

6.6 GET 방식과 POST 방식 요청 동시에 처리하기

6.7 자바스크립트로 서블릿에 요청하기

6.8 서블릿을 이용한 여러 가지 실습 예제

6.1

서블릿의 세 가지 기본 기능

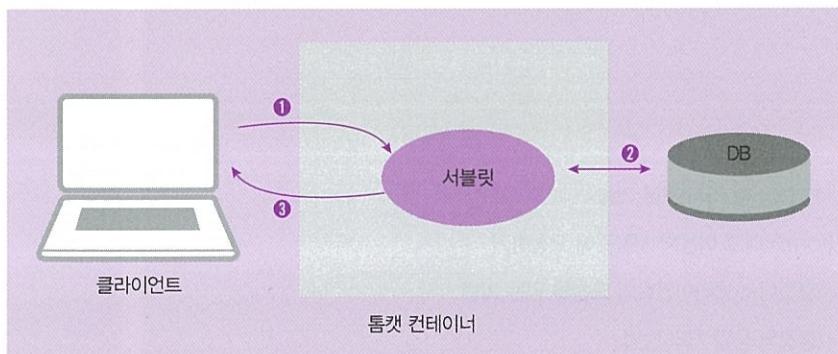
7

톰캣과 같은 WAS(Web Application Server, 웹 애플리케이션 서버)가 처음 나왔을 때 웹 브라우저 요청을 스레드 방식으로 처리하는 기술이 바로 서블릿이었습니다. 6~7장에 걸쳐 살펴볼 서블릿의 기능은 이 책에서 가장 기본이 되면서도 중요한 내용입니다. 모든 웹 프로그램은 6~7장에서 배우는 기능을 뼈대로 하여 동작합니다. 따라서 이 내용을 잘 이해하고 나면 전체 웹 프로그램이 어떤 식으로 동작하는지 쉽게 이해할 수 있습니다. 그 외의 기능들은 주요 서블릿 기능이 효율적으로 동작하기 위해 제공하는 세부 기능이라고 할 수 있습니다.

6.1.1 서블릿 기본 기능 수행 과정

그림 6-1에 서블릿이 수행하는 세 가지 주요 기능을 나타내었습니다. 요약하면 클라이언트로부터 요청을 받아 비즈니스 로직을 처리하고, 그 결과를 다시 클라이언트에 돌려주는 과정입니다.

▼ 그림 6-1 서블릿의 세 가지 주요 기능



- ① 클라이언트로부터 요청을 받습니다.
- ② 데이터베이스 연동과 같은 비즈니스 로직을 처리합니다.
- ③ 처리된 결과를 클라이언트에 돌려줍니다.

초기 웹 프로그램 개발에서는 서블릿이 클라이언트로부터 요청을 받아 데이터베이스 연동 같은 비즈니스 작업을 처리한 후 그 결과를 클라이언트의 브라우저로 전송하는 방식으로 작업했습니다.

클라이언트로부터 요청을 받는 작업에는 어떤 것들이 있을까요? 우리가 자주 사용하는 포털 사이트에서 로그인하려고 ID와 비밀번호를 텍스트 창에 입력한 후 로그인 버튼을 클릭하면 사용자가 입력한 ID와 비밀번호가 서버 쪽의 서블릿에 전송됩니다. 서블릿에서는 여러 가지 메서드(다음 절에서 배웁니다)를 이용해 사용자가 전송한 ID와 비밀번호를 받아옵니다.

그 다음 사용자의 로그인 요청에 대해 데이터베이스와 연동하여 사용자가 이미 등록된 회원인지 조회합니다. 그 결과에 따라 다음 페이지로 가던지 ID나 비밀번호가 틀렸으니 다시 로그인하라는 오류 메시지를 클라이언트에 전송합니다.

6.1.2 서블릿 응답과 요청 수행 API 기능

우선 요청이나 응답과 관련된 서블릿 기능을 알아봅시다. 요청이나 응답과 관련된 API는 모두 `javax.servlet.http` 패키지에 있습니다.

- 요청과 관련된 API: `javax.servlet.http.HttpServletRequest` 클래스
- 응답과 관련된 API: `javax.servlet.http.HttpServletResponse` 클래스

그림 6-2는 요청이나 응답과 관련된 API가 서블릿의 `doGet()`이나 `doPost()` 메서드의 매개변수로 사용되는 예를 나타낸 것입니다.

▼ 그림 6-2 요청과 응답 관련 API 사용 예

```
public class FirstServlet extends HttpServlet {
    @Override
    public void init() throws ServletException {
        System.out.println("init 메서드 호출");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        System.out.println("doGet 메서드 호출");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        System.out.println("doPost 메서드 호출");
    }

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

다시 그림 6-1을 보겠습니다. 클라이언트가 서블릿에 요청을 하면 먼저 톰캣 컨테이너가 받습니다. 그런 다음 사용자의 요청이나 응답에 대한 `HttpServletRequest` 객체와 `HttpServletResponse` 객체를 만들고 서블릿의 `doGet()`이나 `doPost()` 메서드를 호출하면서 이 객체들을 전달합니다.

톰캣이 사용자의 요청에 대한 정보를 모든 `HttpServletRequest` 객체의 속성으로 담아 메서드로 전달하므로 각 `HttpServletRequest`에서 제공하는 메서드들은 매개변수로 넘어온 객체들을 이용하여 사용자가 전송한 데이터를 받아 오거나 응답할 수 있는 것입니다.

표 6-1과 표 6-2에는 각 API에서 제공하는 중요한 메서드들을 정리해 두었습니다. 이 메서드들을 이용해서 여러 가지 요청이나 응답과 관련된 작업을 합니다. 이에 대해서는 다음 절에서 자세히 알아볼 것이므로 여기에서는 이런 메서드들이 있구나 하는 정도로 읽어보고 넘어가도 됩니다.

▼ 표 6-1 `HttpServletRequest`의 여러 가지 메서드

반환형	메서드 이름	기능
boolean	<code>authenticate(HttpServletRequest response)</code>	현재 요청한 사용자가 ServletContext 객체에 대한 인증을 하기 위한 컨테이너 로그인 메커니즘을 사용합니다.
String	<code>changeSessionId()</code>	현재 요청과 연관된 현재 세션의 id를 변경하여 새 세션 id를 반환합니다.
String	<code>getContextPath()</code>	요청한 컨텍스트를 가리키는 URI를 반환합니다.
Cookie[]	<code>getCookies()</code>	클라이언트가 현재의 요청과 함께 보낸 쿠키 객체들에 대한 배열을 반환합니다.
String	<code>.getHeader(String name)</code>	특정 요청에 대한 헤더 정보를 문자열로 반환합니다.
Enumeration<String>	<code>getHeaderNames()</code>	현재의 요청에 포함된 헤더의 name 속성을 enumeration으로 반환합니다.
String	<code>getMethod()</code>	현재 요청이 GET, POST 또는 PUT 방식 중 어떤 HTTP 요청인지 를 반환합니다.
String	<code>getRequestURI()</code>	요청한 URL의 컨텍스트 이름과 파일 경로까지 반환합니다.
String	<code>getServletPath()</code>	요청한 URL에서 서블릿이나 JSP 이름을 반환합니다.
<code>HttpSession</code>	<code>getSession()</code>	현재의 요청과 연관된 세션을 반환합니다. 만약 세션이 없으면 새로 만들어서 반환합니다.

▼ 표 6-2 HttpServletResponse의 여러 가지 메서드

반환형	메서드 이름	기능
void	addCookie (Cookie cookie)	응답에 쿠키를 추가합니다.
void	addHeader(String name, String value)	name과 value를 헤더에 추가합니다.
String	encodeURL(String url)	클라이언트가 쿠키를 지원하지 않을 때 세션 id를 포함한 특정 URL을 인코딩합니다.
Collection <String>	getHeaderNames()	현재 응답의 헤더에 포함된 name을 얻어옵니다.
void	sendRedirect (String location)	클라이언트에게 리다이렉트(redirect) 응답을 보낸 후 특정 URL로 다시 요청하게 합니다.
String	getPathInfo()	클라이언트가 요청 시 보낸 URL과 관련된 추가 경로 정보를 반환합니다.

6.2

〈form〉 태그 이용해 서블릿에 요청하기

JAVA WEB

이번에는 〈form〉 태그를 이용해 브라우저에서 서블릿으로 사용자의 요청이나 데이터를 전송하는 방법과 서블릿이 데이터를 받아 오는 방법에 대해 알아보겠습니다.

6.2.1 〈form〉 태그로 서블릿에 요청하는 과정

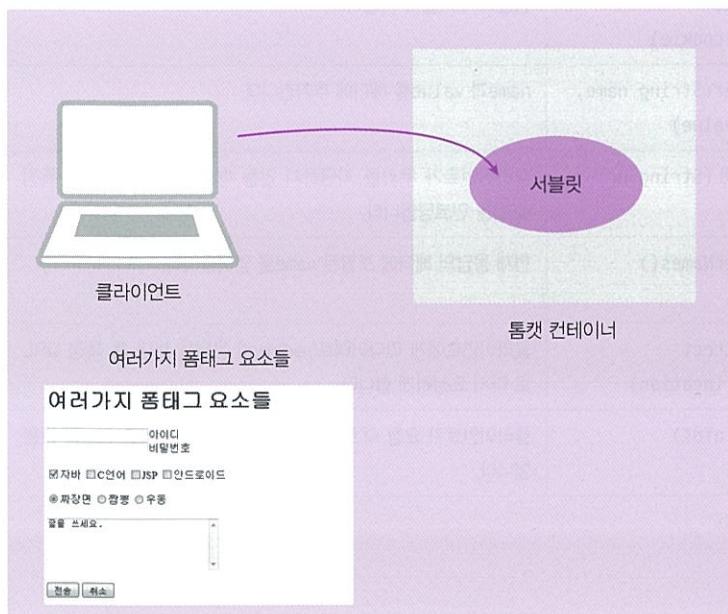
JSP, ASP, PHP가 나오기 전에는 HTML, CSS, 자바스크립트를 이용해 웹 프로그램을 만들었습니다. 서블릿과 JSP는 이러한 HTML, CSS, 자바스크립트 같은 기존의 것을 버리는 것이 아니라 여기에 자신의 기능을 추가하여, 즉 서로 연동하여 동작합니다. 특히 사용자의 요청은 HTML의 〈form〉 태그나 자바스크립트로부터 전송 받아서 처리합니다.



Tip ☆ 서블릿/JSP 프로그래밍을 하려면 기본적으로 HTML이나 자바스크립트에 대해 알아두는 것이 좋습니다. 특히 클라이언트에서 서버로 데이터를 전송하는 기능을 담당하는 〈form〉 태그와 〈input〉 태그의 기능은 자주 사용되므로 반드시 익혀 두기 바랍니다.

클라이언트 웹 브라우저에서 서블릿에 요청하는 방법은 그림 6-3과 같습니다.

▼ 그림 6-3 클라이언트가 서블릿에 요청하는 방법



웹 브라우저에서 여러 가지 입력 서식을 이용해 전송을 클릭하면 사용자가 입력한 데이터가 그림 6-3처럼 서블릿으로 전송됩니다. 그러면 서블릿은 여러 가지 메서드를 이용해서 전송된 데이터를 받아 옵니다(표 6-1 참조).

6.2.2 <form> 태그의 여러 가지 속성

예를 들어 다음과 같이 사용자의 ID와 비밀번호를 입력하는 로그인창이 있다고 합시다.

▼ 그림 6-4 ID와 비밀번호 입력 로그인창

A screenshot of a login form. It contains two text input fields: the first is labeled '아이디:' and contains the text 'hong'; the second is labeled '비밀 번호:' and contains four dots ('....'). Below these fields are two buttons: '로그인' (Login) on the left and '다시 입력' (Re-enter) on the right.

사용자 로그인창의 HTML 태그 구조는 다음과 같습니다.

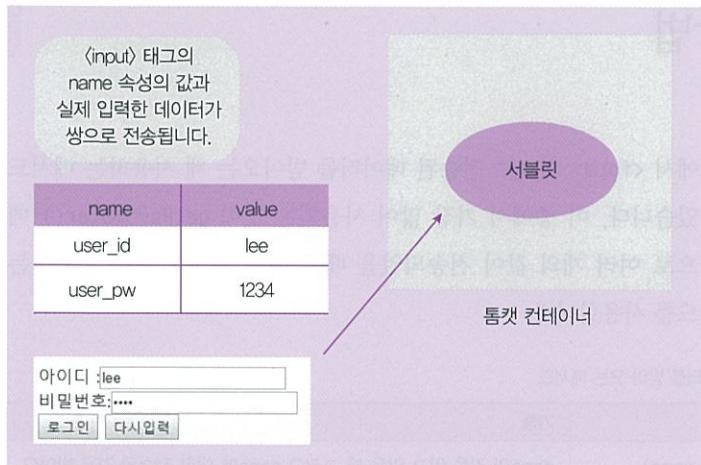
▼ 그림 6-5 로그인창의 HTML 코드

```
<form name="frmLogin" method="get" 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="다시입력" >
</form>
```

사용자가 자신의 ID와 비밀번호를 입력한 후 **로그인**을 클릭하면 **<form>** 태그의 **action** 속성은 데이터를 전송할 서블릿이나 JSP의 이름을 지정합니다. 그러면 지정된 이름이 **login**인 서블릿으로 ID와 비밀번호가 전송됩니다.

다음은 로그인창에서 **로그인**을 클릭했을 때 실제로 데이터가 전송되는 과정입니다.

▼ 그림 6-6 **<form>** 태그 데이터가 전송되는 과정



실제 데이터는 각 **<input>** 태그의 **name** 속성 값과 쌍으로 전송됩니다. 그럼 서블릿에서는 **name** 속성 값으로 같이 전송된 입력 데이터를 받아 옵니다.

그 외 **<form>** 태그의 여러 가지 속성과 기능들은 표 6-3을 참고하세요.

▼ 표 6-3 **<form>** 태그와 관련된 여러 가지 속성

속성	기능
name	<ul style="list-style-type: none"> <form> 태그의 이름을 지정합니다. 여러 개의 form이 존재할 경우 구분하는 역할을 합니다. 자바스크립트에서 <form> 태그에 접근할 때 자주 사용합니다.
method	<ul style="list-style-type: none"> <form> 태그 안에서 데이터를 전송할 때 전송 방법을 지정합니다. GET 또는 POST로 지정합니다(아무것도 지정하지 않으면 GET입니다).

속성	기능
action	<ul style="list-style-type: none"> <form> 태그에서 데이터를 전송할 서블릿이나 JSP를 지정합니다. 서블릿으로 전송할 때는 매팅 이름을 사용합니다.
encType	<ul style="list-style-type: none"> <form> 태그에서 전송할 데이터의 encoding 타입을 지정합니다. 파일을 업로드할 때는 multipart/form-data로 지정합니다.

6.3

서블릿에서 클라이언트의 요청을 얻는 방법

HttpServletRequest 클래스에서 <form> 태그로 전송된 데이터를 받아오는 데 사용하는 메서드로는 표 6-4와 같은 것들이 있습니다. 이 중에서 가장 많이 사용되는 것이 getParameter() 메서드입니다. 만약 같은 name으로 여러 개의 값이 전송되었을 때는 배열 형태로 값을 반환하는 getParameterValues() 메서드를 사용합니다.

▼ 표 6-4 <form> 태그로 전송된 데이터를 받아 오는 메서드

메서드	기능
String getParameter(String name)	name의 값을 알고 있을 때 그리고 name에 대한 전송된 값을 받아오는 데 사용합니다.
String[] getParameterValues (String name)	같은 name에 대해 여러 개의 값을 얻을 때 사용합니다.
Enumeration getParameterNames()	name 값을 모를 때 사용합니다.

6.3.1 HttpServletRequest로 요청 처리 실습

이번에는 실제 이를립스에서 <form> 태그로 전송된 정보를 서블릿에서 받아 와서 출력하는 과정을 실습해 보겠습니다. 로그인창에서 ID와 비밀번호를 입력 받아 HttpServletRequest로 처리하는 간단한 프로그램입니다.

1. pro06이라는 새 프로젝트를 생성합니다. 그리고 톰캣의 servlet-api.jar를 클래스 패스에 지정합니다(5장 참고).

▼ 그림 6-7 pro06 프로젝트 생성



2. WebContent 폴더 하위에 다음과 같이 사용자 정보를 입력 받을 login.html을 생성합니다.

▼ 그림 6-8 실습 파일 위치



3. 다음과 같이 login.html 파일을 작성합니다. 로그인창에서 ID와 비밀번호를 입력 받은 후 서블릿으로 전송하는 내용입니다.

코드 6-1 pro06/WebContent/login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>로그인 창</title>
</head>
<body>
<form name="frmLogin" method="get" 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="다시입력">
</form>
</body>
</html>
```

텍스트 박스에 입력된 ID를 user_id로 전송합니다.

입력된 데이터를 서블릿 매핑 이름이 login인 서블릿으로 전송합니다.

텍스트 박스에 입력된 비밀번호를 user_pw로 전송합니다.

4. sec01.ex01 패키지를 만들고 요청을 받을 서블릿인 LoginServlet 클래스를 생성합니다(5장 애너테이션을 이용한 서블릿 생성 과정을 참고하세요).

▼ 그림 6-9 실습 파일 위치



5. 다음과 같이 LoginServlet.java 코드를 작성합니다. HttpServletRequest 클래스의 getParameter() 메서드로 전송된 ID와 비밀번호를 받아옵니다.

코드 6-2 pro06/src/sec01/ex01/LoginServlet.java

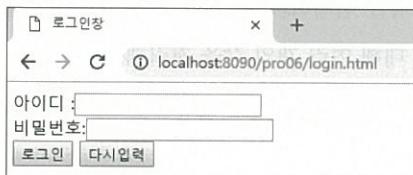
```
package sec01.ex01;  
...  
@WebServlet("/login") ..... 서블릿의 매핑 이름이 login입니다.  
public class LoginServlet extends HttpServlet  
{  
    public void init() throws ServletException  
    {  
        System.out.println("init 메서드 호출");  
    }  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
throws ServletException, IOException  
{  
    request.setCharacterEncoding("utf-8"); ..... 전송된 데이터를 UTF-8로 인코딩합니다.  
    String user_id = request.getParameter("user_id"); ..... getParameter()를 이용해 <input>  
    String user_pw = request.getParameter("user_pw"); ..... 태그의 name 속성 값으로 전송된  
    System.out.println("아이디:" + user_id); ..... value를 받아옵니다.  
    System.out.println("비밀번호:" + user_pw);  
}  
  
    public void destroy()  
    {  
        System.out.println("destroy 메서드 호출");  
    }  
}
```

웹 브라우저에서 전송한 정보를 톰캣 컨테이너가 HttpServletRequest 객체를 생성한 후 doGet()으로 넘겨줍니다.

전송된 데이터를 UTF-8로 인코딩합니다.
getParameter()를 이용해 <input> 태그의 name 속성 값으로 전송된 value를 받아옵니다.

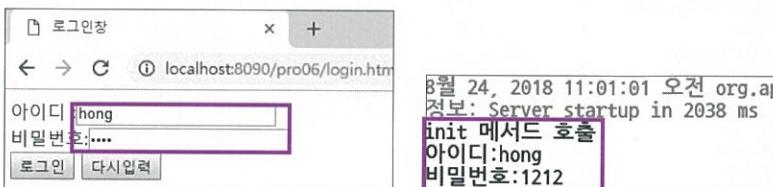
6. pro06 프로젝트를 톰캣에 등록하여 실행한 후 브라우저에서 <http://localhost:8090/pro06/login.html>을 요청합니다.

▼ 그림 6-10 브라우저에서 로그인 화면 요청



7. 텍스트 박스에 ID와 비밀번호를 입력한 후 **로그인**을 클릭하면 서블릿이 ID와 비밀번호를 이 클립스 콘솔에 출력합니다.

▼ 그림 6-11 ID와 비밀번호 입력 후 서블릿으로 전송하면 콘솔에 출력



단, 서블릿이 처리한 후의 응답 기능은 아직 구현하지 않았으므로 웹 브라우저에는 아무것도 출력되지 않습니다.

▼ 그림 6-12 서블릿으로부터 응답이 없는 브라우저



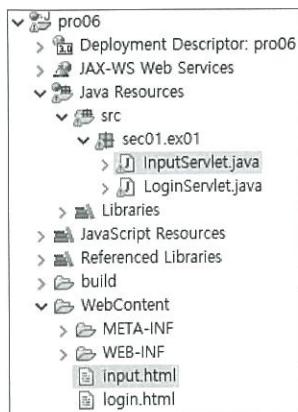
간단한 예제이지만 서블릿을 사용해 로그인을 요청하는 처리를 구현해 보았습니다.

6.3.2 여러 개의 값을 전송할 때의 요청 처리

이번에는 이 프로그램을 조금 더 발전시켜 보겠습니다. 하나의 name으로 여러 값을 서블릿으로 요청하는 방법입니다. 예를 들어 로그인 후 수강할 과목을 입력하되 한 번에 여러 과목을 입력해서 등록하는 예제입니다. 그럼 서블릿에서는 각 과목에 대해 여러 개의 값을 처리해야겠죠?

1. 다음과 같이 input.html을 추가하고 InputServlet 클래스를 새로 만듭니다.

▼ 그림 6-13 실습 파일 위치



2. input.html을 다음과 같이 작성합니다. <input> 태입이 여러 개일 때는 체크박스(Checkbox)를 사용해서 값을 설정하는 것이 좋습니다. 체크박스의 name 속성 값은 모두 subject이므로 서블릿으로 전송할 때 배열로 전송됩니다.

코드 6-3 pro06/WebContent/input.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>여러 가지 input 태입 표시 창</title>
</head>
<body>
<form name="frmInput" method="get" action="input">
    아이디:<input type="text" name="user_id"><br>
    비밀번호:<input type="password" name="user_pw"><br>
    <input type="checkbox" name="subject" value="java" checked>자바
    <input type="checkbox" name="subject" value="C언어">C언어
    <input type="checkbox" name="subject" value="JSP">JSP
    <input type="checkbox" name="subject" value="안드로이드">안드로이드
    <br><br>

```

name 속성이 모두 subject로 같습니다.

```

<input type="submit" value="전송">
<input type="reset" value="초기화">
</form>
</body>
</html>

```

전송을 클릭하면 매핑 이름이 action에 설정한 input 서블릿으로 전송됩니다.

3. InputServlet 클래스를 다음과 같이 작성합니다. getParameterValues()를 이용해 input.html에서 체크박스의 name인 subject로 전송된 값들을 받아 와서 문자열 배열에 저장합니다.

코드 6-4 pro06/src/sec01/ex01/InputServlet.java

```

package sec01.ex01;
...
@WebServlet("/input")
public class InputServlet extends HttpServlet
{
    public void init() throws ServletException
    {
        System.out.println("init 메서드 호출");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        request.setCharacterEncoding("utf-8");
        String user_id = request.getParameter("user_id");
        String user_pw = request.getParameter("user_pw");
        System.out.println("아이디:" + user_id);
        System.out.println("비밀번호:" + user_pw);
        String[] subject = request.getParameterValues("subject");
        for (String str : subject)
        {
            System.out.println("선택한 과목:" + str);
        }
    }

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

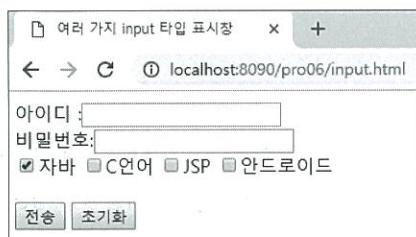
```

한 개씩 전송된 값은 getParameter()를 이용합니다.

하나의 name으로 여러 값을 전송하는 경우 getParameterValues()를 이용해 배열 형태로 반환됩니다.

4. 브라우저에서 <http://localhost:8090/pro06/input.html>로 요청합니다.

▼ 그림 6-14 웹 브라우저에서 input.html로 요청



여러 가지 input 타입 표시창 × +

← → ⌂ localhost:8090/pro06/input.html

아이디 :

비밀번호 :

자바 C언어 JSP 안드로이드

전송 초기화

5. 체크박스에서 여러 개의 값에 체크한 후 전송을 클릭하면 이클립스 콘솔에 해당 과목명이 출력됩니다.

▼ 그림 6-15 값 입력 후 서블릿으로 전송하면 체크한 과목명이 출력



여러 가지 input 타입 표시창 × +

← → ⌂ localhost:8090/pro06/input.html

아이디 : hong

비밀번호 :

자바 C언어 JSP 안드로이드

전송 초기화

init 메서드 호출
아이디:hong
비밀번호:1212
선택한 과목:java
선택한 과목:c언어
선택한 과목:JSP

6.3.3 getParameterNames() 메서드를 이용한 요청 처리

앞에서 사용자로부터 입력을 받아 로직을 처리하는 간단한 수강 신청 프로그램을 만들었습니다. 하지만 우리가 보통 온라인 쇼핑몰에 회원으로 가입하려면 입력해야 할 회원 정보는 이름, 주소, 전화번호, 결제정보 등 최소 10개 이상, 많으면 20개 정도 됩니다. 그럼 이 정보를 서블릿에서 getParameter() 메서드를 이용해서 처리하려면 각 매개변수를 모두 알아야겠죠?

이처럼 전송된 데이터가 많아 일일이 name의 값을 기억하기 힘들 때는 getParameterNames() 메서드를 이용하면 편리합니다. 어떻게 사용하는지 지금부터 실습을 통해 알아봅시다.

1. sec01.ex01 패키지에 InputServlet2 클래스를 생성합니다.

▼ 그림 6-16 실습 파일 위치



2. 그리고 6.3.2절에서 이용했던 input.html을 다음과 같이 수정합니다.

코드 6-5 pro06/WebContent/input.html

```
<!DOCTYPE html>
<html>
<head>...</head>
<body>
<form name="frmInput" method="get" action="input2">———— 매핑 이름을 input2로 수정합니다.  
(중략)
```

3. InputServlet2 클래스를 다음과 같이 작성합니다. 전송되는 데이터가 많은 경우에는 getParameterNames()를 이용해 name 속성만 따로 구할 수 있습니다.

코드 6-6 pro06/src/sec01/ex01/InputServlet2.java

```
package sec01.ex01;
...
@WebServlet("/input2")
public class InputServlet2 extends HttpServlet
{
    public void init() throws ServletException
    {
        System.out.println("init 메서드 호출");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
```

```

{
    request.setCharacterEncoding("utf-8");
    Enumeration enu = request.getParameterNames(); ← 전송되어 온 name 속성
    while (enu.hasMoreElements())
        {
            String name = (String) enu.nextElement();
            String[] values = request.getParameterValues(name);
            for (String value : values)
            {
                System.out.println("name=" + name + ",value=" + value);
            }
        }
}

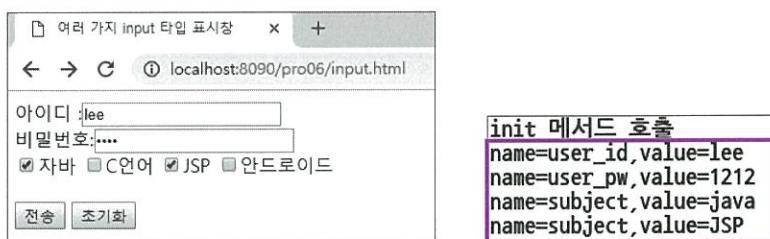
public void destroy() {
    System.out.println("destroy 메서드 호출");
}

```

각 name을 하나씩 가져와 대응해서 전송되어 온 값을 출력합니다.

- 브라우저에서 `http://localhost:8090/pro06/input.html`로 요청하고 값을 입력한 후 전송을 클릭합니다. `getParameterNames()`를 이용해 전송된 name과 값이 모두 출력되는 것을 확인할 수 있습니다.

▼ 그림 6-17 값 입력 후 서블릿으로 전송된 결과



6.4

서블릿의 응답 처리 방법

이번에는 서블릿이 처리한 결과를 클라이언트에게 응답하는 기능을 알아보겠습니다. 서블릿에서 응답을 처리하는 방법은 다음과 같습니다.

- ❶ `doGet()`이나 `doPost()` 메서드 안에서 처리합니다.
- ❷ `javax.servlet.http.HttpServletResponse` 객체를 이용합니다.
- ❸ `setContentType()`을 이용해 클라이언트에게 전송할 데이터 종류(MIME-TYPE)를 지정합니다.
- ❹ 클라이언트(웹 브라우저)와 서블릿의 통신은 자바 I/O의 스트림을 이용합니다.

서블릿의 응답 처리는 `doGet()`이나 `doPost()` 메서드의 두 번째 매개변수인 `HttpServletResponse` 객체를 이용하여 처리합니다. 그리고 웹 브라우저와 서블릿의 응답 과정은 자바 I/O의 기능인 스트림을 이용하여 이루어집니다.

6.4.1 MIME-TYPE

우리가 배우는 웹 애플리케이션은 클라이언트에 해당하는 웹 브라우저와 서버에 해당하는 서블릿이 서로 데이터를 주고받으면서 실행합니다. 웹 브라우저가 네트워크를 통해 서블릿에 데이터를 보내는 경우 서블릿은 네트워크로부터 데이터를 입력 받습니다. 반대로 서블릿이 웹 브라우저로 데이터를 전송하는 경우에는 네트워크로 데이터를 출력합니다. 즉, 네트워크에 대해 자바 I/O 스트림 클래스의 입출력 기능을 이용하면 쉽게 웹 애플리케이션의 네트워크 기능을 구현할 수 있습니다(자바 입문서의 I/O 기능을 참고하세요).

서버(서블릿)에서 웹 브라우저로 데이터를 전송할 때는 어떤 종류의 데이터를 전송하는지 웹 브라우저에 알려줘야 합니다. 그 이유는 웹 브라우저가 전송 받을 데이터의 종류를 미리 알고 있으면 더 빠르게 처리할 수 있기 때문이죠. 따라서 서버(서블릿)에서 웹 브라우저로 데이터를 전송할 때는 톰캣 컨테이너에서 미리 제공하는 여러 가지 전송 데이터 종류 중 하나를 지정해서 웹 브라우저로 전송합니다. 이처럼 톰캣 컨테이너에서 미리 설정해 놓은 데이터 종류들을 MIME-TYPE(마임 타입)이라고 합니다.

서버(서블릿)에서 자바 I/O의 스트림 클래스를 이용하여 웹 브라우저로 데이터를 전송할 때는 MIME-TYPE을 설정해서 전송할 데이터의 종류를 지정합니다.

다음은 MIME_TYPE으로 지정하는 예입니다

- HTML로 전송 시: text/html
- 일반 텍스트로 전송 시: text/plain
- XML 데이터로 전송 시: application/xml

웹 브라우저는 기본적으로 HTML만 인식하므로 서블릿에서 전송하는 대부분의 데이터는 MIME-TYPE을 text/html로 지정합니다.

그 외 톰캣 컨테이너에서는 자주 사용하는 데이터 종류를 MIME-TYPE으로 지정해 놓고 있으므로 서블릿에서 종류를 지정해서 사용하면 됩니다. 더 나아가 새로운 종류의 데이터를 지정하고 싶다면 CATALINA_HOME\conf\web.xml에 추가하면 됩니다.

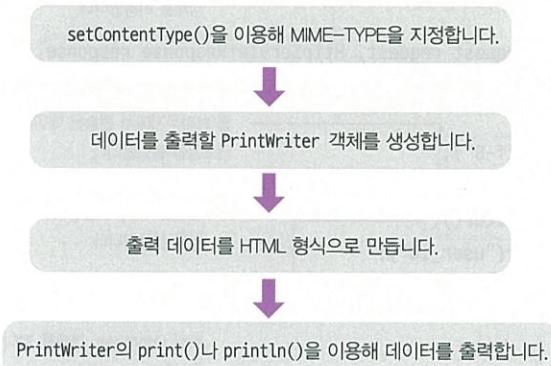
▼ 그림 6-18 톰캣 컨테이너의 web.xml에 정의된 여러 가지 MIME-TYPE

```
4435     <mime-mapping>
4436         <extension>xenc</extension>
4437         <mime-type>application/xenc+xml</mime-type>
4438     </mime-mapping>
4439     <mime-mapping>
4440         <extension>xer</extension>
4441         <mime-type>application/patch-ops-error+xml</mime-type>
4442     </mime-mapping>
4443     <mime-mapping>
4444         <extension>xfdf</extension>
4445         <mime-type>application/vnd.adobe.xfdf</mime-type>
4446     </mime-mapping>
4447     <mime-mapping>
4448         <extension>xfdl</extension>
4449         <mime-type>application/vnd.xfdl</mime-type>
4450     </mime-mapping>
4451     <mime-mapping>
4452         <extension>xht</extension>
4453         <mime-type>application/xhtml+xml</mime-type>
4454     </mime-mapping>
4455     <mime-mapping>
4456         <extension>xhtml</extension>
4457         <mime-type>application/xhtml+xml</mime-type>
4458     </mime-mapping>
```

6.4.2 HttpServletResponse를 이용한 서블릿 응답 실습

이번에는 서블릿이 응답하는 예제를 살펴보겠습니다. 사용자가 입력한 ID와 비밀번호를 전송하면 서블릿이 다시 브라우저에게 응답하는 예제입니다.

서블릿이 클라이언트(웹 브라우저)에 응답하는 과정은 다음과 같습니다.



1. login.html을 다음과 같이 수정합니다. 로그인창에서 ID와 비밀번호를 입력한 후 login2 서블릿으로 전송합니다.

코드 6-7 pro06/WebContent/login.html

```

<!DOCTYPE html>
<html>
<head>..</head>
<body>
    <form name="frmLogin" method="get" action="login2" 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>
  
```

매핑 이름을 login2로 설정합니다.

2. sec02.ex01 패키지에 LoginServlet2 클래스를 추가하고 다음과 같이 작성합니다. 브라우저에서 전달받은 ID와 비밀번호를 HTML 태그로 만든 후 다시 브라우저로 응답합니다.

코드 6-8 pro06/src/sec02/ex01/LoginServlet2.java

```

package sec02.ex01;
...
@WebServlet("/login2")
public class LoginServlet2 extends HttpServlet
{
    public void init() throws ServletException
    {
  
```

```

        System.out.println("init 메서드 호출");
    }

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 id = request.getParameter("user_id");
    String pw = request.getParameter("user_pw");

    String data = "<html>";
    data += "<body>";
    data += "아이디 : " + id;
    data += "<br>";
    data += "패스워드 : " + pw;
    data += "</body>";
    data += "</html>";
    out.print(data);
}

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

```

응답은 HttpServletResponse 객체를 이용합니다.

웹 브라우저에서 전송된 데이터의 인코딩을 설정합니다.

setContentType()을 이용해 응답할 데이터 종류가 HTML임을 설정합니다.

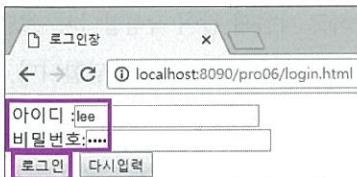
HttpServletResponse 객체의 getWriter()를 이용해 출력 스트림 PrintWriter 객체를 받아옵니다.

브라우저로 출력할 데이터를 문자열로 연결해서 HTML 태그로 만듭니다.

PrintWriter의 print()를 이용해 HTML 태그 문자열을 웹 브라우저로 출력합니다.

3. 브라우저에서 <http://localhost:8090/pro06/login.html>로 접속하여 ID와 비밀번호를 입력한 후 **로그인**을 클릭합니다.

▼ 그림 6-19 login.html로 접속하여 **로그인** 클릭



4. 그러면 서블릿이 ID와 비밀번호를 전달 받아 다시 브라우저로 출력합니다.

▼ 그림 6-20 서블릿의 응답 실행 결과



6.4.3 서블릿을 이용한 환율 계산기 예제 실습

또 다른 서블릿 응답 실습 예제로, 서블릿을 이용하여 환율 계산기를 구현해 보겠습니다. 말 그대로 원화를 입력 받아 외화로 변환해 주는 프로그램입니다.

1. sec02.ex01 패키지에 CalcServlet 클래스를 생성합니다.

▼ 그림 6-21 실습 파일 위치



2. CalcServlet 클래스를 다음과 같이 작성합니다. 최초 맵핑 이름인 /calc로 요청할 경우 command 값이 null이므로 환율 계산기 화면이 나타납니다. 계산기에서 값을 입력한 후 다시 요청할 경우 command 값이 calculate이므로 전달된 원화를 이용해 외화로 변환하여 결과를 출력합니다.

코드 6-9 pro06/src/sec02/ex01/CalcServlet.java

```
package sec02.ex01;
...
@WebServlet("/calc")
public class CalcServlet extends HttpServlet
{
    ...
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
```

```
request.setCharacterEncoding("utf-8");
response.setContentType("text/html; charset=utf-8");
PrintWriter pw = response.getWriter();
String command = request.getParameter("command"); ← 수행할 요청을 받아옵니다.
String won = request.getParameter("won"); ← 변환할 원화를 받아옵니다.
String operator = request.getParameter("operator"); ← 변환할 외화 종류를 받아옵니다.
```

```
if (command != null && command.equals("calculate"))
{
    String result = calculate(Float.parseFloat(won), operator);
    pw.print("<html><font size=10>변환 결과</font><br>");
    pw.print("<html><font size=10>" + result + "</font><br>");
    pw.print("<a href='/pro06/calc'>환율 계산기</a>");
    return;
}
```

최초 요청 시 command 가 null이면 계산기 화면을 출력하고, command 값이 calculate이면 계산 결과를 출력합니다.

```
pw.print("<html><title>환율 계산기</title>");
pw.print("<font size=5>환율 계산기</font><br>");
pw.print("<form name='frmCalc' method='get' action='/pro06/calc' /> ");
pw.print("원화: <input type='text' name='won' size=10 /> ");
pw.print("<select name='operator'>"); ← 셀렉트 박스에서 선택된 값을 name으로 전송합니다.
pw.print("<option value='dollar'>달러</option>");
pw.print("<option value='en'>엔화</option>");
pw.print("<option value='wian'>위안</option>");
pw.print("<option value='pound'>파운드</option>");
pw.print("<option value='euro'>유로</option>");
pw.print("</select>"); ← <hidden> 태그를 이용해 계산 기에서 서블릿으로 수행할 요청을 전달합니다.
pw.print("<input type='hidden' name='command' value='calculate' />"); ← 원화를 선택한 외화로 환산합니다.
pw.println("<input type='submit' value='변환' />"); ← 원화 입력 후 다시 서블릿 calc로 요청합니다.
pw.println("</form>");
```

```
pw.print("</html>"); ← 원화를 선택한 외화로 환산합니다.
pw.close();
```

```
private static String calculate(float won, String operator) {
    String result = null;
    if (operator.equals("dollar")) {
        result = String.format("%.6f", won / USD_RATE);
    } else if (operator.equals("en")) {
        result = String.format("%.6f", won / JPY_RATE);
    } else if (operator.equals("wian")) {
        result = String.format("%.6f", won / CNY_RATE);
    } else if (operator.equals("pound")) {
        result = String.format("%.6f", won / GBP_RATE);
    } else if (operator.equals("euro")) {
```

```

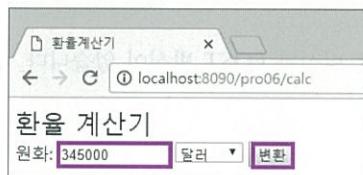
        result = String.format("%.6f", won / EUR_RATE);
    }
    return result;
}
}

```

<input> 태그를 hidden 속성으로 지정하면 화면에는 보이지 않지만 value에 이미 값이 저장되어 있습니다. 따라서 계산기 화면에서 서블릿으로 어떤 기능을 수행할지 명령을 전달할 수 있습니다. <hidden> 태그는 자주 사용하므로 기억해 두기 바랍니다.

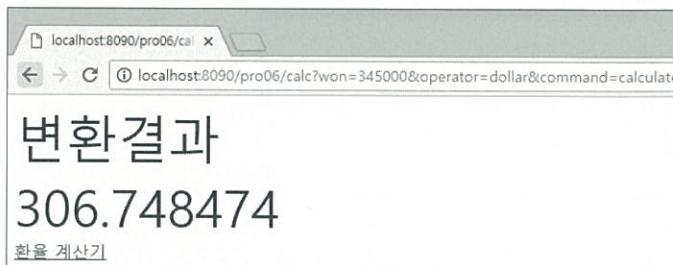
3. 웹 브라우저에서 `http://localhost:8090/pro06/calc`로 요청한 후 원화에 값을 입력하고 변환을 클릭합니다.

▼ 그림 6-22 웹 브라우저로 요청한 계산기 화면



4. 값을 전송한 후 결과를 웹 브라우저에 출력합니다.

▼ 그림 6-23 웹 브라우저로 출력된 변환 결과



6.5

웹 브라우저에서 서블릿으로 데이터 전송하기

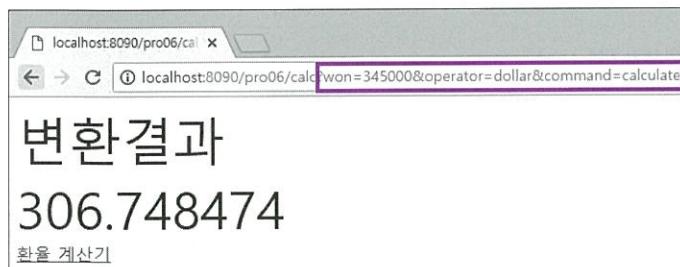
6.4절에서는 웹 브라우저에서 입력한 데이터를 서블릿으로 전송하면 서블릿에서 요청을 받는 방법에 대해 알아보았습니다. 이번에는 웹 브라우저에서 서블릿으로 데이터를 전송하는 방식을 알아보겠습니다.

6.5.1 GET/POST 전송 방식

웹 브라우저에서 서블릿으로 전송하는 방법은 크게 GET 방식과 POST 방식이 있습니다. 6.4절에서 실습한 환율 계산기 결과 화면을 다시 볼까요?

값을 전송한 후 결과를 웹 브라우저에 출력합니다.

▼ 그림 6-24 GET 방식으로 전송한 경우



주소창을 자세히 보면 물음표(?) 뒤에 입력된 값, 외화 종류 그리고 요청을 나타내는 문자열이 서블릿 매핑 이름 뒤에 붙어서 전송됩니다. 이렇게 URL 주소에 데이터를 붙여서 전송하는 방식을 **GET 방식이라고 합니다**. 그런데 로그인창에서 입력한 ID와 비밀번호를 이런 식으로 보이게 전송하면 개인 정보가 유출될 가능성이 높습니다. 이처럼 GET 방식으로 데이터를 전송할 경우에는 전송하는 데이터가 노출되므로 보안에 취약합니다.

반면에 POST 방식은 전송하는 데이터를 숨겨서 전송하므로 보안성이 좋습니다.

즉, GET 방식은 보안과 관련 없는 간단한 데이터를 쉽게 전송할 수 있는 반면, POST 방식은 보안과 관련된 데이터를 전송하는 데 많이 사용된다고 기억해 두면 됩니다. 두 방식의 자세한 특징은 표 6-5에 비교해 두었으니 참고하세요.

▼ 표 6-5 GET 방식과 POST 방식 비교

GET 방식	POST 방식
<ul style="list-style-type: none"> 서블릿에 데이터를 전송할 때는 데이터가 URL 뒤에 name=value 형태로 전송됩니다. 여러 개의 데이터를 전송할 때는 '&'로 구분해서 전송됩니다. 보안에 취약합니다. 전송할 수 있는 데이터는 최대 255자입니다. 기본 전송 방식이고 사용이 쉽습니다. 웹 브라우저에 직접 입력해서 전송할 수도 있습니다. 서블릿에서는 doGet()을 이용해 데이터를 처리합니다. 	<ul style="list-style-type: none"> 서블릿에 데이터를 전송할 때는 TCP/IP 프로토콜 데이터의 body 영역에 숨겨진 채 전송됩니다. 보안에 유리합니다. 전송 데이터 용량이 무제한입니다. 전송 시 서블릿에서는 또다시 가져오는 작업을 해야 하므로 처리 속도가 GET 방식보다 느립니다. 서블릿에서는 doPost()를 이용해 데이터를 처리합니다.

6.5.2 GET 방식으로 서블릿에 요청

로그인창 입력 예제인 코드 6-1를 보면 <form> 태그의 method 속성이 get으로 설정되어 있습니다. 이는 '서블릿에 GET 방식으로 데이터를 전송하겠다'라는 의미입니다.

▼ 그림 6-25 <form> 태그의 method 속성을 get으로 설정

```
<form name="frmLogin" method="get" 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="다시입력" >
</form>
```

마찬가지로 서블릿도 GET 방식으로 전송된 데이터를 doGet() 메서드를 이용해서 처리합니다.

▼ 그림 6-26 GET 방식으로 전송 데이터를 처리하는 doGet() 메서드

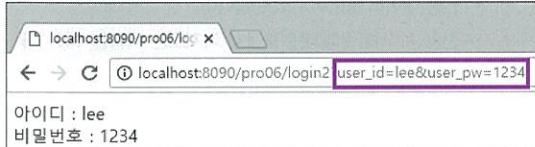
```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html;charset=utf-8");
    PrintWriter out = response.getWriter();

    String id = request.getParameter("user_id");
    String pw = request.getParameter("user_pw");

    String data = "<html>";
    data += "<body>";
    data += "아이디 : " + id;
    data += "<br>";
    data += "비밀번호 : " + pw;
    data += "</body>";
    data += "</html>";
    out.print(data);
}
```

그리고 웹 브라우저의 주소창을 보면 `http://localhost:8090/pro06/login2?user_id=lee&user_pw=1234`처럼 URL 뒤에 ‘name=value’ 쌍으로 붙어서 전송됩니다. 이처럼 GET 방식으로 전송하면 간편하다는 장점은 있으나 어떤 데이터를 전송하는지 다 노출되므로 보안상으로는 좋지 않습니다.

▼ 그림 6-27 주소창의 URL 뒤에 붙어서 전송되는 데이터



6.5.3 POST 방식으로 서블릿에 요청

이번에는 POST 방식으로 서블릿에 요청하여 처리하는 예제를 실습해 보겠습니다.

1. sec03.ex01 패키지를 만들고 LoginServlet3 클래스를 생성합니다.

▼ 그림 6-28 실습 파일 위치



2. login.html은 앞에서 생성한 파일을 편집해서 사용합니다. <form> 태그의 속성 method를 post로, action을 login3으로 수정합니다.

코드 6-10 pro06/WebContent/login.html

```
<form name="frmLogin" method="post" action="login3" encType="UTF-8">  
method 속성을 post로 수정합니다. 매핑 이름을 login3으로 수정합니다.
```

3. 다음과 같이 LoginServlet3 클래스를 작성합니다. 서블릿에서는 반드시 doPost() 메서드를 이용해서 처리해야 합니다.

코드 6-11 pro06/src/sec03/ex01/LoginServlet3.java

```
package sec03.ex01;

...
@WebServlet("/login3")
public class LoginServlet3 extends HttpServlet
{
    public void init() throws ServletException
    {
        System.out.println("init 메서드 호출");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        request.setCharacterEncoding("utf-8");
        String user_id = request.getParameter("user_id");
        String user_pw = request.getParameter("user_pw");
        System.out.println("아이디:" + user_id);
        System.out.println("비밀번호:" + user_pw);
    }

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

POST 방식으로 전송된 데이터를 처리하기 위해 doPost()를 이용합니다.

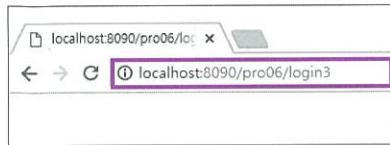
4. 웹 브라우저에서 로그인창을 요청한 후 ID와 비밀번호를 입력하고 **로그인**을 클릭합니다.

▼ 그림 6-29 ID와 비밀번호 입력 후 **로그인** 클릭



5. 웹 브라우저에서 전송되는 데이터는 TCP/IP의 헤더에 숨겨진 채 전송되므로 브라우저의 주소창을 보면 URL 뒤에는 아무것도 표시되지 않습니다.

▼ 그림 6-30 POST 방식으로 전송되는 데이터



이처럼 서블릿에서는 웹 브라우저에서 전송되는 방식에 따라 doGet()이나 doPost() 메서드로 대응해서 처리해야 합니다. 만약 전송 방식과 다른 메서드를 사용하면 브라우저에서 오류가 발생합니다. 예를 들어 웹 브라우저에서는 GET 방식으로 전송하는데 서블릿에서는 doPost() 메서드로 처리하면 GET 방식으로 처리하는 메서드가 없으므로 오류가 발생합니다.

다음은 전송 방식과 다른 메서드를 사용한 경우로, 브라우저에 GET 방식으로 처리하는 메서드가 없다는 오류 메시지를 출력합니다.

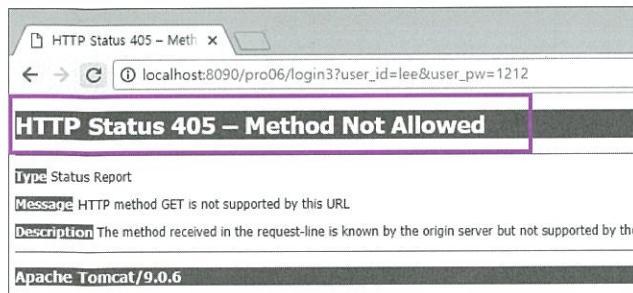
▼ 그림 6-31 로그인창에서 GET 방식으로 전송

```
<form name="frmLogin" method="get" action="login3" 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="다시입력">
</form>
```

▼ 그림 6-32 LoginServlet3.java에서 doPost()로 처리

```
35  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException
36      request.setCharacterEncoding("utf-8");
37      String user_id=request.getParameter("user_id");
38      String user_pw=request.getParameter("user_pw");
39      System.out.println("아이디:"+user_id);
40      System.out.println("비밀번호:"+user_pw);
41
42  }
```

▼ 그림 6-33 전송 방식과 다른 메서드로 처리할 경우 405 오류 발생



이상으로 GET 방식과 POST 방식으로 데이터를 전송하는 경우와 그에 따라 서블릿에서 처리하는 방법을 알아봤습니다. 이 두 방식으로 전송된 데이터는 반드시 HttpServlet에서 오버라이딩 된 doGet() 메서드나 doPost() 메서드를 사용해야 합니다. 만약 두 메서드가 서블릿에 존재하지 않거나 사용자가 임의로 만든 메서드를 사용하면 오류가 발생한다는 점을 꼭 유념하세요.

6.6 GET 방식과 POST 방식 요청 동시에 처리하기

JAVA WEB

웹 프로그램에서는 GET 방식과 POST 방식을 혼합해서 많이 사용합니다. 이때 각 방식마다 일일이 구분해서 구현해야 한다면 불편하겠죠? 이번에는 전송된 방식으로 doGet()이나 doPost() 메서드로 처리한 후 다시 doHandle()을 호출해서 모든 기능을 구현하는 예제를 실습해 보겠습니다.

- 앞에서 실습한 login.html을 다음과 같이 수정합니다. GET 방식으로 로그인하기 위해 method는 get으로, action은 login4로 수정합니다.

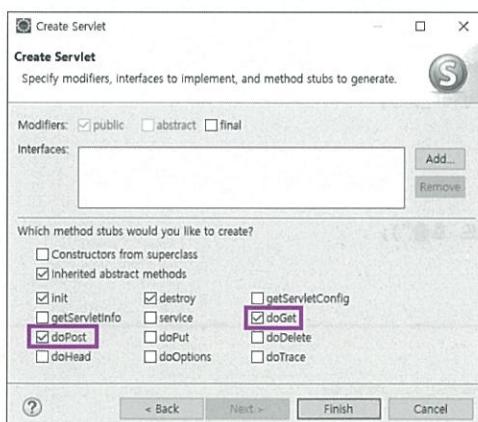
코드 6-12 pro06/WebContent/login.html

```
<form name="frmLogin" method="get" action="login4" encType="utf-8">
```

method 속성을 get으로, action 속성을 login4로 수정합니다.

- sec03.ex02 패키지에 LoginServlet4 서블릿을 만들 때 doGet()과 doPost()를 모두 추가합니다.

그림 6-34 doGet과 doPost의 체크박스에 체크



3. LoginServlet4 클래스를 다음과 같이 작성합니다. doGet()과 doPost() 메서드에서 doHandle() 메서드를 재호출하여 모든 방식의 요청을 처리합니다.

코드 6-13 pro06/src/sec03/ex02/LoginServlet4.java

```
package sec03.ex02;

...
public class LoginServlet4 extends HttpServlet
{
    public void init() throws ServletException
    {
        System.out.println("init 메서드 호출");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        System.out.println("doGet 메서드 호출");
        doHandle(request, response); ────────── GET 방식으로 요청 시 다시 doHandle()을 호출합니다.
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        System.out.println("doPost 메서드 호출");
        doHandle(request, response); ────────── POST 방식으로 요청 시 다시 doHandle()을 호출합니다.
    }

    private void doHandle(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        request.setCharacterEncoding("utf-8"); ────────── 모든 호출 방식에 대해 처리할 수 있습니다.
        String user_id = request.getParameter("user_id");
        System.out.println("doHandle 메서드 호출");
        String user_pw = request.getParameter("user_pw");
        System.out.println("아이디:" + user_id);
        System.out.println("비밀번호:" + user_pw);
    }

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

4. GET 방식과 POST 방식으로 각각 요청해 보겠습니다. 두 방식 모두 `doHandle()` 메서드로 처리한 후 결과를 출력합니다. 로그인창에 접속하여 ID와 비밀번호를 입력한 후 GET 방식으로 요청합니다. 이클립스 콘솔을 보면 `doHandle()` 메서드로 처리한 메시지가 출력된 것을 알 수 있습니다.

▼ 그림 6-35 로그인창에서 GET 방식으로 요청한 결과

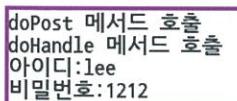


5. 로그인창의 `method` 속성을 `post`로 변경한 후 요청하면 `doHandle()` 메서드로 처리한 후 결과를 출력합니다.

코드 pro06/WebContent/login.html

```
<form name="frmLogin" method="post" action="login4" encType="utf-8">
```

▼ 그림 6-36 POST 방식으로 요청한 결과



6.7

자바스크립트로 서블릿에 요청하기

웹 사이트에 로그인할 때 ID나 비밀번호를 입력하지 않고 로그인하면 오류 메시지가 출력됩니다. 앞 절에서는 <form> 태그에서 바로 서블릿으로 데이터를 전송했지만 전송 전에 로그인하면 ID와 비밀번호 입력 유무 체크하기처럼 전송 데이터에 대해 유효성 검사를 하는 경우가 많습니다. 이런 기능은 자바스크립트로 구현하므로 이번에는 자바스크립트로 서블릿에 요청하는 방법을 알아보겠습니다.

서블릿에 요청할 때 <form> 태그에서 직접 요청하는 것이 아니라 자바스크립트 함수를 호출하고 유효성 검사를 한 후 자바스크립트 함수에서 서블릿에 요청하는 예제를 만들어 보겠습니다.

1. 다음과 같이 sec03_ex03 패키지에 LoginServlet5 클래스를 생성하고 login2.html을 추가로 생성합니다.

▼ 그림 6-37 실습 파일 위치



2. 다음과 같이 login2.html을 작성합니다. 자바스크립트 함수에서 <form> 태그에 접근하여 값 입력 여부를 체크한 후 action 속성에 전송할 서블릿 이름을 지정합니다. 그런 다음 submit() 함수를 호출하여 서블릿으로 전송합니다. <input> 태그의 hidden 속성을 지정하면 화면에는 보이지 않지만 value에 미리 값이 저장됩니다.

코드 6-14 pro06/WebContent/login2.html

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <script type="text/javascript">
        function fn_validate() {
            var frmLogin = document.frmLogin;
            var user_id = frmLogin.user_id.value;           └─> <form> 태그 내 <input> 태그의 name 속성으로
            var user_pw = frmLogin.user_pw.value;           └─> 입력한 ID와 비밀번호를 받아 옵니다.

            if ((user_id.length == 0 || user_id == "") ||
                (user_pw.length == 0 || user_pw == ""))
                alert("아이디와 비밀번호는 필수입니다.");
            } else {
                frmLogin.method = "post";                   └─> <form> 태그의 전송 방식을 post로 설정합니다.
                frmLogin.action = "login5";               └─> action 속성을 서블릿 매핑 이름인 login5로 설정합니다.
                frmLogin.submit();                      └─> 자바스크립트에서 서블릿으로 전송합니다.
            }
        }
    </script>
    <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="button" onClick="fn_validate()" value="로그인">
        <input type="reset" value="다시 입력">
        <input type="hidden" name="user_address" value="서울시 성북구" />           └─> <hidden> 태그를 이용해 화면에는 보이지 않게
    </form>                                         하면서 값을 서블릿으로 전송합니다.
</body>
</html>

```

3. LoginServlet5 클래스를 다음과 같이 작성합니다. 서블릿에서 getParameter() 메서드를 이용해 <hidden> 태그로 전송된 주소를 받아옵니다.

```
코드 6-15 pro06/src/sec03/ex03/LoginServlet5.java
package sec03.ex03;

...
@WebServlet("/login5")
public class LoginServlet5 extends HttpServlet
{
    public void init()
    {
        System.out.println("init 메서드 호출");
    }

    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();
        String id = request.getParameter("user_id");
        String pw = request.getParameter("user_pw");
        String address = request.getParameter("user_address");
        System.out.println("아이디 : " + id);           ← <hidden> 태그로 전송된 값을 받아옵니다.
        System.out.println("비밀번호 : " + pw);

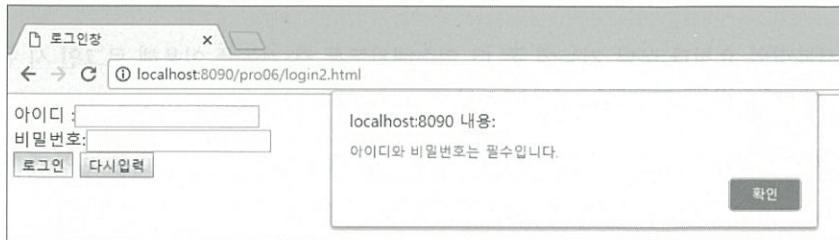
        String data = "<html>";
        data += "<body>";
        data += "아이디 : " + id;
        data += "<br>";
        data += "비밀번호: " + pw;
        data += "<br>";
        data += "주소 : " + address;
        data += "</body>";
        data += "</html>";
        out.print(data);
    }

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

전송된 값을 웹 브라우저로 출력합니다.

4. <http://localhost:8090/pro06/login2.html>로 요청합니다. ID와 비밀번호를 입력하지 않고 로그인을 클릭하면 오류 창이 나타납니다.

▼ 그림 6-38 ID와 비밀번호를 입력하지 않은 경우



반면에 ID와 비밀번호를 정상적으로 입력한 경우에는 웹 브라우저로 입력 값을 출력합니다.

▼ 그림 6-39 ID와 비밀번호를 정상적으로 입력한 경우



6.8

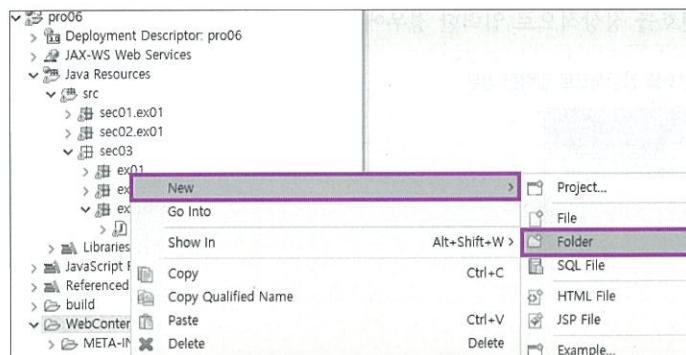
서블릿을 이용한 여러 가지 실습 예제

이번에는 서블릿의 요청과 응답 기능에 좀 더 익숙해지도록 각 기능을 이용해 로그인 시 유효성 검사, 구구단 출력 등 다양한 예제를 실습해 보겠습니다.

- 현재 /WebContent 위치에 실습용 HTML 파일을 따로 저장하는 폴더를 만들겠습니다.

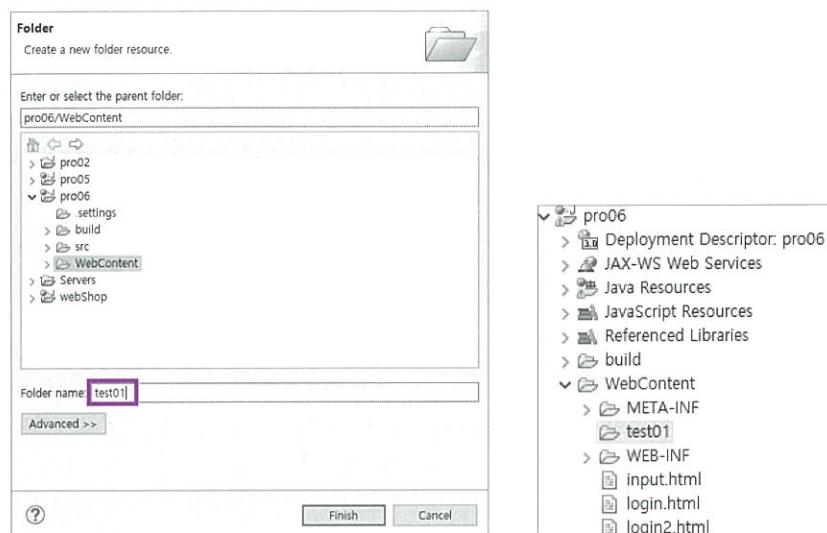
WebContent 폴더를 선택하고 마우스 오른쪽 버튼을 클릭한 후 New > Folder를 선택합니다.

▼ 그림 6-40 New > Folder 선택



- 폴더 이름을 test01로 입력한 후 폴더가 생성되었는지 확인합니다.

▼ 그림 6-41 폴더 test01 생성 후 확인



6.8.1 실습 예제1: 서블릿에 로그인 요청 시 유효성 검사하기

문제: ID를 정상적으로 입력했을 때는 로그인 메시지를 표시하고, ID를 입력하지 않았을 때는 다시 로그인하라는 메시지를 표시하도록 작성하시오.

1. test01 폴더에 login.html을 만들고 다음과 같이 작성합니다.

코드 6-16 pro06/WebContent/test01/login.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>로그인 창</title>
</head>
<body>
    <form name="frmLogin" method="post" action="/pro06/loginTest" 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>
```

test01 폴더에 위치한 HTML에서 서블릿에 요청하므로 action 속성에서 서블릿 매핑 이름 앞에 프로젝트 이름 /pro06을 붙여주어야 합니다.

텍스트 박스에 입력한 ID를 name 속성인 user_id로 전송합니다.

텍스트 박스에 입력한 비밀번호를 name 속성인 user_pw로 전송합니다.

2. LoginTest 클래스를 다음과 같이 작성합니다. ID나 비밀번호를 제대로 입력하지 않으면 오류 메시지를 출력한 후 다시 로그인창으로 이동합니다.

코드 6-17 pro06/src/sec04/ex01/LoginTest.java

```
package sec04.ex01;

@WebServlet("/loginTest")
public class LoginTest extends HttpServlet
{
    public void init()
    {
        System.out.println("init 메서드 호출");
    }

    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();
    }
}
```

```

String id = request.getParameter("user_id");
String pw = request.getParameter("user_pw");

System.out.println("아이디 : " + id);
System.out.println("패스워드 : " + pw);

if(id!= null &&(id.length()!=0)) {
    out.print("<html>");
    out.print("<body>");
    out.print( id +" 님!! 로그인 하셨습니다." );
    out.print("</body>");
    out.print("</html>");

}else{
    out.print("<html>");
    out.print("<body>");
    out.print("아이디를 입력하세요!!!" );
    out.print("<br>");
    out.print("<a href='http://localhost:8090/pro06/test01/login.html'>
                로그인 창으로 이동 </a>");

    out.print("</body>");
    out.print("</html>");
}

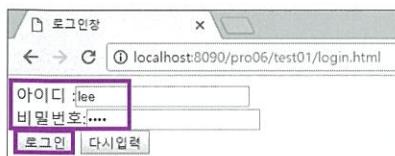
public void destroy() {
    System.out.println("destroy 메서드 호출");
}
}

```

_____ ID와 비밀번호가 없으면 다시 로그인창으로 이동합니다.

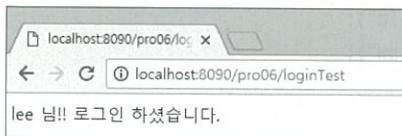
3. <http://localhost:8090/pro06/test01/login.html>로 요청한 후 ID와 비밀번호를 정상적으로 입력하고 **로그인**을 클릭합니다.

▼ 그림 6-42 ID와 비밀번호를 정상적으로 입력



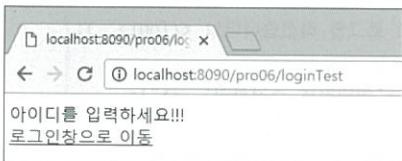
4. 로그인 성공 메시지가 정상적으로 출력됩니다.

▼ 그림 6-43 ID와 비밀번호를 정상적으로 입력한 결과



만약 ID를 입력하지 않고 요청하면 다시 입력하라는 오류 메시지가 출력됩니다.

▼ 그림 6-44 ID를 입력하지 않은 경우



6.8.2 실습 예제2: 서블릿으로 로그인 요청 시 관리자 화면 나타내기

문제: 실습 예제 1을 이용해 로그인 시 admin ID로 로그인하면 회원 관리와 회원 삭제 기능을 보여주도록 작성하시오.

1. LoginTest2 서블릿을 생성하고 다음과 같이 작성합니다. admin ID로 로그인 시 관리자 화면을 보여주는 서블릿으로, 이중 if문을 사용해 ID를 정상적으로 입력해도 다시 ID가 admin인지 체크합니다.

코드 6-18 pro06/src/sec04/ex01/LoginTest2.java

```
package sec04.ex01;

...
@WebServlet("/loginTest2")
public class LoginTest2 extends HttpServlet {
    public void init()
    {
        System.out.println("init 메서드 호출");
    }

    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();
String id = request.getParameter("user_id");
String pw = request.getParameter("user_pw");

System.out.println("아이디 : " + id);
System.out.println("패스워드 : " + pw);
if(id!= null &&(id.length()!=0)) {
    if(id.equals("admin")) {
        out.print("<html>");
        out.print("<body>");
        out.print( "<font size='12'>관리자로 로그인 하셨습니다!! </font>" );
        out.print("<br>");
        out.print("<input type=button value='회원정보 수정하기' />");
        out.print("<input type=button value='회원정보 삭제하기' />");
        out.print("</body>");
        out.print("</html>");
    } else {
        out.print("<html>");
        out.print("<body>");
        out.print( id +" 님!! 로그인 하셨습니다." );
        out.print("</body>");
        out.print("</html>");
    }
} else{
    out.print("<html>");
    out.print("<body>");
    out.print("ID와 비밀번호를 입력하세요!!!!" );
    out.print("<br>");
    out.print("<a href='http://localhost:8090/pro06/test01/login.html'>
                로그인창으로 이동 </a>" );
    out.print("</body>");
    out.print("</html>");
}
}

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

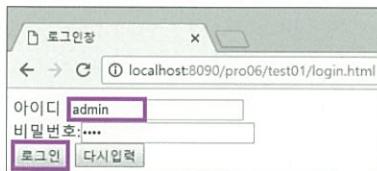
```

이중 if문을 사용해 ID가 admin이면 관리자창을 보여줍니다.

관리자가 아닌 일반 사용자일 경우 로그인 성공 메시지를 보여줍니다.

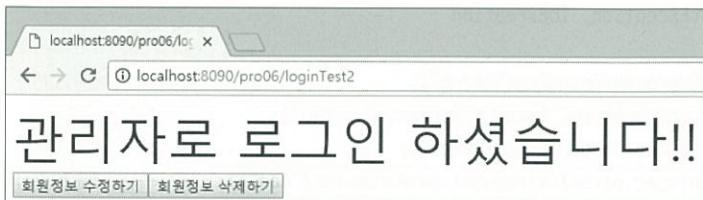
2. login.html에서 LoginTest2를 매핑하도록 수정합니다.
3. http://localhost:8090/pro06/test01/login.html로 요청한 후 ID를 admin으로 입력한 후 로그인 합니다.

▼ 그림 6-45 ID를 admin으로 입력해 로그인



4. 다음과 같이 "관리자로 로그인 하셨습니다!!"라는 메시지가 표시됩니다.

▼ 그림 6-46 관리자창에 표시



6.8.3 실습 예제3: 서블릿으로 요청 시 구구단 출력하기

문제: 구구단 단수를 입력 받아 단수를 출력하시오.

1. 구구단의 단수를 입력 받는 gugu.html을 다음과 같이 작성합니다. 단수를 입력 받아 guguTest 서블릿으로 전송합니다.

코드 6-19 pro06/WebContent/test01/gugu.html

```
<!DOCTYPE html>
<html>
<head>
    <title>단수 입력창</title>
</head>
<body>
    <h1>출력할 구구단의 수를 지정해 주세요.</h1>
    <form method="get" action="/pro06/guguTest"> •———— 매핑 이름이 guguTest인
        출력할 구구단 :<input type="text" name="dan" /> <br>———— 서블릿으로 전송합니다.
        <input type="submit" value="구구단 출력">
    </form>
</body>
</html>
```

2. GuguTest 클래스를 다음과 같이 작성합니다. <table> 태그의 <tr> 태그와 자바의 for문을 이용해 구구단을 연속해서 행으로 출력합니다.

코드 6-20 pro06/src/sec04/ex01/GuguTest.java

```
package sec04.ex01;  
...  
@.WebServlet("/guguTest")  
public class GuguTest extends HttpServlet  
{  
    public void init()  
    {  
        System.out.println("init 메서드 호출");  
    }  
  
    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();  
        int dan = Integer.parseInt(request.getParameter("dan")); ────────── 전송된 dan의 값을  
        out.print(" <table border=1 width=800 align=center>"); ────────── 받아옵니다.  
        out.print("<tr align=center bgcolor=#FFFF66>");  
        out.print("<td colspan=2>" + dan + " 단 출력 </td>");  
        out.print("</tr>");  
        for (int i = 1; i < 10; i++) ────────── for문을 이용해 연속해서 결과를  
        { ────────── 테이블 행으로 출력합니다.  
            out.print("<tr align=center>");  
            out.print("<td width=400>");  
            out.print(dan + " * " + i);  
            out.print("</td>");  
            out.print("<td width=400>");  
            out.print(i * dan);  
            out.print("</td>");  
            out.print("</tr>");  
        }  
        out.print("</table>");  
    }  
  
    public void destroy()  
    {  
        System.out.println("destroy 메서드 호출");  
    }  
}
```

3. 출력 결과 화면에서 구구단 입력창을 요청한 후 단수를 입력합니다.

▼ 그림 6-47 출력할 구구단 수 입력

4. 전송된 단수에 대한 구구단이 브라우저에 행으로 출력됩니다.

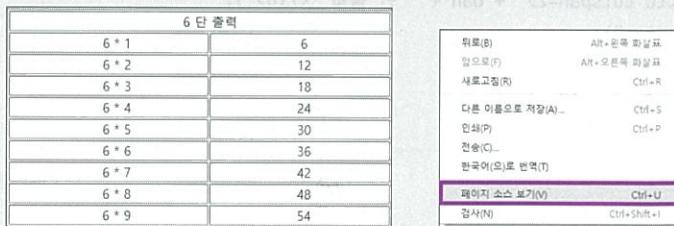
▼ 그림 6-48 구구단 출력하기

6 단 출력	
6 * 1	6
6 * 2	12
6 * 3	18
6 * 4	24
6 * 5	30
6 * 6	36
6 * 7	42
6 * 8	48
6 * 9	54

Note 三 HTML 소스 보는 방법

프로그램을 개발하다 보면 브라우저에서 전송된 HTML 소스를 보고 싶을 때가 있습니다. 그런 경우에는 웹 브라우저에서 마우스 오른쪽 버튼을 클릭한 후 페이지 소스 보기 를 선택합니다. 그러면 서블릿에서 전송된 HTML 소스가 표시됩니다.

▼ 그림 6-49 마우스 오른쪽 버튼 클릭 후 페이지 소스 보기 선택



▼ 그림 6-50 서블릿에서 전송된 HTML 태그

```
<table border=1 width=800 align=center><tr align=center bgcolor="#FFFF66"><td colspan=2>2 단 출력 </td></tr><tr align=center><td width=400>6</td><td width=400>6</td></tr><tr align=center><td width=400>6</td><td width=400>6</td></tr></table>
```

5. 이번에는 서블릿의 응답 기능을 이용해 구구단 테이블의 행 배경색을 교대로 바꾸어 보겠습니다. 다음과 같이 GuguTest2 클래스를 생성하고 코드를 작성합니다.

코드 6-21 pro06/src/sec04/ex02/GuguTest2.java

```
package sec04.ex02;  
...  
@.WebServlet("/guguTest2")  
public class GuguTest2 extends HttpServlet  
{  
    public void init()  
    {  
        System.out.println("init 메서드 호출");  
    }  
  
    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();  
    int dan = Integer.parseInt(request.getParameter("dan"));  
    out.print(" <table border=1 width=800 align=center>");  
    out.print("<tr align=center bgcolor='#FFFF66'>");  
    out.print("<td colspan=2>" + dan + " 단 출력 </td>");  
    out.print("</tr>");  
  
    for (int i = 1; i < 10; i++)  
    {  
        if (i % 2 == 0)  
        {  
            out.print("<tr align=center bgcolor='#ACFA58'> ");  
        } else  
        {  
            out.print("<tr align=center bgcolor='#81BEF7'> ");  
        }  
    }  
}
```

if문을 이용해 행을 나타내는 <tr> 태그에 대해 교대로 다른 배경색을 적용합니다.

```
        out.print("<td width=400>");
        out.print(dan + " * " + i);
        out.print("</td>");
        out.print("<td width=400>");
        out.print(i * dan);
        out.print("</td>");
        out.print("</tr>");
    }

    out.print("</table>");
}

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

6. guguTest2 서블릿을 매핑하도록 gugu.html 파일을 수정한 후 브라우저에 요청합니다. 구구단 수를 입력하면 테이블의 배경색이 교대로 바뀌는 것을 확인할 수 있습니다.

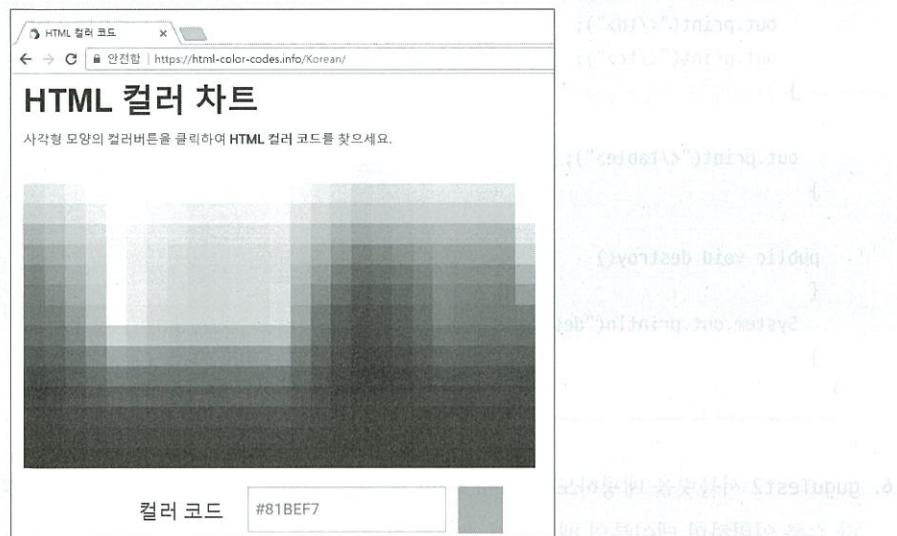
▼ 그림 6-51 테이블 행의 배경색이 교대로 바뀌도록 표시하기

6 단 출력	
6 * 1	6
6 * 2	12
6 * 3	18
6 * 4	24
6 * 5	30
6 * 6	36
6 * 7	42
6 * 8	48
6 * 9	54

Note ≡ 다음 사이트로 접속하면 원하는 색에 대한 코드를 얻을 수 있습니다.

- <https://html-color-codes.info/Korean/>

▼ 그림 6-52 웹 브라우저에서 색상 코드 얻기



7. 이번에는 서블릿의 응답 기능을 조금 더 응용해서 행마다 라디오 박스와 체크박스가 표시되도록 구현해 보겠습니다. 다음과 같이 GuguTest3 클래스를 작성합니다.

코드 6-22 pro06/src/sec04/ex03/GuguTest3.java

```
package sec04.ex01;
...
@WebServlet("/guguTest3")
public class GuguTest3 extends HttpServlet
{
    public void init()
    {
        System.out.println("init 메서드 호출");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        ...
        for (int i = 1; i < 10; i++)
        {
            if (i % 2 == 0)

```

```

{
    out.print("<tr align=center bgcolor='#ACFA58'> ");
} else
{
    out.print("<tr align=center bgcolor='#81BEF7'> ");
}
out.print("<td width=200> ");
out.print("<input type='radio' />" + i);
out.print("</td>");
out.print("<td width=200> ");
out.print("<input type='checkbox' />" + i);
out.print("</td>");
out.print("<td width=400> ");
out.print(dan + " * " + i);
out.print("</td>");
out.print("<td width=400> ");
out.print(i * dan);
out.print("</td>");
out.print("</tr>");
}

out.print("</table>");
}

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

```

각 행에 라디오 박스와
체크박스를 표시합니다.

8. 다음은 실행 결과입니다.

▼ 그림 6-53 서블릿의 응답 기능 이용해 라디오 박스와 체크박스 표시하기

7 단 출력			
<input checked="" type="radio"/> 1	<input type="checkbox"/> 1	7 * 1	7
<input type="radio"/> 2	<input type="checkbox"/> 2	7 * 2	14
<input checked="" type="radio"/> 3	<input type="checkbox"/> 3	7 * 3	21
<input type="radio"/> 4	<input type="checkbox"/> 4	7 * 4	28
<input checked="" type="radio"/> 5	<input type="checkbox"/> 5	7 * 5	35
<input type="radio"/> 6	<input type="checkbox"/> 6	7 * 6	42
<input checked="" type="radio"/> 7	<input type="checkbox"/> 7	7 * 7	49
<input type="radio"/> 8	<input type="checkbox"/> 8	7 * 8	56
<input checked="" type="radio"/> 9	<input type="checkbox"/> 9	7 * 9	63

이상으로 서블릿의 세 가지 주요 기능 중 요청하는 기능과 응답하는 기능을 여러 가지 예제를 통해 알아보았습니다. 눈치가 빠른 사람이라면 서블릿 응답 기능은 결국 웹 애플리케이션 화면을 구현하는 기능이라는 것을 알았을 것입니다. 비록 지금은 서블릿으로 화면을 구현하지 않지만 서블릿이 처음 나왔을 때는 응답 기능을 이용해서 화면을 구현했습니다. 따라서 이 원리를 기억해 두면 웹 개발하는 데 큰 도움이 될 것입니다.

Note ≡ 계층형 패키지 구조 만들기

아클립스에서 패키지를 생성하면 보통 다음과 같이 패키지 이름이 나열됩니다. 패키지를 계층 구조로 표시하면 편리합니다.

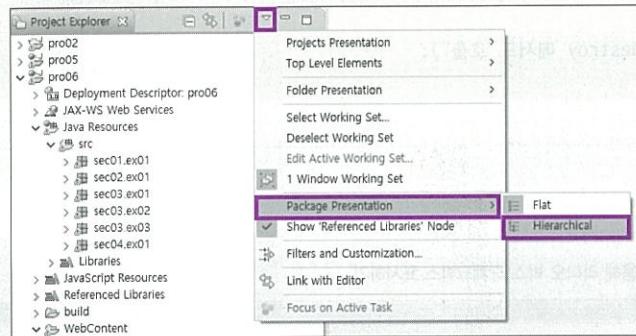
▼ 그림 6-54 패키지가 나열형으로 표시된 모습



그런데 실제로 파일 탐색기에 있는 폴더의 계층 구조로 보면 개발 시 가독성이 조금 더 좋습니다.

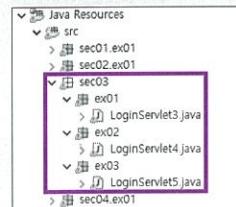
아클립스의 Project Explorer 창에서 상단의 역삼각형을 클릭한 후 Package Presentation > Hierarchical을 선택합니다.

▼ 그림 6-55 Package Presentation > Hierarchical 선택



sec03의 하위 패키지들이 계층 구조로 표시된 것을 볼 수 있습니다.

▼ 그림 6-56 패키지들이 계층 구조로 표시됨



7장

서블릿 비즈니스 로직 처리

- 7.1 서블릿의 비즈니스 로직 처리 방법
- 7.2 서블릿의 데이터베이스 연동하기
- 7.3 DataSource 이용해 데이터베이스 연동하기
- 7.4 DataSource 이용해 회원 정보 등록하기
- 7.5 회원 정보 삭제하기

7.1

서블릿의 비즈니스 로직 처리 방법

웹 프로그램은 클라이언트의 요청에 대해서 비즈니스 처리 기능을 이용해 데이터 저장소에서 데이터를 조회한 후 서블릿의 응답 기능을 이용해 클라이언트에게 결과를 전송합니다.

예를 들어 인터넷 교보문고나 예스24 같은 도서 쇼핑몰(온라인 서점)에서 책 제목을 검색창에 입력하고 검색 버튼을 누르면 책 제목이 서블릿으로 전송됩니다. 그럼 서블릿은 책 제목을 전송 받아 책 제목에 대한 정보를 데이터베이스 연동 기능을 이용해 조회합니다. 그리고 조회한 결과를 서블릿 응답 기능을 이용해 클라이언트 브라우저에 전송하여 결과를 보여줍니다.

▼ 그림 7-1 '자바스크립트'로 검색하면 데이터베이스와 연동해 책 제목 조회 결과가 표시됨

The screenshot shows the YES24 website's search interface. In the search bar, the query '자바스크립트' is entered. Below the search bar, there are filters for category: 국내도서, 외국도서, eBook, DVD/BD, 종고산, 리뷰, 기사, 인터뷰, and 결제내 재검색. The search results are displayed in two sections. The first section shows a thumbnail of a book cover titled '자바스크립트 + 제이쿼리 입문' with the following details: [도서] Do It! 자바스크립트 + 제이쿼리 입문 (전면 개정판), 정인용 저 | 이지스퍼블리싱 | 2018년 04월, 20,000원 → 18,000원(10% 할인) | YES포인트 1,000원(5% 지급). The second section shows another book cover titled 'Learning JavaScript' with the following details: [도서] 러닝 자바스크립트: ES6로 제대로 알아하는 모던 자바스크립트 앱 개발 이선 보라운 저 | 한선웅 역 | 한빛미디어 | 2017년 07월, 28,000원 → 25,200원(10% 할인) | YES포인트 1,400원(5% 지급). Both sections include a '도착 예상일' (Delivery Date) note: 지금 주문하면 오늘 도착예정, and purchase information: 판매자수 15,249 | 회원리뷰 (6개) | 내용 ★★★★☆ 멍腆/ 구성 ★★★★☆. The bottom right corner of the screenshot has a purple rectangular highlight around the second book entry.

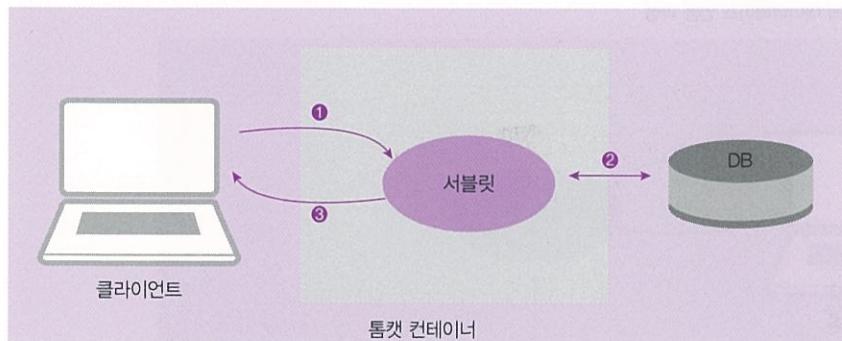
서블릿 비즈니스 처리 작업이란 서블릿이 클라이언트로부터 요청을 받으면 그 요청에 대해 작업을 수행하는 것을 의미합니다. 웹 프로그램에서 대부분의 비즈니스 처리 작업은 데이터베이스 연동 관련 작업이지만 그 외에 다른 서버와 연동해서 데이터를 얻는 작업도 수행합니다. 이 기능은 서블릿의 핵심 기능이라 할 수 있을 만큼 중요합니다.

서블릿의 비즈니스 작업 예로는 여러 가지를 들 수 있지만 대표적인 것들은 다음과 같습니다.

- 웹 사이트 회원 등록 요청 처리 작업
- 웹 사이트 로그인 요청 처리 작업
- 쇼핑몰 상품 주문 처리 작업

그림 7-2는 서블릿의 세 가지 기능 중 비즈니스 처리 과정을 나타낸 것입니다.

▼ 그림 7-2 서블릿의 비즈니스 처리 과정



- ❶ 클라이언트로부터 요청을 받습니다.
- ❷ 데이터베이스 연동과 같은 비즈니스 로직을 처리합니다.
- ❸ 처리 결과를 클라이언트에게 돌려줍니다.

7.2

서블릿의 데이터베이스 연동하기

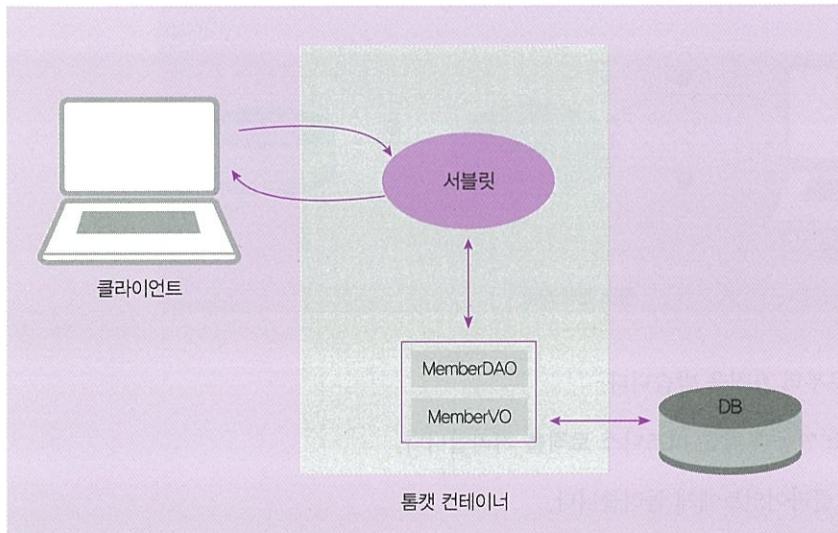
이번에는 서블릿에서 데이터베이스와 연동하여 조회한 데이터를 얻은 후 클라이언트의 웹 브라우저로 응답하는 과정을 알아보겠습니다.



서블릿의 비즈니스 처리 기능을 이해하려면 우선 데이터베이스 접근 명령어인 SQL문을 어느 정도 알아야 합니다. 따라서 데이터베이스 이론이 부족하다면 온라인 강의나 책을 통해 **자바 데이터베이스 기능을 먼저 익히는** 것이 좋습니다.

그림 7-3은 서블릿이 데이터베이스와 연동하는 과정을 나타낸 것입니다.

▼ 그림 7-3 서블릿의 데이터베이스 연동 과정



사실 서블릿에서 데이터베이스와 연동하는 과정은 자바의 데이터베이스 연동 과정과 같습니다. 클라이언트로부터 요청을 받으면 서블릿은 SQL문을 사용해 데이터베이스에 접근하여 작업을 합니다. 이 과정에서 DAO와 VO 클래스가 사용됩니다.

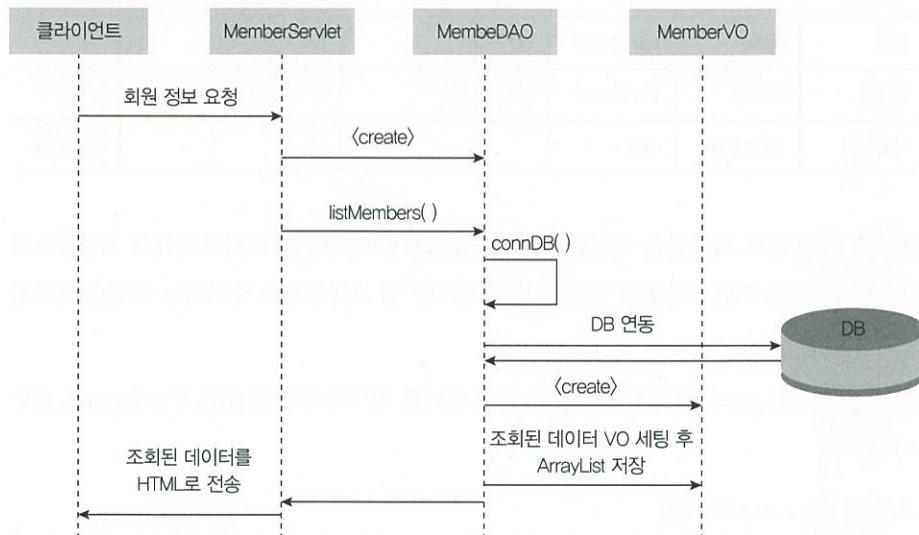


관련 지식이 부족한 초보자는 자바 입문서 또는 필자가 운영하는 자바 카페의 데이터베이스 부분을 참고하기 바랍니다(<http://cafe.naver.com/standardjava>).

7.2.1 서블릿으로 회원 정보 테이블의 회원 정보 조회

그림 7-3을 볼까요? 각 클래스가 연동해서 데이터베이스에 접근합니다. 그런 다음 서블릿에서 회원 정보를 조회한 후 이 정보를 다시 HTML로 만들어 웹 브라우저로 전송해 출력합니다.

▼ 그림 7-4 회원 정보 조회 과정



- ❶ 웹 브라우저가 서블릿에게 회원 정보를 요청합니다.
- ❷ MemberServlet은 요청을 받은 후 MemberDAO 객체를 생성하여 listMembers() 메서드를 호출합니다.
- ❸ listMembers()에서 다시 connDB() 메서드를 호출하여 데이터베이스와 연결한 후 SQL문을 실행해 회원 정보를 조회합니다.
- ❹ 조회된 회원 정보를 MemberVO 속성에 설정한 후 다시 ArrayList에 저장합니다.
- ❺ ArrayList를 다시 메서드를 호출한 MemberServlet으로 반환한 후 ArrayList의 MemberVO를 차례대로 가져와 회원 정보를 HTML 태그의 문자열로 만듭니다.
- ❻ 만들어진 HTML 태그를 웹 브라우저로 전송해서 회원 정보를 출력합니다.

표 7-1은 회원 정보를 저장하는 t_member 테이블의 구성을 나타낸 것입니다.

▼ 표 7-1 테이블 t_member 구성

NO	속성 이름	컬럼 이름	자료형	크기	유일키 여부	NULL 여부	키	기본값
1	ID	id	varchar2	10	Y	N	기본키	
2	비밀번호	pwd	varchar2	10		N		
3	이름	name	varchar2	50		N		
4	이메일	email	varchar2	50		N		
5	가입일자	joinDate	date			N		sysdate

그럼 지금부터 회원 정보 테이블을 생성한 후 회원 정보를 추가하여 웹 브라우저에서 서블릿으로 요청하면 데이터베이스(회원 테이블)와 연동해 회원 정보를 웹 브라우저로 출력하는 작업을 해보겠습니다.

- 먼저 SQL Developer에서 회원 테이블과 회원 정보를 입력하기 위해 SQL Developer를 실행 합니다.

▼ 그림 7-5 SQL Developer 실행



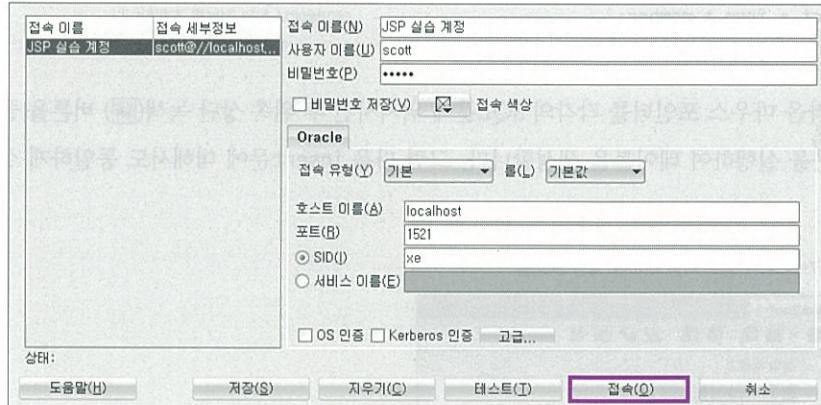
2. 왼쪽 메뉴의 새를 클릭한 후 새 접속...을 선택합니다.

▼ 그림 7-6 새 접속... 선택



3. 왼쪽 메뉴에서 미리 만들어 놓은 접속 이름을 클릭하거나 직접 연결 정보를 입력한 후 접속을 클릭합니다.

▼ 그림 7-7 데이터베이스 연동 정보 입력



4. 접속한 후 생성되는 워크시트에 다음과 같은 테이블 생성 SQL문을 입력합니다.

코드 7-1 회원 정보를 저장하는 테이블을 생성하고 추가하는 SQL문

```
--회원 테이블 생성
create table t_member(
    id varchar2(10) primary key,
    pwd varchar2(10),
    name varchar2(50),
    email varchar2(50),
    joinDate date default sysdate •———— 명시적으로 추가하지 않으면
);                                         현재 시각을 입력합니다.

--회원 정보 추가
insert into t_member
values('hong','1212','홍길동','hong@gmail.com',sysdate);

insert into t_member
values('lee','1212','이순신','lee@test.com',sysdate);

insert into t_member
values('kim','1212','김유신','kim@jweb.com',sysdate);
commit; •———— SQL Developer에서 테이블에 회원 정보를 추가한 후 반드시
                 커밋(commit)을 해줘야 영구적으로 반영이 됩니다.

select * from t_member; •———— 테이블에서 회원 정보를 조회합니다.
```

5. 그런 다음 마우스 포인터를 각각의 SQL문에 위치시킨 후 왼쪽 상단 녹색(▶) 버튼을 클릭해 SQL문을 실행하여 테이블을 생성합니다. 그런 다음 insert문에 대해서도 동일하게 실행합니다.

▼ 그림 7-8 회원 정보 저장 테이블 생성 SQL문

The screenshot shows the Oracle SQL Developer interface with a SQL worksheet titled 'JSP 실습2.sql'. The worksheet contains the following SQL code:

```
--회원 테이블 생성
create table t_member(
    id varchar2(10) primary key,
    pwd varchar2(10),
    name varchar2(50),
    email varchar2(50),
    joinDate date default sysdate
);
```

The code is highlighted with syntax coloring. The 'Execute' button (a green triangle icon) is highlighted with a purple rectangle, indicating it is the target for the next step.

6. 커밋이 완료됐다는 메시지가 나타나고 select문으로 조회 시 회원 정보가 표시됩니다.

▼ 그림 7-9 커밋 후 select문으로 조회한 회원 정보

ID	PWD	NAME	EMAIL	JOINDATE
1 hong	1212	홍길동	hong@gmail.com	18/09/04
2 lee	1212	이순신	lee@test.com	18/09/04
3 kim	1212	김유신	kim@jweb.com	18/09/04

7. 이클립스에서 만든 프로젝트에서 회원 정보를 조회해 보겠습니다. 새 프로젝트 pro07을 생성한 다음 오라클 데이터베이스와 연동하는 데 필요한 드라이버인 ojdbc6.jar를 프로젝트의 /WebContent/WEB-INF/lib 폴더에 복사하여 붙여 넣습니다.

▼ 그림 7-10 오라클 드라이버 설정



오라클 드라이버는 아래 링크를 클릭해 다운로드할 수 있습니다.

- <https://www.oracle.com/technetwork/apps-tech/jdbc-112010-090769.html>

8. sec01.ex01 패키지를 만들고 다음과 같이 회원 조회와 관련된 자바 클래스 파일인 MemberDAO, MemberServlet, MemberVO 클래스를 각각 생성합니다.

❖ 그림 7-11 실습 파일 위치



9. 브라우저의 요청을 받는 MemberServlet 클래스를 다음과 같이 작성합니다.

코드 7-2 pro07/src/sec01/ex01/MemberServlet.java

```
package sec01.ex01;
...
@WebServlet("/member") // 주석 해제
protected class MemberServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        MemberDAO dao = new MemberDAO(); ────────── SQL문으로 조회할 MemberDAO 객체를 생성합니다.
        List<MemberVO> list = dao.listMembers(); ────────── listMembers() 메서드로 회원 정보를 조회
        합니다.

        out.print("<html><body>");
        out.print("<table border=1><tr align='center' bgcolor='lightgreen'>");
        out.print("<td>아이디</td><td>비밀번호</td><td>이름</td><td>이메일</td>
                 <td>가입일</td></tr>");

        for (int i = 0; i < list.size(); i++)
        {
            MemberVO memberVO = list.get(i);
            String id = memberVO.getId();
            String pwd = memberVO.getPwd();
            String name = memberVO.getName();
            String email = memberVO.getEmail();
            Date joinDate = memberVO.getJoinDate();
            out.print("<tr><td>" + id + "</td><td>" + pwd + "</td><td>" +
                     + name + "</td><td>" + email + "</td></tr>")
        }
    }
}
```

조회한 회원 정보를 for문과 <tr> 태그를 이용해 리스트로 출력합니다.

```
        + joinDate + "</td></tr>");  
    }  
    out.print("</table></body></html>");  
}  
}
```

10. MemberDAO 클래스를 다음과 같이 작성합니다. 회원 정보 조회 SQL문을 실행하여 조회한 레코드들의 컬럼 값을 다시 MemberVO 객체의 속성에 설정한 다음 ArrayList에 저장하고 호출할 곳으로 반환합니다.

코드 7-3 pro07/src/sec01/ex01/MemberDAO.java

```

    } catch (Exception e)
    {
        e.printStackTrace();
    }
    return list; •———— 조회한 레코드의 개수만큼 MemberVO
} 객체를 저장한 ArrayList를 반환합니다.

private void connDB()
{
    try
    {
        Class.forName(driver);
        System.out.println("Oracle 드라이버 로딩 성공");
        con = DriverManager.getConnection(url, user, pwd);
        System.out.println("Connection 생성 성공");
        stmt = con.createStatement();
        System.out.println("Statement 생성 성공");
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}
}

```

11. MemberVO 클래스를 다음과 같이 작성합니다. 이는 값을 전달하는 데 사용되는 VO(Value Object) 클래스입니다. 테이블에서 조회한 레코드의 컬럼 값을 속성에 저장해야 하므로 컬럼 이름과 동일한 자료형과 이름으로 속성을 선언하고 getter/setter를 각각 생성합니다.

코드 7-4 pro07/src/sec01/ex01/MemberVO.java

```

package sec01.ex01;

import java.sql.Date;

public class MemberVO
{
    private String id;
    private String pwd;
    private String name;
    private String email;
    private Date joinDate; •———— t_member 테이블의 컬럼 이름과 동일한 자료형과
                                이름으로 속성들을 선언합니다.

    public MemberVO()
    {
    }
}

```

```
    System.out.println("MemberVO 생성자 호출");
}

public String getId()
{
    return id;
}

public void setId(String id)
{
    this.id = id;
}

public String getPwd()
{
    return pwd;
}

public void setPwd(String pwd)
{
    this.pwd = pwd;
}

public String getName()
{
    return name;
}

public void setName(String name)
{
    this.name = name;
}

public String getEmail()
{
    return email;
}

public void setEmail(String email)
{
    this.email = email;
}

public Date getJoinDate()
{
```

getter/setter를 생성합니다.

```

        return joinDate;
    }

    public void setJoinDate(Date joinDate)
    {
        this.joinDate = joinDate;
    }

}

```

12. <http://localhost:8090/pro07/member>로 요청하여 실행 결과를 확인합니다. 회원 정보가 웹 브라우저로 출력되는 것을 확인할 수 있습니다.

▼ 그림 7-12 브라우저로 회원 정보 출력



The screenshot shows a web browser window with the URL localhost:8090/pro07/member. The page displays a table with five columns: 아이디, 비밀번호, 이름, 이메일, and 가입일. The data in the table is as follows:

아이디	비밀번호	이름	이메일	가입일
hong	1212	홍길동	hong@gmail.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
kim	1212	김유신	kim@jweb.com	2018-09-04

7.2.2 PreparedStatement를 이용한 회원 정보 실습

앞 절에서는 회원 정보를 조회하기 위해 MemberDAO에서 Statement 인터페이스를 이용하여 데이터베이스와 연동했습니다. 그런데 Statement를 이용해서 데이터베이스와 연동할 경우에는 연동할 때마다 DBMS에서 다시 SQL문을 컴파일해야 하므로 속도가 느리다는 단점이 있습니다.

이럴 경우 PreparedStatement 인터페이스를 사용하면 SQL문을 미리 컴파일해서 재사용하므로 Statement 인터페이스보다 훨씬 빠르게 데이터베이스 작업을 수행할 수 있습니다. 따라서 데이터베이스와 연동할 때 또는 빠른 반복 처리가 필요할 때는 PreparedStatement 인터페이스를 사용해야 합니다.

PreparedStatement 인터페이스의 특징은 다음과 같습니다.

- PreparedStatement 인터페이스는 Statement 인터페이스를 상속하므로 지금까지 사용한 메서드를 그대로 사용합니다.
- Statement 인터페이스가 DBMS에 전달하는 SQL문은 단순한 문자열이므로 DBMS는 이 문자열을 DBMS가 이해할 수 있도록 컴파일하고 실행합니다. 반면에 PreparedStatement 인터페이스는 컴파일된 SQL문을 DBMS에 전달하여 성능을 향상시킵니다.

- PreparedStatement 인터페이스에서는 실행하려는 SQL문에 '?'를 넣을 수 있습니다. 따라서 '?'의 값만 바꾸어 손쉽게 설정할 수 있어 Statement보다 SQL문 작성하기가 더 간단합니다.

그럼 지금부터 PreparedStatement를 이용해 회원 정보를 조회하는 예제를 실습해 보겠습니다.

- sec01.ex02 패키지를 만든 후 MemberServlet.java와 MemberVO.java는 기존의 것을 복사하여 붙여 넣습니다.

▼ 그림 7-13 실습 파일 위치



Caution MemberServlet 클래스를 붙여 넣었기 때문에 앞서 만든 sec01.ex01 패키지에 있는 MemberServlet 클래스와 서블릿 매핑 이름이 중복됩니다. 매핑 이름이 중복되면 오류가 발생하므로 sec01.ex01의 것을 주석 처리해야 합니다.

- PreparedStatement를 이용해 데이터베이스와 연동하는 MemberDAO 클래스를 작성합니다.

코드 7-5 pro07/src/sec01/ex02/MemberDAO.java

```

package sec01.ex02;
...
public class MemberDAO
{
    private PreparedStatement pstmt;
    private Connection con;
    ...
    public List<MemberVO> listMembers()
    {
        List<MemberVO> list = new ArrayList<MemberVO>();
        try
        {
            connDB();
            String query = "select * from t_member ";
            System.out.println("prepareStatement: " + query);
        }
    }
}
  
```

```

        pstmt = con.prepareStatement(query); •————— prepareStatement() 메서드에 SQL문을 전달해서
        ResultSet rs = pstmt.executeQuery(); PreparedStatement 객체를 생성합니다.

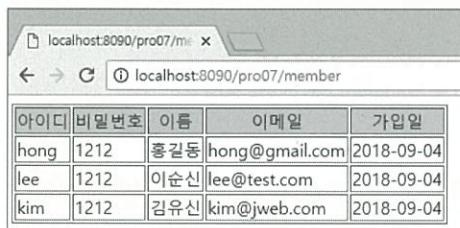
        while (rs.next())
        {
            String id = rs.getString("id");
            String pwd = rs.getString("pwd");
            String name = rs.getString("name");
            String email = rs.getString("email");
            Date joinDate = rs.getDate("joinDate");
            MemberVO vo = new MemberVO();
            vo.setId(id);
            vo.setPwd(pwd);
            vo.setName(name);
            vo.setEmail(email);
            vo.setJoinDate(joinDate);
            list.add(vo);
        }
        rs.close();
        pstmt.close();
        con.close();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    return list;
}

private void connDB()
{
    try
    {
        Class.forName(driver);
        System.out.println("Oracle 드라이버 로딩 성공");
        con = DriverManager.getConnection(url, user, pwd);
        System.out.println("Connection 생성 성공");
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

3. `http://localhost:8090/pro07/member`로 요청해서 실행 결과를 확인합니다. 눈으로 보면 `Statement`을 사용했을 때와 결과는 같습니다. 하지만 데이터베이스와 연동할 경우 수행 속도가 좀 더 빠르다는 차이가 있습니다.

▼ 그림 7-14 `PreparedStatement` 이용해 회원 정보를 조회한 결과



A screenshot of a web browser window titled "localhost:8090/pro07/member". The table has columns: 아이디, 비밀번호, 이름, 이메일, 가입일. The data is as follows:

아이디	비밀번호	이름	이메일	가입일
hong	1212	홍길동	hong@gmail.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
kim	1212	김유신	kim@jweb.com	2018-09-04

7.3 DataSource 이용해 데이터베이스 연동하기

JAVA WEB

앞 절에서는 회원 테이블에서 회원 정보를 조회하는 과정을 실습해 봤습니다. 이러한 데이터베이스 연동 과정은 웹 애플리케이션이 필요할 때마다 데이터베이스에 연결하여 작업하는 방식입니다. 그런데 이런 식으로 필요할 때마다 연동해서 작업할 경우 발생하는 문제가 하나 있습니다. 바로 데이터베이스 연결에 시간이 많이 걸린다는 것입니다.

특히 온라인 쇼핑몰의 경우 동시에 수십 명, 많게는 수백 명까지 접속해서 상품 조회, 주문하기 등의 기능을 사용하는데 앞의 방법처럼 데이터베이스와 연동해 작업해야 한다면 너무 비효율적입니다.

이 문제를 해결하기 위해 현재는 웹 애플리케이션이 실행됨과 동시에 연동할 데이터베이스와의 연결을 미리 설정해 둡니다. 그리고 필요할 때마다 미리 연결해 놓은 상태를 이용해 빠르게 데이터베이스와 연동하여 작업을 합니다. 이렇게 미리 데이터베이스와 연결시킨 상태를 유지하는 기술을 **커넥션풀(ConnectionPool)**이라고 부릅니다.

▼ 그림 7-15 ConnectionPool 등장 배경

기존 데이터베이스 연동 방법의 문제점

- 애플리케이션에서 데이터베이스에 연결하는 과정에서 시간이 많이 걸립니다.



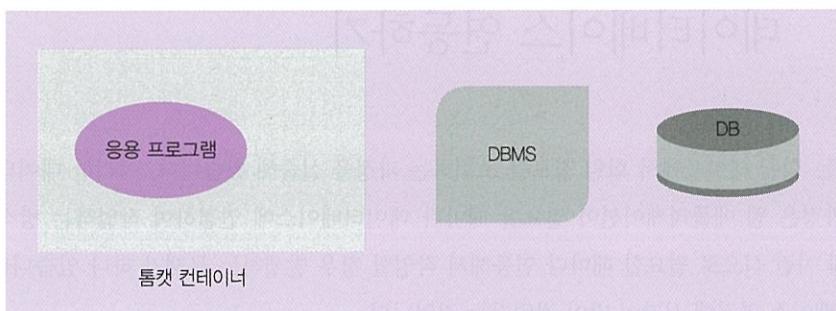
- 애플리케이션 실행 시 미리 ConnectionPool 객체를 생성한 후 데이터베이스와 연결을 맺습니다.
- 애플리케이션은 데이터베이스 연동 작업 발생 시 이 ConnectionPool 객체를 이용해서 작업을 합니다.

7.3.1 커넥션풀 동작 과정

톰캣 컨테이너에서 제공하는 커넥션풀의 동작 과정을 다음 그림을 통해 살펴보겠습니다.

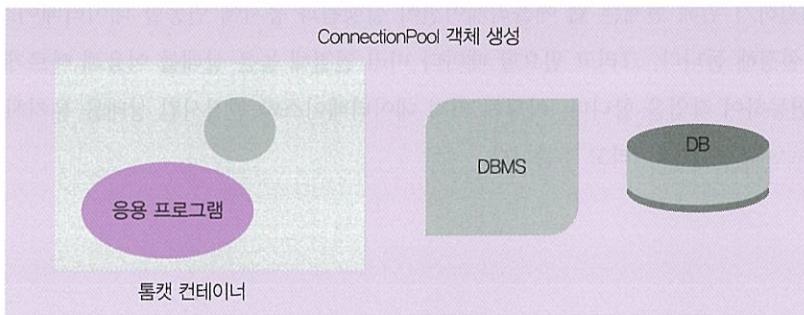
- 톰캣 컨테이너를 실행한 후 응용 프로그램을 실행합니다.

▼ 그림 7-16 톰캣 실행 후 응용 프로그램 실행



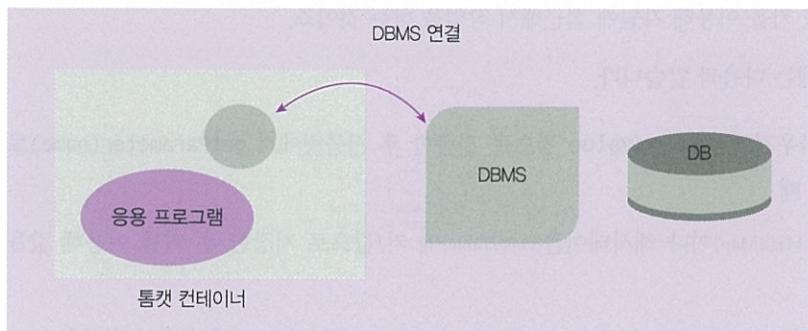
- 톰캣 컨테이너 실행 시 ConnectionPool 객체를 생성합니다.

▼ 그림 7-17 톰캣 실행 시 ConnectionPool 객체 생성



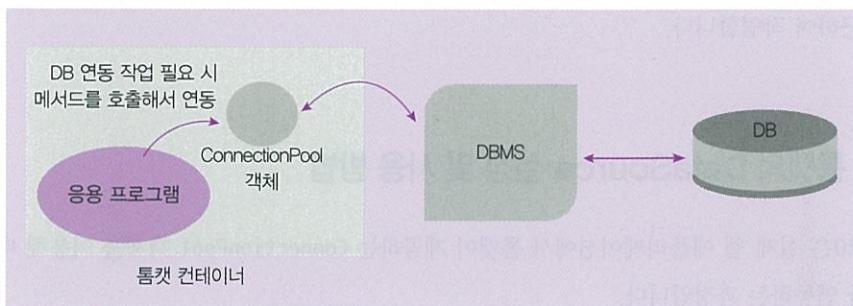
3. 생성된 커넥션 객체는 DBMS와 연결합니다.

▼ 그림 7-18 ConnectionPool 객체와 데이터베이스 연결



4. 데이터베이스와의 연동 작업이 필요할 경우 응용 프로그램은 ConnectionPool에서 제공하는 메서드를 호출하여 연동합니다.

▼ 그림 7-19 응용 프로그램에서 ConnectionPool 객체를 이용해 데이터베이스 연동



톰캣 컨테이너는 자체적으로 ConnectionPool 기능을 제공합니다. 톰캣 실행 시 톰캣은 설정 파일에 설정된 데이터베이스 정보를 이용해 미리 데이터베이스와 연결하여 ConnectionPool 객체를 생성한 후 애플리케이션이 데이터베이스와 연동할 일이 생기면 ConnectionPool 객체의 메서드를 호출해 빠르게 연동하여 작업합니다.

7.3.2 JNDI

실제 웹 애플리케이션에서 ConnectionPool 객체를 구현할 때는 Java SE에서 제공하는 `javax.sql.DataSource` 클래스를 이용합니다. 그리고 웹 애플리케이션 실행 시 톰캣이 만들어 놓은 ConnectionPool 객체에 접근할 때는 JNDI를 이용합니다.

JNDI(Java Naming and Directory Interface)란 필요한 자원을 키/값(key/value) 쌍으로 저장한 후 필요할 때 키를 이용해 값을 얻는 방법입니다. 즉, 미리 접근할 자원에 키를 지정한 후 애플리케이션이 실행 중일 때 이 키를 이용해 자원에 접근해서 작업을 하는 것이죠.

JNDI 사용 예는 다음과 같습니다.

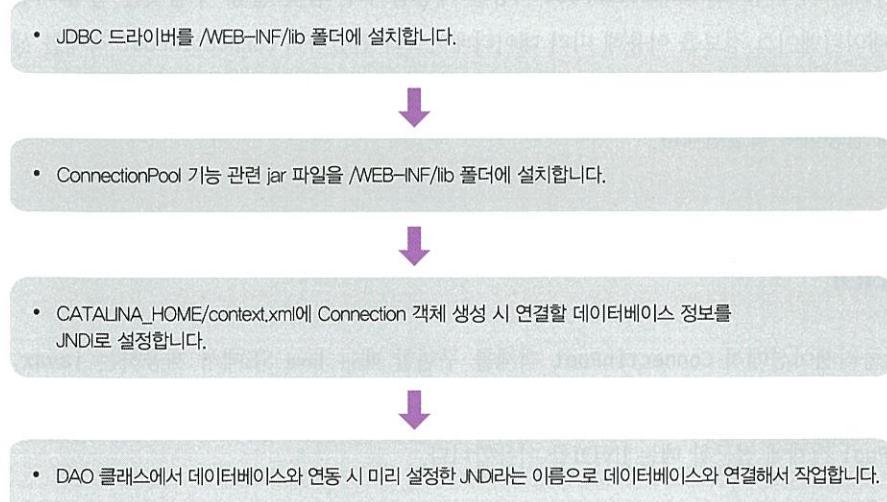
- 웹 브라우저에서 name/value 쌍으로 전송한 후 서블릿에서 getParameter(name)로 값을 가져올 때
- 해시맵(HashMap)이나 해시테이블(HashTable)에 키/값으로 저장한 후 키를 이용해 값을 가져올 때
- 웹 브라우저에서 도메인 네임으로 DNS 서버에 요청할 경우 도메인 네임에 대한 IP 주소를 가져올 때

톰캣 컨테이너가 ConnectionPool 객체를 생성하면 이 객체에 대한 JNDI 이름(key)을 미리 설정해 놓습니다. 그러면 웹 애플리케이션에서 데이터베이스와 연동 작업을 할 때 이 JNDI 이름(key)으로 접근하여 작업합니다.

7.3.3 톰캣의 DataSource 설정 및 사용 방법

그림 7-20은 실제 웹 애플리케이션에서 톰캣이 제공하는 ConnectionPool 객체를 이용해 데이터베이스와 연동하는 과정입니다.

▼ 그림 7-20 톰캣 ConnectionPool 설정 과정



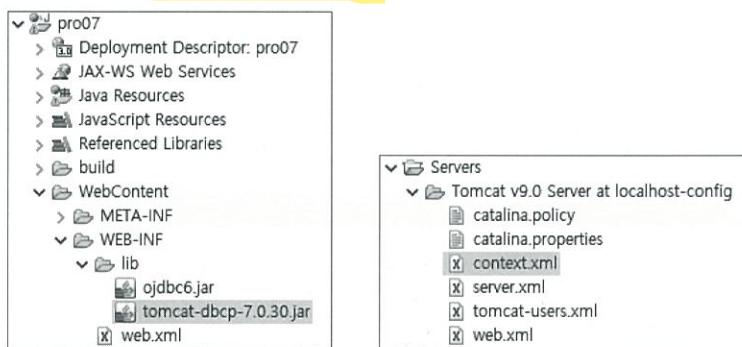
실제 톰캣에서 ConnectionPool 기능을 사용하려면 이 기능을 제공하는 DBCP 라이브러리를 따로 내려 받아야 합니다. 이 라이브러리 파일은 jar 압축 파일 형태로 제공되며, 다음 링크에서 tomcat-dbcp-7.0.30.zip 파일을 내려 받은 후 압축을 풀면 됩니다.

- <http://www.java2s.com/Code/Jar/t/Downloadtomcatdbcp7030jar.htm>

7.3.4 이클립스에서 톰캣 DataSource 설정

그림 7-21을 보면 JDBC 드라이버와 ConnectionPool 관련 jar 파일 및 이클립스에서 생성한 톰캣 서버의 설정 파일인 context.xml의 위치를 알 수 있습니다.

▼ 그림 7-21 ConnectionPool 관련 라이브러리와 context.xml 파일 위치



context.xml 파일을 보면 <Resource> 태그를 이용해 톰캣 실행 시 연결할 데이터베이스를 설정하는 것을 알 수 있습니다.

▼ 그림 7-22 Resource 설정

```

23   <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>
24
25   <!-- Uncomment this to disable session persistence across Tomcat restarts -->
26  <!--
27   <Manager pathname="" />
28   -->
29   <Resource
30     name="jdbc/oracle"
31     auth="Container"
32     type="javax.sql.DataSource"
33     driverClassName="oracle.jdbc.OracleDriver"
34     url="jdbc:oracle:thin:@localhost:1521:XE"
35     username="scott"
36     password="tiger"
37     maxActive="50"

```

The screenshot shows the context.xml file in a code editor. Line 30, which defines a database resource named 'jdbc/oracle', is highlighted with a purple box. The entire file content is as follows:

```

<WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>
<!-- Uncomment this to disable session persistence across Tomcat restarts -->
<!--
<Manager pathname="" />
-->
<Resource
  name="jdbc/oracle"
  auth="Container"
  type="javax.sql.DataSource"
  driverClassName="oracle.jdbc.OracleDriver"
  url="jdbc:oracle:thin:@localhost:1521:XE"
  username="scott"
  password="tiger"
  maxActive="50"

```

자바 클래스에서는 다음과 같이 name 속성의 jdbc/oracle로 DataSource에 접근합니다.

코드 7-6 톰캣 context.xml에 Resource를 설정하는 방법

```
<Resource  
    name="jdbc/oracle" → name의 jdbc/oracle로 DataSource에 접근합니다.  
    auth = "Container"  
    type="javax.sql.DataSource"  
    driverClassName="oracle.jdbc.OracleDriver"  
    url="jdbc:oracle:thin:@localhost:1521:XE"  
    username="scott" → 데이터베이스를 연결하는 데 필요한  
    password="tiger"   네 가지 값을 설정합니다.  
    maxActive="50"  
    maxWait="-1" />
```

오라클 데이터베이스(정확히는 오라클 DBMS)를 연결할 때 다른 속성들은 고정적으로 사용하며, 프로그래머가 주로 설정하는 정보는 `driverClassName`, `user`, `password`, `url`만 변경해서 설정합니다. 각 속성에 대한 자세한 설명은 표 7-2를 참고하기 바랍니다.

▼ 표 7-2 ConnectionPool로 연결할 데이터베이스 속성

속성	설명
<code>name</code>	DataSource에 대한 JNDI 이름
<code>auth</code>	인증 주체
<code>driverClassName</code>	연결할 데이터베이스 종류에 따른 드라이버 클래스 이름
<code>factory</code>	연결할 데이터베이스 종류에 따른 ConnectionPool 생성 클래스 이름
<code>maxActive</code>	동시에 최대로 데이터베이스에 연결할 수 있는 Connection 수
<code>maxIdle</code>	동시에 idle 상태로 대기할 수 있는 최대 수
<code>maxWait</code>	새로운 연결이 생길 때까지 기다릴 수 있는 최대 시간
<code>user</code>	데이터베이스 접속 ID
<code>password</code>	데이터베이스 접속 비밀번호
<code>type</code>	데이터베이스 종별 DataSource
<code>url</code>	접속할 데이터베이스 주소와 포트 번호 및 SID

7.3.5 톰캣의 DataSource로 연동해 회원 정보 조회 실습

이제 설정을 마쳤으니 클래스를 만들어 연동해 보겠습니다.

- sec02.ex01 패키지를 만들고 앞에서 사용한 MemberDAO, MemberServlet, MemberVO 클래스를 복사하여 붙여 넣습니다.

▼ 그림 7-23 실습 파일 위치



- 복사한 MemberServlet 클래스의 서블릿 매핑 이름을 /member2로 변경합니다.

▼ 그림 7-24 복사한 서블릿 클래스 매핑 이름을 /member2로 변경

```

14
15 @WebServlet("/member2")
16 public class MemberServlet extends HttpServlet {
17     public void doGet(HttpServletRequest request, Ht
18
19     response.setContentType("text/html; charset=ut
20     PrintWriter out=response.getWriter();
21     MemberDAO dao=new MemberDAO();
22     List list=dao.listMembers();

```

- DataSource를 이용해 데이터베이스와 연동하는 MemberDAO 클래스를 다음과 같이 수정합니다. 먼저 앞 절에서 데이터베이스와 연동할 때 사용한 connDB() 메서드는 주석 처리합니다. 그리고 생성자에서 톰캣 실행 시 톰캣에서 미리 생성한 DataSource를 name 값인 jdbc/oracle을 이용해 미리 받아옵니다. 마지막으로 서블릿에서 listMembers() 메서드를 호출하면 getConnection() 메서드를 호출하여 DataSource에 접근한 후 데이터베이스와의 연동 작업을 수행합니다.

코드 7-7 pro07/src/sec02/ex01/MemberDAO.java

```

package sec02.ex01;
...
public class MemberDAO
{

```

```

/*
private static final String driver = "oracle.jdbc.driver.OracleDriver";
private static final String url = "jdbc:oracle:thin:@localhost:1521:XE";
private static final String user = "scott";
private static final String pwd = "tiger";
*/

```

더 이상 사용되지 않으므로 주석 처리합니다.

```

private Connection con;
private PreparedStatement pstmt;
private DataSource dataFactory;

public MemberDAO()
{
    try
    {
        Context ctx = new InitialContext();
        Context envContext = (Context) ctx.lookup("java:/comp/env");
        dataFactory = (DataSource) envContext.lookup("jdbc/oracle");
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

public List<MemberVO> listMembers()
{
    List<MemberVO> list = new ArrayList<MemberVO>();
    try
    {
        // connDB();
        con = dataFactory.getConnection(); ← DataSource를 이용해 데이터베이스에
        String query = "select * from t_member "; ← 연결합니다.
        System.out.println("prepareStatement: " + query);
        pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next())
        {
            String id = rs.getString("id");
            String pwd = rs.getString("pwd");
            String name = rs.getString("name");
            String email = rs.getString("email");
            Date joinDate = rs.getDate("joinDate");
            MemberVO vo = new MemberVO();
            vo.setId(id);
            vo.setPwd(pwd);

```

JNDI에 접근하기 위해 기본 경로
(java:/comp/env)를 지정합니다.

톰캣 context.xml에 설정한
name 값인 jdbc/oracle을
이용해 톰캣이 미리 연결한
DataSource를 받아옵니다.

```

        vo.setName(name);
        vo.setEmail(email);
        vo.setJoinDate(joinDate);
        list.add(vo);
    }
    rs.close();
    pstmt.close();
    con.close();
} catch (Exception e)
{
    e.printStackTrace();
}
return list;
}

/*
 * DAO에서 직접 데이터베이스를 연결하는 기능은 주석 처리합니다.
 * private void connDB() {
    ...
}
*/
}

```

4. <http://localhost:8090/pro07/member2>로 요청합니다. 결과는 앞에서 실습했을 때와 같지만 이번에는 커넥션풀을 이용해서 데이터베이스와 연동했다는 점에서 차이가 있습니다.

▼ 그림 7-25 ConnectionPool로 회원 정보를 조회한 결과



The screenshot shows a web browser window with the URL localhost:8090/pro07/member2. The page displays a table of member data:

아이디	비밀번호	이름	이메일	가입일
hong	1212	홍길동	hong@gmail.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
kim	1212	김유신	kim@jweb.com	2018-09-04

7.4

DataSource 이용해 회원 정보 등록하기



이번에는 커넥션풀을 이용해 새 회원을 등록해 보겠습니다.

- sec02.ex02 패키지를 만들고 MemberVO.java를 복사하여 붙여 넣습니다.

▼ 그림 7-26 실습 파일 위치



- 회원 가입창을 작성하기 위해 다음과 같이 memberForm.html을 작성합니다. <hidden> 태그를 이용해 회원 가입창에서 새 회원 등록 요청을 서블릿에 전달합니다.

코드 7-8 pro07/WebContent/memberForm.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>회원 가입창</title>
    <script type="text/javascript">
        function fn_sendMember() {
            var frmMember = document.frmMember;
            var id = frmMember.id.value;
            var pwd = frmMember.pwd.value;
            var name = frmMember.name.value;
            var email = frmMember.email.value;
```

자바스크립트에서 <form> 태그의 name으로 접근해 입력한 값들을 얻습니다.

```

if (id.length == 0 || id == "") {
    alert("아이디는 필수입니다.");
} else if (pwd.length == 0 || pwd == "") {
    alert("비밀번호는 필수입니다.");
}
else if (name.length == 0 || name == "") {
    alert("이름은 필수입니다.");
} else if (email.length == 0 || email == "") {
    alert("이메일은 필수입니다.");
} else {
    frmMember.method = "post";           ← 전송 방법을 post로 지정합니다.
    frmMember.action = "member3";        ← 서블릿 매핑 이름을 member3으로 지정합니다.
    frmMember.submit();                 ← 서블릿으로 전송합니다.
}
}
</script>
</head>
<body>
<form name="frmMember">
    <table>
        <th>회원 가입창</th>
        <tr>
            <td>아이디</td>
            <td><input type="text" name="id"></td>           ← 입력한 ID를 서블릿으로 전송합니다.
        </tr>
        <tr>
            <td>비밀번호</td>
            <td><input type="password" name="pwd"></td>           ← 입력한 비밀번호를 서블릿으로 전송합니다.
        </tr>
        <tr>
            <td>이름</td>
            <td><input type="text" name="name"></td>           ← 입력한 이름을 서블릿으로 전송합니다.
        </tr>
        <tr>
            <td>이메일</td>
            <td><input type="text" name="email"></td>           ← 입력한 이메일을 서블릿으로 전송합니다.
        </tr>
    </table>
    <input type="button" value="가입하기" onclick="fn_sendMember()">
    <input type="reset" value="다시 입력">
    <input type="hidden" name="command" value="addMember" />           ← <hidden> 태그를 이용해
                                                                서블릿에게 회원 등록임
                                                                을 알립니다.
</form>
</body>
</html>

```

3. MemberServlet 클래스를 다음과 같이 작성합니다. command 값을 먼저 받아 와 addMember 이면 같이 전송된 회원 정보를 받아 옵니다. 회원 정보를 MemberVO 객체에 설정한 후 MemberDAO의 메서드로 전달해 SQL문을 이용하여 테이블에 추가합니다.

코드 7-9 pro07/sec02/ex02/MemberServlet.java

```
package sec02.ex02;  
...  
@WebServlet("/member3")  
public class MemberServlet extends HttpServlet  
{  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException  
    {  
        doHandle(request, response);  
    }  
  
    protected void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException  
    {  
        doHandle(request, response);  
    }  
  
    private void doHandle(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException  
    {  
        request.setCharacterEncoding("utf-8");  
        response.setContentType("text/html; charset=utf-8");  
        MemberDAO dao = new MemberDAO();  
        PrintWriter out = response.getWriter();  
        String command = request.getParameter("command"); ────────── command 값을 받아 옵니다.  
        if (command != null && command.equals("addMember")) ────────── 회원 가입창에서 전송된  
        {  
            String _id = request.getParameter("id");  
            String _pwd = request.getParameter("pwd");  
            String _name = request.getParameter("name");  
            String _email = request.getParameter("email");  
            MemberVO vo = new MemberVO();  
            vo.setId(_id);  
            vo.setPwd(_pwd);  
            vo.setName(_name);  
            vo.setEmail(_email);  
            dao.addMember(vo);  
        }  
    }  
}
```

```

} else if (command != null && command.equals("delMember"))
{
    String id = request.getParameter("id");
    dao.delMember(id);
}
List<MemberVO> list = dao.listMembers();
out.print("<html><body>");
out.print("<table border=1><tr align='center' bgcolor='lightgreen'>");
out.print("<td>아이디</td><td>비밀번호</td><td>이름</td><td>이메일</td><td>가입일"
</td><td >삭제</td></tr>");

for (int i = 0; i < list.size(); i++)
{
    MemberVO memberVO = (MemberVO) list.get(i);
    String id = memberVO.getId();
    String pwd = memberVO.getPwd();
    String name = memberVO.getName();
    String email = memberVO.getEmail();
    Date joinDate = memberVO.getJoinDate();
    out.print("<tr><td>" + id + "</td><td>" + pwd + "</td><td>"
+ name + "</td><td>" + email + "</td><td>" + joinDate
+ "</td><td>" + "<a href='/pro07/member3?command=delMember&id="
+ id + "'> 삭제 </a></td></tr>");
}
out.print("</table></body></html>");
out.print("<a href='/pro07/memberForm.html'>새 회원 등록하기 </a>");
```

클릭하면 다시 회원 가입창으로
이동합니다.

Note 三 PrepareStatement에서 insert문 사용하는 방법

- ❶ PreparedStatement의 insert문은 회원 정보를 저장하기 위해 ?(물음표)를 사용합니다.
- ❷ ?는 id, pwd, name, age에 순서대로 대응합니다.
- ❸ 각 ?에 대응하는 값을 지정하기 위해 PreparedStatement의 setter를 이용합니다.
- ❹ setter() 메서드의 첫 번째 인자는 '?'의 순서를 지정합니다.
- ❺ ?은 1부터 시작합니다.
- ❻ insert, delete, update문은 executeUpdate() 메서드를 호출합니다.

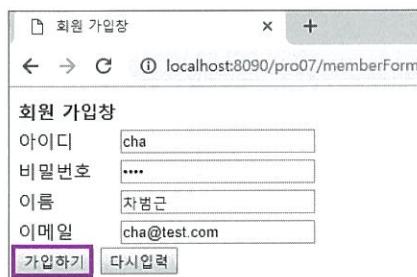
4. MemberDAO 클래스를 다음과 같이 수정합니다

코드 7-10 pro07/sec02/ex02/MemberDAO.java

```
...  
public void addMember(MemberVO memberVO)  
{  
    try  
    {  
        con = dataFactory.getConnection(); •————— DataSource를 이용해 데이터베이스와 연결합니다.  
        String id = memberVO.getId();  
        String pwd = memberVO.getPwd();  
        String name = memberVO.getName();  
        String email = memberVO.getEmail();  
  
        String query = "insert into t_member"; •————— insert문을 문자열로 만듭니다.  
        query += " (id,pwd,name,email)";  
        query += " values(?, ?, ?, ?)";  
        System.out.println("prepareStatement: " + query);  
        pstmt = con.PreparedStatement(query);  
        pstmt.setString(1, id); •————— insert문의 각 '?'에 순서대로 회원 정보를 세팅합니다.  
        pstmt.setString(2, pwd);  
        pstmt.setString(3, name);  
        pstmt.setString(4, email);  
        pstmt.executeUpdate(); •————— 회원 정보를 테이블에 추가합니다.  
        pstmt.close();  
    } catch (Exception e)  
    {  
        e.printStackTrace();  
    }  
}
```

5. <http://localhost:8090/pro07/memberForm.html>로 요청하여 회원 정보를 입력한 후 가입하기를 클릭합니다.

▼ 그림 7-27 회원 정보 입력 후 가입하기 클릭



6. 다음과 같이 회원 정보가 출력됩니다.

▼ 그림 7-28 회원 등록 실행 결과

아이디	비밀번호	이름	이메일	가입일
hong	1212	홍길동	hong@gmail.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
kim	1212	김유신	kim@jweb.com	2018-09-04
cha	1234	차범근	cha@test.com	2018-09-04
새 회원 등록하기				

7.5 회원 정보 삭제하기

JAVA WEB

이번에는 회원 정보를 삭제하는 기능을 구현해 보겠습니다.

1. MemberServlet 클래스를 다음과 같이 수정합니다. <a> 태그를 이용해 회원 정보를 삭제할 수 있는 링크를 추가합니다. 브라우저에서 삭제 요청도 전송하므로 if문에 else if문을 추가하여 삭제 기능을 수행합니다.

코드 7-11 pro07/src/sec02/ex02/MemberServlet3.java

```

...
private void doHandle(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html;charset=utf-8");
    MemberDAO dao = new MemberDAO();
    PrintWriter out = response.getWriter();
    String command = request.getParameter("command");
    if (command != null && command.equals("addMember"))
    {
        ...
    }
    else if (command != null && command.equals("delMember"))
    {
        String id = request.getParameter("id");
        dao.delMember(id);
    }
}

```

command 값이 delMember인 경우 ID를 가져와 SQL문으로 전달해서 삭제합니다.

```

List<MemberVO> list = dao.listMembers();
out.print("<html><body>");
out.print("<table border=1><tr align='center' bgcolor='lightgreen'>");
out.print("<td>아이디</td><td>비밀번호</td><td>이름</td><td>나이</td><td>가입일</td><td>삭제</td></tr>");

for (int i = 0; i < list.size(); i++)
{
    MemberVO memberVO = (MemberVO) list.get(i);
    String id = memberVO.getId();
    String pwd = memberVO.getPwd();
    String name = memberVO.getName();
    int age = memberVO.getAge();
    Date joinDate = memberVO.getJoinDate();
    out.print("<tr><td>" + id + "</td><td>" + pwd + "</td><td>" + name + "</td><td>" + age + "</td><td>" + joinDate + "</td><td>" + "<a href='/pro07/member3?command=delMember&id=" + id + "'>삭제 </a></td></tr>");

}
out.print("</table></body></html>");
out.print("<a href='/pro07/memberForm.html'>새 회원 등록하기</a>");
}
}

```

2. MemberDAO 클래스를 다음과 같이 수정합니다. delete문의 첫 번째 '?'에 전달된 ID를 인자로 executeUpdate() 메서드를 호출합니다.

코드 7-12 pro07/src/sec02/ex02/MemberDAO.java

```

public void delMember(String id)
{
    try
    {
        con = dataFactory.getConnection();

        String query = "delete from t_member" + " where id=?"; ————— delete문을 문자열로 만듭니다.
        System.out.println("prepareStatement:" + query);
        pstmt = con.PreparedStatement(query);
        pstmt.setString(1, id); ————— 첫 번째 '?'에 전달된 ID를 인자로 넣습니다.
        pstmt.executeUpdate(); ————— delete문을 실행해 테이블에서 해당 ID의
        pstmt.close(); ————— 회원 정보를 삭제합니다.

    } catch (Exception e)
    {

```

```

        e.printStackTrace();
    }
}

```

3. <http://localhost:8090/member3>로 요청한 후 삭제를 클릭합니다.

▼ 그림 7-29 삭제를 클릭해 회원 정보 삭제

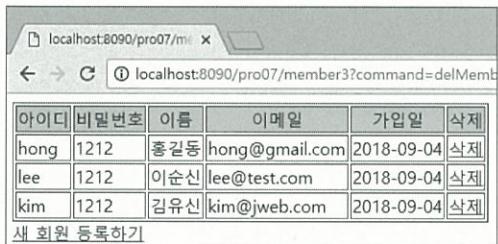


아이디	비밀번호	이름	이메일	가입일	삭제
hong	1212	홍길동	hong@gmail.com	2018-09-04	삭제
lee	1212	이순신	lee@test.com	2018-09-04	삭제
kim	1212	김유신	kim@jweb.com	2018-09-04	삭제
cha	1234	차범근	cha@test.com	2018-09-04	삭제

새 회원 등록하기

4. 회원 정보를 삭제한 후 남은 회원 정보가 다시 출력됩니다.

▼ 그림 7-30 삭제 후 회원 정보 출력



아이디	비밀번호	이름	이메일	가입일	삭제
hong	1212	홍길동	hong@gmail.com	2018-09-04	삭제
lee	1212	이순신	lee@test.com	2018-09-04	삭제
kim	1212	김유신	kim@jweb.com	2018-09-04	삭제

새 회원 등록하기

지금까지 서블릿의 세 가지 기능 중 마지막 기능인 비즈니스 로직 처리 작업을 알아보았습니다. 일반적으로 서블릿에서 데이터베이스와 연동하는 작업은 크게 CRUD(Create, Read, Update, Delete) 작업으로 나눌 수 있습니다. 회원 기능에서 CRUD 중 update 작업은 뒤에서 JSP를 배운 후에 적용해 보겠습니다.

Note ≡ 이클립스 디버깅 기능 사용하기

데이터베이스 연동 등 여러 기능이 추가되면 당연히 소스 코드의 양이 많아질 수밖에 없습니다. 따라서 일반적인 자바 문법 오류보다 실행 중 오류나 결괏값이 다르게 출력되는 논리 오류가 더 많이 발생합니다. 이런 오류를 소스를 보면서 직접 해결하려면 매우 불편하고 시간도 많이 걸립니다.

이때 이클립스의 디버깅 기능을 사용하면 빠르게 오류를 해결할 수 있습니다. 그럼 이클립스의 디버깅 기능을 사용해 보겠습니다.

1. sec02/ex02.MemberServlet 클래스의 doHandle() 메서드 28번 줄 번호 옆을 마우스로 더블클릭해 중단점 (breakpoint)을 만듭니다.

▼ 그림 7-31 소스에 중단점 설정

```
27~ private void doHandle(HttpServletRequest request, HttpServletResponse
28     request.setCharacterEncoding("utf-8");
29     response.setContentType("text/html; charset=utf-8");
30     MemberDAO dao=new MemberDAO();
31     PrintWriter out=response.getWriter();
32     String command=request.getParameter("command");
33     if(command!= null && command.equals("addMember")){
34         String _id=request.getParameter("id");
35         String _pwd=request.getParameter("pwd");
36         String _name=request.getParameter("name");
37         String _email=request.getParameter("email");
38
39         MemberVO vo=new MemberVO();
40         vo.setId(_id);
41         vo.setPwd(_pwd);
42         vo.setName(_name);
43         vo.setEmail(_email);
44         dao.addMember(vo);
45     }else if(command!= null && command.equals("delMember")){
46         String id = request.getParameter("id");
47     }
48 }
```

2. 톰캣 실행 시 버그(🐞) 아이콘을 클릭해 디버그 모드로 실행합니다.

▼ 그림 7-32 디버그 모드로 톰캣 실행



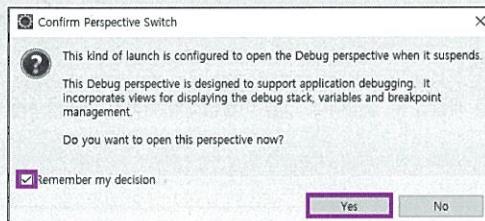
3. 회원 가입 페이지를 열어 새 회원 정보를 입력한 후 가입하기를 클릭합니다.

▼ 그림 7-33 디버깅을 위한 회원 정보 입력

아이디	park
비밀번호	****
이름	박찬호
이메일	park@test.com
<input type="button" value="가입하기"/> <input type="button" value="다시입력"/>	

4. 웹 브라우저의 요청을 받은 이클립스가 디버그 모드로 전환하기 위한 동의 요청창이 나타나면 Remember my decision 옵션 체크박스에 체크한 후 Yes를 클릭합니다.

▼ 그림 7-34 디버그 모드 전환 동의 요청창



5. 이클립스가 디버그 모드로 전환되고 실행은 중단점에서 정지합니다.

▼ 그림 7-35 웹 브라우저의 요청을 받은 후 중단점에서 실행이 중지된 상태

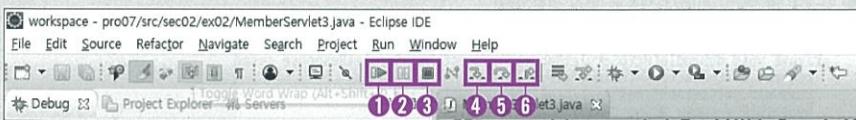
```

27 private void doHandle(HttpServletRequest request, HttpServletResponse response)
28     request.setCharacterEncoding("utf-8");
29     response.setContentType("text/html; charset=utf-8");
30     MemberDAO dao=new MemberDAO();
31     PrintWriter out=response.getWriter();
32     String command=request.getParameter("command");
33     if(command!= null && command.equals("addMember")){
34         String _id=request.getParameter("id");
35         String _pwd=request.getParameter("pwd");
36         String _name=request.getParameter("name");
37         String _email=request.getParameter("email");
38
39         MemberVO vo=new MemberVO();
40         vo.setId(_id);
41         vo.setPwd(_pwd);
42         vo.setName(_name);
43         vo.setEmail(_email);
44         dao.addMember(vo);

```

6. 이클립스 상단의 여러 가지 버튼을 이용해 디버깅을 수행합니다.

▼ 그림 7-36 디버깅 관련 기능을 하는 여러 가지 버튼들



- ① Resume: 다음 중단점을 만날 때까지 진행합니다([F8]).
- ② Suspend: 현재 동작하고 있는 스레드를 멈춥니다.
- ③ Terminate: 프로그램을 종료합니다([Ctrl] + [F2]).
- ④ Step Into: 메서드가 존재할 경우 그 메서드로 이동합니다([F5]).
- ⑤ Step Over: 한 라인씩 실행합니다([F6]).
- ⑥ Step Return: 'Step Into'로 이동한 메서드에서 원래 위치로 복귀합니다([F7]).

7. 가장 자주 사용하는 Step Over([]) 아이콘을 클릭해 중단점에서 다음 라인으로 이동합니다.

▼ 그림 7-37 Step Over 아이콘을 클릭해 한 라인씩 실행

```

18 public class MemberServlet extends HttpServlet {
19     protected void doGet(HttpServletRequest request, HttpServletResponse response)
20         doHandle(request, response);
21     }
22
23     protected void doPost(HttpServletRequest request, HttpServletResponse response)
24         doHandle(request, response);
25     }
26
27     private void doHandle(HttpServletRequest request, HttpServletResponse response)
28         request.setCharacterEncoding("utf-8");
29         response.setContentType("text/html; charset=utf-8");
30         MemberDAO dao=new MemberDAO();
31         PrintWriter out=response.getWriter();
32         String command=request.getParameter("command");

```

8. 계속해서 Step Over 아이콘을 클릭해 32행의 실행문을 실행한 후 변수 command 위에 마우스 포인터를 놓으면 command의 값을 팝업창으로 표시해 줍니다.

▼ 그림 7-38 Step Over 클릭해 실행한 후 변수 위에 마우스 포인터 올려 변수 값 확인

```

27 private void doHandle(HttpServletRequest request, HttpServletResponse response) {
28     request.setCharacterEncoding("utf-8");
29     response.setContentType("text/html;charset=utf-8");
30     MemberDAO dao=new MemberDAO();
31     PrintWriter out=response.getWriter();
32     String command=request.getParameter("command");
33     if(command!=null && command.equals("addMember")){
34         String id=request.getParameter("id");
35         String pwd=request.getParameter("pwd");
36         String name=request.getParameter("name");
37         String email=request.getParameter("email");
38         MemberVO vo=new MemberVO();
39         vo.setId(id);
40         vo.setPwd(pwd);
41         vo.setName(name);
42         vo.setEmail(email);
43     }
44 }

```

9. Step Over 아이콘을 계속 클릭하면 if문이 참이므로 회원 정보를 가져옵니다.

▼ 그림 7-39 웹 브라우저에서 전송된 회원 정보 확인

```

27 private void doHandle(HttpServletRequest request, HttpServletResponse response) {
28     request.setCharacterEncoding("utf-8");
29     response.setContentType("text/html;charset=utf-8");
30     MemberDAO dao=new MemberDAO();
31     PrintWriter out=response.getWriter();
32     String command=request.getParameter("command");
33     if(command!=null && command.equals("addMember")){
34         String id=request.getParameter("id");
35         String pwd=request.getParameter("pwd");
36         String name=request.getParameter("name");
37         String email=request.getParameter("email");
38         MemberVO vo=new MemberVO();
39         vo.setId(id);
40         vo.setPwd(pwd);
41         vo.setName(name);
42         vo.setEmail(email);
43         dao.addMember(vo);
44 }

```

10. 디버깅이 끝났으면 Resume(▶) 아이콘을 클릭해 다음 중단점으로 이동합니다(중단점은 여러 개를 지정할 수 있습니다). 중단점이 더 없으면 종료합니다.

▼ 그림 7-40 Resume 아이콘 클릭해 디버깅 종료

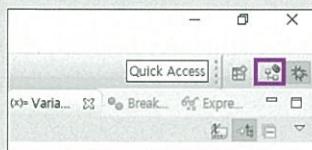
```

27 private void doHandle(HttpServletRequest request, HttpServletResponse response) {
28     request.setCharacterEncoding("utf-8");
29     response.setContentType("text/html;charset=utf-8");
30     MemberDAO dao=new MemberDAO();
31     PrintWriter out=response.getWriter();
32     String command=request.getParameter("command");
33     if(command!=null && command.equals("addMember")){
34         String id=request.getParameter("id");
35         String pwd=request.getParameter("pwd");
36         String name=request.getParameter("name");
37         String email=request.getParameter("email");
38         MemberVO vo=new MemberVO();
39         vo.setId(id);
40         vo.setPwd(pwd);
41         vo.setName(name);
42         vo.setEmail(email);
43     }
44 }

```

11. 이클립스를 다시 편집 모드로 되돌리기 위해 오른쪽 상단의 Java EE Perspective(Java EE) 아이콘을 클릭합니다.

▼ 그림 7-41 이클립스를 디버그 모드에서 편집 모드로 전환



12. 이클립스를 편집 모드로 전환합니다.

▼ 그림 7-42 편집 모드로 전환된 이클립스

```

private void doHandle(HttpServletRequest request, HttpServletResponse response) {
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html;charset=utf-8");
    MemberDAO dao=new MemberDAO();
    PrintWriter out=response.getWriter();
    String command=request.getParameter("command");
    if(command!= null && command.equals("addMember")){
        String _id=request.getParameter("id");
        String _pwd=request.getParameter("pwd");
        String _name=request.getParameter("name");
        String _email=request.getParameter("email");
        MemberVO vo=new MemberVO();
        vo.setId(_id);
        vo.setPwd(_pwd);
        vo.setName(_name);
        vo.setEmail(_email);
        dao.addMember(vo);
    }else if(command!= null && command.equals("delMember")){
        String id = request.getParameter("id");
        dao.delMember(id);
    }
    List list=dao.listMembers();
    out.print("<html><body>");
}

```

13. 정상적으로 회원이 등록된 것을 확인할 수 있습니다.

▼ 그림 7-43 디버그 모드에서 실행한 결과

아이디	비밀번호	이름	이메일	가입일	삭제
hong	1212	홍길동	hong@gmail.com	2018-09-04	삭제
lee	1212	이순신	lee@test.com	2018-09-04	삭제
kim	1212	김유신	kim@jweb.com	2018-09-04	삭제
park	1234	박찬호	park@test.com	2018-09-04	삭제

새 회원 등록하기

8장

서블릿 확장 API 사용하기

- 8.1 서블릿 포워드 기능 사용하기
- 8.2 서블릿의 여러 가지 포워드 방법
- 8.3 `dispatch`를 이용한 포워드 방법
- 8.4 바인딩
- 8.5 `ServletContext`와 `ServletConfig` 사용법
- 8.6 `load-on-startup` 기능 사용하기

8.1

서블릿 포워드 기능 사용하기



웹 프로그래밍 개발 초기에는 지금까지 배운 기본적인 서블릿 기능을 이용해 실제 웹 사이트의 기능을 구현했습니다. 즉, 서블릿 요청과 비즈니스 로직 처리 작업, 웹 브라우저의 화면 표시 응답 기능 등을 모두 사용했습니다.

이 장에서는 이 외에 서블릿 프로그래밍을 개발할 때 사용하는 기능인 포워드, 바인딩, 애너테이션 등 다양한 기능에 대해 살펴보겠습니다.

8.1.1 포워드 기능

실제 온라인 쇼핑몰 같은 웹 애플리케이션은 여러 기능을 합쳐 하나의 프로그램을 실행합니다. 회원 관리 기능, 게시판 관리 기능, 주문 관리 기능 등에 대해 각각의 서블릿이 기능을 수행하는 것이죠.

그런데 프로그램을 실행하다 보면 서블릿끼리 또는 서블릿과 JSP를 연동해서 작업해야 하는 경우가 있습니다.

예를 들어 쇼핑몰의 경우 상품 관리 서블릿과 조회된 상품을 화면에 표시하는 JSP는 각각 따로 존재합니다. 따라서 사용자가 상품 조회를 요청하면 상품 관리 서블릿은 데이터베이스에서 상품 정보를 조회한 후 다시 JSP에게 전달하여 상품 정보를 표시합니다.

이처럼 하나의 서블릿에서 다른 서블릿이나 JSP와 연동하는 방법을 **포워드(forward)**라고 합니다. 포워드 기능이 사용되는 용도는 여러 가지이며 요약하면 다음과 같습니다.

- 요청에 대한 추가 작업을 다른 서블릿에게 수행하게 합니다.
- 요청(request)에 포함된 정보를 다른 서블릿이나 JSP와 공유할 수 있습니다.
- 요청(request)에 정보를 포함시켜 다른 서블릿에 전달할 수 있습니다.
- 모델2 개발 시 서블릿에서 JSP로 데이터를 전달하는 데 사용됩니다.

한마디로 포워드 기능은 서블릿에서 다른 서블릿이나 JSP로 요청을 전달하는 역할을 합니다. 그리고 이 요청(request)을 전달할 때 추가 데이터를 포함시켜서 전달할 수도 있습니다. 모델2 개발 방식으로 웹 애플리케이션을 개발할 경우 서블릿에서 JSP로 데이터를 전달할 때 주로 사용됩니다(모델2 개발 방식은 17장에서 자세히 설명합니다).

8.2

서블릿의 여러 가지 포워드 방법

서블릿에서 사용되는 포워드 방법에는 다음 네 가지가 있습니다.

- redirect 방법

고객이 납득할 수 있다

- HttpServletResponse 객체의 `sendRedirect()` 메서드를 이용합니다.

- 웹 브라우저에 재요청하는 방식입니다.

- 형식: `sendRedirect("포워드할 서블릿 또는 JSP");`

- Refresh 방법

은행 주식 → 주가 절반 경

- HttpServletResponse 객체의 `addHeader()` 메서드를 이용합니다.

- 웹 브라우저에 재요청하는 방식입니다.

- 형식: `response.addHeader("Refresh", 경과시간(초);url=요청할 서블릿 또는 JSP");`

- location 방법

- 자바스크립트 `location` 객체의 `href` 속성을 이용합니다.

- 자바스크립트에서 재요청하는 방식입니다.

- 형식: `location.href='요청할 서블릿 또는 JSP';`

- dispatch 방법

고객이 알 수 X

- 일반적으로 포워딩 기능을 지칭합니다.

- 서블릿이 직접 요청하는 방법입니다.

- RequestDispatcher 클래스의 `forward()` 메서드를 이용합니다.

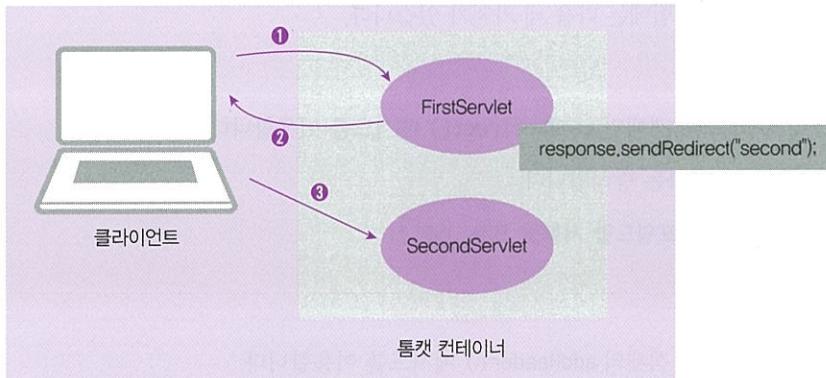
- 형식: `RequestDispatcher dis= request.getRequestDispatcher("포워드할 서블릿 또는 JSP");
dis.forward(request,response);`

`redirect`, `refresh`, `location` 방법은 서블릿이 웹 브라우저를 거쳐 다른 서블릿이나 JSP에게 요청하는 방법입니다. 반면에 `dispatch` 방법은 서블릿에서 클라이언트를 거치지 않고 바로 다른 서블릿에게 요청하는 방법입니다. 네 가지 모두 자주 사용하므로 각각의 사용법과 차이점을 익혀두는 것이 좋습니다.

8.2.1 redirect를 이용한 포워딩

redirect 방법은 서블릿의 요청이 클라이언트의 웹 브라우저를 다시 거쳐 요청되는 방식입니다.

▼ 그림 8-1 서블릿의 redirect 방법 수행 과정



- ① 클라이언트의 웹 브라우저에서 첫 번째 서블릿에 요청합니다.
- ② 첫 번째 서블릿은 `sendRedirect()` 메서드를 이용해 두 번째 서블릿을 웹 브라우저를 통해서 요청합니다.
- ③ 웹 브라우저는 `sendRedirect()` 메서드가 지정한 두 번째 서블릿을 다시 요청합니다.

8.2.2 redirect를 이용한 포워딩 실습

1. 새 프로젝트 pro08을 만들고 sec01.ex01 패키지를 추가합니다. FirstServlet 클래스와 SecondServlet 클래스를 추가합니다.

▼ 그림 8-2 실습 파일 위치



2. FirstServlet 클래스를 다음과 같이 작성합니다. redirect 기능을 구현한 서블릿입니다.

코드 8-1 pro08/src/sec01/ex01/FirstServlet.java

```
package sec01.ex01;
...
@WebServlet("/first")
public class FirstServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        response.sendRedirect("second");
    }
}
```

이 책에서 제공하는 예제 파일에는 매핑 이름의 중복을 피하기 위해 주석 처리되어 있으므로 주석 처리를 해제합니다.

sendRedirect() 메서드를 이용해 웹 브라우저에게 다른 서블릿인 second로 재요청합니다.

3. SecondServlet 클래스는 첫 번째 서블릿에서 요청을 받아 실행하는 두 번째 서블릿입니다.

코드 8-2 pro08/src/sec01/ex01/SecondServlet.java

```
package sec01.ex01;
...
@WebServlet("/second")
public class SecondServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("sendRedirect를 이용한 redirect 실습입니다.");
        out.println("</body></html>");
    }
}
```

제공하는 예제 파일에는 매핑 이름의 중복을 피하기 위해 주석 처리되어 있으므로 주석 처리를 해제합니다.

브라우저로 출력합니다.

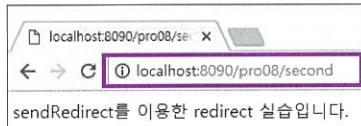
4. <http://localhost:8090/pro08/first>로 요청합니다.

▼ 그림 8-3 매핑 이름 /first로 요청



5. 최종적으로 웹 브라우저에 표시되는 매팅 이름은 /second입니다. 즉, /first로 요청하면 `sendRedirect()`를 호출해 웹 브라우저에게 다시 /second를 요청하는 것입니다.

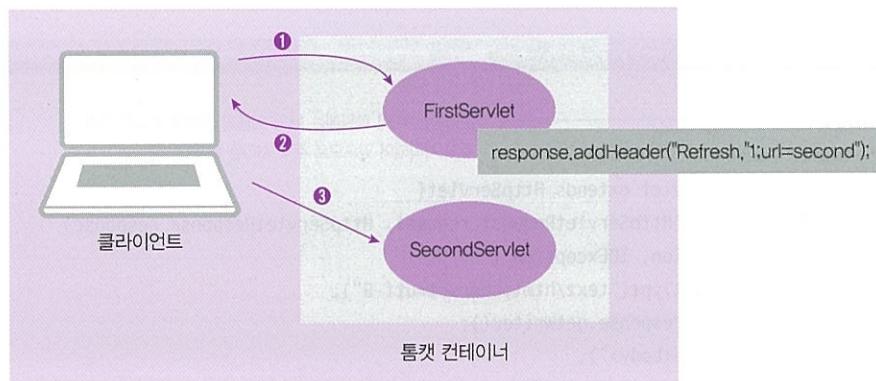
▼ 그림 8-4 매팅 이름 /second로 웹 브라우저에서 재요청



8.2.3 refresh를 이용한 포워딩

refresh를 이용한 포워딩 역시 redirect처럼 웹 브라우저를 거쳐서 요청을 수행합니다.

▼ 그림 8-5 서블릿의 refresh 이용해 포워딩하는 과정

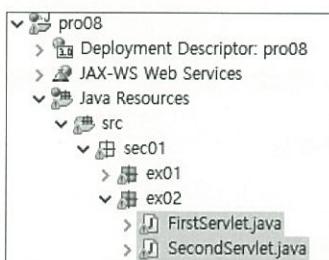


- ① 클라이언트의 웹 브라우저에서 첫 번째 서블릿에 요청합니다.
- ② 첫 번째 서블릿은 `addHeader()` 메서드를 이용해 두 번째 서블릿을 웹 브라우저를 통해서 요청합니다.
- ③ 웹 브라우저는 `addHeader()` 메서드가 지정한 두 번째 서블릿을 다시 요청합니다.

8.2.4 refresh를 이용한 포워딩 실습

- sec01.ex02 패키지를 만들고 redirect 포워딩 실습 때와 마찬가지로 두 개의 서블릿 클래스를 추가합니다.

▼ 그림 8-6 실습 파일 위치



- FirstServlet 클래스를 다음과 같이 작성합니다. response의 addHeader() 메서드에 Refresh 를 설정하고 1초 후 url=second에 지정한 second 서블릿에 브라우저에서 재요청하게 합니다.

코드 8-3 pro08/src/sec01/ex02/FirstServlet.java

```

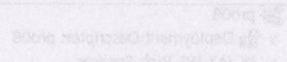
package sec01.ex02;
...
@WebServlet("/first")
public class FirstServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        response.addHeader("Refresh", "1;url=second"); ────────── 제공하는 예제 파일에는 주석 처리되어 있으므로 주석 처리를 해제합니다.
    }
}
  
```

웹 브라우저에 1초 후 서블릿 second로 재요청합니다.

Tip 웹 브라우저에서는 first라는 매팅 이름으로 동일하게 요청할 것이므로 앞서 실습한 redirect 예제 소스 (코드 8-1, 코드 8-2)에 있는 다음 두 부분은 주석 처리해야 합니다. 이후 실습을 진행할 때도 마찬가지입니다.

```

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



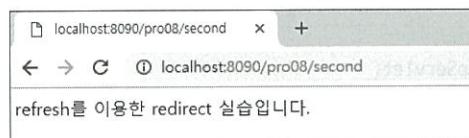
3. SecondServlet 클래스를 다음과 같이 작성합니다. 이는 브라우저에서 재요청하면 브라우저로 메시지를 출력하는 서블릿입니다.

코드 8-4 pro08/src/sec01/ex02/SecondServlet.java

```
package sec01.ex02;  
....  
@WebServlet("/second")  
public class SecondServlet extends HttpServlet{  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=utf-8");  
        PrintWriter out = response.getWriter();  
        out.println("<html><body>");  
        out.println("refresh를 이용한 redirect 실습입니다.");  
        out.println("</body></html>");  
    }  
}
```

4. 브라우저에서 http://localhost:8090/pro08/first로 요청하면 /second로 재요청합니다

▼ 그림 8-7 브라우저를 통해 서블릿 second로 재요청



8.2.5 location을 이용한 포워딩

이번에는 자바스크립트의 location 객체를 이용하는 방법을 알아보겠습니다

1. sec01.ex03 패키지를 만들고 다음과 같이 두 개의 서블릿 클래스를 추가합니다.

▼ 그림 8-8 실습 파일 위치



2. FirstServlet 클래스를 다음과 같이 작성합니다. 서블릿에서 PrintWriter로 자바스크립트 코드를 출력해 서블릿 second로 재요청합니다.

코드 8-5 pro08/src/sec01/ex03/FirstServlet.java

```
package sec01.ex03;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/first")
public class FirstServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.print("<script type='text/javascript'>");
        out.print("location.href='second';");
        out.print("</script>");
    }
}
```

제공하는 예제 파일에는 주석 처리되어 있으므로 주석 처리를 해제합니다.

자바스크립트 location의 href 속성에 서블릿 second를 설정해 재요청합니다.

3. 마찬가지로 브라우저에서 재요청하면 브라우저로 메시지를 출력하는 두 번째 서블릿을 작성합니다.

코드 8-6 pro08/src/sec01/ex03/SecondServlet.java

```
package sec01.ex03;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/second")
public class SecondServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("location을 이용한 redirect 실습입니다.");
        out.println("</body></html>");
    }
}
```

제공하는 예제 파일에는 주석 처리되어 있으므로 주석 처리를 해제합니다.

4. `http://localhost:8090/pro08/first`로 요청하면 `/second`로 재요청합니다.

▼ 그림 8-9 브라우저 요청 결과



8.2.6 redirect 방식으로 다른 서블릿에 데이터 전달하기

redirect 방식을 이용하면 웹 브라우저를 통해 다른 서블릿을 호출하면서 원하는 데이터를 전달할 수도 있습니다.

1. 이번에는 `redirect` 방법으로 최초 요청한 서블릿에서 GET 방식으로 다른 서블릿으로 데이터를 전달하는 예제를 같은 방법으로 작성해 보겠습니다. `FirstServlet` 클래스를 다음과 같이 작성합니다.

코드 8-7 pro08/src/sec02/ex01/FirstServlet.java

```
package sec02.ex01;  
...  
@WebServlet("/first")  
public class FirstServlet extends HttpServlet{  
    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();  
        response.sendRedirect("second?name=lee");  
    }  
}
```

GET 방식을 이용해 이름/값 쌍으로 데이터를 다른 서블릿으로 전달합니다.

2. `SecondServlet` 클래스를 다음과 같이 작성합니다. 이전 서블릿에서 전달된 값을 `getParameter()` 메서드를 이용해 가져옵니다.

코드 8-8 pro08/src/sec02/ex01/SecondServlet.java

```
package sec02.ex01;  
...  
@WebServlet("/second")  
public class SecondServlet extends HttpServlet{
```

```

protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=utf-8");
    PrintWriter out = response.getWriter();
    String name=request.getParameter("name"); •————— name으로 이전 서블릿에서
    out.println("<html><body>");
    out.println("이름:"+name);
    out.println("</body></html>");
}
}

```

3. 다음은 실행 결과입니다. GET 방식을 이용해 redirect되는 서블릿 second로 이름이 전달됩니다.

▼ 그림 8-10 GET 방식으로 데이터를 다른 서블릿으로 전달



refresh나 location 역시 GET 방식을 이용해 다른 서블릿으로 데이터를 전달할 수 있습니다. 각자 실습해 보기 바랍니다.

8.3

dispatch를 이용한 포워드 방법

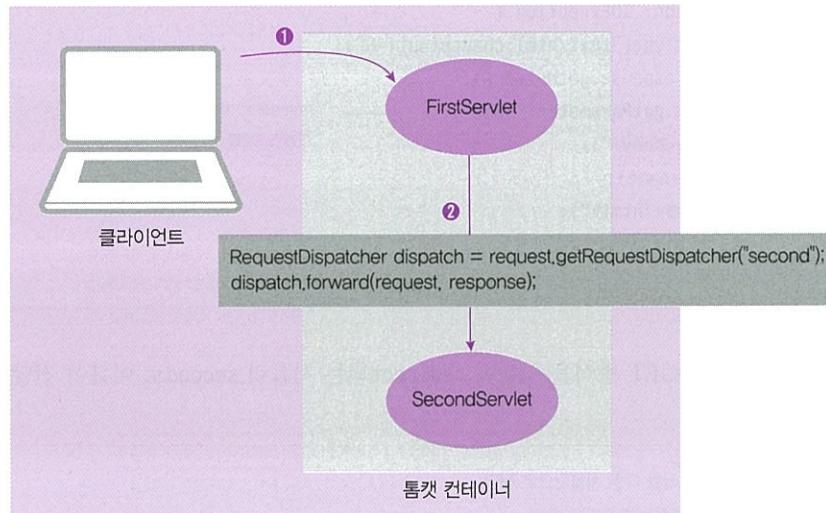
JAVA WEB

이번에는 dispatch를 이용해 포워드하는 방법을 알아보겠습니다.

8.3.1 dispatch를 이용한 포워딩 과정

dispatch를 이용한 포워딩 과정이 redirect 방법과 다른 점은 클라이언트의 웹 브라우저를 거치지 않고 바로 서버에서 포워딩이 진행된다는 것입니다. 따라서 웹 브라우저 주소창의 URL이 변경되지 않습니다. 즉, 클라이언트 측에서는 포워드가 진행되었는지 알 수 없습니다.

▼ 그림 8-11 dispatch를 이용한 포워딩 수행 과정



- ① 클라이언트의 웹 브라우저에서 첫 번째 서블릿에 요청합니다.
- ② 첫 번째 서블릿은 `RequestDispatcher`를 이용해 두 번째 서블릿으로 포워드합니다.

Tip `dispatch` 방법은 17장에서 배우는 모델2 방식이나 스트럿츠(struts), 스프링(spring) 프레임워크에서 포워딩할 때 사용합니다.

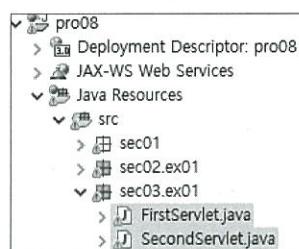
↳ 지금은 X

8.3.2 서블릿을 이용한 dispatch 포워딩 실습

이번에는 `dispatch` 방법으로 포워딩 기능을 구현해 보겠습니다.

1. sec03.ex01 패키지에 다음과 같이 두 개의 서블릿 클래스를 추가합니다.

▼ 그림 8-12 실습 파일 위치



2. FirstServlet 클래스를 다음과 같이 작성합니다. RequestDispatcher 클래스를 이용해 두 번째 서블릿인 second를 지정한 후 forward() 메서드를 이용해 포워드합니다.

코드 8-9 pro08/src/sec03/ex01/FirstServlet.java

```
package sec03.ex01;
...
@WebServlet("/first")
public class FirstServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");
        RequestDispatcher dispatch = request.getRequestDispatcher("second");
        dispatch.forward(request, response);
    }
}
```

dispatch 방법을 이용해 second로 전달합니다.

3. 두 번째 서블릿인 SecondServlet 클래스를 다음과 같이 작성합니다

코드 8-10 pro08/src/sec03/ex01/SecondServlet.java

```
package sec03.ex01;
...
@WebServlet("/second")
public class SecondServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("dispatch를 이용한 forward 실습입니다.");
        out.println("</body></html>");
    }
}
```

4. 실행해 보면 웹 브라우저 주소 창의 URL이 변경되지 않고 그대로입니다. 이는 서블릿의 포워드가 서버에서 수행되었기 때문입니다.

▼ 그림 8-13 매팅 이름 first로 요청한 결과



5. 이번에는 `dispatch`를 이용해 전송할 때 GET 방식으로 데이터를 전송해 봅시다. 앞의 서블릿 클래스를 다음과 같이 수정합니다. 서블릿 이름 다음에 `?name=lee`를 추가하여 GET 방식으로 name 값을 두 번째 서블릿으로 전달합니다.

코드 8-11 pro08/src/sec03/ex01/FirstServlet.java

```
package sec03.ex01;  
...  
@WebServlet("/first")  
public class FirstServlet extends HttpServlet{  
    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();  
        RequestDispatcher dispatcher =  
            request.getRequestDispatcher("second?name=lee");  
        dispatcher.forward(request, response);  
    }  
}
```

GET 방식으로 데이터를 전달합니다.

6. `dispatch`를 이용해 전달된 name 값을 출력합니다.

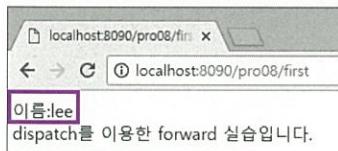
코드 8-12 pro08/src/sec03/ex01/SecondServlet.java

```
package sec03.ex01;  
...  
@WebServlet("/second")  
public class SecondServlet extends HttpServlet{  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=utf-8");  
        PrintWriter out = response.getWriter();  
        String name=request.getParameter("name");  
        out.println("<html><body>");  
        out.println("이름:"+name);  
        out.println("<br>");  
        out.println("dispatch를 이용한 forward 실습입니다.");  
        out.println("</body></html>");  
    }  
}
```

다른 서블릿에서 전달된 데이터를 가져옵니다.

7. GET 방식으로 `dispatch`를 이용해 데이터를 전달해도 웹 브라우저의 URL은 변경되지 않습니다.

▼ 그림 8-14 매핑 이름 first로 요청한 결과



8.4 바인딩

JAVA WEB

앞 절에서는 서블릿에서 다른 서블릿으로 포워딩할 때 GET 방식으로 데이터를 전달하는 방법을 알아봤습니다. 전달하는 데이터 양이 적을 때는 이 방법이 편리합니다. 그러나 서블릿에서 조회한 대량의 상품 정보를 JSP로 전달할 때는 GET 방식이 불편합니다. 따라서 서블릿에서 다른 서블릿 또는 JSP로 대량의 데이터를 공유하거나 전달하고 싶을 때는 **바인딩(binding)** 기능을 사용합니다.

바인딩의 사전적 의미는 “**두 개를 하나로 묶는다**”는 것입니다. 이는 웹 프로그램 실행 시 자원(데이터)을 서블릿 관련 객체에 저장하는 방법으로, 주로 `HttpServletRequest`, `HttpSession`, `ServletContext` 객체에서 사용되며 저장된 자원(데이터)은 프로그램 실행 시 서블릿이나 JSP에서 공유하여 사용합니다.

실제 모델2, 스트럿츠, 스프링 프레임워크로 구현하는 웹 프로그램은 이 바인딩 기능을 이용해 서블릿이나 JSP 간 데이터를 전달하고 공유합니다.

표 8-1은 서블릿 관련 객체에서 바인딩 관련 기능을 제공하는 여러 가지 메서드입니다.

▼ 표 8-1 서블릿 객체에서 사용되는 바인딩 관련 메서드

관련 메서드	기능
<code>setAttribute(String name, Object obj)</code>	자원(데이터)을 각 객체에 바인딩합니다.
<code>getAttribute(String name)</code>	각 객체에 바인딩된 자원(데이터)을 name으로 가져옵니다.
<code>removeAttribute(String name)</code>	각 객체에 바인딩된 자원(데이터)을 name으로 제거합니다.

8.4.1 HttpServletRequest를 이용한 redirect 포워딩 시 바인딩

먼저 HttpServletRequest 객체를 이용한 바인딩 기능을 알아보겠습니다. 브라우저에서 전달 받은 request를 서블릿에서 redirect 방식으로 다른 서블릿에 전달하는 예제입니다.

1. 다음과 같이 실습 파일을 준비합니다.

▼ 그림 8-15 실습 파일 위치



2. FirstServlet 클래스를 다음과 같이 작성합니다. HttpServletRequest의 setAttribute() 메서드를 이용해 (address, "서울시 성북구")를 바인딩합니다.

코드 8-13 pro08/src/sec04/ex01/FirstServlet.java

```
package sec04.ex01;  
...  
@WebServlet("/first")  
public class FirstServlet extends HttpServlet{  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        request.setCharacterEncoding("utf-8");  
        response.setContentType("text/html; charset=utf-8");  
        request.setAttribute("address", "서울시 성북구"); •———— 웹 브라우저에서 요청한 request  
        response.sendRedirect("second"); 객체에 address의 값으로 "서울시  
    }                                         성북구"를 바인딩합니다.  
}                                         두 번째 서블릿으로 전달하기 위해  
                                         sendRedirect()를 호출합니다.
```

3. 두 번째 서블릿에서는 HttpServletRequest의 getAttribute() 메서드를 이용해 전달된 주소를 받습니다.

코드 8-14 pro08/src/sec04/ex01/SecondServlet.java

```
package sec04.ex01;  
...  
@WebServlet("/second")
```

```

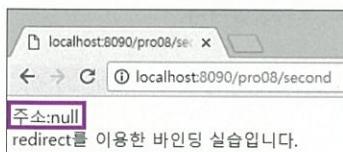
public class SecondServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        String address=(String)request.getAttribute("address");
        out.println("<html><body>");
        out.println("주소:"+address);
        out.println("<br>");
        out.println("redirect를 이용한 바인딩 실습입니다.");
        out.println("</body></html>");
    }
}

```

전달된 request에서 getAttribute()를
이용해 address의 값을 가져옵니다.

4. 실행 결과를 보면 정상적으로는 '서울시 성북구'가 출력되어야 하는데 null이 출력됩니다.
왜 그럴까요?

▼ 그림 8-16 매핑 이름 first로 요청한 결과



그 이유는 8.1절 그림 8-1의 `redirect` 방식 포워드 과정 때문입니다. 포워딩 과정 1단계인 웹 브라우저에서 요청할 때 서블릿에 전달되는 첫 번째 `request`는 웹 브라우저를 통해 재요청되는 3단계의 두 번째 `request`와 다른 요청입니다. 즉, `redirect` 방식으로는 서블릿에서 바인딩한 데이터를 다른 서블릿으로 전송할 수 없다는 것입니다.

그럼 이런 의문이 들 수 있을 것입니다. 앞서 `redirect` 방식을 실습했을 때처럼 GET 방식으로 전송하면 되지 않느냐고 말이죠. 물론 전달하고자 하는 데이터가 보안과 상관이 없으며, 데이터 양이 적다면 그렇게 해도 괜찮습니다. 하지만 데이터베이스에서 조회된 수십 개의 회원 정보나 상품 정보를 전달해야 한다면 확실히 `redirect` 방식에는 문제가 있습니다.

8.4.2 HttpServletRequest를 이용한 dispatch 포워딩 시 바인딩

이번에는 `dispatch` 방법으로 바인딩 기능을 사용해 보겠습니다.

1. 다음과 같이 실습 파일을 준비합니다.

▼ 그림 8-17 실습 파일 위치



2. `FirstServlet` 클래스를 다음과 같이 작성합니다. 브라우저에서 전달된 `request`에 주소를 바인딩한 후 `dispatch` 방법을 이용해 다른 서블릿으로 포워딩합니다.

코드 8-15 pro08/src/sec04/ex02/FirstServlet.java

```
package sec04.ex02;
...
@WebServlet("/first")
public class FirstServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");
        request.setAttribute("address", "서울시 성북구");
        RequestDispatcher dispatcher = request.getRequestDispatcher("second");
        dispatcher.forward(request, response);
    }
}
```

웹 브라우저의 최초 요청 request에
바인딩합니다.

바인딩된 request를 다시 두 번째
서블릿으로 포워드합니다.

3. `SecondServlet` 클래스를 다음과 같이 작성합니다. 전달된 `request`에서 주소를 받은 후 브라우저로 출력합니다.

코드 8-16 pro08/src/sec04/ex02/SecondServlet.java

```
package sec04.ex02;
...

```

```

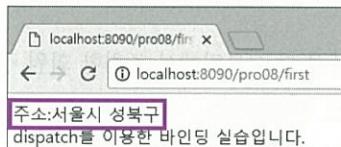
@WebServlet("/second")
public class SecondServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        String address=(String)request.getAttribute("address");
        out.println("<html><body>");
        out.println("주소:"+address);
        out.println("<br>");
        out.println("dispatch를 이용한 바인딩 실습입니다.");
        out.println("</body></html>");
    }
}

```

전달된 request에서 getAttribute()를
이용해 주소를 받아 옵니다.

4. 이번에는 화면에 정상적으로 주소가 출력됩니다. 그 이유는 8.3절의 그림 8-11을 보면 알 수 있습니다.

▼ 그림 8-18 dispatch 방법으로 바인딩된 데이터 전달



이 포워딩 과정을 보면 첫 번째 서블릿에서 두 번째 서블릿으로 전달되는 request가 브라우저를 거치지 않고 바로 전달되었습니다. 따라서 첫 번째 서블릿의 request에 바인딩된 데이터가 그대로 전달된 것입니다.

모델2, 스트럿츠, 스프링 프레임워크로 개발할 때는 dispatch 방식으로 바인딩된 데이터를 서블릿이나 JSP로 전달합니다. 자세한 것은 17장에서 알아보겠습니다.

8.4.3 두 서블릿 간 회원 정보 조회 바인딩 실습

이번에는 데이터베이스에서 조회된 회원 정보를 화면 기능을 담당하는 서블릿에 전달해서 웹 브라우저에 출력해 보겠습니다.

- 7장에서 실습한 MemberDAO와 MemberVO 클래스를 다음과 같이 복사하여 붙여 넣습니다. 그리고 데이터베이스 연동을 위한 DataSource 기능도 7장을 참고하여 설정합니다.

▼ 그림 8-19 실습 파일 위치



- MemberServlet 클래스를 다음과 같이 작성합니다. 첫 번째 서블릿에서 조회한 회원 정보를 List에 저장한 후 다시 바인딩하여 두 번째 서블릿으로 전달합니다.

코드 8-17 pro08/src/sec04/ex03/MemberServlet.java

```
package sec04.ex03;  
...  
@WebServlet("/member")  
public class MemberServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        doHandle(request, response);  
    }  
    protected void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        doHandle(request, response);  
    }  
    private void doHandle(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        request.setCharacterEncoding("utf-8");  
        response.setContentType("text/html; charset=utf-8");  
        PrintWriter out = response.getWriter();  
    }  
}
```

```

MemberDAO dao=new MemberDAO();
List memberList=dao.listMembers();
request.setAttribute("membersList", membersList);
RequestDispatcher dispatcher = request.getRequestDispatcher("viewMembers");
dispatcher.forward(request, response);
}
}

```

조회된 회원 정보를 ArrayList 객체에 저장한 후 request에 바인딩합니다.

바인딩한 request를 viewMembers 서블릿으로 포워딩합니다.

3. ViewServlet 클래스를 다음과 같이 작성합니다. getAttribute() 메서드를 이용해 첫 번째 서블릿에서 바인딩한 회원 정보를 List로 가져옵니다.

코드 8-18 pro08/src/sec04/ex03/ViewServlet.java

```

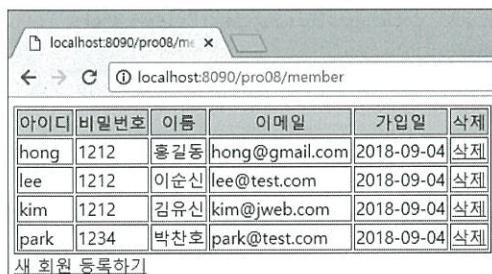
package sec04.ex03;
...
@WebServlet("/viewMembers")
public class ViewServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out=response.getWriter();
        List membersList = (List) request.getAttribute("membersList");
        out.print("<html><body>");
        out.print("<table border=1><tr align='center' bgcolor='lightgreen'>");
        out.print("<td>아이디</td><td>비밀번호</td><td>이름</td><td>이메일</td>
                <td>가입일</td><td>삭제</td></tr>");
        for (int i = 0; i < membersList.size(); i++) {
            MemberVO memberVO = (MemberVO) membersList.get(i);
            String id = memberVO.getId();
            String pwd = memberVO.getPwd();
            String name = memberVO.getName();
            String email = memberVO.getEmail();
            Date joinDate = memberVO.getJoinDate();
            out.print("<tr><td>" + id + "</td><td>" + pwd + "</td><td>" + name + "</td><td>"
                    + email + "</td><td>" + joinDate + "</td><td>" +
                    "<a href='/pro08/member3?command=delMember&id=" + id
                    + "'>삭제 </a></td></tr>");
        }
        out.print("</table></body></html>");
        out.print("<a href='/pro08/memberForm.html'>새 회원 등록하기</a>");
    }
}

```

바인딩해서 넘어온 request에서 회원 정보를 가져옵니다.

4. <http://localhost:8090/pro08/member>로 요청하여 실행 결과를 확인합니다.

▼ 그림 8-20 실행 결과



A screenshot of a web browser window titled "localhost:8090/pro08/member". The page displays a table with member information. The columns are labeled: 아이디 (ID), 비밀번호 (Password), 이름 (Name), 이메일 (Email), 가입일 (Join Date), and 삭제 (Delete). The table contains five rows of data:

아이디	비밀번호	이름	이메일	가입일	삭제
hong	1212	홍길동	hong@gmail.com	2018-09-04	삭제
lee	1212	이순신	lee@test.com	2018-09-04	삭제
kim	1212	김유신	kim@jweb.com	2018-09-04	삭제
park	1234	박찬호	park@test.com	2018-09-04	삭제

At the bottom left of the table, there is a link "새 회원 등록하기".

ViewServlet 클래스는 웹 브라우저에서 화면 기능을 담당하는데 이러한 기능을 하는 서블릿이 분화되어 발전된 것이 바로 JSP입니다.

8.5

ServletContext와 ServletConfig 사용법

JAVA WEB

이번에는 서블릿과 더불어 웹 프로그래밍 개발 시 유용한 기능을 제공하는 클래스들을 알아보겠습니다.

8.5.1 ServletContext 클래스

ServletContext 클래스는 톰캣 컨테이너 실행 시 각 컨텍스트(웹 애플리케이션)마다 한 개의 ServletContext 객체를 생성합니다. 그리고 톰캣 컨테이너가 종료하면 ServletContext 객체 역시 소멸됩니다. ServletContext 객체는 웹 애플리케이션이 실행되면서 애플리케이션 전체의 공통 자원이나 정보를 미리 바인딩해서 서블릿들이 공유하여 사용합니다.

ServletContext 클래스의 특징은 다음과 같습니다.

- javax.servlet.ServletContext로 정의되어 있습니다.
- 서블릿과 컨테이너 간의 연동을 위해 사용합니다.
- 컨텍스트(웹 애플리케이션)마다 하나의 ServletContext가 생성됩니다.
- 서블릿끼리 자원(데이터)을 공유하는 데 사용합니다.
- 컨테이너 실행 시 생성되고 컨테이너 종료 시 소멸됩니다.

ServletContext가 제공하는 기능은 다음과 같습니다.

- 서블릿에서 파일 접근 기능
- 자원 바인딩 기능
- 로그 파일 기능
- 컨텍스트에서 제공하는 설정 정보 제공 기능

그림 8-21에는 톰캣 컨테이너를 실행할 때 각 애플리케이션에서 생성되는 ServletContext와 ServletConfig 객체를 나타내었습니다. ServletContext는 컨텍스트당 생성되는 반면에 ServletConfig는 각 서블릿에 대해 생성됩니다.

▼ 그림 8-21 톰캣 컨테이너의 ServletContext와 ServletConfig 생성 상태

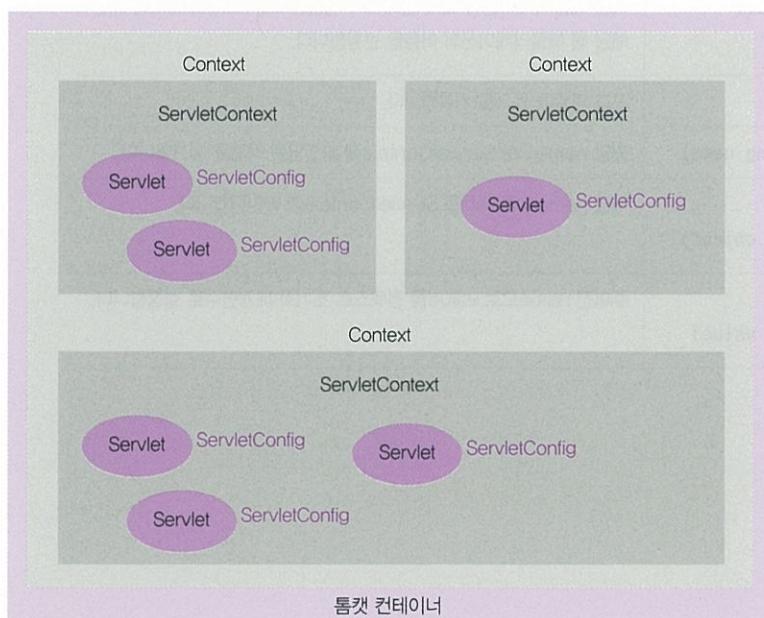


표 8-2는 ServletContext에서 제공하는 여러 가지 메서드의 기능을 정리한 것입니다.

▼ 표 8-2 ServletContext에서 제공하는 여러 가지 메서드

메서드	기능
getAttribute(String name)	<ul style="list-style-type: none">주어진 name을 이용해 바인딩된 value를 가져옵니다.name이 존재하지 않으면 null을 반환합니다.
getAttributeNames()	바인딩된 속성들의 name을 반환합니다.
getContext(String uripath)	지정한 uripath에 해당되는 객체를 반환합니다.
getInitParameter(String name)	<ul style="list-style-type: none">name에 해당되는 매개변수의 초기화 값을 반환합니다.name에 해당되는 매개변수가 존재하지 않으면 null을 반환합니다.
getInitParameterNames()	<ul style="list-style-type: none">컨텍스트의 초기화 관련 매개변수들의 이름들을 String 객체가 저장된 Enumeration 타입으로 반환합니다.매개변수가 존재하지 않으면 null을 반환합니다.
getMajorVersion()	서블릿 컨테이너가 지원하는 주요 서블릿 API 버전을 반환합니다.
getRealPath(String path)	지정한 path에 해당되는 실제 경로를 반환합니다.
getResource(String path)	지정한 path에 해당되는 Resource를 반환합니다.
getServerInfo()	현재 서블릿이 실행되고 있는 서블릿 컨테이너의 이름과 버전을 반환합니다.
getServletContextName()	해당 애플리케이션의 배치 관리자가 지정한 ServletContext에 대한 해당 웹 애플리케이션의 이름을 반환합니다.
log(String msg)	로그 파일에 로그를 기록합니다.
removeAttribute(String name)	해당 name으로 ServletContext에 바인딩된 객체를 제거합니다.
setAttribute (String name, Object object)	해당 name으로 객체를 ServletContext에 바인딩합니다.
setInitParameter (String name, String value)	주어진 name으로 value를 컨텍스트 초기화 매개변수로 설정합니다.

8.5.2 ServletContext 바인딩 기능

이번에는 ServletContext의 바인딩 기능을 알아보겠습니다.

1. 다음과 같이 GetServletContext, SetServletContext 클래스 파일을 준비합니다.

▼ 그림 8-22 실습 파일 위치



2. SetServletContext 클래스를 다음과 같이 작성합니다. getServletContext() 메서드를 이용해 ServletContext 객체에 접근한 다음 ArrayList에 이름과 나이를 저장한 후 다시 ServletContext 객체에 setAttribute() 메서드를 이용해 바인딩합니다.

코드 8-19 pro08/src/sec05/ex01/SetServletContext.java

```
package sec05.ex01;
...
@WebServlet("/cset")
public class SetServletContext extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        ServletContext context = getServletContext(); ← ServletContext 객체를 가져옵니다.
        List member = new ArrayList();
        member.add("이순신");
        member.add(30);
        context.setAttribute("member", member); ← ServletContext 객체에 데이터를
        out.print("<html><body>");
        out.print("이순신과 30 설정");
        out.print("</body></html>");
    }
}
```

3. GetServletContext 클래스를 다음과 같이 작성합니다. getServletContext() 메서드를 이용해 ServletContext 객체에 접근합니다. 그리고 getAttribute() 메서드를 이용해 다른 서블릿에서 바인딩한 ArrayList를 가져와 회원 정보를 출력합니다.

코드 8-20 pro08/src/sec05/ex01/GetServletContext.java

```
package sec05.ex01;  
...  
@WebServlet("/cget")  
public class GetServletContext extends HttpServlet{  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html; charset=utf-8");  
        PrintWriter out = response.getWriter();  
        ServletContext context = getServletContext(); ────────── ServletContext 객체를 가져옵니다.  
        List member = (ArrayList)context.getAttribute("member");  
        String name = (String)member.get(0); ─────────── member로 이전에 바인딩된  
        int age = (Integer)member.get(1);  
        out.print("<html><body>");  
        out.print(name + "<br>");  
        out.print(age + "<br>");  
        out.print("</body></html>");  
    }  
}
```

ServletContext 객체를 가져옵니다.
member로 이전에 바인딩된
회원 정보를 가져옵니다.

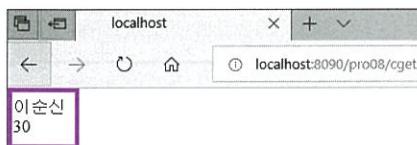
4. 크롬 브라우저에서 <http://localhost:8090/pro08/cset>으로 요청하면 ServletContext 객체에 데이터를 바인딩합니다.

▼ 그림 8-23 첫 번째 브라우저에서 /cset으로 요청



5. 이번에는 인터넷 익스플로러에서 <http://localhost:8090/pro08/cget>으로 요청합니다. 마찬가지로 바인딩된 데이터를 브라우저에 표시합니다.

▼ 그림 8-24 IE에서 /cget으로 요청



이처럼 ServletContext에 바인딩된 데이터는 모든 서블릿들(사용자)이 접근할 수 있습니다. 따라서 웹 애플리케이션에서 모든 사용자가 공통으로 사용하는 데이터는 ServletContext에 바인딩해 놓고 사용하면 편리합니다.

8.5.3 ServletContext의 매개변수 설정 기능

대부분의 웹 애플리케이션에서 메뉴는 공통으로 사용하는 기능입니다. 따라서 web.xml에 설정해 놓고 프로그램 시작 시 초기화할 때 가져와서 사용하면 편리합니다. 그러면 새로운 메뉴 항목이 생성되거나 기존 메뉴 항목을 추가, 삭제할 때도 쉽게 수정할 수 있습니다.

1. 다음과 같이 ContextParamServlet 클래스 파일과 web.xml 파일을 준비합니다.

▼ 그림 8-25 실습 파일 위치



2. web.xml에 메뉴 항목을 설정합니다. <context-param> 태그 안에 <param-name> 태그와 <param-value> 태그를 이용해 메뉴에 대한 하위 메뉴 항목을 설정합니다.

코드 8-21 pro08/WebContent/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <context-param>
        <param-name>menu_member</param-name>
        <param-value>회원등록 회원조회 회원수정</param-value>
    </context-param>
```

→ <context-param> 태그 안에 다시 <param-name>과 <param-value> 태그로 초기 값을 설정합니다.

```

<context-param>
    <param-name>menu_order</param-name>
    <param-value>주문조회 주문등록 주문수정 주문취소</param-value>
</context-param>
<context-param>
    <param-name>menu_goods</param-name>
    <param-value>상품조회 상품등록 상품수정 상품삭제</param-value>
</context-param>
...
</web-app>

```

3. ContextParamServlet 클래스를 다음과 같이 작성합니다. getServletContext() 메서드로 ServletContext 객체에 접근합니다. 그리고 getInitParameter() 메서드의 인자로 각각의 메뉴 이름을 전달한 후 메뉴 항목들을 가져와 이를 브라우저로 출력합니다.

코드 8-22 pro08/src/sec05/ex02/ContextParamServlet.java

```

package sec05.ex02;
...
@WebServlet("/initMenu")
public class ContextParamServlet extends HttpServlet {
    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();
        ServletContext context = getServletContext(); ← ServletContext 객체를 가져옵니다.
        String menu_member = context.getInitParameter("menu_member");
        String menu_order = context.getInitParameter("menu_order");
        String menu_goods = context.getInitParameter("menu_goods");
        out.print("<html><body>");
        out.print("<table border=1 cellspacing=0><tr>메뉴 이름</tr>");
        out.print("<tr><td>" + menu_member + "</td></tr>");
        out.print("<tr><td>" + menu_order + "</td></tr>");
        out.print("<tr><td>" + menu_goods + "</td></tr>");
        out.print("</tr></table></body></html>");
    }
}

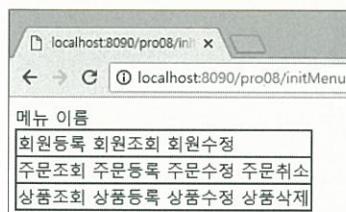
```

ServletContext 객체를 가져옵니다.

web.xml의 <param-name> 태그의 이름으로 <param-value> 태그의 값인 메뉴 이름들을 받아옵니다.

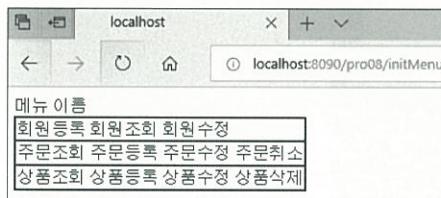
4. 크롬 브라우저에서 <http://localhost:8090/pro08/initMenu>로 서블릿을 요청합니다.

▼ 그림 8-26 크롬에서 initMenu로 요청한 결과



5. 인터넷 익스플로러에서도 요청해 봅니다.

▼ 그림 8-27 IE에서 /initMenu로 요청한 결과



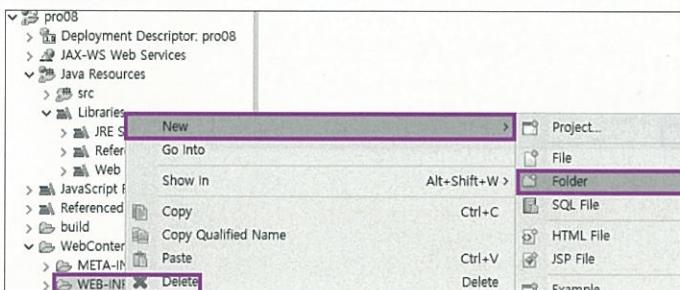
모든 브라우저에서 같은 메뉴를 출력하는 것을 확인할 수 있습니다. 즉, 메뉴는 ContextServlet 객체를 통해 접근하므로 모든 웹 브라우저에서 공유하면서 접근할 수 있습니다.

8.5.4 ServletContext의 파일 입출력 기능

이번에는 ServletContext의 파일에서 데이터를 읽어 오는 기능을 알아보겠습니다. 먼저 폴더를 하나 생성합니다.

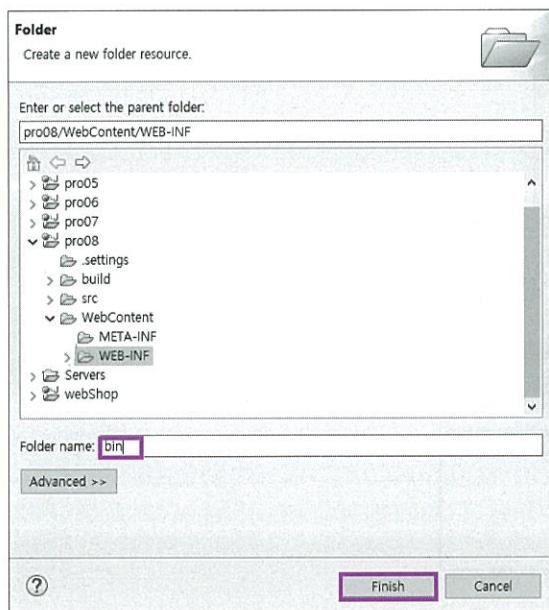
- 프로젝트 pro08의 WebContent/WEB-INF 폴더를 선택하고 마우스 오른쪽 버튼을 클릭한 후 New > Folder를 선택합니다.

▼ 그림 8-28 New > Folder 선택



2. 폴더 이름으로 bin을 입력하고 Finish를 클릭합니다.

▼ 그림 8-29 폴더 이름으로 bin 입력 후 Finish 클릭



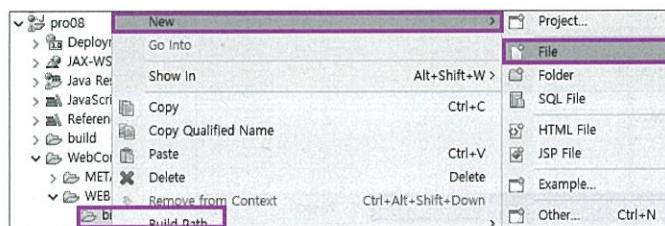
3. bin 폴더가 생성된 것을 확인할 수 있습니다.

▼ 그림 8-30 bin 폴더 생성 확인



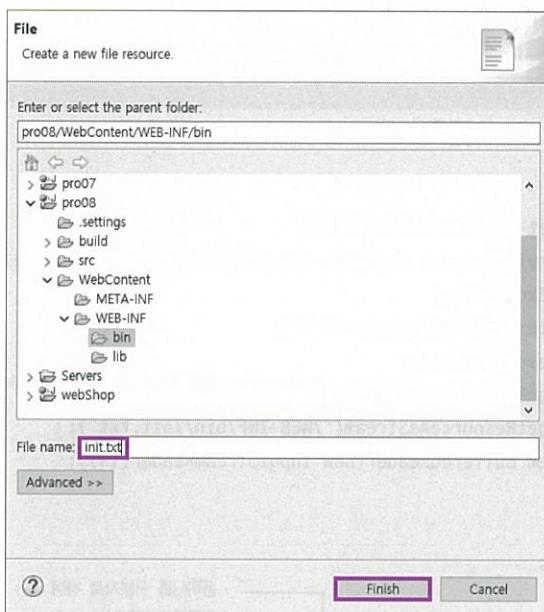
4. bin 폴더를 선택하고 마우스 오른쪽 버튼을 클릭한 후 NEW > File을 선택합니다.

▼ 그림 8-31 NEW > File 선택



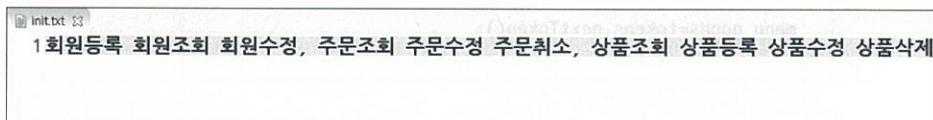
5. 파일 이름으로 init.txt를 입력하고 Finish를 클릭합니다.

▼ 그림 8-32 파일 이름으로 init.txt 입력 후 Finish 클릭



6. 생성된 파일에 메뉴 항목을 입력한 후 저장합니다

▼ 그림 8-33 파일에 메뉴 항목 입력 후 저장



7. 이제 init.txt에서 메뉴 데이터를 읽어와 출력하는 기능을 구현해 보겠습니다. 다음과 같이 ContextFileServlet 클래스를 준비합니다.

▼ 그림 8-34 실습 파일 위치



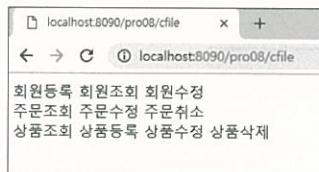
8. ContextFileServlet 클래스를 다음과 같이 작성합니다. getServletContext() 메서드로 ServletContext에 접근하여 getResourceAsStream() 메서드에서 읽어 들일 파일 위치를 지정한 후 파일에서 데이터를 입력 받습니다.

코드 8-23 pro08/src/sec05/ex03/ContextFileServlet.java

```
package sec05.ex03;  
...  
@WebServlet("/cfile")  
public class ContextFileServlet extends HttpServlet{  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=utf-8");  
        PrintWriter out = response.getWriter();  
        ServletContext context = getServletContext(); └───────── 해당 위치의 파일을 읽어 들입니다.  
        InputStream is = context.getResourceAsStream("/WEB-INF/bin/init.txt");  
        BufferedReader buffer = new BufferedReader(new InputStreamReader(is));  
  
        String menu= null;  
        String menu_member = null;  
        String menu_order = null;  
        String menu_goods = null;  
        while((menu=buffer.readLine()) !=null){  
            StringTokenizer tokens = new StringTokenizer(menu, ",");  
            menu_member=tokens.nextToken();  
            menu_order=tokens.nextToken();  
            menu_goods=tokens.nextToken();  
        }  
        out.print("<html><body>");  
        out.print(menu_member + "<br>");  
        out.print(menu_order+ "<br>");  
        out.print(menu_goods + "<br>");  
        out.print("</body></html>");  
        out.close();  
    }  
}
```

9. `http://localhost:8090/pro08/cfile`로 요청하면 다음과 같이 파일의 메뉴 항목을 읽어와 브라우저로 출력합니다.

▼ 그림 8-35 실행 결과



8.5.5 ServletConfig

이번에는 다른 서블릿 확장 API인 `ServletConfig`에 대해 알아보겠습니다.

`ServletConfig`는 각 `Servlet` 객체에 대해 생성됩니다(그림 8-21 참조). 그리고 5.2절에서 언급한 서블릿 API 계층 구조를 보면 `ServletConfig` 인터페이스를 `GenericServlet` 클래스가 실제로 구현하고 있습니다.

`ServletConfig`에서 제공하는 여러 가지 메서드를 이용해 서블릿에 관련된 기능을 사용할 수 있습니다. 대표적인 기능이 앞에서 실습한 `ServletContext` 객체를 가져오는 기능입니다.

`ServletConfig`는 `javax.servlet` 패키지에 인터페이스로 선언되어 있으며, 서블릿에 대한 여러 가지 기능을 제공합니다. 각 서블릿에서만 접근할 수 있으며 공유는 불가능합니다. `ServletConfig`는 서블릿과 동일하게 생성되고 서블릿이 소멸되면 같이 소멸됩니다.

`ServletConfig`가 제공하는 기능은 다음과 같습니다.

- `ServletContext` 객체를 얻는 기능
- 서블릿에 대한 초기화 작업 기능

이번에는 서블릿에서 사용할 설정 정보를 읽어 들여와 초기화하는 기능을 알아보겠습니다. 서블릿에서 초기화하는 방법으로는 `@WebServlet` 애너테이션을 이용하는 방법과 `web.xml`에 설정하는 방법이 있습니다. 먼저 애너테이션을 이용하는 방법을 알아보겠습니다.

8.5.6 @WebServlet 애너테이션을 이용한 서블릿 설정

표 8-3은 @WebServlet의 중요한 구성 요소에 대한 설명입니다.

▼ 표 8-3 @WebServlet 구성 요소들

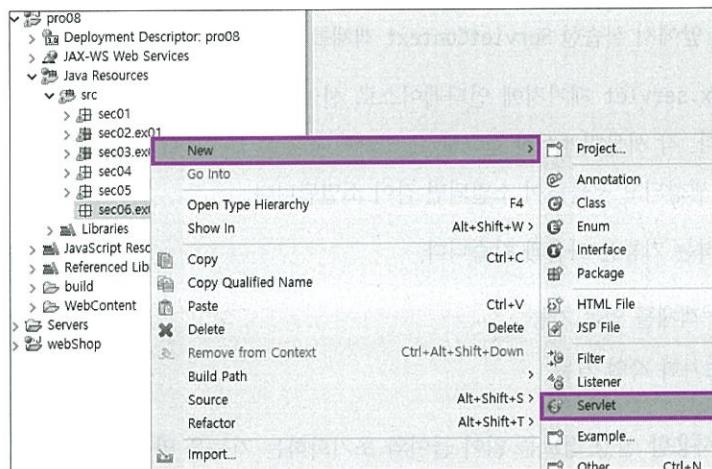
요소	설명
urlPatterns	웹 브라우저에서 서블릿 요청 시 사용하는 맵핑 이름
name	서블릿 이름
loadOnStartup	컨테이너 실행 시 서블릿이 로드되는 순서 지정
initParams	@WebInitParam 애너테이션 이용해 매개변수를 추가하는 기능
description	서블릿에 대한 설명

아클립스에서 서블릿을 생성할 때 @WebServlet의 값들을 편리하게 설정할 수 있습니다.

@WebServlet으로 서블릿을 생성할 때 사용할 매개변수를 설정해 보겠습니다.

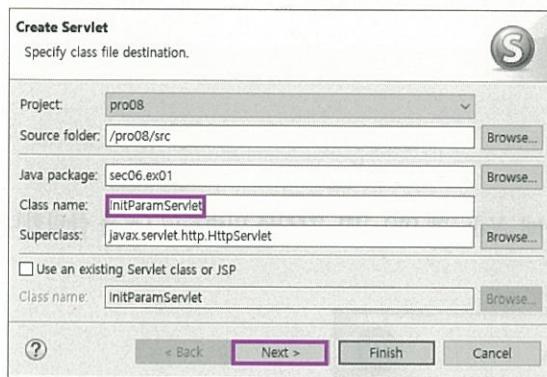
1. sec06_ex01 패키지를 생성하고 마우스 오른쪽 버튼을 클릭한 후 New > Servlet을 선택합니다.

▼ 그림 8-36 New > Servlet 선택



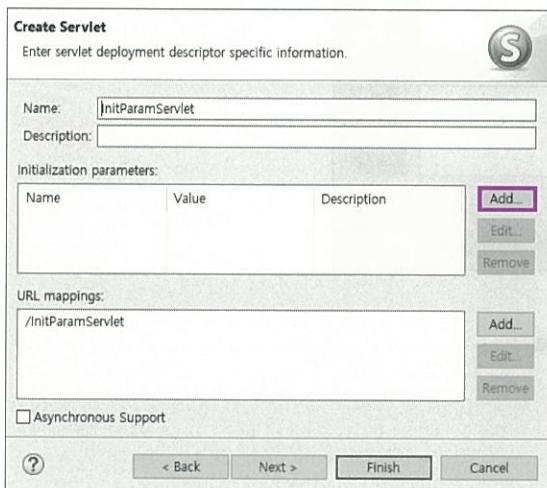
2. 클래스 이름으로 InitParamServlet을 입력한 후 Next를 클릭합니다.

▼ 그림 8-37 클래스 이름으로 InitParamServlet 입력 후 Next 클릭



3. Initialization parameters 항목의 Add...를 클릭합니다.

▼ 그림 8-38 Add...를 클릭



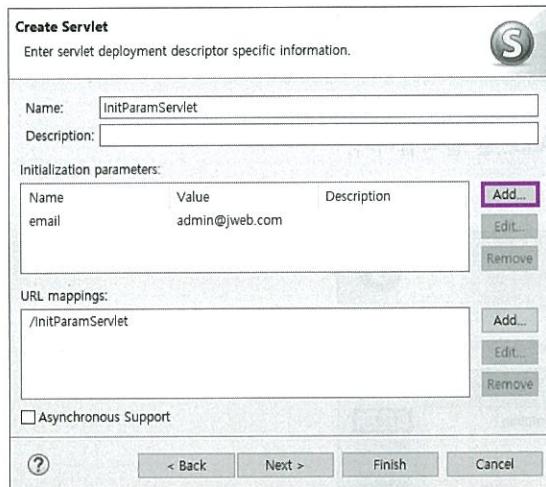
4. Name에 email, Value에 admin@jweb.com을 입력한 후 OK를 클릭합니다.

▼ 그림 8-39 email 정보 입력 후 OK 클릭

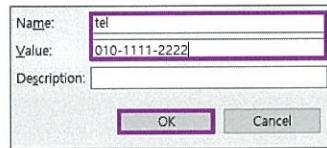


5. 다시 Add...를 클릭한 후 Name에 tel, Value에 010-1111-2222를 입력하고 OK를 클릭합니다.

▼ 그림 8-40 Add... 클릭

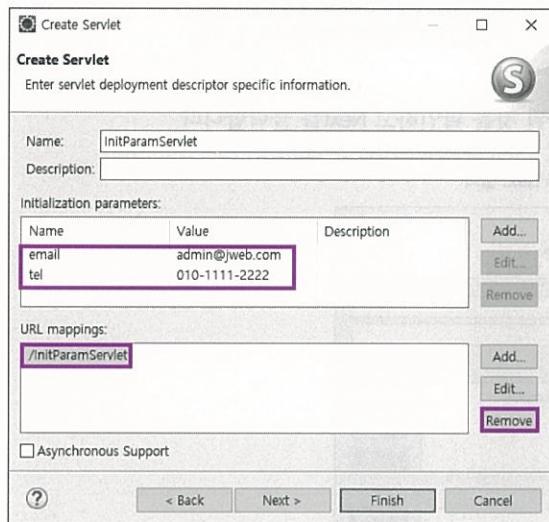


▼ 그림 8-41 전화번호 정보 입력 후 OK 클릭



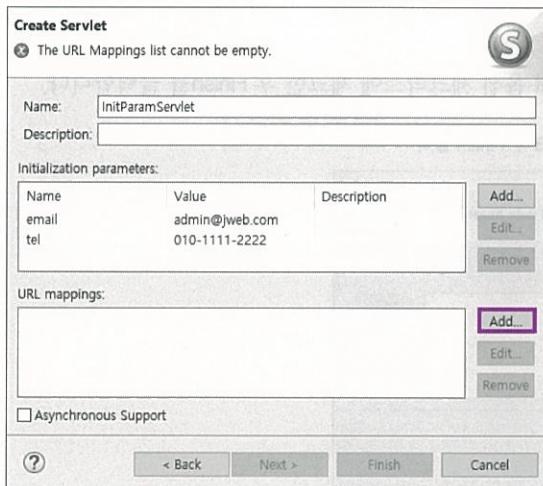
6. 두 개의 서블릿 매개변수가 추가되었음을 확인한 후 URL mappings의 /InitParamServlet을 선택하고 Remove를 클릭해 삭제합니다.

▼ 그림 8-42 Remove 클릭



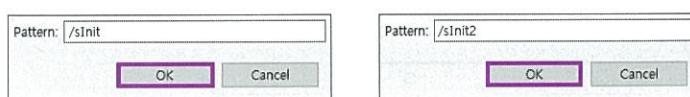
7. 새로운 맵핑 이름을 추가하기 위해 Add...를 클릭합니다.

▼ 그림 8-43 Add... 클릭



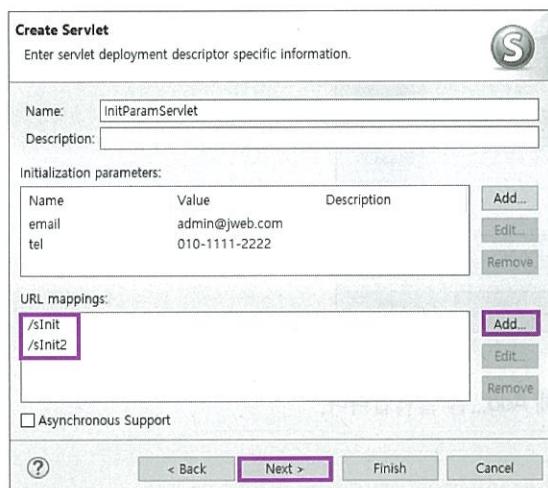
8. 첫 번째 매핑 이름은 /sInit로, 두 번째 매핑 이름은 /sInit2로 입력하고 각각 OK를 클릭합니다.

▼ 그림 8-44 /sInit와 sInit2 매핑 이름 입력



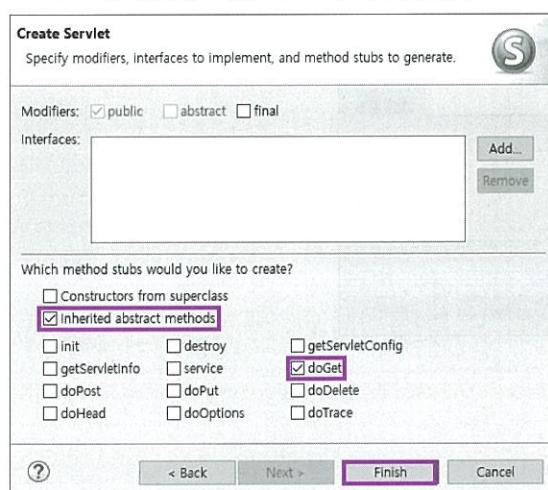
9. 두 개의 서블릿 매핑 이름이 추가된 것을 확인하고 Next를 클릭합니다.

▼ 그림 8-45 매개변수와 매핑 이름 확인 후 Next 클릭



10. Inherited abstract methods와 doGet 옵션 체크박스에 체크한 후 Finish를 클릭합니다.

▼ 그림 8-46 서블릿에서 사용할 메서드 체크 후 Finish 클릭



11. 이클립스에서 확인하면 설정한 대로 @WebServlet으로 표시되는 것을 확인할 수 있습니다.

▼ 그림 8-47 @WebServlet에 추가된 매핑 이름과 매개변수 확인

```

1 package sec06.ex01;
2
3 import java.io.IOException;
10
11 /**
12 * Servlet implementation class InitParamServlet
13 */
14 @WebServlet(
15     urlPatterns = {
16         "/sInit",
17         "/sInit2"
18     },
19     initParams = {
20         @WebInitParam(name = "email", value = "admin@jweb.com"),
21         @WebInitParam(name = "tel", value = "010-1111-2222")
22     }
23 public class InitParamServlet extends HttpServlet {
24     private static final long serialVersionUID = 1L;

```

12. InitParamServlet 클래스를 다음과 같이 작성합니다. getInitParameter() 메서드에 애너테이션으로 매개변수를 설정할 때 지정한 email과 name을 인자로 전달하여 각 값을 가져옵니다.

코드 8-24 pro08/sec06/ex01/InitParamServlet.java

```

package sec06.ex01;
...
@WebServlet(name="InitParamServlet",
            urlPatterns = {"/sInit", "/sInit2"},           urlPatterns를 이용해 매핑 이름을
                                                       여러 개 설정할 수 있습니다.
            initParams = {@WebInitParam(name="email", value="admin@jweb.com"),
                          @WebInitParam(name="tel", value="010-1111-2222")})           @WebInitParam을 이용해 여러 개의
                                                               매개변수를 설정할 수 있습니다.

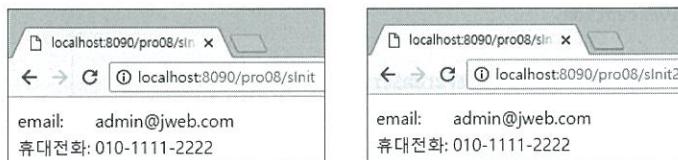
public class InitParamServlet extends HttpServlet{
    protected void doGet( HttpServletRequest request , HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType( "text/html;charset=utf-8" );
        PrintWriter out = response.getWriter();
        String email = getInitParameter("email");
        String tel = getInitParameter("tel");           설정한 매개변수의 name으로 값을
                                                       가져옵니다.

        out.print( "<html><body>" );
        out.print( "<table><tr>" );
        out.print( "<td>email: </td><td>" +email+ "</td></tr>" );
        out.print( "<tr><td>휴대전화: </td><td>" +tel+ "</td>" );
        out.print( "</tr></table></body></html>" );
    }
}

```

13. 브라우저에서 각각 매핑 이름 /sInit과 /sInit2로 요청합니다. 동일한 결과가 출력되는 것을 확인할 수 있습니다.

▼ 그림 8-48 매핑 이름 /sInit과 /sInit2로 요청한 결과



Note ≡ **web.xml을 이용한 방법**

지금은 잘 이용하지 않는 방법이지만 다음과 같이 web.xml을 이용해 매개변수를 설정할 수도 있습니다.

코드 8-25 pro08/WebContent/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <servlet>
        <servlet-name>sinit</servlet-name>
        <servlet-class>sec06.ex01.initParamServlet</servlet-class>
        <init-param>
            <param-name>email</param-name>
            <param-value>admin@jweb.com</param-value>
        </init-param>
        <init-param>
            <param-name>tel</param-name>
            <param-value>010-111-2222</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>sinit</servlet-name>
        <url-pattern>/first</url-pattern>
    </servlet-mapping>
</web-app>
```

→ <init-param> 태그 안에 매개변수를 설정합니다.

8.6

load-on-startup 기능 사용하기

서블릿은 브라우저에서 최초 요청 시 `init()` 메서드를 실행한 후 메모리에 로드되어 기능을 수행합니다. 따라서 최초 요청에 대해서는 실행 시간이 길어질 수밖에 없습니다. 이런 단점을 보완하기 위해 이용하는 기능이 `load-on-startup`입니다.

`load-on-startup`의 특징은 다음과 같습니다.

- 톰캣 컨테이너가 실행되면서 미리 서블릿을 실행합니다.
- 지정한 숫자가 0보다 크면 톰캣 컨테이너가 실행되면서 서블릿이 초기화됩니다.
- 지정한 숫자는 우선순위를 의미하며 작은 숫자부터 먼저 초기화됩니다.

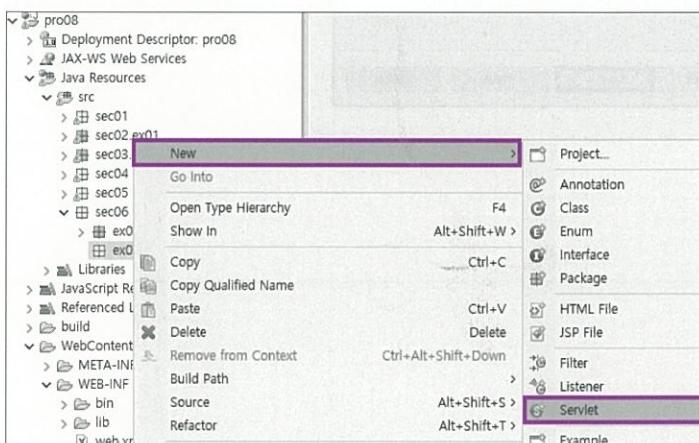
`load-on-startup` 기능을 구현하는 방법으로는 애너테이션을 이용하는 방법과 `web.xml`에 설정하는 방법이 있습니다.

8.6.1 애너테이션을 이용하는 방법

먼저 애너테이션을 이용해 `web.xml`에서 공통 메뉴를 읽어 오는 실습을 해보겠습니다(8.5.3절 참조).

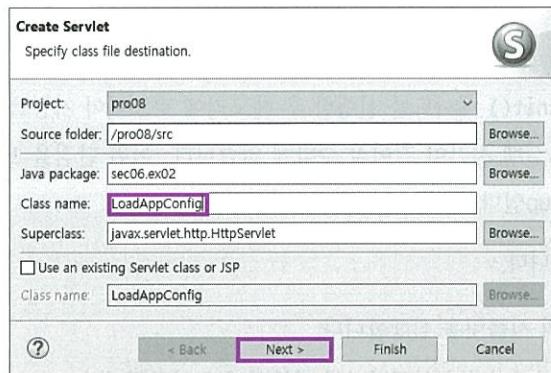
1. `sec06.ex02` 패키지를 생성하고 마우스 오른쪽 버튼을 클릭한 후 `New > Servlet`을 선택합니다.

▼ 그림 8-49 New > Servlet 선택



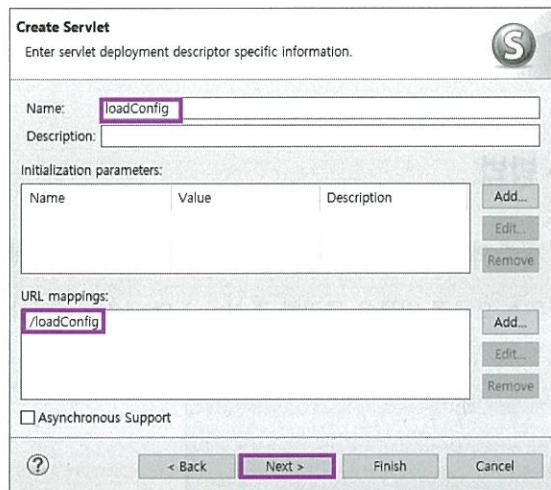
2. 클래스 이름으로 LoadAppConfig를 입력하고 Next를 클릭합니다.

▼ 그림 8-50 클래스 이름으로 LoadAppConfig 입력 후 Finish 클릭



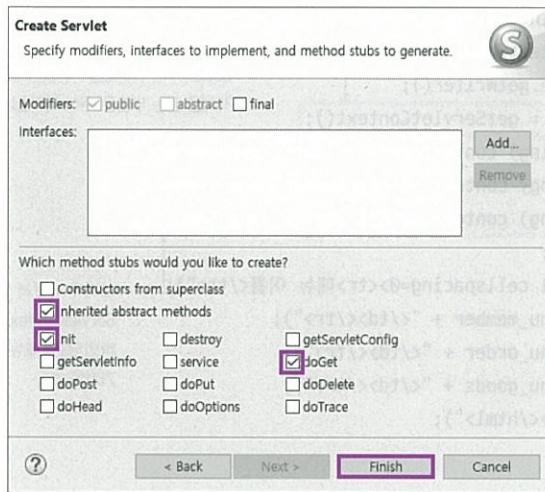
3. Name과 URL mappings을 loadConfig로 변경하고 Next를 클릭합니다.

▼ 그림 8-51 Name과 URL mappings 입력 후 Next 클릭



4. Inherited abstract methods, init, doGet 옵션 체크박스에 체크한 후 Finish를 클릭합니다.

▼ 그림 8-52 Inherited abstract methods, init, doGet 항목 체크 후 Finish 클릭



5. LoadAppConfig 클래스를 다음과 같이 작성합니다. 애너테이션으로 설정한 매개변수에 loadOnStartup 속성을 추가한 후 우선순위를 1로 설정합니다.

코드 8-26 pro08/src/sec06/ex02/LoadAppConfig.java

```
package sec06.ex02;
...
@WebServlet(name = "loadConfig", urlPatterns = { "/loadConfig" }, loadOnStartup = 1)
public class LoadAppConfig extends HttpServlet
{
    private ServletContext context;           ← 변수 context를 멤버 변수로 선언합니다.

    @Override
    public void init(ServletConfig config) throws ServletException
    {
        System.out.println("LoadAppConfig의 init 메서드 호출");
        context = config.getServletContext();   ← init() 메서드에서 ServletContext 객체를 얻습니다.
        String menu_member = context.getInitParameter("menu_member");   ← getInitParameter() 메서드로 web.xml 의 메뉴 정보를 읽어 들입니다.
        String menu_order = context.getInitParameter("menu_order");
        String menu_goods = context.getInitParameter("menu_goods");
        context.setAttribute("menu_member", menu_member);
        context.setAttribute("menu_order", menu_order);
        context.setAttribute("menu_goods", menu_goods);
    }

    @Override
}
```

loadOnStartup 속성을 추가하고 우선순위를 1로 설정합니다.

메뉴 정보를 ServletContext 객체에 바인딩합니다.

```

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();
    // ServletContext context = getServletContext(); doGet() 메서드 호출 시 ServletContext
    String menu_member = (String) context.getAttribute("menu_member"); 객체를 얻는 부분은 주석 처리합니다.
    String menu_order = (String) context.getAttribute("menu_order");
    String menu_goods = (String) context.getAttribute("menu_goods");
    out.print("<html><body>");
    out.print("<table border=1 cellspacing=0><tr>메뉴 이름</tr>"); 브라우저에서 요청 시
    out.print("<tr><td>" + menu_member + "</td></tr>"); ServletContext 객체의
    out.print("<tr><td>" + menu_order + "</td></tr>"); 바인딩된 메뉴 항목을
    out.print("<tr><td>" + menu_goods + "</td></tr>"); 가져옵니다.
    out.print("</table></body></html>");

}
}

```

우선순위는 양의 정수로 지정하며 숫자가 작으면 우선순위가 높으므로 먼저 실행합니다.

톰캣 실행 시 init() 메서드를 호출하면 getInitParameter() 메서드를 이용해 web.xml의 메뉴 정보를 읽어 들인 후 다시 ServletContext 객체에 setAttribute() 메서드로 바인딩합니다. 브라우저에서 요청하면 web.xml이 아니라 ServletContext 객체에서 메뉴 항목을 가져온 후 출력하기 때문에 파일에서 읽어 들여와 출력하는 것보다 빨리 출력할 수 있습니다.

6. 브라우저에서 /loadConfig로 최초 요청 시 기다리지 않고 바로 공통 메뉴를 출력합니다. 또한 톰캣을 실행하면 서블릿의 init() 메서드를 호출하므로 로그에 메시지가 출력됩니다.

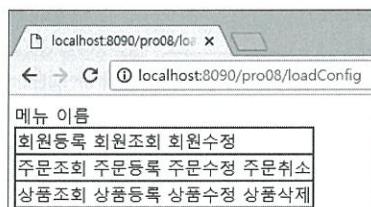
▼ 그림 8-53 톰캣 실행 시 init() 메서드의 메시지 출력

```

8월 29, 2018 3:16:18 오후 org.apache.jasper.JspApplicationContextImpl$JspContext
정보: At least one JAR was scanned for TLDs.
LoadAppConfig의 init 메서드 호출
8월 29, 2018 3:16:18 오후 org.apache.tomcat.util.threads.TaskThread
경고: Name = oracle Property maxActive is
8월 29, 2018 3:16:18 오후 org.apache.tomcat.util.threads.TaskThread

```

▼ 그림 8-54 브라우저에서 /loadConfig로 요청한 결과



8.6.2 web.xml에 설정하는 방법

이번에는 web.xml에 설정해서 사용하는 방법을 알아봅시다. 애너테이션 방법을 이용하기 전에는 web.xml에 설정해서 사용했습니다.

1. 다음과 같이 web.xml을 작성합니다. <servlet-name> 태그의 값은 반드시 서블릿을 생성할 때 name으로 지정한 값으로 설정해야 합니다.

```
코드 8-27 pro08/WebContent/WEB-INF/web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <servlet>
    <servlet-name>loadConfig</servlet-name> 애너테이션으로 서블릿 생성 시
    <servlet-class>sec06.ex02.LoadAppConfig</servlet-class> name 속성 값으로 설정합니다.
    <load-on-startup>1</load-on-startup> 패키지 이름까지 포함된 서블릿
  </servlet> 클래스 이름을 설정합니다.
  ...
</web-app>
```

2. 톰캣을 다시 실행한 후 브라우저에서 /loadConfig로 요청합니다. 실행 결과는 애너테이션으로 실행했을 때와 같습니다.

在地圖上標出你所知道的中國四大盆地。

在地圖上標出你所知道的中國四大高原。

在地圖上標出你所知道的中國四大平原。

在地圖上標出你所知道的中國四大丘陵。

在地圖上標出你所知道的中國四大盆地。

在地圖上標出你所知道的中國四大高原。

在地圖上標出你所知道的中國四大平原。

在地圖上標出你所知道的中國四大丘陵。

9^장

쿠키와 세션 알아보기

- 9.1 웹 페이지 연결 기능
- 9.2 <hidden> 태그와 URL Rewriting 이용해 웹 페이지 연동하기
- 9.3 쿠키를 이용한 웹 페이지 연동 기능
- 9.4 세션을 이용한 웹 페이지 연동 기능
- 9.5 encodeURL() 사용법
- 9.6 세션을 이용한 로그인 예제

9.1

웹 페이지 연결 기능

보통 웹 프로그램에서 사용되는 정보는 서블릿의 비즈니스 로직 처리 기능을 이용해 데이터베이스에서 가져옵니다. 그러나 동시 사용자 수가 많아지면 데이터베이스 연동 속도도 영향을 받게 되므로 정보의 종류에 따라 어떤 정보들은 클라이언트 PC나 서버의 메모리에 저장해두고 사용하면 좀 더 프로그램을 빠르게 실행시킬 수 있습니다. 이 장에서는 그 방법과 함께 서블릿이 로그인 시 사용자의 로그인 상태를 일정하게 유지시키는 기능에 대해 살펴보겠습니다.

9.1.1 세션 트래킹

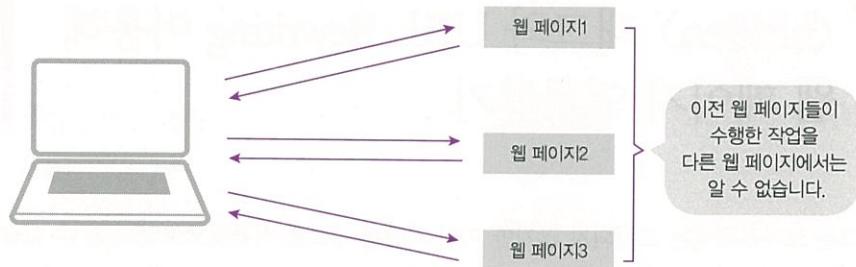
온라인 쇼핑몰을 이용하다 보면 메인 페이지에서 미리 로그인한 후 다른 웹 페이지에서 상품에 관한 댓글을 달거나 게시판에 상품평을 달곤 합니다. 글쓰기창에서는 따로 로그인하지 않아도 됩니다. 그러나 메인 페이지에서 미리 로그인하지 않고 새 글을 작성하려면 ‘로그인 후 이용하라’는 메시지가 나타납니다. 그러면 사용자는 로그인한 후 글쓰기창으로 이동하게 되죠.

쇼핑몰을 이용하는 일반 사용자들은 로그인 상태를 각각의 웹 페이지들이 자동적으로 알고 있을 것이라 생각합니다. 그러나 실제 HTTP 프로토콜 방식으로 통신하는 웹 페이지들은 서로 어떤 정보도 공유하지 않습니다.

사용자 입장에서 웹 페이지 사이의 상태나 정보를 공유하려면 프로그래머가 세션 트래킹(Session Tracking)이라는 웹 페이지 연결 기능을 구현해야 합니다.

그림 9-1은 HTTP 프로토콜로 각각의 웹 페이지를 요청해서 클라이언트의 브라우저에 표시해 주는 과정을 나타낸 것입니다.

▼ 그림 9-1 HTTP로 웹 페이지 요청해 보여주는 과정



HTTP 프로토콜은 서버-클라이언트 통신 시 **stateless** 방식으로 통신을 합니다. 즉, 브라우저에서 새 웹 페이지를 열면 기존의 웹 페이지나 서블릿에 관한 어떤 연결 정보도 새 웹 페이지에서는 알 수 없습니다.

정리하면 HTTP 프로토콜은 각 웹 페이지의 상태나 정보를 다른 페이지들과 공유하지 않는 **stateless** 방식으로 통신을 합니다. 따라서 웹 페이지나 서블릿끼리 상태나 정보를 공유하려면 웹 페이지 연결 기능, 즉 세션 트래킹을 이용해야 합니다.

웹 페이지를 연동하는 방법은 다음과 같습니다.

- <hidden> 태그: HTML의 <hidden> 태그를 이용해 웹 페이지들 사이의 정보를 공유합니다.
- URL Rewriting: GET 방식으로 URL 뒤에 정보를 붙여서 다른 페이지로 전송합니다.
- 쿠키: 클라이언트 PC의 Cookie 파일에 정보를 저장한 후 웹 페이지들이 공유합니다.
- 세션: 서버 메모리에 정보를 저장한 후 웹 페이지들이 공유합니다.

지금부터 이 방법을 하나씩 알아보겠습니다.

9.2

〈hidden〉 태그와 URL Rewriting 이용해 웹 페이지 연동하기

〈hidden〉 태그는 브라우저에는 표시되지 않지만 미리 저장된 정보를 서블릿으로 전송할 수 있습니다. 지금부터 〈hidden〉 태그를 이용해 클라이언트의 데이터를 서버에 보내는 예제를 수행해 보겠습니다.

9.2.1 〈hidden〉 태그를 이용한 세션 트래킹 실습

- 새 프로젝트 pro09를 만들고 sec01.ex01 패키지를 생성한 후 다음과 같이 LoginServlet 클래스 파일과 login.html을 준비합니다.

▼ 그림 9-2 실습 파일 위치



- login.html을 다음과 같이 작성합니다. 로그인창에서 ID와 비밀번호를 입력하면 미리 〈hidden〉 태그에 저장된 주소, 이메일, 휴대폰 번호를 서블릿으로 전송합니다.

코드 9-1 pro09/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">
        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">
        <input type="hidden" name="user_address" value="서울시 성북구" />
        <input type="hidden" name="user_email" value="test@gmail.com" />
        <input type="hidden" name="user_hp" value="010-111-2222" />
    </form>
</body>
</html>

```

〈hidden〉 태그의 value 속성에 주소, 이메일, 전화번호를 저장한 후 서블릿으로 전송합니다.

3. LoginServlet 클래스를 다음과 같이 작성합니다. getParameter() 메서드를 이용해 전송된 회원 정보를 가져온 후 브라우저로 다시 출력합니다.

Tip 이 책에서 제공하는 예제 소스에서 프로젝트를 불러와서 사용하는 경우에는 매핑 이름이 충돌하지 않도록 sec01.ex02 패키지의 LoginServlet.java에 있는 @WebServlet("/login") 명령을 주석 처리하세요.

코드 9-2 pro09/src/sec01/ex01/LoginServlet.java

```

package sec01.ex01;
...
@WebServlet("/login")
public class LoginServlet extends HttpServlet{
    public void init(){
        System.out.println("init 메서드 호출");
    }

    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();
        String user_id = request.getParameter("user_id");
        String user_pw = request.getParameter("user_pw");
        String user_address=request.getParameter("user_address");
        String user_email=request.getParameter("user_email");
        String user_hp=request.getParameter("user_hp");

        <hidden> 태그로 전송된 값을
        getParameter() 메서드를 이용
        해 가져옵니다.

        String data="안녕하세요!<br> 로그인하셨습니다.<br><br>";
        data+="<html><body>";
    }
}

```

```

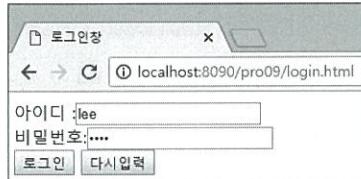
        data+="아이디 : "+user_id ;
        data+="<br>";
        data+="패스워드 : "+user_pw;
        data+="<br>";
        data+="주소 : "+user_address;
        data+="<br>";
        data+="email : "+user_email;
        data+="<br>";
        data+="휴대전화 : "+user_hp;
        data+="</body></html>";
        out.print(data);
    }

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

```

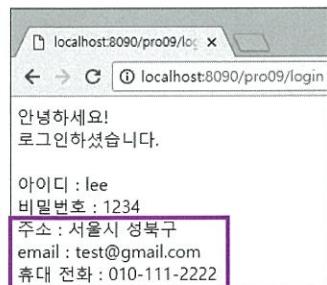
4. <http://localhost:8090/pro09/login.html>로 요청하고 ID와 비밀번호를 입력한 후 서블릿으로 전송합니다.

▼ 그림 9-3 로그인창 요청



5. <hidden> 태그로 전송된 데이터도 출력합니다.

▼ 그림 9-4 주소, 이메일, 전화번호를 서블릿으로 전송



9.2.2 URL Rewriting을 이용한 세션 트래킹 실습

이번에는 URL Rewriting을 이용해 로그인창에서 입력 받은 ID와 비밀번호를 다른 서블릿으로 전송하여 로그인 상태를 확인해 보겠습니다.

- 새로운 패키지를 만들고 LoginServlet, SecondServlet 클래스 파일을 준비합니다.

▼ 그림 9-5 실습 파일 위치



- LoginServlet 클래스를 다음과 같이 작성합니다. 로그인창에서 입력 받은 ID와 비밀번호를 <a> 태그의 두 번째 서블릿으로 보내기를 클릭하면 로그인창에서 입력한 ID와 비밀번호 그리고 다른 정보들을 GET 방식을 이용해 두 번째 서블릿으로 전송합니다.

코드 9-3 pro09/src/sec01/ex02/LoginServlet.java

```
package sec01.ex02;
...
@WebServlet("/login")
public class LoginServlet extends HttpServlet{
    public void init(){
        System.out.println("init 메서드 호출");
    }
    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();
        String user_id = request.getParameter("user_id");
        String user_pw = request.getParameter("user_pw");
        String user_address=request.getParameter("user_address");
```

코드 9-2에서 주석 처리했다면 다시 주석 처리를 해제합니다.

```

String user_email=request.getParameter("user_email");
String user_hp=request.getParameter("user_hp");

String data="안녕하세요!<br>로그인하셨습니다.<br><br>";
data+="<body>";
data+="아이디 : "+user_id ;
data+="  
";
data+="패스워드 : "+user_pw;
data+="  
";
data+="주소 : "+user_address;
data+="  
";
data+="email : "+user_email;
data+="  
";
data+="휴대전화 : "+user_hp;
data+="  
";
out.print(data);

user_address=URLEncoder.encode(user_address, "utf-8");
out.print("<a href='/pro09/second?user_id="+user_id+"&user_pw="+user_pw+
        "&user_address="+user_address+">두 번째 서블릿으로 보내기</a>");
data="</body></html>";
out.print(data);
}

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

```

GET 방식으로 한글을 전송하기 위해 인코딩합니다.

〈a〉 태그를 이용해 링크 클릭 시 서블릿 /second로 다시 로그인 정보를 전송합니다.

3. SecondServlet 클래스를 다음과 같이 작성합니다. 첫 번째 서블릿에서 전송한 데이터 중 ID와 비밀번호를 가져왔으면 이미 첫 번째 서블릿에서 로그인한 것이므로 로그인 상태를 유지하도록 해줍니다.

코드 9-4 pro09/src/sec01/ex02/SecondServlet.java

```

package sec01.ex02;

...
@WebServlet("/second")
public class SecondServlet extends HttpServlet{
    public void init(){
        System.out.println("init 메서드 호출");
    }
}

```

```

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 user_id = request.getParameter("user_id");
    String user_pw = request.getParameter("user_pw");
    String user_address=request.getParameter("user_address");

    out.println("<html><body>");
    if(user_id!=null && user_id.length()!=0){
        out.println("이미 로그인 상태입니다!<br><br>");
        out.println("첫 번째 서블릿에서 넘겨준 아이디: " + user_id +"<br>");
        out.println("첫 번째 서블릿에서 넘겨준 비밀번호: "+user_pw +"<br>");
        out.println("첫 번째 서블릿에서 넘겨준 주소: "+user_address +"<br>");
        out.println("</body></html>");
    }else{
        out.println("로그인 하지 않았습니다.<br><br>");
        out.println("다시 로그인하세요!!<br>");
        out.println("<a href='/pro09/login.html'>로그인창으로 이동하기 </a>");
    }
}

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

```

첫 번째 서블릿에서 전송한
로그인 정보를 가져옵니다.

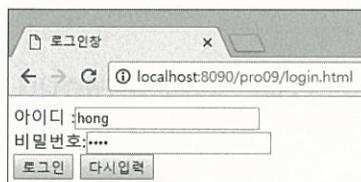
첫 번째 서블릿의 ID 정보를 이용해
로그인 상태를 유지합니다.

로그인창을 거치지 않고 바로 요청한 경우에는
로그인창으로 다시 이동하도록 안내합니다.

4. <hidden> 태그와 URL Rewriting 방식으로 데이터를 전송한 결과를 볼까요?

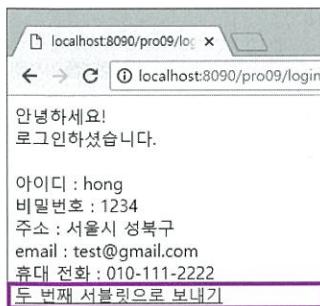
<http://localhost:8090/pro09/login.html>로 요청한 후 ID와 비밀번호를 입력하고 첫 번째 서블릿으로 전송합니다.

▼ 그림 9-6 로그인창 요청



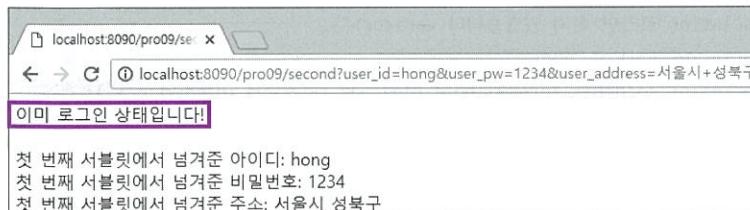
5. 첫 번째 서블릿에서 전달받은 로그인 정보를 출력한 후 두 번째 서블릿으로 보내기를 클릭합니다.

▼ 그림 9-7 입력한 로그인 정보가 표시되면 두 번째 서블릿으로 보내기 클릭



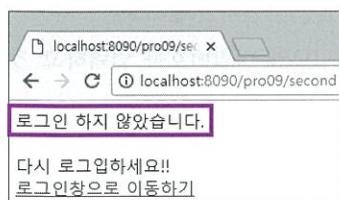
6. 두 번째 서블릿에서 현재 로그인 상태와 회원 정보를 출력합니다.

▼ 그림 9-8 로그인창에서 입력한 정보가 두 번째 서블릿에 전달됨



7. 만약 브라우저에서 로그인창을 거치지 않고 바로 서블릿 /second를 요청하면 “로그인 하지 않았습니다.”라는 상태 안내 문구와 함께 ‘로그인창으로 이동하기’가 표시됩니다.

▼ 그림 9-9 로그인창을 거치지 않고 바로 서블릿 /second로 요청한 경우



지금까지 `<hidden>` 태그와 GET 방식으로 웹 페이지들을 연동하는 방법을 알아봤습니다. 그런데 이 방법은 여러 가지 단점이 있습니다. 일단 웹 페이지가 많아지면 일일이 로그인 상태를 확인하기 위해 로그인 정보를 다른 웹 페이지로 전송해야 한다는 것입니다. 그리고 ID와 비밀번호를 GET 방식으로 전송하므로 브라우저에 노출되어 보안상으로도 좋지 않습니다. 또한 전송할 수 있는 데이터 용량에도 한계가 있습니다. 따라서 이 방식은 웹 페이지 사이에 간단한 정보 정도를 공유할 때만 사용하는 것이 좋습니다.

9.3

쿠키를 이용한 웹 페이지 연동 기능

이번에는 웹 페이지끼리 정보를 공유하는 기능 중 쿠키를 이용하는 기능에 대해 알아보겠습니다.

쿠키(Cookie)란 웹 페이지들 사이의 공유 정보를 클라이언트 PC에 저장해 놓고 필요할 때 여러 웹 페이지들이 공유해서 사용할 수 있도록 매개 역할을 하는 방법입니다.

쿠키의 특징은 다음과 같습니다.

- 정보가 클라이언트 PC에 저장됩니다.
- 저장 정보 용량에 제한이 있습니다(파일 용량은 4kb).
- 보안이 취약합니다.
- 클라이언트 브라우저에서 사용 유무를 설정할 수 있습니다.
- 도메인당 쿠키가 만들어집니다(웹 사이트당 하나의 쿠키가 만들어집니다).

☞ (우선 표기) ✕

쿠키는 클라이언트 PC에 정보를 저장해서 사용하므로 보안에 취약합니다. 따라서 쿠키를 이용한 방법은 주로 보안과 무관한 경우에 한해 사용합니다. 예를 들어 우리가 웹 페이지를 방문했을 때 어떤 팝업창이 나타나면 ‘오늘은 더 이상 보지 않기’를 체크하는데, 이처럼 팝업창이 나타나지 않게 하는 경우 등에 사용합니다.

쿠키는 다음과 같이 두 종류로 나눌 수 있습니다.

▼ 표 9-1 쿠키의 종류

속성	Persistence 쿠키	Session 쿠키
생성 위치	파일로 생성	브라우저 메모리에 생성
종료 시기	쿠키를 삭제하거나 쿠키 설정 값이 종료된 경우	브라우저를 종료한 경우
최초 접속 시 전송 여부	최초 접속 시 서버로 전송	최초 접속 시 서버로 전송되지 않음
용도	로그인 유무 또는 팝업창을 제한할 때	사이트 접속 시 Session 인증 정보를 유지할 때

Persistence 쿠키는 클라이언트에 파일로 정보를 저장하는 기능을 합니다. 파일로 생성된 쿠키는 사용자가 만료 시간을 지정할 수 있는 반면에 Session 쿠키는 브라우저가 사용하는 메모리에 생성되는 쿠키입니다. 브라우저가 종료되면 메모리의 Session 쿠키도 자동으로 소멸됩니다. Session 쿠키는 다음 절에서 배우는 Session 기능과 같이 사용됩니다.

그럼 실제로 클라이언트 PC에서 쿠키 파일이 생성되는 위치를 확인해 볼까요? 이미 여러 웹 사이트에서 사용하는 쿠키가 생성된 것을 볼 수 있을 것입니다.

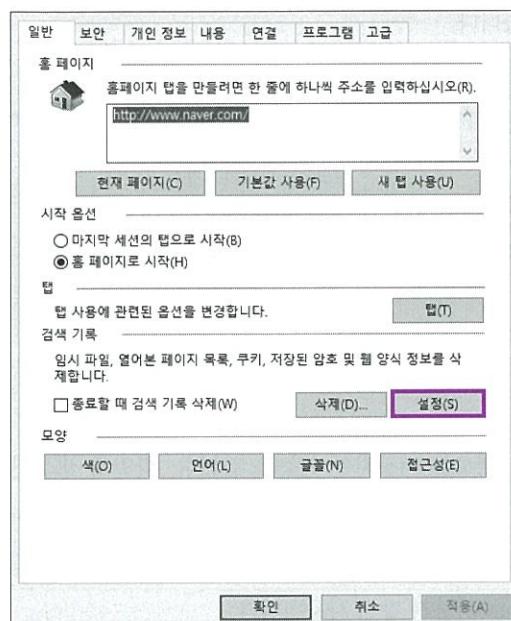
- 원도 탐색기를 열고 C:\Users\사용자\AppData\Local\Google\Chrome\User Data\Default\Cache로 이동하면 크롬에서 사용하는 쿠키 파일이 보일 것입니다.

▼ 그림 9-10 크롬에서 사용하는 쿠키 파일 위치



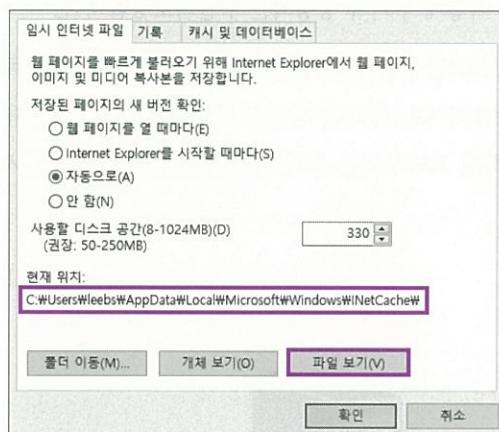
- 인터넷 익스플로러의 경우 브라우저 상단 메뉴에서 도구 > 인터넷 옵션을 선택한 후 팝업창에서 설정을 클릭합니다.

▼ 그림 9-11 익스플로러 메뉴에서 도구 > 인터넷 옵션 선택



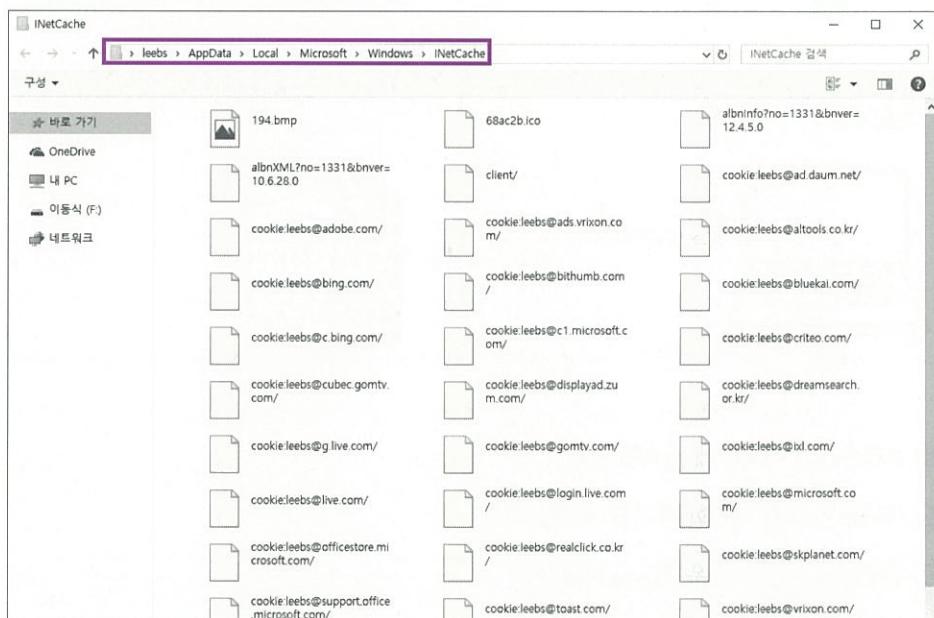
3. ‘임시 인터넷 파일’ 탭에서 파일 보기를 클릭합니다.

▼ 그림 9-12 파일 보기 클릭



4. 인터넷 익스플로러에서 사용하는 쿠키 파일이 저장되어 있는 폴더가 나타납니다.

▼ 그림 9-13 익스플로러에서 사용하는 쿠키 파일들



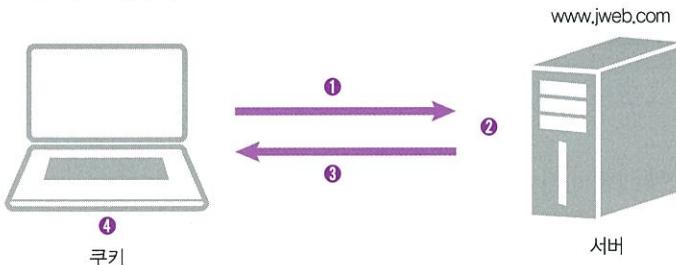
9.3.1 쿠키 기능 실행 과정

클라이언트 브라우저가 웹 서버에 요청하면 어떻게 쿠키가 생성되는지 살펴보겠습니다.

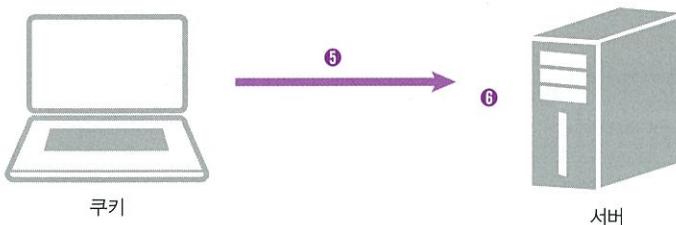
그림 9-14를 보면 브라우저에서 웹 사이트(www.jweb.com)에 최초 접속하면 웹 서버에서 쿠키를 생성해 클라이언트로 전송합니다. 그리고 브라우저는 쿠키를 파일로 저장합니다. 이후 다시 접속해 서버가 브라우저에게 쿠키 전송을 요청하면 브라우저는 쿠키 정보를 서버에 전송하고 서버는 쿠키 정보를 이용해 작업을 합니다.

▼ 그림 9-14 쿠키 생성 과정

최초 사이트 접속 시



재접속 시



- ① 브라우저로 사이트에 접속합니다.
- ② 서버는 정보를 저장한 쿠키를 생성합니다.
- ③ 생성된 쿠키를 브라우저로 전송합니다.
- ④ 브라우저는 서버로부터 받은 쿠키 정보를 쿠키 파일에 저장합니다.
- ⑤ 브라우저가 다시 접속해 서버가 브라우저에게 쿠키 전송을 요청하면 브라우저는 쿠키 정보를 서버에 넘겨줍니다.
- ⑥ 서버는 쿠키 정보를 이용해 작업을 합니다.

9.3.2 쿠키 API

실제로 서블릿에서 쿠키 기능 사용 시 이와 관련된 API에 대해 알아보겠습니다. 쿠키는 `Cookie` 클래스 객체를 생성하여 정보를 저장한 후 서버에서 클라이언트로 전송해 파일로 저장됩니다. 쿠키 관련 API의 특징은 다음과 같습니다.

- `javax.servlet.http.Cookie`를 이용합니다.
- `HttpServletResponse`의 `addCookie()` 메서드를 이용해 클라이언트 브라우저에 쿠키를 전송한 후 저장합니다.
- `HttpServletRequest`의 `getCookie()` 메서드를 이용해 쿠키를 서버로 가져옵니다.

표 9-2는 `Cookie` 클래스에서 제공하는 여러 가지 메서드의 기능을 정리한 것입니다.

▼ 표 9-2 쿠키 클래스의 여러 가지 메서드

메서드	설명
<code>getComment()</code>	쿠키에 대한 설명을 가져옵니다.
<code>getDomain()</code>	쿠키의 유효한 도메인 정보를 가져옵니다.
<code>getMaxAge()</code>	쿠키 유효 기간을 가져옵니다.
<code>getName()</code>	쿠키 이름을 가져옵니다.
<code>getPath()</code>	쿠키의 디렉터리 정보를 가져옵니다.
<code>getValue()</code>	쿠키의 설정 값을 가져옵니다.
<code>setComment(String)</code>	쿠키에 대한 설명을 설정합니다.
<code>setDomain(String)</code>	쿠키의 유효한 도메인을 설정합니다.
<code>setMaxAge(int)</code>	쿠키 유효 기간을 설정합니다.
<code>setValue(String)</code>	쿠키 값을 설정합니다.
<code>setPath(String)</code>	쿠키의 디렉터리 정보를 설정합니다.

쿠키 생성 시 `setMaxAge()` 메서드 인자 값의 종류를 지정해서 파일에 저장하는 Persistence 쿠키를 만들거나 메모리에만 저장하는 Session 쿠키를 만들 수 있습니다. 즉, `setMaxAge()` 메서드를 이용한 쿠키 저장 방식은 다음 두 가지로 나눌 수 있습니다.

인자 값으로 음수나 `setMaxAge()` 메서드를 사용하지 않고 쿠키를 만들면 Session 쿠키로 저장됩니다.

인자 값으로 양수를 지정하면 Persistence 쿠키로 저장됩니다.

9.3.3 서블릿에서 쿠키 사용하기

서블릿에서 쿠키 API를 이용해 직접 쿠키를 만들어 보겠습니다.

1. GetCookieValue, SetCookieValue 클래스 파일을 준비합니다.

▼ 그림 9-15 실행 파일 위치



2. SetCookieValue 클래스를 다음과 같이 작성합니다. Cookie 객체를 생성한 후 쿠키 이름을 cookieTest로 값을 저장합니다. 그리고 setMaxAge() 메서드에 쿠키 유효 시간을 24시간으로 설정합니다. 그런 다음 response의 addCookie() 메서드를 이용해 생성된 쿠키를 브라우저로 전송합니다.

코드 9-5 pro09/src/sec02/ex01/SetCookieValue.java

```
package sec02.ex01;  
...  
@WebServlet("/set")  
public class SetCookieValue extends HttpServlet{  
    protected void doGet(HttpServletRequest request,HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=utf-8");  
        PrintWriter out=response.getWriter();  
        Date d=new Date();  
        Cookie c=new Cookie("cookieTest",URLEncoder.encode("JSP프로그래밍입니다."  
            ,"utf-8"));  
        c.setMaxAge(24*60*60);  
        response.addCookie(c);  
        out.println("현재시간 : "+d);  
        out.println("문자열을 Cookie에 저장합니다.");  
    }  
}
```

Cookie 객체를 생성한 후 cookieTest 이름으로
한글 정보를 인코딩해서 쿠키에 저장합니다.

유효 기간을 설정합니다.

생성된 쿠키를 브라우저로 전송합니다.

3. GetCookieValue 클래스를 다음과 같이 작성합니다. 두 번째 서블릿 요청 시에는 request의 getCookies() 메서드를 호출해 브라우저로부터 쿠키를 전달받습니다. 그리고 전달된 쿠키에서 저장할 때 사용한 이름인 cookieTest로 검색해 값을 가져옵니다.

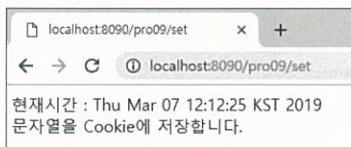
코드 9-6 pro09/src/sec02/ex01/GetCookieValue.java

```
package sec02.ex01;
...
@WebServlet("/get")
public class GetCookieValue extends HttpServlet{
    protected void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out=response.getWriter(); [request의 getCookies() 메서드를 호출해 브라우저에게 쿠키 정보를 요청한 후 쿠키 정보를 배열로 가져옵니다.]
        Cookie[] allValues=request.getCookies();
        for(int i=0; i<allValues.length;i++){
            if(allValues[i].getName().equals("cookieTest")){
                out.println("<h2>Cookie 값 가져오기 : "+URLDecoder.decode(allValues[i].getValue(),"utf-8"));
            }
        }
    }
}
```

[배열에서 저장할 때 사용한 쿠키 이름인 cookieTest로 검색해 쿠키 값을 가져옵니다.]

4. 우선 set으로 첫 번째 서블릿을 요청합니다. 쿠키에 cookieTest 이름으로 문자열을 저장합니다.

▼ 그림 9-16 /set으로 쿠키에 데이터 저장



5. get으로 두 번째 서블릿을 요청하여 cookieTest로 쿠키 값을 가져와 브라우저에 출력합니다.

▼ 그림 9-17 /get으로 쿠키 데이터 얻기

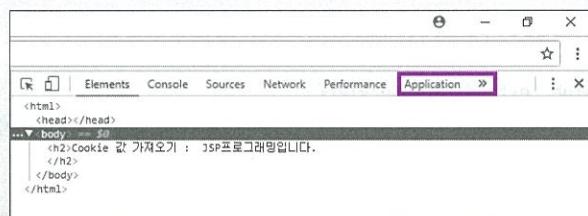


Note ≡ 쿠키 생성 상태 확인하기

다음은 클라이언트 쿠키의 생성 상태를 크롬에서 확인하는 방법입니다.

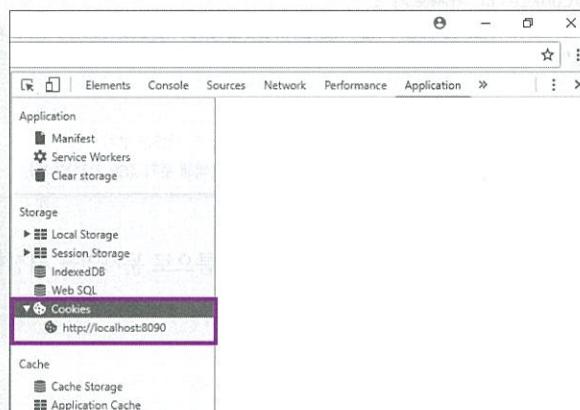
1. 크롬 브라우저를 실행하고 F12를 눌러 디버그창을 나타냅니다. 그리고 상단 메뉴 바에서 Application을 클릭합니다.

▼ 그림 9-18 Application 클릭



2. 왼쪽 메뉴에서 Cookies를 선택한 후 하위에 있는 http://localhost:8090을 클릭합니다.

▼ 그림 9-19 메뉴에서 Cookies > http://localhost:8090 클릭



3. 현재 애플리케이션에서 사용하고 있는 쿠키 정보가 표시됩니다.

▼ 그림 9-20 현재 애플리케이션에서 사용 중인 쿠키 정보 표시

Name	Value	Do...	...	S...	S...	S...
cookieTest	JSP%ED%94%84...	local...	...	86		

9.3.4 세션 쿠키 사용하기

다음은 쿠키를 파일에 저장하는 것이 아닌, 브라우저가 사용하는 메모리에 저장하는 Session 쿠키를 만들어 보겠습니다.

1. 다음과 같이 Cookie의 setMaxAge() 메서드를 이용해 유효 시간을 -1로 설정하면 세션 쿠키가 생성됩니다.

코드 9-7 pro09/src/sec02/ex01/SetCookieValue.java

```
package sec02.ex01;
...
@WebServlet("/set")
public class SetCookieValue extends HttpServlet{
    protected void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out=response.getWriter();
        Date d=new Date();
        Cookie c=new Cookie("cookieTest",URLEncoder.encode("JSP프로그래밍입니다. ","utf-8"));
        //c.setMaxAge(24*60*60); ────────── 주석 처리합니다.
        c.setMaxAge(-1); ────────── 유효 시간을 음수로 지정하여 Session
        response.addCookie(c); ────────── 쿠키를 만듭니다.
        out.println("현재 시간 : "+d);
        out.println("현재 시간을 Cookie로 저장합니다.");
    }
}
```

2. 톰캣을 재실행합니다. 출력 결과는 앞의 쿠키 예제(9.3.3절)와 같습니다.

9.3.5 쿠키 이용해 팝업창 제한하기

쿠키를 이용해 팝업창을 제한하는 기능을 구현해 보겠습니다. 팝업창 제어는 서버에서 쿠키를 다루지 않고 자바스크립트를 이용해 쿠키에 직접 접근합니다.

1. popUp.html, popupTest.html 이렇게 두 개의 html 파일을 준비합니다.

▼ 그림 9-21 실습 파일 위치



2. 먼저 popupTest.html을 다음과 같이 작성합니다. 웹 페이지가 브라우저에 로드될 때 pageLoad() 함수를 호출한 후 쿠키에 접근해 팝업창 관련 정보를 가져옵니다. getCookieValue() 함수를 호출하여 쿠키 이름 notShowPop의 값이 true가 아니면 팝업창을 나타내고, notShowPop의 값이 true면 팝업창을 나타내지 않습니다.

코드 9-8 pro09/WebContent/popupTest.html

```
<html>
<head>
<meta charset="UTF-8">
<title> 자바스크립트에서 쿠키 사용 </title>
<script type = "text/javascript">
window.onload = pageLoad; ← 브라우저에 웹 페이지가 로드될 때 pageLoad() 함수를 호출하여 실행합니다.

function pageLoad(){
    notShowPop =getCookieValue(); ← notShowPop의 쿠키 값을 getCookieValue()를 호출하여 얻습니다.

    if(notShowPop!="true"){
        window.open("popUp.html","pop","width=400,height=500,history=no,
                    resizable=no,status=no,scrollbars=yes,menubar=no");
    }
}

function getCookieValue(){
    var result="false";
    if(document.cookie != ""){
        cookie = document.cookie.split(";");
        for(var i=0; i<cookie.length;i++){
            element=cookie[i].split("=");
            value=element[0];
            value=value.replace(/\s*/,'');
            if(value=="notShowPop"){ ← 정규식을 이용해 쿠키 이름 문자열의 공백(\s)을 제거합니다.

                result= element[1]; ← 쿠키 이름이 notShowPop이면 해당하는 쿠키 값을 가져와 반환합니다.
            }
        }
    }
}
```

```

        }
    }
    return result;
}
function deleteCookie(){
    document.cookie = "notShowPop=" + "false" + ";path=/; expires=-1" ;
}
</script>
</head>
<body>
<form>
    <input type=button value="쿠키삭제" onClick="deleteCookie()" >
</form>
</body>
</html>

```

'쿠키삭제' 클릭 시 호출됩니다.
notShowPop 쿠키 값을 false로 설정합니다.

쿠키를 삭제합니다.

3. popUp.html에서는 오늘 더 이상 팝업창 띄우지 않기에 체크하면 자바스크립트 함수인 setPopUpStart() 함수를 호출해 notShowPop의 값을 true로 설정하여 재접속 시 팝업창을 나타내지 않도록 설정합니다.

코드 9-9 pro09/WebContent/popUp.html

```

<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
function setPopUpStart(obj){
    if(obj.checked==true){
        var expireDate = new Date();
        var days = 1;
        expireDate.setDate(expireDate.getDate() + days);
        document.cookie ="notShowPop=" +"true" + ";path=/; expires=" +
                        expireDate.toGMTString();
        window.close();
    }
}
</script>
</head>
<body>
    알림 팝업창입니다.
    <br><br><br><br><br><br>
    <form>
        <input type=checkbox onClick="setPopUpStart(this)">오늘 더 이상 팝업창 띄우지 않기
    </form>
</body>
</html>

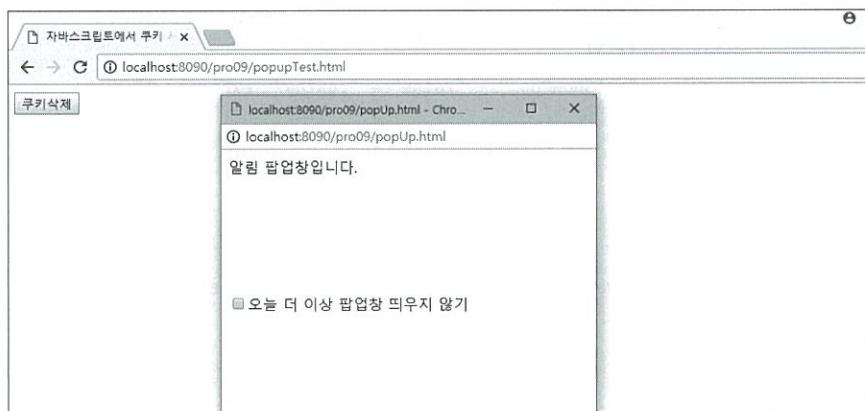
```

쿠키 유효 시간을 하루로 설정합니다.

오늘 더 이상 팝업창 띄우지 않기에 체크하면
notShowPop 쿠키 값을 true로 설정하여 재접속
시 팝업창을 나타내지 않습니다.

4. 브라우저에 최초 접속 시 팝업창을 나타냅니다.

▼ 그림 9-22 최초 요청 시 팝업창 나타내기



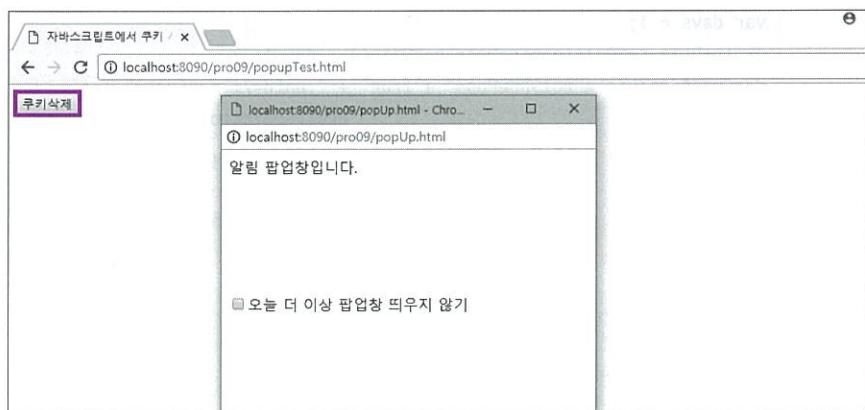
5. 오늘 더 이상 팝업창 띄우지 않기에 체크하고 재요청하면 더 이상 팝업창이 나타나지 않습니다.

▼ 그림 9-23 오늘 더 이상 팝업창 띄우지 않기 체크 후 재요청 결과



6. 쿠키삭제를 클릭한 후 재요청하면 다시 팝업창이 나타납니다.

▼ 그림 9-24 쿠키삭제 클릭 후 재요청 결과



9.4

세션을 이용한 웹 페이지 연동 기능

이번에는 세션을 이용해 웹 페이지들을 연동하는 방법을 알아보겠습니다.

세션 역시 웹 페이지들 사이의 공유 정보를 서버에 저장해 두고 웹 페이지들을 매개해 주는 방법이라는 점에서는 쿠키와 같습니다. 하지만 쿠키는 사용 시 웹 페이지들의 정보가 클라이언트 PC에 저장되므로 정보가 쉽게 노출될 수 있다는 단점이 있는 반면, 세션은 서버의 메모리에 생성되어 정보를 저장합니다. 따라서 웹 페이지에서 사용되는 정보 중에 로그인 정보처럼 보안이 요구되는 정보는 대부분 세션을 이용합니다.

세션은 각 브라우저당 한 개, 즉 사용자당 한 개가 생성됩니다. 사용자의 로그인 상태나 쇼핑몰의 장바구니 담기 기능 같은 정보를 해당 브라우저의 세션에 저장해 두고 사용하면 편리합니다.

세션의 특징은 다음과 같습니다.

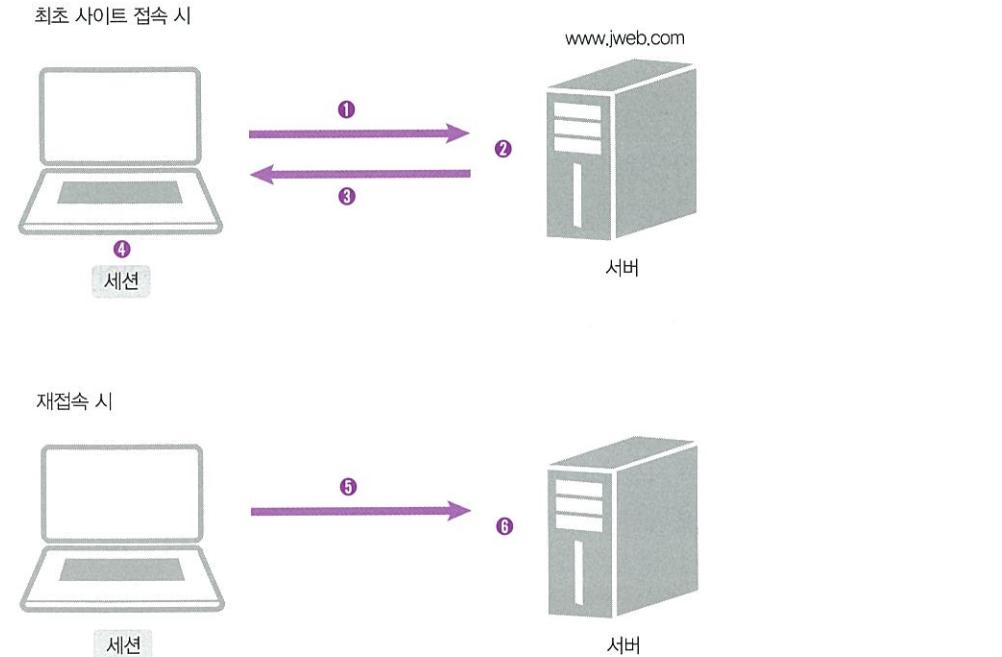
- 정보가 서버의 메모리에 저장됩니다.
- 브라우저의 세션 연동은 세션 쿠키를 이용합니다.
- 쿠키보다 보안에 유리합니다.
- 서버에 부하를 줄 수 있습니다.
- 브라우저(사용자)당 한 개의 세션(세션 id)이 생성됩니다.
- 세션은 유효 시간을 가집니다(기본 유효 시간은 30분입니다).
- 로그인 상태 유지 기능이나 쇼핑몰의 장바구니 담기 기능 등에 주로 사용됩니다.

9.4.1 세션 기능 실행 과정

클라이언트의 브라우저가 서버에 최초 접속하면 서버의 서블릿은 세션 객체를 생성한 후 세션 객체에 대한 세션 id를 브라우저에 전송합니다. 그러면 브라우저는 이 세션 id를 브라우저가 사용하는 세션 쿠키에 저장합니다. 즉, 서버로부터 전송된 세션 id도 쿠키이며, 쿠키 이름은 jsessionId입니다.

그리고 재접속하여 세션 쿠키에 저장된 세션 id(jsessionId)를 다시 서버로 전송하면 서버에서는 전송된 세션 id를 이용해 브라우저의 세션 객체에 접근하여 브라우저에 대한 작업을 수행합니다.

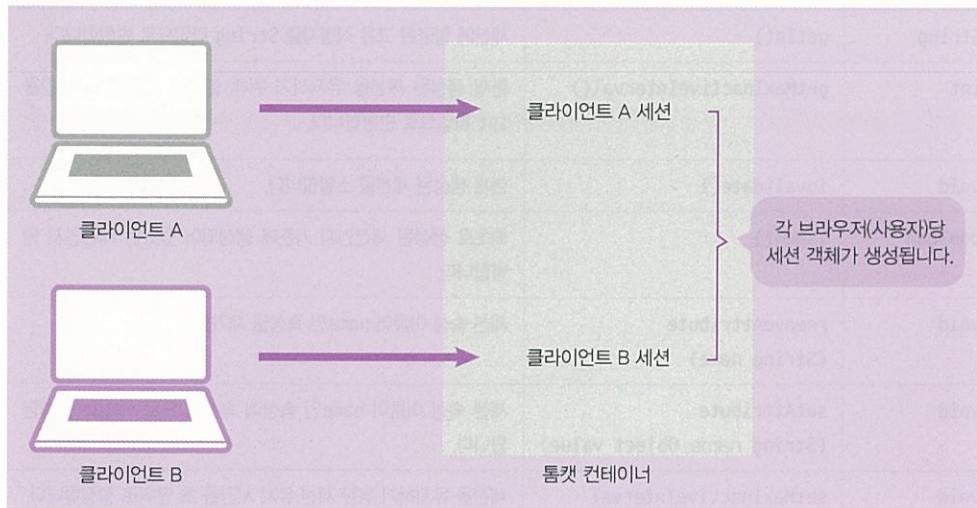
▼ 그림 9-25 세션 실행 과정



- ① 브라우저로 사이트에 접속합니다.
- ② 서버는 접속한 브라우저에 대한 세션 객체를 생성합니다.
- ③ 서버는 생성된 세션 id를 클라이언트 브라우저에 응답합니다.
- ④ 브라우저는 서버로부터 받은 세션 id를 브라우저가 사용하는 메모리의 세션 쿠키에 저장합니다(쿠키 이름은 jsessionId).
- ⑤ 브라우저가 재접속하면 브라우저는 세션 쿠키에 저장된 세션 id를 서버에 전달합니다.
- ⑥ 서버는 전송된 세션 id를 이용해 해당 세션에 접근하여 작업을 수행합니다.

세션의 중요한 특징은 브라우저당 한 개씩 생성된다는 점입니다. 그러므로 브라우저가 서버에 접속하여 브라우저에 저장된 세션 id(jsessionId)를 전송하면 서버는 그 값을 이용해서 해당 브라우저에 대한 세션을 구분하고 각 브라우저에 대한 세션 작업을 수행합니다.

▼ 그림 9-26 각 브라우저에 대한 세션 생성



9.4.2 세션 API의 특징과 기능

서블릿에서 세션을 이용하려면 HttpSession 클래스 객체를 생성해서 사용해야 합니다. HttpSession 객체는 HttpServletRequest의 getSession() 메서드를 호출해서 생성합니다.

세션을 얻는 getSession() 메서드로는 다음과 같은 것들이 있습니다.

- getSession(): 기존의 세션 객체가 존재하면 반환하고, 없으면 새로 생성합니다.
- getSession(true): 기존의 세션 객체가 존재하면 반환하고, 없으면 새로 생성합니다.
- getSession(false): 기존의 세션 객체가 존재하면 반환하고, 없으면 null을 반환합니다.

또한 HttpSession 클래스에서 제공하는 세션 기능 관련 메서드는 표 9-3에 정리했습니다.

▼ 표 9-3 HttpSession 클래스의 여러 가지 메서드

반환 타입	메서드	설명
Object	getAttribute (String name)	속성 이름이 name인 속성 값을 Object 타입으로 반환합니다. 해당되는 속성 이름이 없을 경우 null 값을 반환합니다.
Enumeration	getAttributeNames()	세션 속성 이름들을 Enumeration 객체 타입으로 반환합니다.
long	getCreationTime()	1970년 1월 1일 0시 0초를 기준으로 현재 세션이 생성된 시간 까지 경과한 시간을 계산하여 1/1000초 값으로 반환합니다.

반환 타입	메서드	설명
String	getId()	세션에 할당된 고유 식별자를 String 타입으로 반환합니다.
int	getMaxInactiveInterval()	현재 생성된 세션을 유지하기 위해 설정된 세션 유지 시간을 int 타입으로 반환합니다.
void	invalidate()	현재 생성된 세션을 소멸합니다.
boolean	isNew()	최초로 생성된 세션인지 기준에 생성되어 있었던 세션인지 판별합니다.
void	removeAttribute (String name)	세션 속성 이름이 name인 속성을 제거합니다.
void	setAttribute (String name, Object value)	세션 속성 이름이 name인 속성에 속성 값으로 value를 할당합니다.
void	setMaxInactiveInterval (int interval)	세션을 유지하기 위한 세션 유지 시간을 초 단위로 설정합니다.

9.4.3 서블릿에서 세션 API 이용하기

그럼 지금부터는 세션 API를 이용해 직접 세션을 생성해 보겠습니다.

1. 다음과 같이 세션 테스트를 위한 실습 파일인 SessionTest 클래스를 준비합니다.

▼ 그림 9-27 실습 파일 위치



2. SessionTest 클래스를 다음과 같이 작성합니다. request의 인자 없는 getSession() 메서드를 호출하여 세션이 없으면 새로 생성하고, 세션이 있으면 기존 세션을 가져옵니다. 또한 세션 객체의 getMaxInactiveInterval()를 호출하여 생성된 세션의 유효 시간을 가져옵니다.

코드 9-10 pro09/src/sec03/ex01/SessionTest.java

```
package sec03.ex01;
```

```
...
```

```

@WebServlet("/sess")
public class SessionTest extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession();
        out.println("세션 아이디: "+session.getId()+"<br>");
        out.println("최초 세션 생성 시각: "+new Date(session.getCreationTime())+"<br>");
        out.println("최근 세션 접근 시각 : "+
                    new Date(session.getLastAccessedTime())+"<br>");
        out.println("세션 유효 시간 : "+session.getMaxInactiveInterval()+"<br>");
        if(session.isNew()){
            out.print("새 세션이 만들어졌습니다.");
        }
    }
}

```

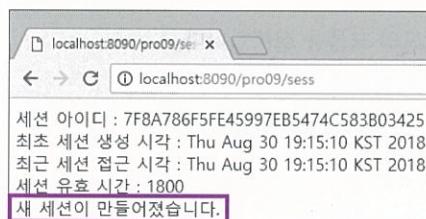
getSession()을 호출하여 최초 요청 시 세션 객체를 새로 생성하거나 기존 세션을 반환합니다.
생성된 세션 객체의 id를 가져옵니다.
최초 세션 객체 생성 시간을 가져옵니다.
세션 객체에 가장 최근에 접근한 시간을 가져옵니다.
세션 객체의 유효 시간을 가져옵니다.
최초 생성된 세션인지 판별합니다.



세션 유효 시간을 따로 설정하지 않으면 톰캣에서 설정한 기본 유효 시간 30분이 적용됩니다.

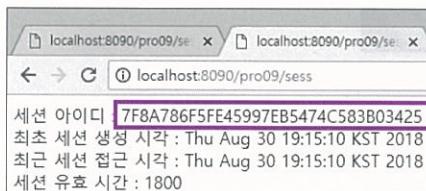
3. 브라우저에서 최초 요청 시 생성된 세션 객체에 할당된 세션 id와 여러 가지 정보를 출력합니다. 최초 생성된 세션이므로 “새 세션이 만들어졌습니다.”라는 메시지가 출력됩니다.

▼ 그림 9-28 최초 요청 시 출력되는 세션 정보



4. 같은 브라우저에서 다른 탭을 열고 요청하면 같은 세션 id를 출력하므로 최초 생성된 세션을 재사용합니다. 따라서 “새 세션이 만들어졌습니다.”라는 메시지는 출력되지 않습니다.

▼ 그림 9-29 다른 탭에서 재요청 시 출력 결과



다음은 서블릿에서 생성된 세션 id, 즉 브라우저로 전송되어 세션 쿠키에 쿠키 이름 jsessionID로 저장된 세션 id입니다.

▼ 그림 9-30 브라우저 세션 쿠키에 저장된 세션 id

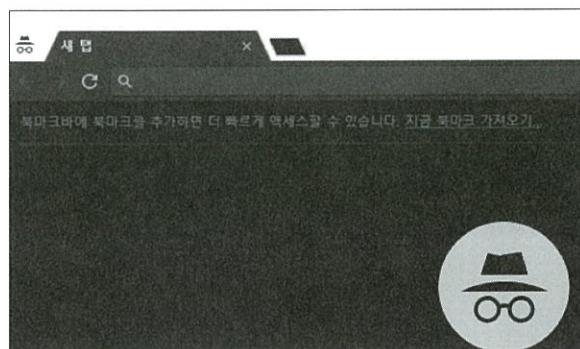
The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. In the left sidebar, under 'Cookies', there is a section for the domain 'http://localhost:8090'. A single cookie entry is highlighted: 'Name' is 'JSESSIONID' and 'Value' is '7F8A786F5FE45997EB5474C583B03425'. Other sections like 'Manifest', 'Service Workers', and 'Clear storage' are also visible in the sidebar.

현재 윈도 10의 운영체제에서는 같은 PC의 브라우저나 다른 탭에서 요청을 해도 기존 세션을 유지합니다. 그러면 이번에는 같은 PC에서 기존 브라우저의 세션으로 연결되지 않고, 다른 브라우저로 다른 세션을 만드는 방법을 알아보겠습니다. 즉, 새 세션으로 브라우저를 여는 과정입니다.

9.4.4 다른 브라우저에서 새 세션 만들기

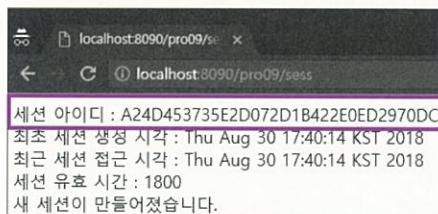
브라우저에서 [Ctrl] + [Shift] + [N]을 눌러 시크릿 모드의 크롬을 실행합니다.

▼ 그림 9-31 크롬을 시크릿 모드로 실행



주소창에서 /sess로 요청하면 새로운 세션을 생성한 후 다른 세션 id를 출력합니다.

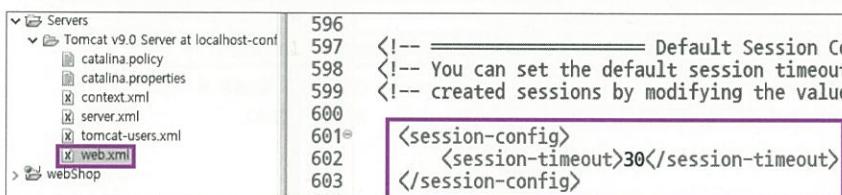
▼ 그림 9-32 /sess로 요청 시 새로운 세션이 생성



세션 기본 유효 시간은 톰캣 컨테이너에서 지정한 30분(1800초)입니다. 그러나 HttpSession의 `setMaxInactiveInterval()` 메서드를 이용하면 사용자가 원하는 세션 유효 시간을 설정할 수 있습니다. 그리고 `invalidate()` 메서드를 이용하면 세션을 언제든지 삭제할 수도 있습니다.

다음과 같이 톰캣 컨테이너의 web.xml에 세션 기본 유효 시간이 설정된 것을 확인할 수 있습니다.

▼ 그림 9-33 기본 유효 시간이 30분으로 설정



이번에는 제공되는 메서드들을 이용해서 직접 세션 유효 시간을 재설정해 보겠습니다.

1. 다음과 같이 SessionTest2 클래스를 준비합니다.

▼ 그림 9-34 실습 파일 위치



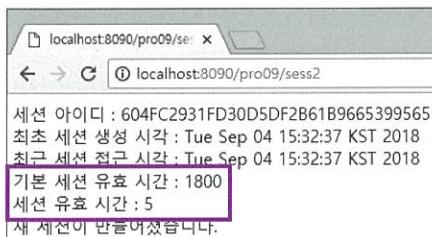
2. SessionTest2 클래스를 다음과 같이 작성합니다. setMaxInactiveInterval() 메서드를 이용해 세션 유효 시간을 5초로 재설정합니다.

코드 9-11 pro09/src/sec03/ex02/SessionTest2.java

```
package sec03.ex02;  
...  
@WebServlet("/sess2")  
public class SessionTest2 extends HttpServlet{  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html; charset=utf-8");  
        PrintWriter out = response.getWriter();  
        HttpSession session = request.getSession();  
        out.println("세션 아이디: " + session.getId() + "<br>");  
        out.println("최초 세션 생성 시각: " + new Date(session.getCreationTime()) + "<br>");  
        out.println("최근 세션 접근 시각: " + new Date(session.getLastAccessedTime()) + "<br>");  
        out.println("기본 세션 유효 시간: " + session.getMaxInactiveInterval() + "<br>");  
        session.setMaxInactiveInterval(5); ─────────── 세션의 유효 시간을 5초로 설정합니다.  
        out.println("세션 유효 시간: " + session.getMaxInactiveInterval() + "<br>");  
        if(session.isNew()){  
            out.print("새 세션이 만들어졌습니다.");  
        }  
    }  
}
```

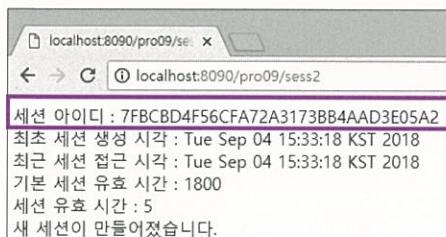
3. 최초에 /sess2로 요청하여 설정 전 유효 시간과 설정 후 유효 시간을 출력합니다.

▼ 그림 9-35 최초 요청 시 결과



4. 5초가 지난 후 같은 브라우저에서 재요청하면 다시 새 세션이 생성됩니다.

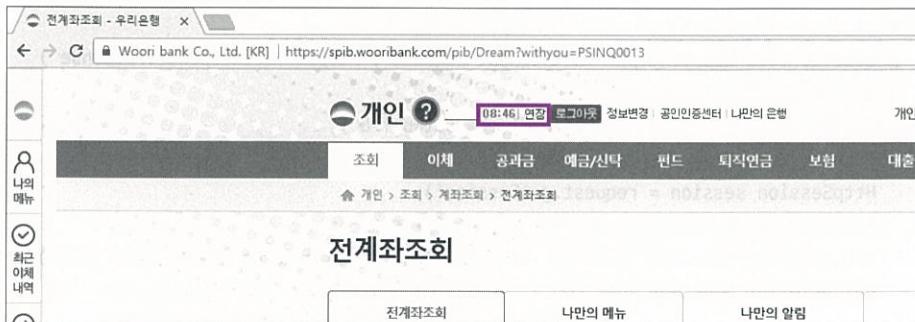
▼ 그림 9-36 5초 경과 후 요청 시 결과



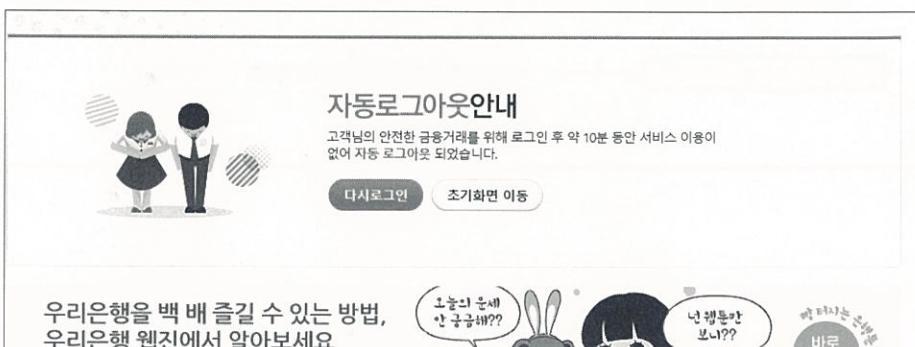
최초 요청 시 세션의 id와 5초 경과 후 요청 시 세션의 id가 다릅니다. 즉, 5초 이후에는 기존 세션은 삭제되고 재요청 시 새 세션이 생성됩니다.

세션 유효 시간을 재설정하는 경우는 주로 은행 사이트에서 자주 발생합니다. 우리가 보통 은행 사이트에 로그인하면 화면 위쪽에 10분에서 초 단위로 역카운팅이 되는 것이 보입니다. 만약 10분 이상 아무 작업도 하지 않으면 로그인 상태를 기억하는 세션이 자동 삭제되어 자동 로그아웃 화면을 출력합니다.

▼ 그림 9-37 로그인 시 10분 유효 시간을 체크하는 은행 사이트



▼ 그림 9-38 10분간 아무 작업을 하지 않으면 자동 로그아웃 화면 출력



이번에는 사용자가 강제로 세션을 삭제하는 기능을 실습해 보겠습니다.

1. 다음과 같이 실습 파일을 준비합니다.

▼ 그림 9-39 실습 파일 위치



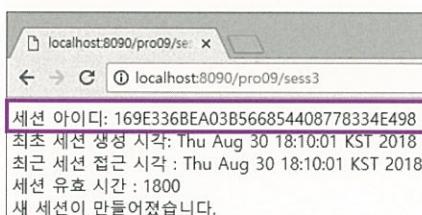
2. SessionTest3 클래스를 다음과 같이 작성합니다. invalidate() 메서드를 이용해 강제로 세션을 삭제합니다.

코드 9-12 pro09/sec03/ex03/SessionTest3.java

```
package sec03.ex03;  
...  
@WebServlet("/sess3")  
public class SessionTest3 extends HttpServlet{  
    protected void doGet(HttpServletRequest request , HttpServletResponse response )  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=utf-8");  
        PrintWriter out = response.getWriter();  
        HttpSession session = request.getSession();  
        out.println("세션 아이디: "+session.getId()+"<br>");  
        out.println("최초 세션 생성 시각: "+ new Date(session.getCreationTime())+"<br>");  
        out.println("최근 세션 접근 시각 : "+  
                    new Date(session.getLastAccessedTime())+"<br>");  
        out.println("세션 유효 시간 : "+ session.getMaxInactiveInterval()+"<br>");  
        if(session.isNew()){  
            out.print("새 세션이 만들어졌습니다.");  
        }  
        session.invalidate(); •———— invalidate()를 호출하여 생성된 세션 객체를  
    }  
} }  
    강제로 삭제합니다.
```

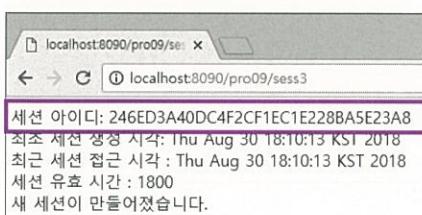
3. 최초 요청 시 새 세션이 생성된 후 invalidate() 메서드가 호출되므로 바로 소멸됩니다.

▼ 그림 9-40 최초 요청 시 결과



4. 재요청 시 다른 세션이 생성됩니다.

▼ 그림 9-41 재요청 시 결과

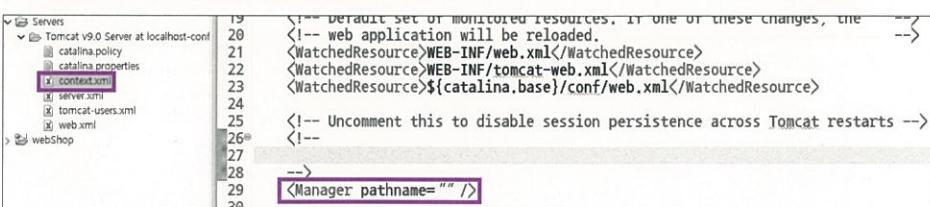


9.4.5 세션을 이용한 로그인 정보 바인딩 실습

지금까지는 HttpSession과 HttpServletRequest의 바인딩 기능을 사용했습니다. 그런데 로그인 상태처럼 사용자와 관련된 정보를 바인딩해서 사용할 때는 세션을 이용하는 것이 편리합니다. 세션은 사용자당 한 개씩 생성되기 때문이죠. 이번 절에서는 로그인을 이용해 HttpSession의 바인딩 기능을 알아보겠습니다.

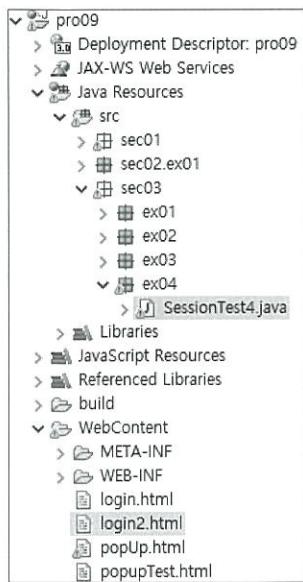
1. 실습하기 전에 해야 할 일이 있습니다. 톰캣이 종료된 후에도 세션이 메모리에서 삭제되지 않는 경우가 있으므로 톰캣 설정 파일인 context.xml을 열어 <Manager pathname="" /> 태그의 주석을 해제해야 합니다.

▼ 그림 9-42 context.xml의 Manager 태그 주석 해제



2. 다음과 같이 실습 파일을 준비합니다.

▼ 그림 9-43 실습 파일 위치



3. 로그인창에서 ID와 비밀번호를 입력한 후 서블릿으로 전송할 수 있도록 login2.html 파일을 작성합니다.

코드 9-13 pro09/WebContent/login2.html

```
<!DOCTYPE html>
<html>
<head>...</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>
```

4. SessionTest4 클래스를 다음과 같이 작성합니다. 로그인창에서 로그인한 경우 ID와 비밀번호를 가져오고, 최초 요청 시 세션에 setAttribute() 메서드를 이용해 user_id로 사용자ID를 바인딩하도록 구현합니다. <a> 태그를 이용해 재요청하고 세션의 getSttribute() 메서드를 이용하여 user_id 값을 가져와 로그인 여부를 확인합니다.

코드 9-14 pro09/src/sec03/ex04/SessionTest4.java

```

package sec03.ex04;
...
@WebServlet("/login")
public class SessionTest4 extends HttpServlet{
    protected void doGet(HttpServletRequest request , HttpServletResponse response )
        throws ServletException, IOException {
        doHandle(request, response);
    }

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

    private void doHandle(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");
        if (session.isNew()){
            if(user_id != null){
                session.setAttribute("user_id", user_id);
                out.println("<a href='login'>로그인 상태 확인</a>");
            }else {
                out.print("<a href='login2.html'>다시 로그인 하세요!!</a>");
                session.invalidate();
            }
        }else {
            user_id = (String) session.getAttribute("user_id");
            if (user_id != null && user_id.length() != 0) {
                out.print("안녕하세요 " + user_id + "님!!!");
            } else {
                out.print("<a href='login2.html'>다시 로그인 하세요!!</a>");
                session.invalidate();
            }
        }
    }
}

```

제공하는 예제 파일에서는 주석 처리를 해제하세요.

로그인창에서 전송된 ID와 비밀번호를 가져옵니다.

최초 요청 시 수행합니다.

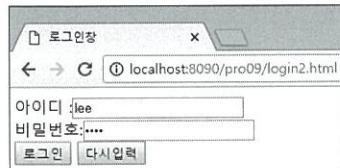
로그인창에서 서블릿으로 요청했다면 ID가 null이 아니므로 세션에 ID를 바인딩합니다.

재요청 시 세션에서 ID를 가져와 이전에 로그인했는지 여부를 확인합니다.

```
    }  
}
```

5. 로그인창 요청 후 ID와 비밀번호를 입력하고 전송합니다.

▼ 그림 9-44 로그인창에서 로그인



6. 최초 로그인 시 세션에 ID를 바인딩합니다.

▼ 그림 9-45 로그인 후 다시 /login으로 재요청



결과가 제대로 나오지 않는다면 다른 서블릿 파일에서 login이라는 매핑 이름이 중복되지 않았는지 확인하세요.

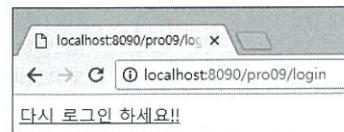
7. 다시 로그인 상태 확인을 클릭해 /login으로 재요청하면 현재 로그인 상태를 출력합니다.

▼ 그림 9-46 /login으로 재요청 시 로그인 상태 출력



8. 톰캣 재실행 후 로그인창을 거치지 않고 바로 /login으로 요청하면 세션에 ID가 없으므로 “다시 로그인 하세요!!”라는 메시지가 출력됩니다.

▼ 그림 9-47 로그인창을 거치지 않고 /login 요청 시 결과



9.5 encodeURL() 사용법

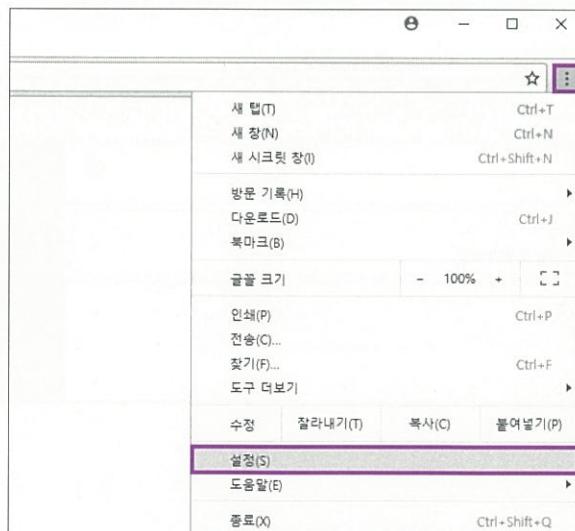
앞 절에서는 쿠키와 세션 기능을 알아봤습니다. 세션 역시 클라이언트의 세션 쿠키를 이용해 각 브라우저에 대한 세션 기능을 사용합니다. 그런데 만약 브라우저에서 쿠키 기능을 사용할 수 없게 설정했다면 쿠키 기능은 물론 세션 기능도 사용할 수 없습니다. 이럴 때는 `encodeURL()` 메서드를 이용해 직접 서버에서 브라우저로 응답을 먼저 보낸 후 URL Rewriting 방법을 이용해 `jsessionId`를 서버로 전송하여 세션 기능을 사용하면 됩니다.

9.5.1 브라우저에서 쿠키 사용 금지하기

이번에는 크롬 브라우저에서 쿠키 사용을 금지하는 방법을 알아보겠습니다.

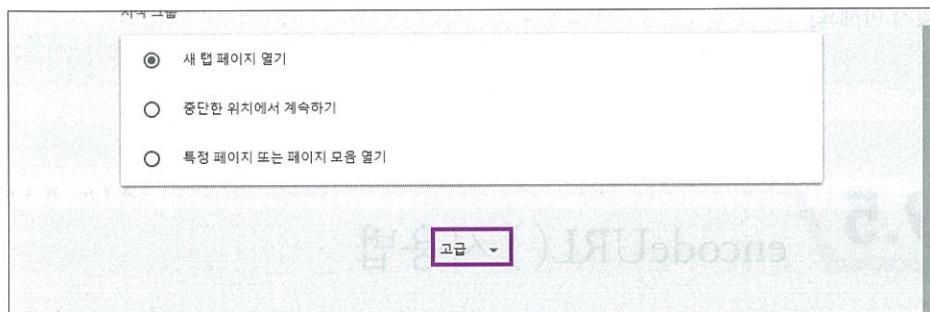
1. 크롬 브라우저를 실행하고 오른쪽 상단에서 더 보기(⋮) 아이콘 클릭 후 설정을 클릭합니다.

▼ 그림 9-48 더 보기 아이콘 클릭 후 설정 클릭



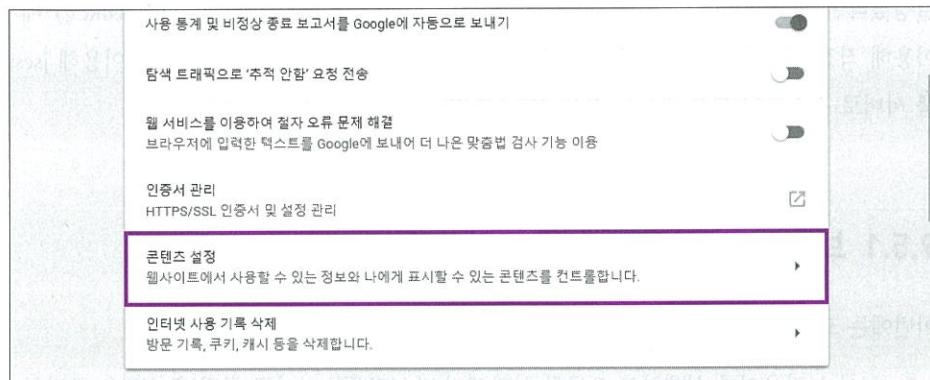
2. 하단에서 고급을 클릭합니다.

▼ 그림 9-49 고급 클릭



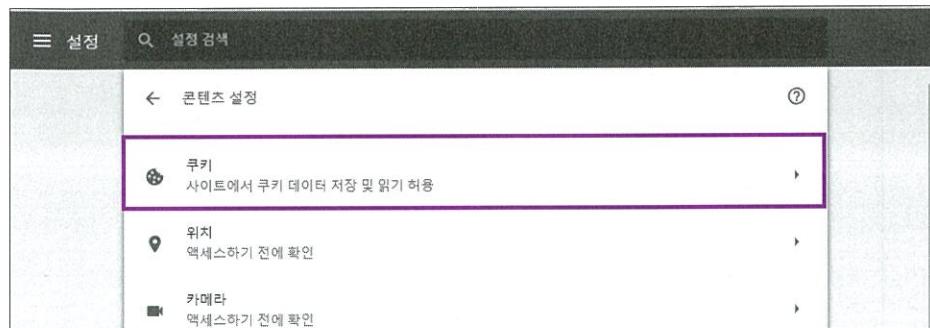
3. ‘개인정보 보호 및 보안’에서 콘텐츠 설정을 클릭합니다.

▼ 그림 9-50 콘텐츠 설정 클릭



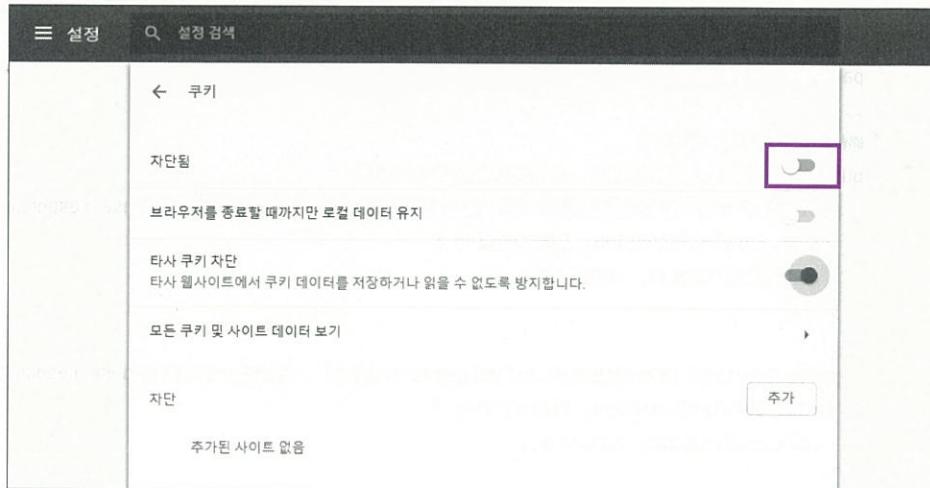
4. 쿠키를 클릭해 ‘사이트에서 쿠키 데이터 저장 및 읽기 허용’을 차단하도록 합니다.

▼ 그림 9-51 쿠키 클릭



5. ‘차단됨’ 옆의 옵션을 클릭하여 차단으로 설정합니다.

▼ 그림 9–52 ‘사이트에서 쿠키 데이터 저장 및 읽기 허용’을 차단으로 설정

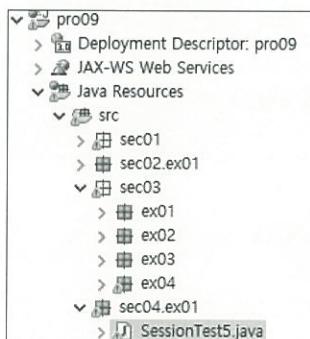


9.5.2 encodeURL() 메서드를 이용한 세션 사용 실습

이번에는 세션 쿠키를 사용하지 않고 encodeURL() 메서드를 이용해 jsessionId의 세션 id를 브라우저에 응답으로 전송한 후 세션을 사용해 보겠습니다.

1. 다음과 같이 실습 파일을 준비합니다.

▼ 그림 9–53 실습 파일 위치



2. SessionTest5 클래스를 다음과 같이 작성합니다. 다시 /login으로 요청해 jsessionId를 URL rewriting 방식으로 가져온 후 세션에 접근하여 로그인 상태 유무를 판단하도록 구현합니다.

코드 9-15 pro09/src/sec04/ex01/SessionTest5.java

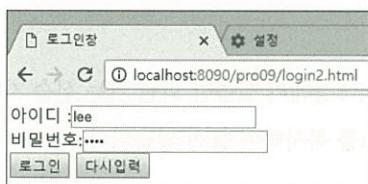
```
package sec04.ex01;  
...  
@WebServlet("/login")  
public class SessionTest5 extends HttpServlet{  
    protected void doGet(HttpServletRequest request , HttpServletResponse response )  
        throws ServletException, IOException {  
        doHandle(request, response);  
    }  
  
    protected void doPost(HttpServletRequest request , HttpServletResponse response )  
        throws ServletException, IOException {  
        doHandle(request, response);  
    }  
  
    private void doHandle(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");  
        if (session.isNew()) {  
            if(user_id != null){  
                session.setAttribute("user_id", user_id);  
                String url=response.encodeURL("login");  
                out.println("<a href='"+url+">로그인 상태 확인</a>");  
            } else {  
                out.print("<a href='login2.html'>다시 로그인하세요!!</a>");  
                session.invalidate();  
            }  
        } else {  
            user_id = (String) session.getAttribute("user_id");  
            if (user_id != null && user_id.length() != 0) {  
                out.print("안녕하세요 " + user_id + "님!!!");  
            } else {  
                out.print("<a href='login2.html'>다시 로그인하세요!!</a>");  
                session.invalidate();  
            }  
        }  
    }  
}
```

변수 url|| encodeURL()을 이용해 응답 시
미리 jsessionId를 저장합니다.

로그인 상태 확인 클릭 시
jsessionId를 서블릿으로 다시 전송합니다.

3. 로그인창에서 ID와 비밀번호를 입력하고 로그인합니다.

▼ 그림 9-54 로그인창에서 로그인



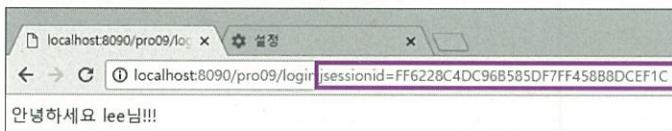
4. 로그인 상태 확인을 클릭합니다.

▼ 그림 9-55 로그인 상태 확인 클릭



5. 서블릿에 sessionId 쿠키 값은 전송해 로그인 상태를 유지합니다.

▼ 그림 9-56 서블릿에 sessionId 쿠키 값 전송



대부분의 브라우저는 쿠키 사용을 기본으로 설정하고 있지만 쿠키를 사용할 수 없을 경우에는 이렇게 encodeURL() 메서드를 이용해 세션 기능을 사용할 수 있습니다.

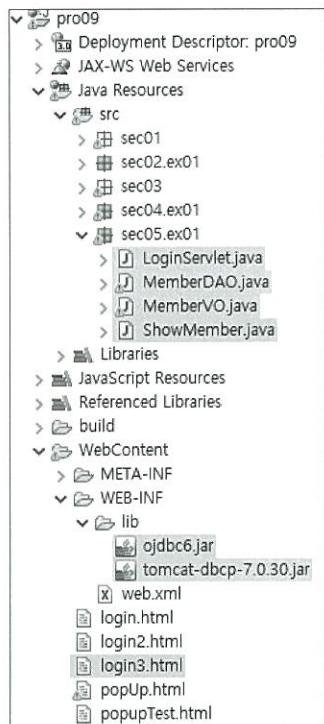
9.6

세션을 이용한 로그인 예제

이번에는 실제 웹 사이트에서 데이터베이스와 연동한 후 세션을 이용해 로그인 상태를 유지하는 예제를 실습해 보겠습니다.

- 먼저 데이터베이스 연동과 관련된 설정을 해줍니다. 8장에서 실습한 회원 기능 실습 자바 클래스 파일인 MemberDAO.java와 MemberVO.java를 복사하여 붙여 넣습니다.

▼ 그림 9-57 실습 파일 위치



- 사용자의 ID와 비밀번호를 입력한 후 /login 서블릿으로 전송하도록 login3.html을 작성합니다.

코드 9-16 pro09/WebContent/login3.html

```
<!DOCTYPE html>
<html>
<head>..</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_pwd"><br>
    <input type="submit" value="로그인">
    <input type="reset" value="초기화">
</form>
</body>
</html>
```

3. 로그인창의 요청을 처리하는 LoginServlet 클래스를 다음과 같이 작성합니다. 로그인창에서 전송된 ID와 비밀번호를 가져와 MemberVO 객체를 생성한 후 속성에 ID와 비밀번호를 설정합니다. 그런 다음 MemberDAO의 isExisted() 메서드를 호출하면서 memberVO를 인자로 전달합니다. 조회된 결과가 true이면 isLogOn 속성을 true로 세션에 저장하고, ID와 비밀번호도 세션에 저장합니다.

코드 9-17 pro09/src/sec05/ex01/LoginServlet.java

```
package sec06.ex01;
...
@WebServlet("/login")
public class LoginServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doHandle(request, response);
    }

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

    private void doHandle(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        String user_id = request.getParameter("user_id");
        String user_pwd = request.getParameter("user_pwd"); ← 로그인창에서 전송된 ID와 비밀번호를 가져옵니다.
        MemberVO memberVO = new MemberVO();
        memberVO.setId(user_id); ← MemberVO 객체를 생성하고 속성에 ID와 비밀번호를 설정합니다.
        memberVO.setPwd(user_pwd);
        MemberDAO dao = new MemberDAO();
        boolean result = dao.isExisted(memberVO); ← MemberDAO의 isExisted() 메서드를 호출하면서 memberVO를 전달합니다.
    }
}
```

```

if (result) {
    HttpSession session = request.getSession();
    session.setAttribute("isLogon", true); ← 조회한 결과가 true이면 isLogOn 속성을
    session.setAttribute("login.id", user_id); ← true로 세션에 저장합니다.
    session.setAttribute("login.pwd", user_pwd); ← 조회한 결과가 true이면 ID와 비밀번호
    out.print("<html><body>");
    out.print("안녕하세요 " + user_id + "님!!!<br>");
    out.print("<a href='show'>회원정보 보기</a>");
    out.print("</body></html>");
} else {
    out.print("<html><body><center>회원 아이디가 틀립니다."</center>"); ←
    out.print("<a href='login3.html'> 다시 로그인하기</a>"); ←
    out.print("</body></html>"); ←
}
}
}

```

4. MemberDAO 클래스를 다음과 같이 작성합니다. 오라클에서 제공하는 decode() 함수를 이용해 SQL문으로 회원 정보를 조회합니다. 정보가 존재하면 true를, 존재하지 않으면 false를 반환합니다.

코드 9-18 pro09/src/sec05/ex01/MemberDAO.java

```

package sec05.ex01;
...
public class MemberDAO {
    private DataSource dataFactory;
    public MemberDAO(){
        try{
            Context ctx=new InitialContext();
            Context envContext = (Context) ctx.lookup("java:/comp/env");
            dataFactory = (DataSource) envContext.lookup("jdbc/oracle");
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    ...
    public boolean isExisted(MemberVO memberVO) {
        boolean result = false;
        String id = memberVO.getId();
        String pwd = memberVO.getPwd();
        try {
            con = dataFactory.getConnection();

```

```

String query = "select decode(count(*),1,'true','false') as result from t_member";
query += " where id=? and pwd=?";
pst = con.prepareStatement(query);
pst.setString(1, id);
pst.setString(2, pwd);
ResultSet rs = pst.executeQuery();
rs.next(); 커서를 첫 번째 레코드로 위치시킵니다.
result = Boolean.parseBoolean(rs.getString("result"));
System.out.println("result=" + result);
} catch (Exception e) {
e.printStackTrace();
}
return result;
}
}

```

오라클의 decode() 함수를 이용해 조회하여 ID와 비밀번호가 테이블에 존재하면 true를, 존재하지 않으면 false를 조회합니다.

메서드로 전달된 ID와 비밀번호를 이용해 SQL문을 작성한 후 데이터베이스에 조회합니다.

- ShowMember 클래스를 다음과 같이 작성합니다. 두 번째 서블릿은 사용자가 로그인할 때 회원 정보를 표시해 줍니다. 먼저 로그인 상태를 확인하기 위해 getSession(false) 메서드를 호출하여 세션을 얻은 다음 getAttribute() 메서드에 isLogOn을 인자로 전달해 로그인 상태를 가져옵니다. isLogOn의 값이 true면 회원 정보를 세션에서 가져와 출력합니다. 만약 세션이 존재하지 않거나 isLogOn이 false면 다시 로그인창으로 이동합니다.

코드 9-19 pro09/src/sec05/ex01>ShowMember.java

```

package sec05.ex01;
...
@WebServlet("/show")
public class ShowMember extends HttpServlet{
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html;charset=utf-8");
PrintWriter out = response.getWriter();
String id = "", pwd="";
Boolean isLogon=false;
HttpSession session = request.getSession(false);
if( session != null){ 먼저 세션이 생성되어 있는지 확인합니다.
isLogon=(Boolean)session.getAttribute("isLogon"); isLogOn 속성을 가져와 로그인 상태를 확인합니다.
if(isLogon==true){
id = (String)session.getAttribute("login.id");
pwd = (String)session.getAttribute("login.pwd");
out.print("<html><body>");
out.print("아이디: " + id+"<br>");
out.print("비밀번호: " + pwd+"<br>");
}
}
}

```

이미 세션이 존재하면 세션을 반환하고, 없으면 null을 반환합니다.

isLogOn이 true면 로그인 상태이므로 회원 정보를 브라우저에 표시합니다.

```

        out.print("</body></html>");
    }else{
        response.sendRedirect("Login3.html"); ← 로그인 상태가 아니면 로그인창
    }                                         으로 이동합니다.

}else{                                     세션이 생성되지 않았으면 로그인
    response.sendRedirect("login3.html"); ← 창으로 이동합니다.
}
}
}

```

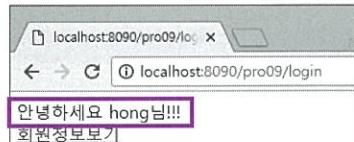
6. 로그인창에서 ID와 비밀번호를 입력한 후 전송합니다.

▼ 그림 9-58 로그인창에서 로그인



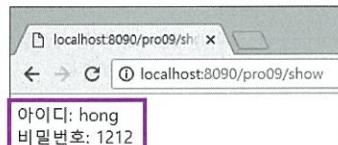
7. 회원 테이블에 입력한 ID와 비밀번호가 존재하면 로그인 성공 메시지가 출력됩니다.

▼ 그림 9-59 로그인 성공 시 메시지 출력



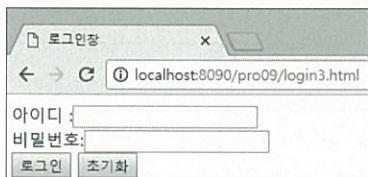
8. 회원정보보기를 클릭하면 이미 로그인 상태이므로 세션에 저장된 회원 정보가 표시됩니다.

▼ 그림 9-60 세션에 저장된 로그인 정보 표시



9. 다음은 톰캣을 재실행한 후 로그인창을 거치지 않고 바로 /show로 요청한 경우입니다. 로그인을 하지 않았으므로 다시 로그인창으로 리다이렉트됩니다.

▼ 그림 9-61 비로그인 상태 시 다시 로그인창으로 이동



현재 웹 사이트들은 사용자가 로그인하면 한 번만 데이터베이스에서 회원 정보를 조회한 후 로그인 상태를 세션에 저장해 놓고, 각 웹 페이지를 열 때마다 세션에 접근해 앞 페이지에서 로그인을 했는지 판단합니다.

10^장

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

이 장에서는 서블릿의 기능을 도와주는 다른 API들에 대해 알아보겠습니다. 우선 서블릿의 요청과 응답 작업하기 전에 수행하는 필터(Filter) 기능을 알아본 다음 서블릿의 속성과 스코프(scope) 개념에 대해 살펴보겠습니다. 그리고 서블릿 관련 API에서 특정 이벤트가 발생했을 때 이벤트를 처리할 수 있는 여러 가지 리스너(Listener)에 대해서도 알아봅니다. 이 장에 나오는 기능을 알아두면 좀 더 고급 기능을 구현할 수 있습니다.

10.1 서블릿 속성과 스코프

서블릿 속성(attribute)이란 다음 세 가지 서블릿 API 클래스에 저장되는 객체(정보)라고 보면 됩니다.

- ServletContext
- HttpSession
- HttpServletRequest

각 속성은 앞 장에서 이미 사용해 봤습니다. 서블릿 API의 `setAttribute(String name, Object value)`로 바인딩하고, 필요할 때 `getAttribute(String name)`으로 바인딩된 속성을 가져오면 됩니다. 또한 `removeAttribute(String name)`을 이용해 속성을 서블릿 API에서 제거할 수도 있습니다.

서블릿의 스코프(scope)는 서블릿 API에 바인딩된 속성에 대한 접근 범위를 의미합니다.

앞 장에서도 사용해 봤듯이 `ServletContext`에 바인딩된 속성은 애플리케이션 전체에서 접근할 수 있으므로 애플리케이션 스코프를 갖습니다. `HttpSession`에 바인딩된 속성은 그 `HttpSession`에 해당하는 브라우저에만 접근할 수 있으므로 세션 스코프를 갖습니다. `HttpServletRequest`는 해당 요청/응답에 대해서만 접근하므로 리퀘스트 스코프를 갖습니다.

스코프의 기능은 다음과 같습니다.

- 로그인 상태 유지 기능
- 장바구니 기능
- MVC의 Model과 View의 데이터 전달 기능

▼ 표 10-1 스코프의 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.

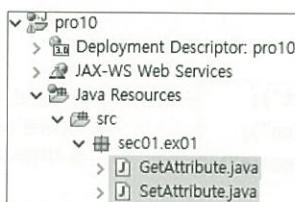
☞ Git Hub, 구글

+ Page

각 서블릿 API에 바인딩된 속성의 스코프를 알아보겠습니다.

1. 다음과 같이 GetAttribute, SetAttribute 클래스 파일을 준비합니다.

▼ 그림 10-1 실습 파일 위치



2. SetAttribute 클래스를 다음과 같이 작성합니다. ServletContext, HttpSession, HttpServletRequest 객체의 setAttribute() 메서드를 이용해 속성을 바인딩합니다.

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

```

package sec01.ex01;
...
@WebServlet("/set")
public class SetAttribute extends HttpServlet{
    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.

3. 두 번째 서블릿인 GetAttribute 클래스를 다음과 같이 작성합니다. 각 서블릿 API들의 getAttribute() 메서드를 이용해 속성 이름으로 바인딩한 값을 가져와 브라우저로 출력합니다.

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

```
package sec01.ex01;  
...  
@WebServlet("/get")  
public class GetAttribute extends HttpServlet{  
    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.

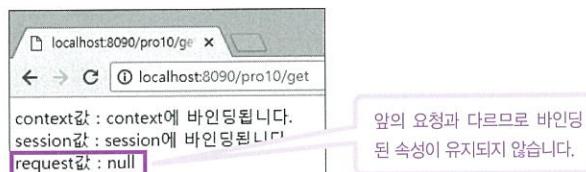
4. 브라우저에서 /set으로 요청해 속성을 바인딩합니다.

▼ 그림 10-2 /set으로 요청 시 출력 결과



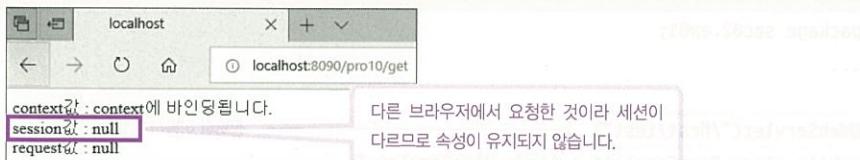
5. Context와 Session 객체에 바인딩된 속성은 같은 브라우저에서 접근할 수 있으므로 값을 출력합니다. 그러나 기존에 바인딩된 request 객체는 /get으로 요청하여 생성된 request 객체와 다르므로 null이 출력됩니다.

▼ 그림 10-3 같은 브라우저에서 /get으로 요청 시 출력 결과



6. 인터넷 익스플로러에서 /get으로 요청해 볼까요? 익스플로러에서 요청했기 때문에 이번에는 크롬의 세션 객체에는 접근할 수 없어 null을 출력합니다. 반면에 Context 객체에 바인딩된 레이터는 모든 브라우저에서 같은 결과를 출력합니다.

▼ 그림 10-4 다른 브라우저에서 /get으로 요청했을 때



10.2

서블릿의 여러 가지 URL 패턴

JAVA WEB

이번에는 서블릿에 요청할 때 사용하는 URL 패턴에 대해 자세히 알아보겠습니다.

URL 패턴이란 실제 서블릿의 매핑 이름을 말합니다. 즉, 서블릿 매핑 시 사용되는 가상의 이름으로, 클라이언트가 브라우저에서 요청할 때 사용되며 반드시 /(슬래시)로 시작해야 합니다.

서블릿 매핑 이름으로 사용되는 URL 패턴의 종류는 정확히 이름까지 일치하는지, 디렉터리까지만 일치하는지, 확장자만 일치하는지에 따라 세 가지로 나누어집니다.

10.2.1 서블릿에 여러 가지 URL 패턴 적용 실습

여러 가지 URL 패턴을 사용해 서블릿에 요청하는 방법을 알아보겠습니다.

1. 다음과 같이 TestServlet1~3 클래스 파일을 준비합니다.

▼ 그림 10-5 실습 파일 위치



2. 첫 번째 서블릿인 TestServlet1 클래스를 다음과 같이 작성합니다. 이 서블릿은 /first/test로 요청할 때 실행됩니다. 브라우저의 요청 URL에 대해 서블릿의 여러 가지 메서드를 이용하여 요청 관련 정보를 가져옵니다.

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

3. 두 번째 서블릿인 TestServlet2 클래스를 다음과 같이 작성합니다. /first/ 디렉터리 이름으로 시작되는 요청에 대해 실행됩니다.

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

```
package sec02.ex01;  
...  
@WebServlet("/first/*") ━━━━━━━━ 딜렉터리 이름만 일치하는 URL 패턴
```

```

public class TestServlet2 extends HttpServlet {
    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>");
```

(1) 헤더 출력
(2) 본문 출력
(3) 터미네이션

```

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

```

4. 세 번째 서블릿인 TestServlet3 클래스를 다음과 같이 작성합니다. 이는 매핑 이름에 상관 없이 확장자가 .do면 실행됩니다.

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 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();
```

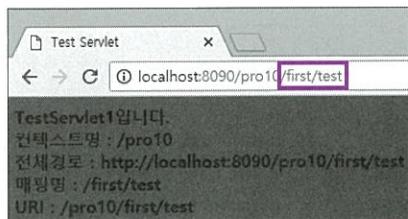
```

String uri = request.getRequestURI();
out.println("<html>");
out.println("<head>");
out.println("<title>Test Servlet3</title>");
out.println("</head>");
out.println("<body bgcolor='red'>");
out.println("<b>TestServlet3입니다.</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>" );
out.println("</body>");
out.println("</html>");
out.close();
}
}

```

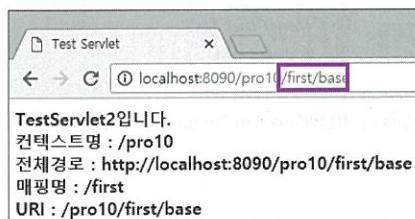
5. 각각의 매핑 이름으로 요청해 보겠습니다. 우선 정확한 매핑 이름(/first/test)으로 요청한 경우에는 다음과 같이 출력됩니다.

▼ 그림 10-6 /first/test로 요청 시

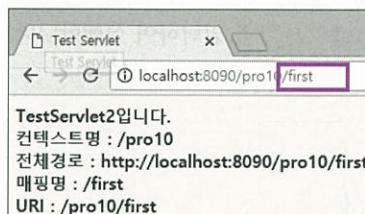


6. 디렉터리 이름만 일치하는 경우에는 각각 다음과 같이 출력됩니다.

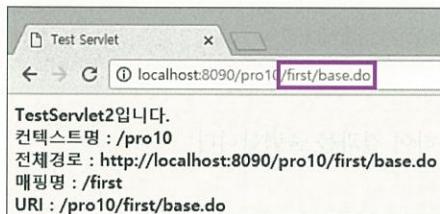
▼ 그림 10-7 /first/base로 요청 시



▼ 그림 10-8 디렉터리 이름 /first로 요청 시

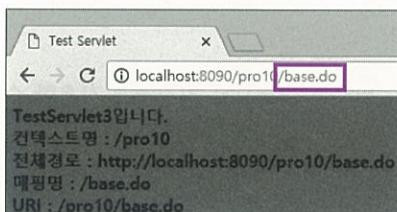


▼ 그림 10-9 디렉터리 이름 /first/base.do로 요청 시

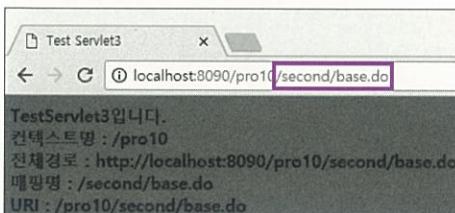


7. 다음은 확장자가 일치했을 경우의 출력 결과로, 각각 /base.do와 /second/base.do로 요청했을 때의 출력 결과입니다.

▼ 그림 10-10 /base.do로 요청 시



▼ 그림 10-11 /second/base.do로 요청 시



/first/base.do로 요청하면 확장자명이 .do로 끝나지만 앞의 디렉터리 이름이 우선하므로 TestServlet2가 실행됩니다. 반면에 /second/base.do로 요청하면 /second 디렉터리는 존재하지 않으므로 확장자명 .do를 우선하여 TestServlet3이 실행됩니다.

8. 다음은 TestServlet3 클래스의 URL 패턴을 /*로 설정한 후 요청한 결과입니다.

@WebServlet("*.do")를 주석 처리하고, @WebServlet("/")을 입력하여 실행합니다.

▼ 그림 10-12 @WebServlet("/")으로 설정

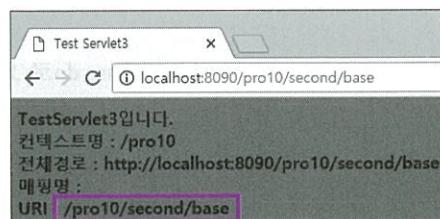
```
14 */  
15 /*@WebServlet("*.do")*/  
16 @WebServlet("/")  
17 public class TestServlet3 extends HttpServlet {  
18     private static final long serialVersionUID = 1L;  
19     /**
```

Tip *

실습 파일을 사용할 경우 주석 처리를 해제하기 바랍니다.

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

▼ 그림 10-13 확장자명 없이 요청한 결과



확장자명은 지정하지 않을 수도 있고, do 대신 자신이 원하는 이름으로 지정해서 사용할 수도 있습니다(do는 일반적으로 MVC나 프레임워크에서 자주 사용하는 확장자명입니다).

10.3 Filter API

이번에는 필터에 대해 알아보겠습니다. 필터란 브라우저에서 서블릿에 요청하거나 응답할 때 미리 요청이나 응답과 관련해 여러 가지 작업을 처리하는 기능입니다. 프로그래밍을 하다가 한글 인코딩처럼 각 서블릿에서 반복적으로 처리해야 하는 작업이 있을 수 있는데, 이런 경우 서블릿의 공통 작업을 미리 필터에서 처리하면 반복해서 작업할 필요가 없겠죠.

▼ 그림 10-14 필터 기능 수행 과정

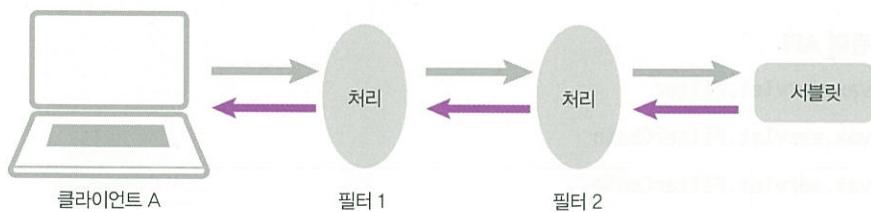


그림 10-15는 9장에서 로그인 예제를 실습할 때 사용한 서블릿 코드의 일부분입니다.

▼ 그림 10-15 request에 한글 인코딩 설정

```

private void doHandle(HttpServletRequest request, HttpServletResponse response) throws ServletException {
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html;charset=utf-8");
    PrintWriter out = response.getWriter();
    String user_id = request.getParameter("user_id");
    String user_pwd = request.getParameter("user_pwd");

    MemberVO memberVO = new MemberVO();
    memberVO.setId(user_id);
    memberVO.setPwd(user_pwd);
    MemberDAO dao = new MemberDAO();
    boolean result = dao.isExisted(memberVO);

    if (result) {
        HttpSession session = request.getSession();
        session.setAttribute("isLogon", true);
        session.setAttribute("login.id", user_id);
        session.setAttribute("login.pwd", user_pwd);
    }
}
  
```

이처럼 웹 페이지에서 입력한 한글을 서블릿에 전달하려면 `setCharacterEncoding()` 메서드를 이용해 한글 인코딩 설정을 서블릿마다 상단에 추가해야 했습니다. 하지만 모든 서블릿에서 공통으로 처리하는 작업을 먼저 필터에서 처리해 주면 편리하겠습니다.

필터는 용도에 따라 크게 요청 필터와 응답 필터로 나뉘며 다음과 같은 API가 있습니다.

- 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

표 10-2와 표 10-3은 실제 서블릿에서 제공하는 필터 관련 API의 여러 가지 메서드들입니다.

▼ 표 10-2 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.

▼ 표 10-3 FilterConfig의 메서드

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

10.3.1 사용자 정의 Filter 만들기

그럼 이번에는 직접 필터를 만들어 보겠습니다. 사용자 정의 필터는 반드시 `Filter` 인터페이스를 구현해야 합니다. 그리고 `init()`, `doFilter()`, `destroy()`의 추상 메서드를 구현해 주어야 합니다. 사용자 정의 필터를 생성하면 필터를 각각의 요청에 맞게 적용하기 위해 필터 매팅을 해야 하는데, 필터를 매팅하는 방법은 다음 두 가지입니다.

- 애너테이션을 이용하는 방법
- `web.xml`에 설정하는 방법

일반적으로 애너테이션을 이용하는 방법이 편리하므로 많이 사용합니다.

10.3.2 Filter를 이용한 한글 인코딩 실습

우선 한글 인코딩 처리를 통해 필터 기능을 실습해 보겠습니다.

1. 다음과 같이 `LoginTest`, `EncoderFilter` 클래스 파일을 준비합니다.



2. 로그인창에서 ID 대신 이름을 입력한 후 서블릿으로 전송하도록 `login.html`을 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>
```

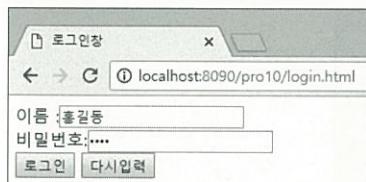
3. LoginTest 클래스를 다음과 같이 작성합니다. 서블릿에서는 setCharacterEncoding() 메서드를 주석 처리하여 한글 처리를 하지 않도록 합니다.

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

```
package sec03.ex01;  
...  
@.WebServlet("/login")  
public class LoginTest extends HttpServlet {  
    protected void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        //request.setCharacterEncoding( "utf-8" ); ← post 방식으로 한글 전송 시  
        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("이름은 " + user_name +"<br>");  
        out.println("비밀번호는 "+user_pw +"<br>");  
        out.println("</body></html>");  
    }  
}
```

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

▼ 그림 10-17 로그인창에서 로그인

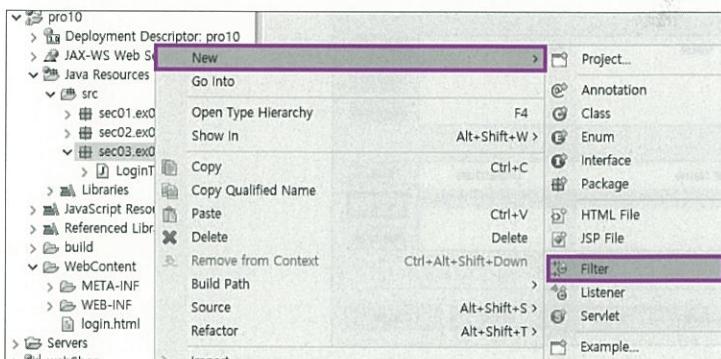


▼ 그림 10-18 전송된 한글이 깨져서 출력



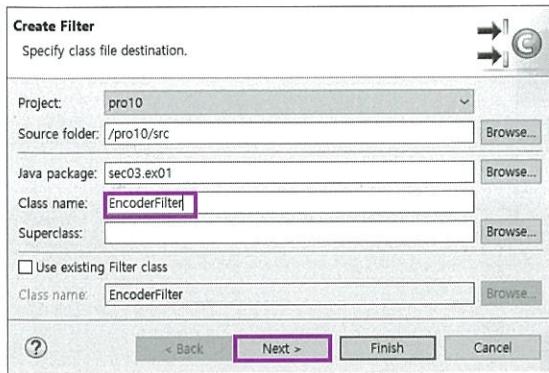
5. 이번에는 필터를 이용해 한글 인코딩 기능을 구현해 보겠습니다. sec03.ex01 패키지를 선택하고 마우스 오른쪽 버튼을 클릭한 후 New > Filter를 선택합니다.

◀ 그림 10-19 New > Filter 선택



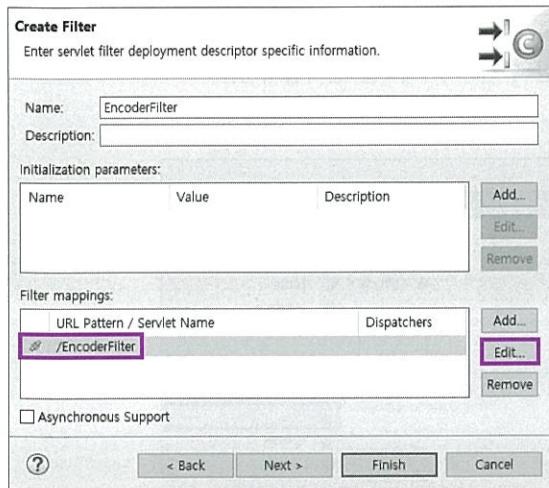
6. Class name으로 EncoderFilter를 입력하고 Next를 클릭합니다.

▼ 그림 10-20 클래스 이름으로 EncoderFilter 입력 후 Next 클릭



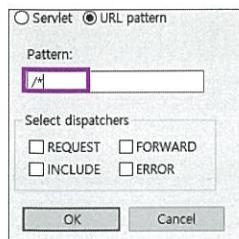
7. Filter mappings에서 /EncoderFilter를 선택한 후 Edit를 클릭합니다.

▼ 그림 10-21 필터 매핑 이름 수정



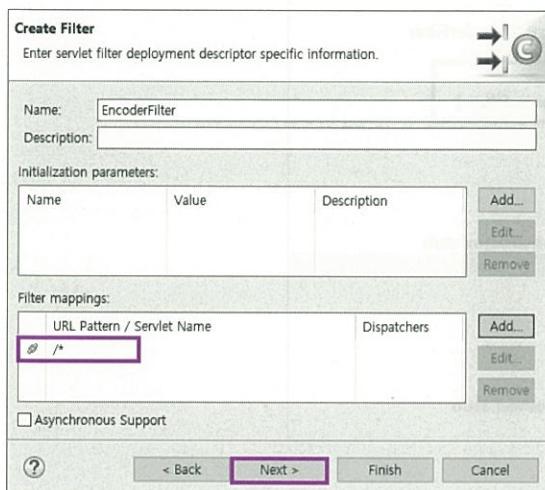
8. 모든 요청에 대해 필터 기능을 수행하도록 Pattern을 /*로 수정합니다.

▼ 그림 10-22 모든 요청에 대해 필터를 처리하도록 설정



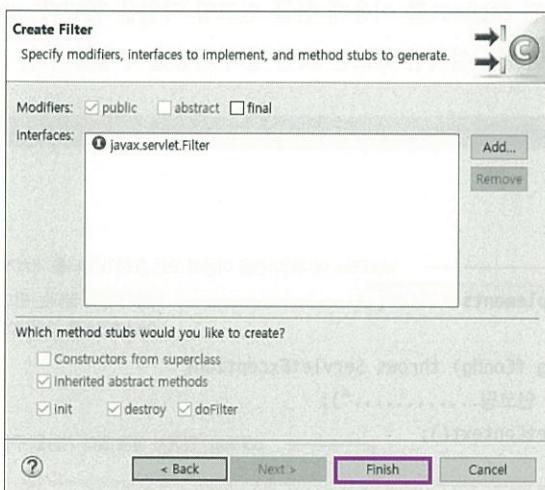
9. URL Pattern에서 /*을 확인하고 Next를 클릭합니다.

▼ 그림 10-23 필터 맵핑 이름 확인 후 Next 클릭



10. Finish를 클릭하여 필터 클래스가 생성된 것을 확인합니다.

▼ 그림 10-24 Finish 클릭



▼ 그림 10-25 @WebFilter 애너테이션으로 필터 생성 확인

```
1 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     }
}
```

11. 이제 다음과 같이 EncoderFilter 클래스를 작성합니다. 사용자 정의 필터 클래스는 반드시 Filter 인터페이스를 구현해야 합니다. 브라우저 요청 시 doFilter() 메서드의 매개변수로 request와 response가 전달되며, doFilter() 메서드는 FilterChain 타입인 chain을 세 번째 매개변수로 가집니다. 전달된 request를 이용해 한글 인코딩 작업을 합니다. chain.doFilter() 메서드를 기준으로 위쪽에 위치한 코드는 요청 필터 기능을 수행합니다.

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

```
package sec03.ex01;
...
@WebFilter("/*")           ← WebFilter 애너테이션을 이용해 모든 요청이 필터를 거치게 합니다.
public class EncoderFilter implements Filter{           ← 사용자 정의 필터는 반드시 Filter
    ServletContext context;                           ← 인터페이스를 구현해야 합니다.
    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");           ← 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.
    }
}
```

```

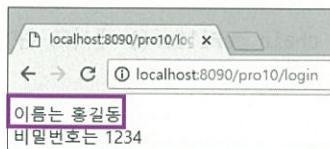
String mesg = " Context 정보:" + context
+ "\n URI 정보 : " + pathinfo
+ "\n 물리적 경로: " + realPath;
System.out.println(mesg);
chain.doFilter( request, response ); ----- 다음 필터로 넘기는 작업을 수행합니다.
}

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

```

12. 톰캣을 재실행하고 로그인창에서 한글을 입력합니다. 이번에는 필터를 거쳐 한글이 제대로 출력되는 것을 확인할 수 있습니다. 요청 필터 기능을 수행할 때마다 `doFilter()`가 수행되므로 이클립스 콘솔에도 다음과 같은 메시지가 출력됩니다.

▼ 그림 10-26 필터를 거친 출력 결과



▼ 그림 10-27 필터 호출 시 메시지 출력

```

9월 03, 2018 2:58:10 오후 org.apache.catalina.core.StandardContext reload
정보: 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

```

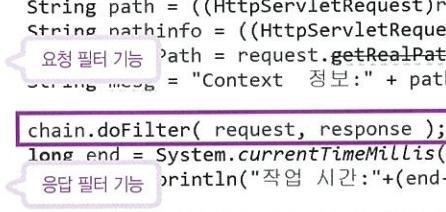
10.3.3 응답 필터 사용

이번에는 응답에 대해 수행하는 응답 필터를 알아보겠습니다. 서블릿에서 요청과 응답에 대한 필터 기능은 동일한 필터가 수행합니다.

한 필터에서 요청과 응답 기능을 수행하는 방법을 그림 10-28에 나타내었습니다. 필터에서 `doFilter()` 메서드를 기준으로 위쪽에 위치한 코드는 요청 필터 기능을 수행하고, 아래에 위치한 코드는 응답 필터 기능을 수행합니다.

▼ 그림 10-28 한 개의 필터에서 요청 필터와 응답 필터의 수행 방법

```
15  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {  
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 realPath = request.getRealPath(pathinfo);  
23      String mesg = "Context 정보:" + path + " URI 정보 : " + pathinfo + "  
24      + "\n 물리적 경로: " + realPath;  
25      chain.doFilter(request, response);  
26      long end = System.currentTimeMillis();  
27      System.out.println("작업 시간:"+ (end - begin) + "ms");  
28  }
```



10.3.4 응답 필터 기능으로 작업 시간 구하기

응답 필터 기능을 이용해 로그인 요청 시 작업 수행 시간을 구해 보겠습니다.

- 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.

```

chain.doFilter( request, response );

long end = System.currentTimeMillis();           ← 응답 필터에서 요청 처리 후의 시각을 구합니다.
System.out.println("작업 시간:"+(end-begin)+"ms"); ← 작업 요청 전과 후의 시각 차를 구
}                                                 해 작업 수행 시간을 구합니다.

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

```

`long begin = System.currentTimeMillis()` 메서드는 `chain.doFilter()` 메서드 위쪽에 위치하므로 요청 시 시작을 구합니다. `long end = System.currentTimeMillis()` 메서드는 `chain.doFilter()` 메서드 아래에 위치하므로 응답 시 시작을 구합니다.

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

▼ 그림 10-29 응답 필터를 이용한 작업 수행 시간 출력 결과

```

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

```

이상으로 필터 기능에 대해 알아봤습니다. 서블릿이나 JSP에서 공통으로 처리해야 할 작업을 필터에 구현해 놓고 사용하면 편리하다는 것을 기억해 두세요.

10.4

여러 가지 서블릿 관련 Listener API

자바 GUI에서는 마우스 클릭과 같은 이벤트 발생 시 여러 가지 이벤트 핸들러를 이용해 화면의 기능을 구현합니다. 이처럼 서블릿에서도 서블릿에서 발생하는 이벤트에 대해 적절한 처리를 해주는 여러 가지 리스너를 제공합니다.

▼ 표 10-4 서블릿 관련 여러 가지 리스너들

서블릿 관련 Listener	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 알려주는 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

10.4.1 HttpSessionBindingListener 이용해 로그인 접속자수 표시

HttpSessionBindingListener를 이용해 현재 웹 페이지에 로그인한 접속자수를 알아보는 기능을 구현해 보겠습니다.

1. 다음과 같이 실습 파일을 새로 준비합니다.

▼ 그림 10-30 실습 파일 위치



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>
```

3. LoginTest 클래스를 다음과 같이 작성합니다. LoginImpl loginUser=new LoginImpl(user_id,user_pw)를 실행하여 전송된 ID와 비밀번호를 저장합니다. 또 session.setAttribute("loginUser", loginUser)으로 세션에 바인딩 시 미리 HttpSessionBindingListener를 구현한 LoginImpl의 valueBound() 메서드를 호출합니다.

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); ← 이벤트 핸들러를 생성한 후  
        if(session.isNew()){ ← 세션에 저장합니다.  
            session.setAttribute("loginUser", loginUser); ← 세션에 바인딩 시  
        } ← LoginImpl의 valueBound()  
        out.println("<head>"); ← 메서드를 호출합니다.  
        out.println("<script type='text/javascript'>");  
        out.println("setTimeout('history.go(0)', 5000)"); ← 자바스크립트의 setTimeout() 함수를  
        out.println("</script>"); ← 이용해 5초마다 서블릿에 재요청하여  
        out.println("</head>"); ← 현재 접속자수를 표시합니다.  
        out.println("<html><body>");  
        out.println("아이디는 " + loginUser.user_id +"  
        out.println("총 접속자수는"+LoginImpl.total_user +"  
        out.println("</body></html>"); ← 접속자수를 브라우저로  
    } ← 출력합니다.  
}
```

4. LoginImpl 클래스를 다음과 같이 작성합니다. HttpSessionBindingListener를 구현하여 세션에 바인딩 이벤트를 처리하는 이벤트 핸들러가 구현되어 있습니다. 세션에 바인딩 시 valueBound()가 호출되어 static 변수인 total_user의 값을 1 증가시킵니다.

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

```
package sec04.ex01;  
  
import javax.servlet.http.HttpSessionBindingEvent;
```

```

import javax.servlet.http.HttpSessionBindingListener;
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;
    }

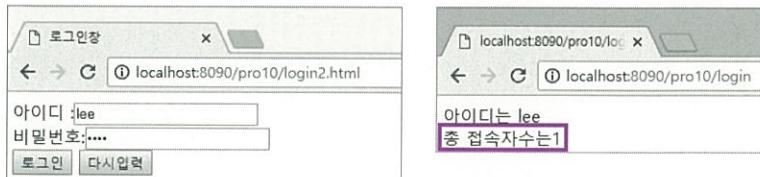
    @Override
    public void valueBound(HttpSessionBindingEvent arg0) {
        System.out.println("사용자 접속");
        ++total_user;
    }

    @Override
    public void valueUnbound(HttpSessionBindingEvent arg0) {
        System.out.println("사용자 접속 해제");
        total_user--;
    }
}

```

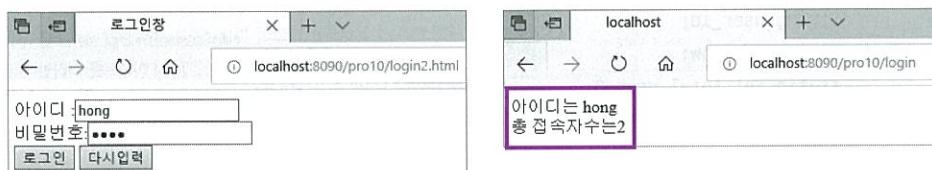
5. 서로 다른 종류의 브라우저에서 접속하여 실행 결과를 확인해 보겠습니다. 우선 크롬에서 로그인하면 접속자 ID와 접속자수가 표시됩니다.

▼ 그림 10-31 크롬에서 로그인한 결과



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

▼ 그림 10-32 익스플로러에서 로그인한 결과



7. 5초 후 크롬에서는 접속자수가 갱신되어 표시됩니다.

▼ 그림 10-33 크롬 화면 재접속 시 갱신된 접속자 수



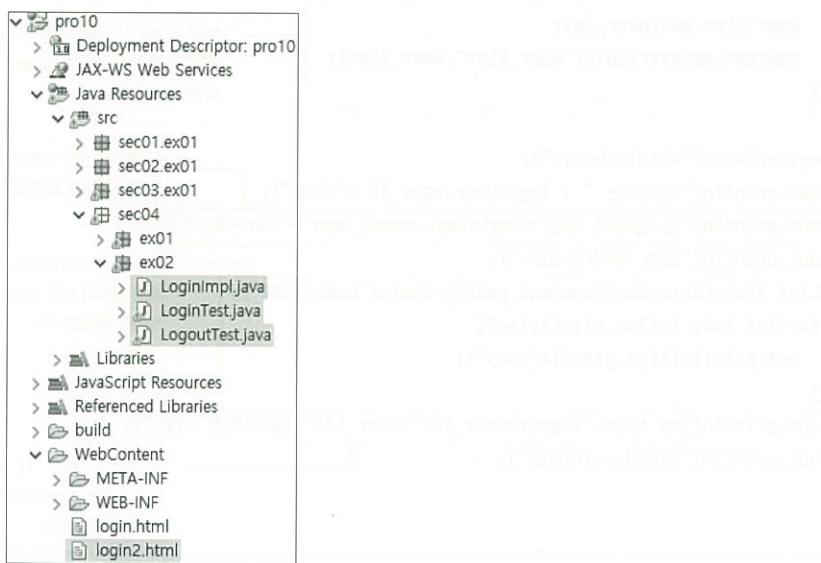
참고로 HttpSessionBindingListener를 구현한 LoginImpl 클래스는 리스너를 따로 등록할 필요가 없습니다.

10.4.2 HttpSessionListener 이용해 로그인 접속자수 표시

이번에는 HttpSessionListener를 이용해 웹 페이지 로그인 시 접속자수와 모든 접속자 ID를 표시해 주는 기능을 구현해 보겠습니다.

1. 다음과 같이 실습 파일을 준비합니다.

▼ 그림 10-34 실습 파일 위치



2. 첫 번째 서블릿인 LoginTest 클래스 파일을 다음과 같이 수정합니다. `setAttribute()`를 이용해 `loginUser`를 세션에 바인딩하면 `LoginImpl` 클래스에 구현된 이벤트 핸들러를 이용해 접속자수를 1 증가시킵니다. 그리고 `user_list`에 접속자 ID를 저장한 다음 `ServletContext` 객체에 바인딩합니다.

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.

    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");
```

```

LoginImpl loginUser=new LoginImpl(user_id,user_pw);
if(session.isNew()){
    session.setAttribute("loginUser", loginUser);
    user_list.add(user_id);
    context.setAttribute("user_list",user_list);
}

out.println("<html><body>");
out.println("아이디는 " + loginUser.user_id + "<br>"); ← LoginImpl 객체를 생성한 후
out.println("총 접속자 수는"+LoginImpl.total_user + "<br><br>"); ← 전송된 ID와 비밀번호를 저장합니다.

out.println("접속 아이디:<br>"); ← 최초 로그인 시 접속자 ID를 ArrayList에
List list=(ArrayList)context.getAttribute("user_list"); 차례로 저장한 후 다시 context 객체에
for(int i=0; i<list.size();i++){ 속성으로 저장합니다.
    out.println(list.get(i)+"<br>"); ← 세션에 바인딩 이벤트 처리 후
} ← 총 접속자수를 표시합니다.
out.println("<a href='logout?user_id="+user_id+">로그아웃 </a>"); ← context 객체의 ArrayList를 가져와 접속자 ID를 차례로 브라우저로 출력합니다.
out.println("</body></html>"); ←
}
}

```

로그아웃 클릭 시 서블릿 logout으로 접속자 ID를 전송해 로그아웃합니다.

- LogoutTest 클래스를 다음과 같이 작성합니다. 여기서는 로그아웃 링크를 클릭하면 접속자 수를 1 감소시키고 user_list에서 로그아웃한 접속자 ID를 삭제한 후 다시 user_list를 ServletContext 객체에 바인딩하도록 설정합니다.

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

```

package sec04.ex02;
...
@WebServlet("/logout")
public class LogoutTest extends HttpServlet {
    ServletContext context;

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

    private void doHandle(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"); ─────────── user_list에서 삭제할 ID를 가져옵니다.
session.invalidate(); ─────────── 로그아웃 시 세션을 소멸시킵니다.
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.");
}
}

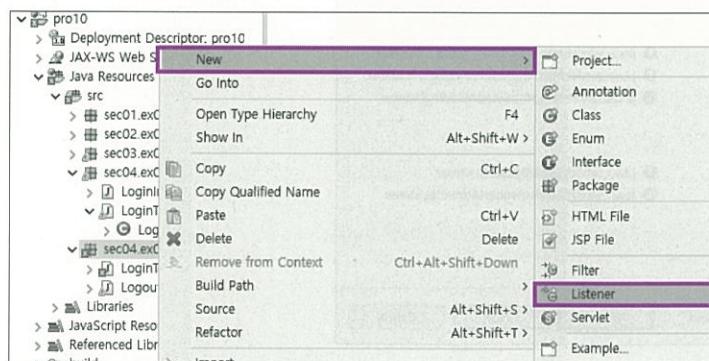
```

↑ user_list에서 로그아웃한 접속자 ID를 삭제한
후 다시 user_list를 컨텍스트에 저장합니다.

LoginImpl 클래스는 HttpSessionListener를 구현해 세션 생성과 소멸 시 이벤트를 처리하는 핸들러입니다. 중요한 것은 앞의 LoginImpl에서 구현한 HttpSessionBindingListener와는 다르게 HttpSessionListener는 반드시 리스너를 구현한 이벤트 핸들러를 애너테이션을 이용해서 등록해야 한다는 것입니다. 직접 구현해 보겠습니다.

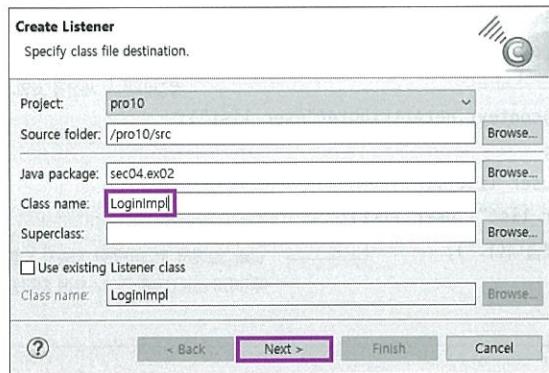
- sec04.ex02 패키지를 선택하고 마우스 오른쪽 버튼을 클릭한 후 New > Listener를 선택합니다.

▼ 그림 10-35 New > Listener 선택



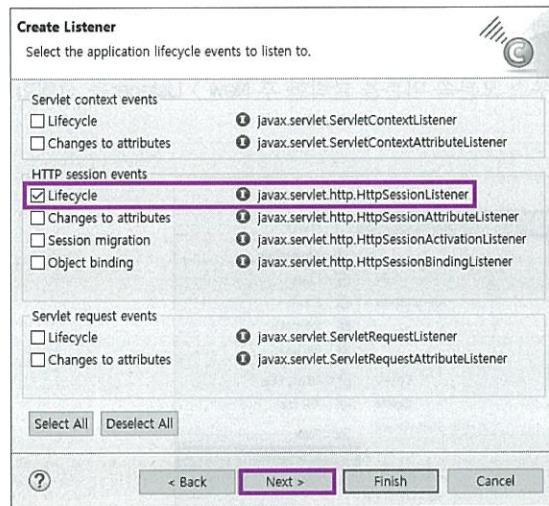
2. Class name으로 LoginImpl을 입력하고 Next를 클릭합니다.

▼ 그림 10-36 Class name으로 LoginImpl 입력 후 Next 클릭



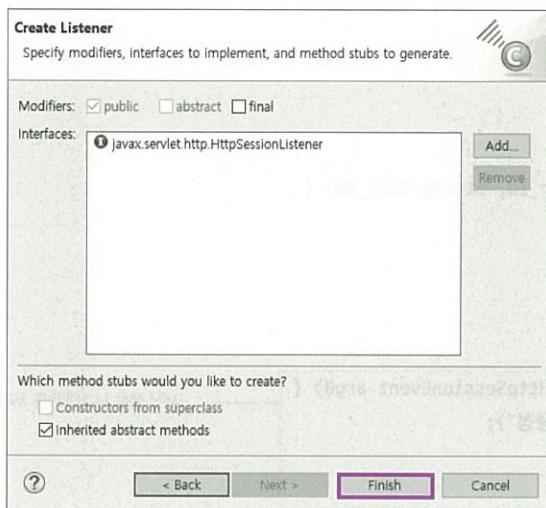
3. HttpSessionListener이 체크하고 Next를 클릭합니다.

▼ 그림 10-37 HttpSessionListener 선택 후 Next 클릭



4. Finish를 클릭합니다.

그림 10-38 Finish 클릭



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

그림 10-39 어너테이션으로 리스너 등록 확인

```
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* /**
```

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 어너테이션을 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;
    }

    @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;
}
}

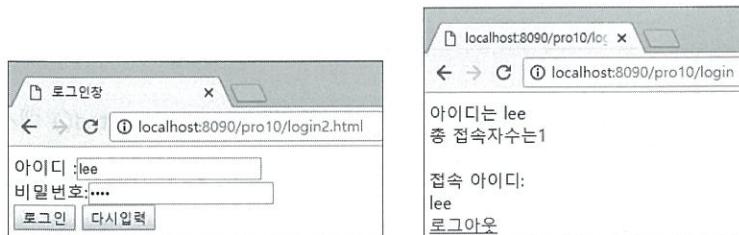
세션 소멸 시 이벤트를 처리합니다.

세션 소멸 시 접속자수를 1 감소시킵니다.

```

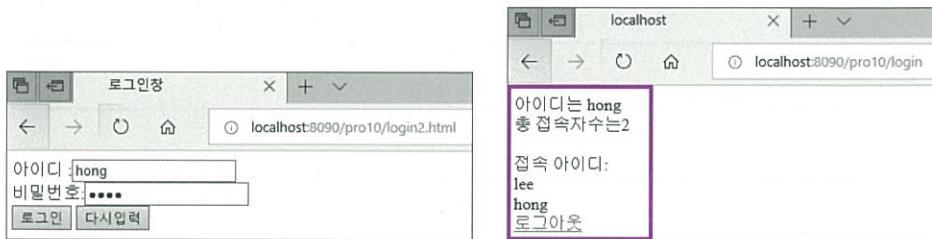
7. 실행하면 사용자마다 로그인/로그아웃 시 접속자수와 접속자 ID를 표시해 줍니다. 다음은 첫 번째 아이디로 로그인한 결과입니다.

❖ 그림 10-40 크롬에서 로그인 시 결과



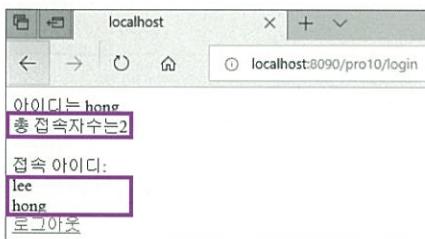
8. 이번에는 인터넷 익스플로러에서 두 번째 ID로 로그인하면 다음과 같이 현재 접속자수와 접속자 ID가 출력됩니다.

▼ 그림 10-41 익스플로러에서 로그인 시 결과



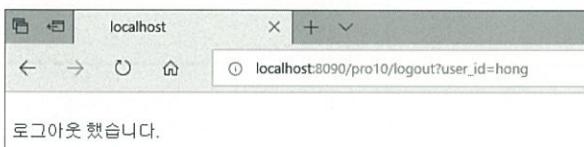
9. 다시 크롬에서 화면을 갱신하면 다음과 같이 현재 접속자수와 접속자 ID가 표시됩니다.

▼ 그림 10-42 크롬에서 현재 접속자 정보 표시



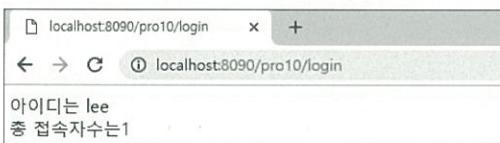
10. 익스플로러에서 로그아웃을 클릭합니다.

▼ 그림 10-43 익스플로러에서 로그아웃 클릭



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

▼ 그림 10-44 크롬 화면 재요청 시 접속자 정보 표시



이상으로 서블릿에서 제공하는 리스너의 기능에 대해 알아봤습니다. 이처럼 다른 리스너에 대해서도 세부 기능을 익히고 나면 고급 기능도 쉽게 구현할 수 있습니다.

Digitized by srujanika@gmail.com

• 100% Satisfaction Guaranteed •

Digitized by srujanika@gmail.com

Digitized by srujanika@gmail.com

Digitized by srujanika@gmail.com

www.nature.com/scientificreports/

Digitized by srujanika@gmail.com

Digitized by srujanika@gmail.com

Digitized by srujanika@gmail.com

www.nature.com/scientificreports/

Digitized by srujanika@gmail.com

Digitized by srujanika@gmail.com

Digitized by srujanika@gmail.com

11 장

JSP 정의와 구성 요소

11.1 JSP 등장 배경

11.2 JSP의 3단계 작업 과정

11.3 JSP 페이지 구성 요소

11.4 디렉티브 태그

11.1

JSP 등장 배경

초기 웹 프로그램은 서블릿을 이용해서 구현했습니다. 그런데 인터넷 사용자가 폭발적으로 증가하고 사용자에게 보여주는 화면의 기능이나 구성이 복잡해짐에 따라 사용자를 고려하는 화면 요구 사항도 점점 늘어났습니다. 그래서 현재는 프로그래머가 서블릿으로 화면을 구현하지 않고 주로 디자이너가 이 일을 담당하는 추세입니다.

앞 장에서 살펴봤듯이 기존 서블릿에서 화면을 구현할 때는 서블릿의 응답 기능을 이용했습니다. 자바 코드를 이용해 HTML 태그를 브라우저로 전송하는 방식이죠. 그런데 화면 구성이 복잡해짐에 따라 디자이너의 역할이 커지기 시작했는데 디자이너는 개발자와 달리 자바 코드에는 익숙하지 않은 경우가 많아 화면 기능 구현 시 많은 불편함이 있었습니다. 그래서 서블릿의 기능 중 별도로 화면 기능을 디자이너가 작업하기 쉽게 하기 위해 JSP가 등장했습니다.

11.1.1 서블릿으로 화면 구현 시 문제점

코드 11-1은 웹 프로그램 기능을 구현할 때 사용하는 서블릿 코드의 예입니다. 서블릿은 이처럼 응답 기능을 이용해 화면을 구현합니다. 서블릿 코드는 간단한 기능이므로 화면 기능을 구현하기도 그리 어렵지 않습니다.

코드 11-1 서블릿 코드 예시(LoginServlet.java)

```
...
@WebServlet("/login")
public class LoginServlet extends HttpServlet{
    public void init() throws ServletException {
        System.out.println("init 메서드 호출");
    }

    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 id = request.getParameter("user_id");
        String pw = request.getParameter("user_pw");
    }
}
```

비즈니스 로직

```

System.out.println("아이디 : "+ id);
System.out.println("패스워드 : "+ pw);

String data="<html>";
data+="<body>";
data+="아이디 : " + id ;
data+="  
";
data+="패스워드 : " + pw;
data+="</html>";
data+="</body>";
out.print(data);
}

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

```

그림 11-1은 실제 운영 중인 온라인 서점, 즉 도서 쇼핑몰 화면입니다. 그리고 그림 11-2는 그 화면을 구성하는 자바스크립트 및 HTML 코드입니다.

▼ 그림 11-1 온라인 서점 화면



▼ 그림 11-2 온라인 서점 구현 HTML과 자바스크립트 코드

```

771 function getExtension(filePath) { //파일의 확장자를 가짐
772     var testIndex = -1;
773     var extension = "";
774
775     testIndex = filePath.lastIndexOf(".");
776     if (testIndex != -1) {
777         extension = filePath.substring(testIndex + 1, filePath.length);
778     }
779     return extension.toLowerCase();
780 }
781
782 function regReview() {
783     var form = document.reviewForm;
784
785     alert("로그인 후 작성 가능합니다.");
786     in_coolLogin(location.href);
787     return;
788 }
789
790 if(jQuery('#reviewForm input[name="imageFile"]').val() != null || jQuery('#inputImage').val() == ""){
791     var form = document.reviewForm;
792     var extension = getExtension(form.imageFile.value);
793     if(extension != ".jpg" & extension != ".jpeg"){
794         alert("jpg, jpeg 확장만 사용할 수 있습니다.");
795         return;
796     }
797 }
798
799 if(jQuery('#book_review_textarea').val() == "" || jQuery('#book_review_textarea').val() == "내용을 입력해주세요. 주제와 무관한 댓글, 악플은 삭제될 수 있습니다."){
800     alert("내용을 입력하세요.");
801     return;
802 }
803
804 if(jQuery('#inputName rating').val() == "0"){
805     alert("인강료 평가를 해주세요.");
806     return;
807 }
808
809 if(jQuery('#coolForm').submit() == false){
810     return;
811 }
812
813 if(confirm("교과 도록하시겠습니까?")){
814 }
```

```

1429 function divShow(divName){
1430     document.getElementById(divName).style.display = "block";
1431     for (i=0;i<inSelect.length;i++){inSelect[i].style.display = "none"};
1432 }
1433 function divHidden(divName){
1434     document.getElementById(divName).style.display = "none";
1435     for (i=0;i<inSelect.length;i++){inSelect[i].style.display = ""};
1436 }
1437 function setHeight(hidden){
1438     for (i=0;i<inSelect.length;i++){inSelect[i].style.display = "none"};
1439 }
1440 function setHeightShow(){
1441     for (i=0;i<inSelect.length;i++){inSelect[i].style.display = ""};
1442 }
1443
1444 //function setHeightShow(){
1445 //    for (i=0;i<inSelect.length;i++){inSelect[i].style.display = ""};
1446 //}
1447
1448 //## [이름, 바로 등록] ###
1449 function bookmark(url) {
1450     var formname = document.ForceFav;
1451     formname.target = '_framew';
1452     formname.action = "http://www.kyobobook.co.kr/index.fav.jsp?%24NNH00bookmarkborderClick<<23";
1453     formname.method = 'post';
1454     formname.submit();
1455 }
1456
1457 function bookmark(title,url) {
1458     var elem = document.createElement('a');
1459     window.sidebar.addPanel(title, url, '');
1460     else if(window.cores && window.print)
1461     { // opera
1462         var elem = document.createElement('x');
1463         elem.setAttribute('href',url);
1464         elem.setAttribute('title',title);
1465         elem.setAttribute('rel','sidebar');
1466         elem.click();
1467     }
1468     else if(document.all) // ie
1469     window.external.AddFavorite(url, title);
1470 }
1471 
```

뭔가 복잡해 보이죠? 이처럼 웹 사이트 화면 기능이 복잡해지면 화면을 나타내는 코드 구현 역시 복잡해집니다. 따라서 기존 서블릿이 처리하는 기능 중 비즈니스 로직 기능과 화면 기능을 분리해야 한다는 목소리가 점점 높아지고 있습니다.

JSP는 디자이너 입장에서 화면의 수월한 기능 구현과 개발 후 화면의 편리한 유지관리를 목적으로 도입되었습니다. 기존 서블릿에서는 자바 코드를 기반으로 문자열을 사용해 HTML과 자바스크립트로 화면을 구현했으나 JSP는 이와 반대로 HTML, CSS와 자바스크립트를 기반으로 JSP 요소들을

사용해 화면을 구현합니다.

즉, JSP의 등장 배경을 정리하면 다음과 같습니다.

[문제점]

- 웹 프로그램의 화면 기능이 복잡해지므로 서블릿의 자바 기반으로 화면 기능 구현 시 어려움이 발생한다.
- 디자이너 입장에서 화면 구현 시 자바 코드로 인해 작업이 어렵다.
- 서블릿에 비즈니스 로직과 화면 기능이 같이 있다 보니 개발 후 유지관리가 어렵다.

[해결책]

- 서블릿의 비즈니스 로직과 결과를 보여주는 화면 기능을 분리하자!
- 비즈니스 로직과 화면을 분리함으로써 개발자는 비즈니스 로직 구현에 집중하고, 디자이너는 화면 기능 구현에만 집중하자!
- 개발 후 재사용성과 유지관리가 훨씬 수월해진다!

11.1.2 JSP의 구성 요소

JSP라고 해서 기존의 웹 페이지를 구현하는 HTML이나 자바스크립트와 관련이 없다는 말이 아닙니다. 여러 번 언급했듯이 JSP는 HTML과 CSS와 자바스크립트를 기반으로 JSP에서 제공하는 여러 가지 구성 요소들을 사용해 화면을 구현하는 기술입니다. 주로 웹 프로그램의 화면 기능과 모델2 기반 MVC에서 뷰(View) 기능을 담당하죠.

JSP의 구성 요소는 다음과 같습니다.

- HTML 태그, CSS 그리고 자바스크립트 코드
- JSP 기본 태그
- **JSP 액션 태그**
- 개발자가 직접 만들거나 프레임워크에서 제공하는 커스텀(custom) 태그

11.2

JSP의 3단계 작업 과정

서블릿에서는 자바 코드와 함께 원하는 HTML 태그를 사용해 브라우저로 전송해서 화면을 구현했습니다(6.4절 참고). 즉, `println()`과 같은 자바 코드를 사용해 HTML 화면을 구성했습니다. 따라서 서블릿으로 화면을 구현하려면 화면에 해당하는 HTML 태그를 브라우저로 전송해 주기만 하면 브라우저가 받아서 실시간으로 구현해 줍니다.

그런데 JSP는 HTML, CSS와 자바스크립트는 물론이고 JSP에서 제공하는 여러 가지 구성 요소가 화면을 구현하는 데 사용됩니다. 그러다 보니 JSP 파일 자체를 브라우저로 전송하면 브라우저는 JSP 요소들을 인식하지 못합니다. 따라서 JSP는 톰캣 컨테이너에 의해 브라우저로 전송되기 전에 실행 단계를 거쳐야 합니다.

11.2.1 톰캣 컨테이너에서 JSP 변환 과정

JSP 파일은 다음과 같이 3단계를 거쳐 실행됩니다.

1. **변환 단계**(Translation Step): 컨테이너는 JSP 파일을 자바 파일로 변환합니다.
2. **컴파일 단계**(Compile Step): 컨테이너는 변환된 자바(java) 파일을 클래스(class) 파일로 컴파일합니다.
3. **실행 단계**(Interpret Step): 컨테이너는 class 파일을 실행하여 그 결과(HTML, CSS와 자바스크립트 코드)를 브라우저로 전송해 출력합니다.

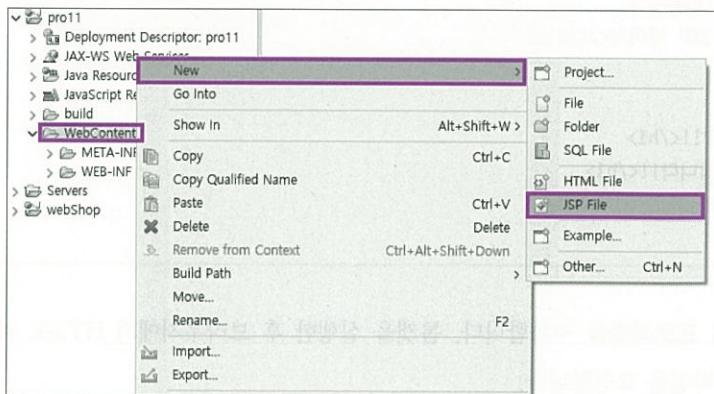
브라우저에서 JSP 파일을 요청하면 톰캣 컨테이너는 요청된 JSP 파일을 자바 파일(.java)로 변환합니다. 그리고 변환된 자바 파일을 클래스 파일(.class)로 컴파일합니다. 이 클래스 파일을 실행하여 브라우저로 결과값을 전송하면 JSP가 브라우저 화면에 표시됩니다. 즉, 브라우저로 전송되는 결과는 HTML, CSS와 자바스크립트로 변환된 파일입니다.

11.2.2 이클립스에서 JSP 변환 과정 실습

지금부터 이클립스에서 JSP 파일을 만든 후 브라우저에서 요청하는 과정을 살펴보겠습니다.

1. 이클립스에서 새 프로젝트 pro11을 만들고 WebContent 폴더에서 마우스 오른쪽 버튼을 클릭한 후 New > JSP File을 선택합니다.

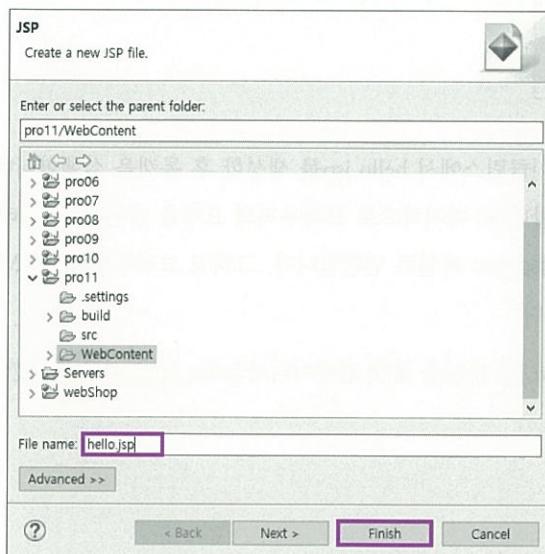
▼ 그림 11-3 New > JSP File 선택



반드시 servlet_api.jar를 설정해 줍니다(5.4절 참고).

2. 파일 이름으로 hello.jsp를 입력한 후 Finish를 클릭합니다.

▼ 그림 11-4 파일 이름으로 hello.jsp 입력 후 Finish 클릭



3. 생성된 JSP 파일에 간단한 HTML 태그와 메시지를 작성합니다.

코드 11-2 pro11/WebContent/hello.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>첫 번째 JSP 페이지</title>
</head>
<body>
<h1>hello JSP!!</h1>
<h1>JSP 실습입니다!!</h1>
</body>
</html>
```

4. 톰캣 컨테이너에 프로젝트를 추가합니다. 톰캣을 실행한 후 브라우저에서 HTML 파일을 요청하듯이 JSP 파일을 요청합니다.

- <http://ip주소:포트번호/프로젝트이름/JSP파일이름>

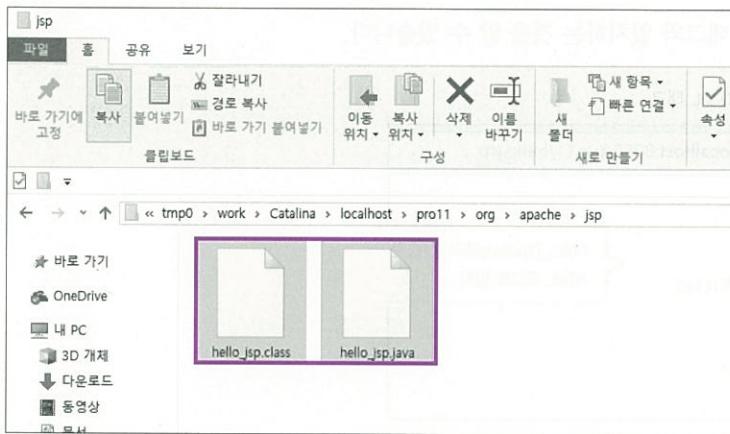
▼ 그림 11-5 브라우저에서 hello.jsp로 요청



다시 한번 JSP 실행 과정을 정리하면 이클립스에서 hello.jsp를 생성한 후 톰캣을 실행합니다. 그런 다음 브라우저에서 hello.jsp로 요청합니다. 마지막으로 브라우저의 요청을 받은 톰캣 컨테이너는 해당 JSP 파일을 읽어 들여와 hello.jsp.java 파일로 변환합니다. 그리고 브라우저로 HTML 형식의 결과를 전송하여 화면에 표시합니다.

그림 11-6에 브라우저에서 요청한 hello.jsp 파일을 톰캣 컨테이너가 hello.jsp.java로 변환한 자바 파일과 클래스 파일을 나타내었습니다.

▼ 그림 11-6 hello_jsp.java 파일로 변환된 상태



Tip ☆ 이클립스에서 개발할 경우 이클립스의 workspace 아래에 생성됩니다.

%이클립스_workspace% \metadata\plugins\

org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\pro11\org\apache\jsp

hello.jsp.java를 VS Code나 편집기로 열어보면 앞에서 배운 서블릿의 여러 가지 기능으로 변환되어 hello.jsp의 HTML 태그를 브라우저로 전송해 주는 것을 알 수 있습니다. 서블릿에서는 개발자가 일일이 HTML 태그를 만들어서 `println()`으로 전송해 주었으나 JSP는 요청 시 컨테이너에서 자동으로 JSP 파일에 있는 HTML 태그와 자바스크립트를 브라우저로 전송해 줍니다.

▼ 그림 11-7 hello.jsp.java로 변환한 후 브라우저로 전송한 HTML 태그

```

105 try {
106     response.setContentType("text/html; charset=UTF-8");
107     pageContext = _jspxFactory.getPageContext(this, request, response,
108         null, true, 8192, true);
109     _jspx_page_context = pageContext;
110     application = pageContext.getServletContext();
111     config = pageContext.getServletConfig();
112     session = pageContext.getSession();
113     out = pageContext.getOut();
114     _jspx_out = out;
115
116     out.write("\r\n");
117     out.write("<!DOCTYPE html>\r\n");
118     out.write("<html>\r\n");
119     out.write("<head>\r\n");
120     out.write("<meta charset='UTF-8'>\r\n");
121     out.write("<title>첫번째 JSP 페이지</title>\r\n");
122     out.write("</head>\r\n");
123     out.write("<body>\r\n");
124     out.write("  <h1>Hello JSP!!</h1>\r\n");
125     out.write("  <h1>JSP 실습입니다!!</h1>\r\n");
126     out.write("</body>\r\n");
127     out.write("</html>\r\n");
128     out.write("\r\n");
129     out.write("\r\n");
130     out.write("\r\n");
131     out.write("\r\n");
132     out.write("\r\n");
133     out.write("\r\n");
134 } catch (java.lang.Throwable t) {

```

그림 11-8은 hello.jsp 요청 시 브라우저로 전송된 HTML 태그를 나타낸 것입니다. 앞서 hello.jsp.java가 전송한 HTML 태그와 일치하는 것을 알 수 있습니다.

▼ 그림 11-8 브라우저로 전송된 HTML 태그

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>첫번째 JSP 페이지</title>
6 </head>
7 <body>
8   <h1>Hello JSP!!</h1>
9   <h1>JSP 실습입니다!!</h1>
10 </body>
11 </html>
```

hello.jsp.java에서 전송된
HTML 태그와 일치

11.3 JSP 페이지 구성 요소

JAVA WEB

JSP의 동작 원리를 알았으니 이번에는 JSP에서 HTML 태그와 같이 사용되는 여러 가지 JSP 구성 요소들의 기능을 알아보겠습니다.

JSP 페이지에서 사용되는 여러 가지 구성 요소들은 다음과 같습니다.

- 디렉티브 태그(Directive Tag) → %
- 스크립트 요소(Scripting Element): 주석문, 스크립트릿(Scriptlet), 표현식, 선언식
- 표현 언어(Expression Language)
- 내장 객체(내장 변수)
- 액션 태그(Action Tag)
- 커스텀 태그(Custom Tag)

이 중 디렉티브 태그와 스크립트 요소는 JSP가 처음 나왔을 때 많이 사용했던 기능이고 그 외 요소들은 JSP에서 추가한 기능들입니다.

11.4

디렉티브 태그



디렉티브 태그는 주로 JSP 페이지에 대한 전반적인 설정 정보를 지정할 때 사용하는 태그입니다.

디렉티브 태그의 종류는 다음과 같습니다.

- **페이지 디렉티브 태그**(Page Directive Tag): JSP 페이지의 전반적인 정보를 설정할 때 사용합니다.
- **인클루드 디렉티브 태그**(Include Directive Tag): 공통으로 사용하는 JSP 페이지를 다른 JSP 페이지에 추가할 때 사용합니다. → **Component**
- **태그라이브 디렉티브 태그**(Taglib Directive Tag): 개발자나 프레임워크에서 제공하는 태그를 사용할 때 사용합니다.

11.4.1 페이지 디렉티브 태그 정의와 사용법

먼저 페이지 디렉티브 태그에 대해 알아보겠습니다. 표 11-1은 페이지 디렉티브 태그를 이용해 지정하는 여러 가지 속성을 나타낸 것입니다.

▼ 표 11-1 페이지 디렉티브 태그로 설정하는 여러 가지 JSP 속성

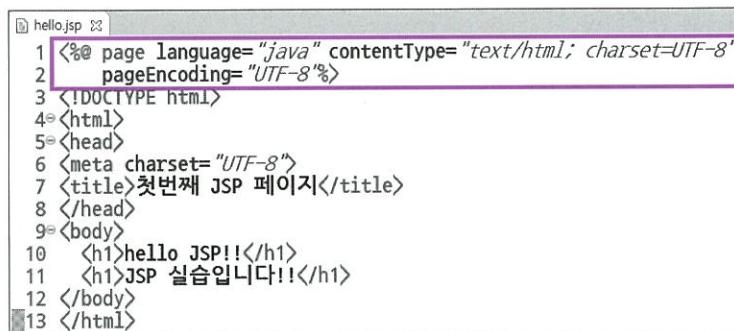
속성	기본값	설명
info	없음	JSP 페이지를 설명해 주는 문자열을 지정합니다.
language	"java"	JSP 페이지에서 사용할 언어를 지정합니다.
contentType	"text/html"	JSP 페이지 출력 형식을 지정합니다.
import	없음	JSP 페이지에서 다른 패키지의 클래스를 임포트할 때 지정합니다.
session	"true"	JSP 페이지에서 HttpSession 객체의 사용 여부를 지정합니다.
buffer	"8kb"	JSP 페이지 출력 시 사용할 버퍼 크기를 지정합니다.
autoFlush	"true"	JSP 페이지의 내용이 출력되기 전 버퍼가 다 채워질 경우 동작을 지정합니다.
errorPage	"false"	JSP 페이지 처리 도중 예외가 발생할 경우 예외 처리 담당 JSP 페이지를 지정합니다.
isErrorPage	"false"	현재 JSP 페이지가 예외 처리 담당 JSP 페이지인지를 지정합니다.
pageEncoding	"ISO-8859-1"	JSP 페이지에서 사용하는 문자열 인코딩을 지정합니다.
isELIgnored	"true"	JSP 2.0 버전에서 추가된 기능으로 EL 사용 유무를 지정합니다.

페이지 디렉티브 형식은 다음과 같이 <%@page %> 안에 속성과 값을 나열하면 됩니다.

```
<%@ page 속성1="값1" 속성2="값2" 속성3="값3".... %>
```

그런데 이클립스에서 JSP 페이지를 만들면 자동으로 페이지 디렉티브 태그가 생성됩니다.

▼ 그림 11-9 이클립스에서 자동으로 생성된 페이지 디렉티브 태그



```
hello.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>첫번째 JSP 페이지</title>
8 </head>
9 <body>
10 <h1>hello JSP!!</h1>
11 <h1>JSP 실습입니다!!</h1>
12 </body>
13 </html>
```

11.4.2 페이지 디렉티브 태그 사용 예제

다음은 페이지 디렉티브 태그를 적용한 JSP 페이지입니다.

코드 11-2 pro11/WebContent/hello2.jsp

```
<%@ page contentType="text/html; charset=utf-8"
    import="java.util.*"
    language="java"
    session="true"
    buffer="8kb"
    autoFlush="true"
    isThreadSafe="true"
    info="(ShoppingMall.....)"
    isErrorPage="false"
    errorPage="" %>
```

import 속성을 제외한 다른 속성은
한 번만 선언해야 합니다.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>페이지 디렉티브 연습</title>
</head>

<body>
```

```

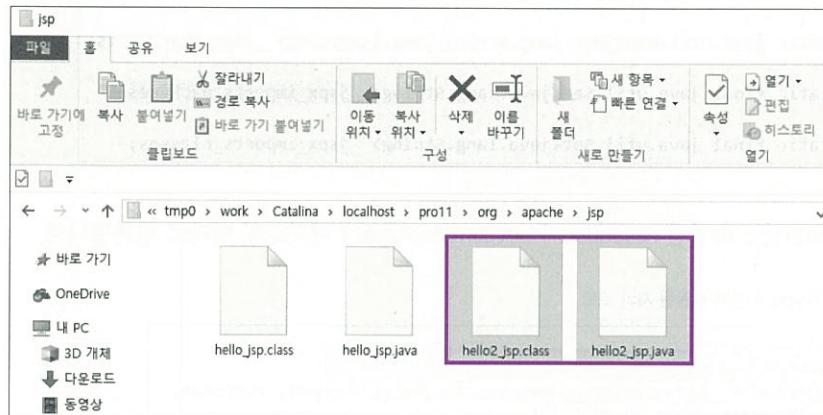
<h1>쇼핑몰 구현 중심 JSP입니다.!!!</h1>
</body>

</html>

```

hello2.jsp를 브라우저에서 요청 시 톰캣 컨테이너는 JSP 파일을 자바 파일로 변환합니다.

▼ 그림 11-10 JSP 파일이 변환되어 생성된 java 파일



hello2.jsp.java 파일을 열어 보면 hello2.jsp의 페이지 디렉티브에서 설정한 정보가 모두 자바 코드로 변환된 것을 알 수 있습니다. 우선 그림 11-11을 보면 import 속성이 변환되어 import문에 추가됩니다.

▼ 그림 11-11 import 속성이 변환된 자바 import문

```

9 package org.apache.jsp;
10
11 import javax.servlet.*;
12 import javax.servlet.http.*;
13 import javax.servlet.jsp.*;
14 import java.util.*;

```

그리고 info 속성이 변환되어 getServletInfo() 메서드에서 서블릿 정보를 반환합니다.

▼ 그림 11-12 info 속성이 변환된 자바 코드

```
16 public final class hello2_jsp extends org.apache.jasper.runtime.HttpJspBase
17     implements org.apache.jasper.runtime.JspSourceDependent,
18             org.apache.jasper.runtime.JspSourceImports {
19
20     public java.lang.String getServletInfo() {
21         return "<ShoppingMall.....>";
22     }
23
24     private static final javax.servlet.jsp.JspFactory _jspxFactory =
25         javax.servlet.jsp.JspFactory.getDefaultFactory();
26
27     private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;
28
29     private static final java.util.Set<java.lang.String> _jspx_imports_packages;
30
31     private static final java.util.Set<java.lang.String> _jspx_imports_classes;
```

마지막으로 contentType 속성인 response가 setContentType() 메서드의 인자로 변환됩니다.

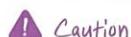
▼ 그림 11-13 contentType 속성이 변환된 자바 코드

```
111 try {
112     response.setContentType("text/html;charset=utf-8");
113     pageContext = _jspxFactory.getPageContext(this, request, response,
114         "", true, 8192, true);
115     _jspx_page_context = pageContext;
116     application = pageContext.getServletContext();
117     config = pageContext.getServletConfig();
118     session = pageContext.getSession();
119     out = pageContext.getOut();
120     _jspx_out = out;
```

서블릿에서는 필요한 클래스 파일을 import문을 이용해서 일일이 설정해 주었습니다. 하지만 이제는 JSP 페이지에서 페이지 디렉티브 태그를 이용해서 설정합니다. 그리고 페이지 디렉티브 태그는 import 속성을 제외한 다른 속성을 한 번만 선언해 주어야 합니다.

▼ 그림 11-14 실행 결과





페이지 디렉티브 속성을 설정할 때는 대소문자에 유의하세요!

JSP 페이지에서 페이지 디렉티브 태그의 속성 이름을 잘못 설정하면 다음과 같은 오류가 발생합니다.

그림 11-15 autoFlush 속성 이름을 autoflush로 잘못 설정한 경우

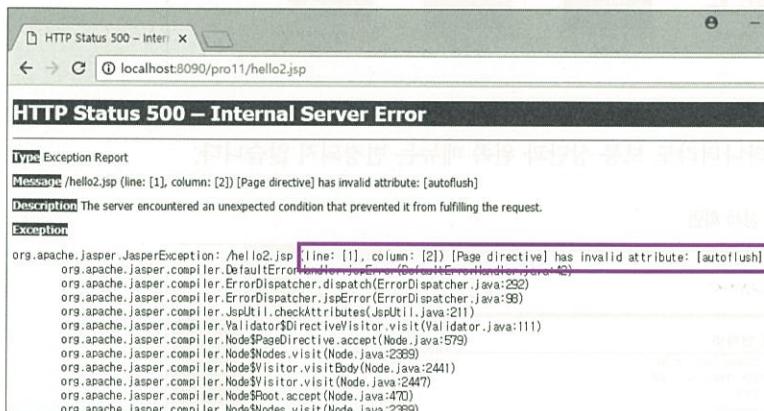
```

1 <%@ page contentType="text/html; charset=utf-8"
2         import="java.util.*"
3         language="java"
4         session="true"
5         buffer="8kb"
6         autoFlush="true"           autoFlush 속성 이름이
7         isThreadSafe="true"        잘못 설정되었습니다.
8         info="(ShoppingMall....."
9         isErrorPage="false"
10        errorPage="" %>
11 <!DOCTYPE html>

```

브라우저에서 재요청 시 수정된 JSP 파일을 다시 컴파일하는 과정에서 잘못된 속성에 대해 오류를 발생시킵니다.

그림 11-16 페이지 디렉티브 속성을 잘못 설정한 경우 브라우저 요청 결과



페이지 디렉티브 태그에 사용되는 속성 이름은 대소문자를 정확히 사용해야 한다는 점을 유의하세요(표 11-1 참고).

11.4.3 인클루드 디렉티브 태그 정의와 사용법

JSP 페이지로 웹 페이지를 만들다 보면 제목이나 로고를 표시하는 상단, 메뉴를 표시하는 왼쪽 단 은 화면이 바뀌더라도 일정하게 유지되는 경우가 많습니다.

▼ 그림 11-17 쇼핑몰 메인 화면

상품 상세 화면이 나타나더라도 보통 상단과 왼쪽 메뉴는 변경되지 않습니다.

▼ 그림 11-18 쇼핑몰 상품 상세 화면

이런 공통 화면을 일일이 JSP 페이지마다 만들어 사용해야 한다면 불편하겠죠. 그래서 JSP에서는 공통으로 사용되는 JSP 페이지를 미리 만들어 놓고 다른 JSP 페이지 요청 시 인클루드 디렉티브 태그를 사용합니다. 그러면 재사용성이 높아질 뿐 아니라 유지관리도 수월해집니다.

인클루드 디렉티브 태그란 여러 JSP 페이지에서 사용되는 공통 JSP 페이지를 만든 후 다른 JSP 페이지에서 공통 JSP 페이지를 포함시켜 사용하는 기능을 말합니다.

인클루드 디렉티브 태그의 특징은 다음과 같습니다.

- 재사용성이 높다.
- JSP 페이지의 유지관리가 쉽다.

인클루드 디렉티브 태그의 형식은 다음과 같습니다.

```
<%@ include file="공통기능.jsp" %>
```

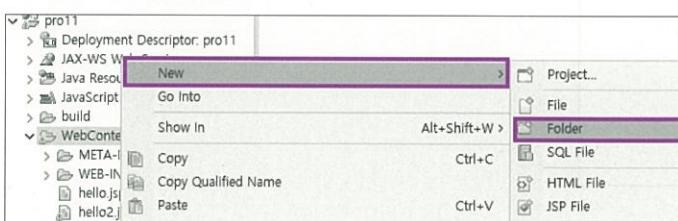
11.4.4 인클루드 디렉티브 태그 이용해 이미지 삽입하기

인클루드 디렉티브 태그를 이용해 다른 JSP 파일의 이미지를 삽입하는 예제를 살펴보겠습니다.

우선 프로젝트에 웹 프로그램에서 사용할 이미지를 저장할 image 폴더를 생성한 후 이미지를 복사합니다.

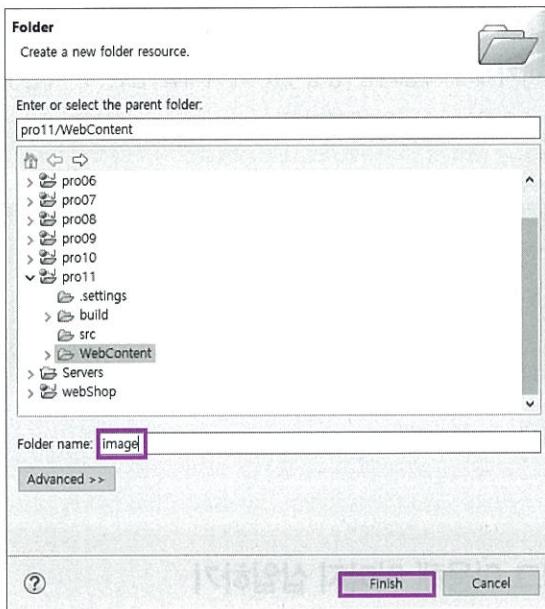
1. 프로젝트의 WebContent에서 마우스 오른쪽 버튼을 클릭한 후 New > Folder를 선택합니다.

▼ 그림 11-19 New > Folder 선택



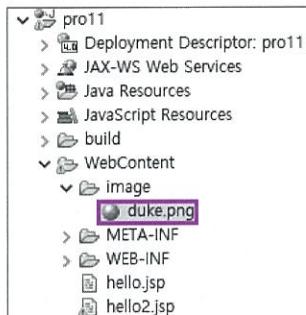
2. 폴더 이름으로 image를 입력한 후 Finish를 클릭합니다.

▼ 그림 11-20 폴더 이름으로 image 입력 후 Finish 클릭



3. 이미지를 복사한 후 image 폴더에 붙여 넣습니다.

▼ 그림 11-21 이미지 폴더 생성



4. 다음과 같이 인클루드 디렉티브 태그를 이용해 두 개의 jsp 파일을 작성합니다. duke_image.jsp

jsp는 이미지를 화면에 표시하는 기능을 하고, include.jsp는 인클루드 디렉티브 태그를 이용해 duke_image.jsp를 삽입하는 기능을 합니다.

코드 11-3 pro11/WebContent/duke_image.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"  
pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>duke_image</title>
</head>
<body>
     ----- image 폴더의 duke.png를 표시합니다.
</body>
</html>
```

코드 11-4 pro11/WebContent/include.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>인클루드 디렉티브</title>
</head>
<body>
    <h1>안녕하세요. 쇼핑몰 중심 JSP 시작입니다!!! </h1><br>
    <%@ include file="duke_image.jsp" %><br>
    <h1>안녕하세요. 쇼핑몰 중심 JSP 끝 부분입니다.!!!</h1>
</body>
</html>
```

----- 인클루드 디렉티브 태그를 이용해 duke_image.jsp를 포함합니다.

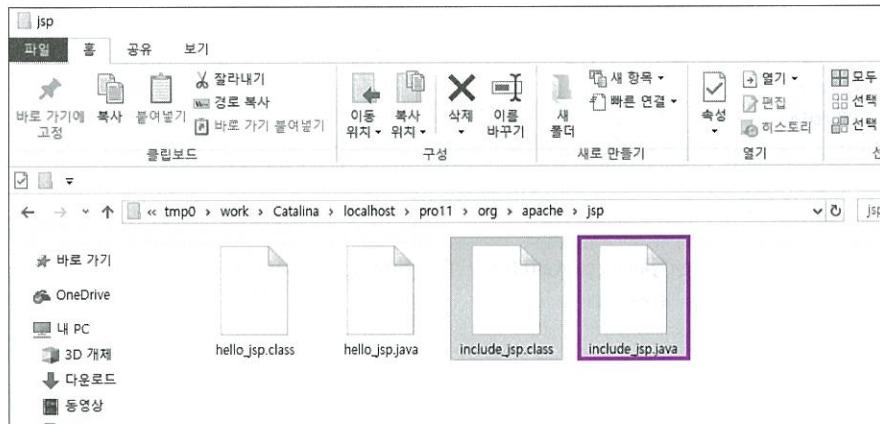
5. 브라우저에서 요청하면 include.jsp 안에 duke_image.jsp가 포함되어 표시됩니다.

▼ 그림 11-22 실행 결과



6. 윈도 탐색기에서 다음 경로에 들어가면 브라우저에서 include.jsp를 요청할 때 변환된 자바 파일이 생성된 것을 볼 수 있습니다. 자바 파일을 열어보면 인클루트 디렉티브 태그로 포함된 duke_image.jsp의 HTML 태그가 합쳐져 있습니다.

▼ 그림 11-23 브라우저 요청 후 자바로 변환된 JSP 파일



▼ 그림 11-24 인클루드 디렉티브 태그에 의해 합쳐진 HTML 태그

```

122     out.write("\r\n");
123     out.write("<!DOCTYPE html>\r\n");
124     out.write("<html>\r\n");
125     out.write("<head>\r\n");
126     out.write("  <meta charset=\"UTF-8\">\r\n");
127     out.write("  <title>인클루드 디렉티브</title>\r\n");
128     out.write("</head>\r\n");
129     out.write("<body>\r\n");
130     out.write("  <h1>안녕하세요. 쇼핑몰 중심 JSP 시작입니다!!! </h1><br>\r\n");
131     out.write("  ");
132     out.write("\r\n");
133     out.write("\r\n");
134     out.write(<!DOCTYPE html>\r\n");
135     out.write("<html>\r\n");
136     out.write("<head>\r\n");
137     out.write("  <meta charset=\"UTF-8\">\r\n");
138     out.write("  <title>duke_image</title>\r\n");
139     out.write("</head>\r\n");
140     out.write("<body>\r\n");
141     out.write("  <img src=\"./image/duke.png\" />\r\n");
142     out.write("</body>\r\n");
143     out.write("</html>\r\n");
144     out.write("<br>\r\n");
145     out.write("  <h1>안녕하세요. 쇼핑몰 중심 JSP 끝 부분입니다.!!!</h1>\r\n");
146     out.write("</body>\r\n");
147     out.write("</html>\r\n");
148 } catch (java.lang.Throwable t) {

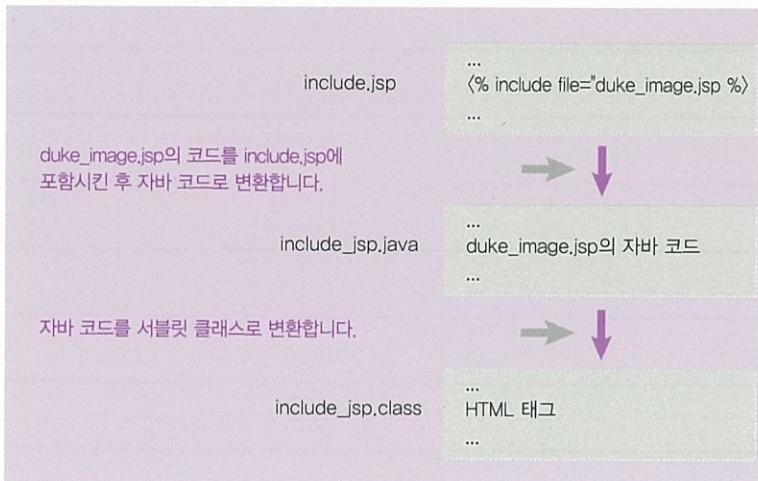
```

duke_image.jsp 페이지가 포함됨

인클루드 디렉티브 태그를 이용해 JSP 페이지를 요청하면 포함되는 duke_image.jsp의 자바 코드가 include_jsp.java 파일과 합쳐져 브라우저로 전송됩니다.

인클루드 디렉티브 태그를 이용해 JSP 페이지가 호출되는 과정을 살펴볼까요?

▼ 그림 11-25 인클루드 디렉티브 태그 실행 과정



이처럼 인클루드 디렉티브 태그를 이용해 JSP 페이지를 요청하면 요청하는 JSP 페이지에 대해 실행하는 자바 파일은 단 한 개만 생성됩니다(그림 11-23 참조).

12장

JSP 스크립트 요소 기능

- 12.1 JSP 스크립트 요소
- 12.2 선언문 사용하기
- 12.3 스크립트릿 사용하기
- 12.4 표현식 사용하기
- 12.5 JSP 주석문 사용하기
- 12.6 스크립트 요소 이용해 실습하기
- 12.7 내장 객체(내장 변수) 기능
- 12.8 JSP 페이지 예외 처리하기
- 12.9 JSP welcome 파일 지정하기
- 12.10 스크립트 요소 이용해 회원 정보 조회하기

지금까지 컨테이너에서 JSP의 동작 과정을 알아보았습니다. HTML 태그는 컨테이너 작업 없이 바로 브라우저로 전송되어 화면을 구현하기 때문에 HTML 태그로 화면을 구현하면 조건에 따라 화면을 동적으로 구성할 수 없습니다. 반면에 JSP는 컨테이너에서 자바로 변환되는 과정을 거쳐므로 JSP에서 제공하는 스크립트 요소를 이용하면 조건이나 상황에 맞게 HTML 태그를 선택적으로 전송할 수 있습니다. 즉, 화면을 동적으로 구성할 수 있습니다.

지금부터 JSP 스크립트 요소 기능에 대해 좀 더 알아보겠습니다.

12.1 JSP 스크립트 요소

JAVA WEB

JSP 스크립트 요소(Scripting Element)란 JSP 페이지에서 여러 가지 동적인 처리를 제공하는 기능으로, <% %> 기호 안에 자바 코드로 구현합니다. <% %> 기호를 **스크립트릿**(scriptlet)이라고 부릅니다.

스크립트 요소의 종류는 다음과 같이 세 가지입니다

- **선언문**(declaration tag): JSP에서 변수나 메서드를 선언할 때 사용합니다.
- **스크립트릿**(scriptlet): JSP에서 자바 코드를 작성할 때 사용합니다.
- **표현식**(expression tag): JSP에서 변수의 값을 출력할 때 사용합니다.

먼저 선언문의 기능부터 알아보겠습니다.

12.2 선언문 사용하기

JAVA WEB

선언문(declaration tag)은 JSP 페이지에서 사용하는 멤버 변수나 멤버 메서드를 선언할 때 사용합니다. 선언문 안의 멤버는 서블릿 변환 시 서블릿 클래스의 멤버로 변환됩니다. 선언문의 형식은 다음과 같습니다.

```
<%! 멤버 변수 or 멤버 메서드 %>
```

JSP가 처음 나온 초기에는 이처럼 자바 코드를 이용해 JSP 페이지의 필요한 변수나 메서드를 구현했습니다.

12.2.1 JSP에서 선언문 실습

- 새 프로젝트 pro12를 만들고 hello.jsp 파일을 생성합니다.

▼ 그림 12-1 실습 파일 위치



- 선언문을 사용한 hello.jsp를 다음과 같이 작성합니다. 선언문은 일반적으로 JSP 페이지의 상단에서 주로 사용합니다.

코드 12-1 pro12/WebContent/hello.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%!
    String name = "듀크";
    public String getName(){ return name;}
%>
```

선언문을 이용해 멤버 변수 name과 멤버 메서드 getName()을 선언합니다.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>선언문 연습</title>
</head>
<body>
    <h1>안녕하세요 <%=name %> !!!</h1>
</body>
</html>
```

표현식을 이용해 선언문에서 선언한 name의 값을 출력합니다.

3. 브라우저에서 <http://localhost:8090/pro12/hello.jsp>로 요청합니다.

▼ 그림 12-2 실행 결과



4. 변환된 자바 코드를 보면 선언문에서 선언된 변수와 메서드는 서블릿 클래스의 멤버 변수와 멤버 메서드로 변환된 것을 알 수 있습니다. 따라서 선언문에서 선언된 변수는 JSP(서블릿 클래스) 안에서 자유롭게 접근할 수 있습니다.

▼ 그림 12-3 서블릿 클래스의 멤버로 변환된 상태

```
9 package org.apache.jsp;
10
11 import javax.servlet.*;
12 import javax.servlet.http.*;
13 import javax.servlet.jsp.*;
14
15 public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
16     implements org.apache.jasper.runtime.JspSourceDependent,
17     org.apache.jasper.runtime.JspSourceImports {
18
19
20     String name = "듀크";
21     public String getName(){ return name; } 서블릿 클래스의 멤버 변수와  
멤버 메서드로 변환됩니다.
22
23     private static final javax.servlet.jsp.JspFactory _jspxFactory =
24         javax.servlet.jsp.JspFactory.getDefaultFactory();
25
26     private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;
27
28     private static final java.util.Set<java.lang.String> _jspx_imports_packages;
29
30     private static final java.util.Set<java.lang.String> _jspx_imports_classes;
31
```

12.3

스크립트릿 사용하기

보통 웹 페이지는 디자이너가 주도적으로 구현하는 부분인 만큼, 웹 페이지 구현 시 디자이너에게 어려운 자바 코드는 거의 사용되지 않습니다. 스크립트릿은 초기의 JSP에서 자바 코드를 이용해 화면의 동적인 기능을 구현했습니다. 비록 현재 JSP 페이지에서는 거의 사용되지 않지만 자바 코드로 화면의 동적인 기능을 구현할 수 있다면 자바 코드를 대체해서 나온 여러 가지 태그들을 이해하는 데에도 분명 도움이 될 것입니다.

스크립트릿의 형식은 다음과 같습니다.

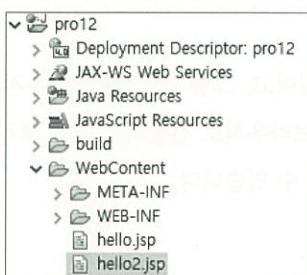
<% 자바 코드 %>

그럼 본격적으로 실습을 진행해 보겠습니다.

12.3.1 JSP에서 스크립트릿 실습

1. JSP에서 스크립트릿 실습을 위해 hello2.jsp 파일을 준비합니다.

▼ 그림 12-4 실습 파일 위치



2. 브라우저에서 JSP로 전송된 값을 얻기 위해 <% %> 안에 자바 코드를 사용하여 age 값을 가져옵니다.

코드 12-2 pro12/WebContent/hello2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%!
String name = "이순신";
```

```

public String getName(){ return name;}
%>
<% String age=request.getParameter("age"); %>
    ↑
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>스크립트릿 연습</title>
</head>
<body>
<h1>안녕하세요 <%=name %>님!!</h1>
<h1>나이는 <%=age %>살입니다!!</h1>
</body>
</html>

```

스크립트릿을 이용해 자바 코드를 작성합니다.

표현식을 이용해 전송된 나이를 출력합니다.

3. <http://localhost:8090/pro12/hello2.jsp?age=22>로 요청합니다.

▼ 그림 12-5 실행 결과

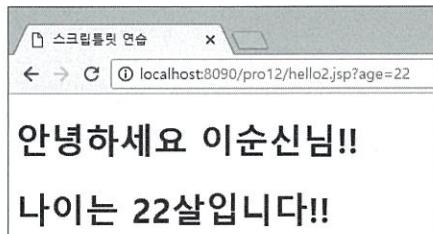


그림 12-6은 브라우저에서 JSP 페이지 요청 시 변경된 서블릿이고, 그림 12-7은 브라우저로 전송된 HTML 태그입니다. 이를 통해 JSP의 스크립트 요소는 브라우저로 전송되지 않고 브라우저로 전송되기 전에 컨테이너에서 자바 코드로 변환되는 것을 알 수 있습니다.

▼ 그림 12-6 서블릿으로 변경된 상태

```

109 try {
110     response.setContentType("text/html; charset=UTF-8");
111     pageContext = _jspxFactory.getPageContext(this, request, response,
112         null, true, 8192, true);
113     _jspx_page_context = pageContext;
114     application = pageContext.getServletContext();
115     config = pageContext.getServletConfig();
116     session = pageContext.getSession();
117     out = pageContext.getOut();
118     _jspx_out = out;
119
120     out.write('\r');
121     out.write('\n');
122     out.write("\r\n");
123     String age=request.getParameter("age");
124     out.write("\r\n");
125     out.write("\r\n");
126     out.write("<!DOCTYPE html>\r\n");
127     out.write("<html>\r\n");
128     out.write("<head>\r\n");
129     out.write("<meta charset=\"UTF-8\">\r\n");
130     out.write("<title>스크립틀릿 연습</title>\r\n");
131     out.write("</head>\r\n");
132     out.write("<body>\r\n");
133     out.write("<h1>안녕하세요 ");
134     out.print(name );
135     out.write("님!!</h1>\r\n");
136     out.write("    <h1>나이는 ");
137     out.print(age );
138     out.write("살입니다!!</h1>\r\n");

```

서블릿의 _jspService() 메서드
안의 자바 코드로 변환됩니다.

name과 age의 값이 print()로
브라우저로 전송됩니다.

▼ 그림 12-7 브라우저로 전송된 HTML 태그

```

5  <!DOCTYPE html>
6  <html>
7  <head>
8  <meta charset="UTF-8">
9      <title>스크립틀릿 연습</title>
10 </head>
11 <body>
12     <h1>안녕하세요 미순신님!!</h1>
13     <h1>나이는 22살입니다!!</h1>
14 </body>
15 </html>

```

<% %> 안에는 자바 코드만 쓸 수 있다는 점 꼭 기억하세요.

12.4

표현식 사용하기

표현식(expression tag)은 JSP 페이지의 정한 위치에 값을 출력하는 기능입니다. 즉, JSP 페이지에서 변수나 메서드의 결과값 등을 브라우저에 출력하는 용도로 사용합니다.

표현식의 형식은 다음과 같습니다.

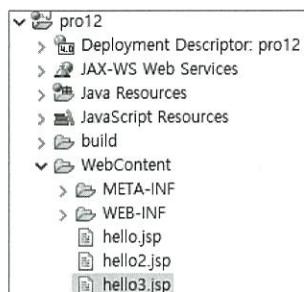
<%=값 or 자바 변수 or 자바 식%>

긴 설명보다는 바로 실습을 통해 표현식에 대해 알아보겠습니다.

12.4.1 JSP 페이지에서 표현식 실습

1. 다음과 같이 hello3.jsp 파일을 준비합니다.

▼ 그림 12-8 실습 파일 위치



2. 다음과 같이 hello3.jsp를 작성합니다. 표현식을 이용해 JSP 페이지에서 선언한 변수와 여러 가지 값을 HTML의 원하는 위치에 출력합니다. 이때 <%= %> 안의 자바 변수나 자바 식에는 세미콜론(;)이 있으면 안 됩니다.

코드 12-3 pro12/WebContent/hello3.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
String name = "이순신";
public String getName(){ return name;}
%>
```

```

<% String age=request.getParameter("age"); %>

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>표현식 연습</title>
</head>
<body>
    <h1>안녕하세요 <%=name %>님!!</h1>
    <h1>나이는 <%=age %>살입니다!!</h1>
    <h1>키는 <%=180 %>cm입니다!!</h1>
    <h1>나이+10은 <%=Integer.parseInt(age)+10 %>살입니다!!</h1>
</body>
</html>

```

(%)를 이용해 값을 출력합니다.

age의 값에 10을 더한 값을 출력합니다.

3. <http://localhost:8090/pro12/hello3.jsp?age=22>로 요청하여 결과를 확인합니다.

▼ 그림 12-9 실행 결과



그림 12-10은 서블릿으로 변환된 코드입니다. 표현식 안의 값은 `print()`를 이용해 브라우저에 출력됩니다.

▼ 그림 12-10 서블릿으로 변환된 상태

```
120 out.write('\r');
121 out.write('\n');
122 out.write(" \r\n");
123 String age=request.getParameter("age");
124 out.write(" \r\n");
125 out.write(" \r\n");
126 out.write("<!DOCTYPE html>\r\n");
127 out.write("<html>\r\n");
128 out.write("<head>\r\n");
129 out.write(" <meta charset=\"UTF-8\">\r\n");
130 out.write(" <title>표현식 연습</title>\r\n");
131 out.write("</head>\r\n");
132 out.write("<body>\r\n");
133 out.write(" <h1>안녕하세요!");
134 out.print(name );
135 out.write("님!!</h1>\\" 表현식의 원하는 위치에서 print()를
136 out.write(" <h1>나이는 ");
137 out.print(age );
138 out.write("살입니다!!</h1>\r\n");
139 out.write(" <h1>키는 ");
140 out.print(180 );
141 out.write("cm입니다!!</h1>\r\n");
142 out.write(" <h1>나이+10은 ");
143 out.print(Integer.parseInt(age)+10 );
144 out.write("살입니다!!</h1>\r\n");
145 out.write("</body>\r\n");
146 out.write("</html>\r\n");
147 } catch (java.lang.Throwable t) {
```

만약 선언문 안에 다음과 같이 세미콜론(:)을 추가하면 어떻게 될까요?

▼ 그림 12-11 선언문에 세미콜론(:) 추가

```
9 <!DOCTYPE html>
10 <html>
11 <head>
12   <meta charset="UTF-8">
13   <title>표현식 연습</title>
14 </head>
15 <body>
16   <h1>안녕하세요 <%=name %>님!!</h1>
17   <h1>나이는 <%=age %>살입니다!!</h1>
18   <h1>키는 <%=180 %>cm입니다!!</h1>
19   <h1>나이+10은 <%=Integer.parseInt(age)+10; %>살입니다!!</h1>
20 </body>
21 </html>
```

▼ 그림 12-12 세미콜론(:) 추가 후 요청 결과

```

HTTP Status 500 – Internal Server Error

Type Exception Report

Message Unable to compile class for JSP:

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

org.apache.jasper.JasperException: Unable to compile class for JSP:
An error occurred at line: [19] in the jsp file: [/hello3.jsp]
Syntax error on token ; , delete this token
16: <h1>◆문◆답◆필◆활◆꽃 ◆<%=name %>◆답!!</h1>
17: <h1>◆꽃◆씨◆꽃 ◆<%=age %>◆질◆였◆꽃◆답!!</h1>
18: <h1>◆꽃◆꽃 ◆<%=180 %>cm◆였◆꽃◆답!!</h1>
19: <h1>◆꽃◆씨◆+10◆ ◆<%=Integer.parseInt(age)+10 %>◆질◆였◆꽃◆답!!</h1>
20: </body>
21: </html>

```

<%= %> 안의 자바 변수나 자바 식에는 세미콜론(:)이 있으면 안 된다는 것 꼭 기억하세요!

스크립트 요소는 브라우저에서 JSP 페이지 요청 시 모두 서블릿의 자바 코드로 변환됩니다. 즉, 스크립트 요소는 브라우저로 전송되지 않습니다. 브라우저는 HTML 태그, CSS, 자바스크립트만 전달받습니다.

12.5

JSP 주석문 사용하기

JAVA WEB

다음은 JSP 페이지에 사용되는 주석문들입니다.

- HTML 주석
- 자바 주석
- JSP 주석

JSP 페이지에서는 HTML이 사용되므로 HTML 주석문이 있고, 스크립트릿 안에서는 자바 코드가 사용되므로 자바 주석문이 있습니다. 그리고 스크립트 요소에 대해 주석 처리를 하는 JSP 주석문도 있습니다.

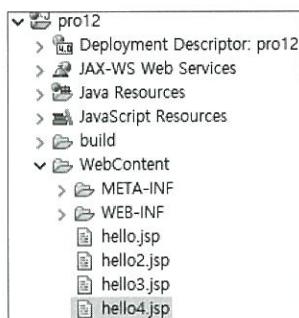
다음과 같이 <% %> 부분에 '--'을 붙이면 JSP 주석문이 됩니다.

<%-- 내용 --%>

12.5.1 JSP 페이지에서 주석문 사용하기

1. 다음과 같이 hello4.jsp 파일을 준비합니다.

▼ 그림 12-13 실습 파일 위치



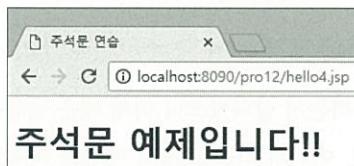
2. hello4.jsp를 다음과 같이 작성합니다. JSP 페이지에서 사용되는 여러 가지 주석문이 포함되어 있습니다.

코드 12-4 pro12/WebContent/hello4.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
<%
/*
String age=request.getParameter("age");
*/
%>
<!DOCTYPE html>
<!-- HTML 주석문입니다. --> •————— HTML 태그에 대한 주석문
<html>
<head>
    <title>주석문 연습</title>
</head>
<body>
    <h1>주석문 예제입니다!!</h1>
    <%-- <%=Integer.parseInt(age)+10 %> --%> •————— JSP 페이지에 대한 주석문
</body>
</html>
```

3. . http://localhost:8090/pro12/hello4.jsp로 요청합니다.

▼ 그림 12-14 브라우저 출력 결과



브라우저로 전달된 HTML 태그를 보면 HTML 주석문도 브라우저로 전달됩니다.

▼ 그림 12-15 HTML 소스

```

4 <!DOCTYPE html>
5 <!-- HTML 주석문입니다. -->
6 <html>
7 <head>
8   <title>주석문 연습</title>
9 </head>
10 <body>
11   <h1>주석문 예제입니다!!</h1>
12 </body>
13 </html>
14
15

```

HTML 주석문은 브라우저로 전달됩니다.

자바 주석문은 서블릿으로 변환 시 자바 주석문으로 표시됩니다.

▼ 그림 12-16 서블릿으로 변환된 상태

```

113     out = pageContext.getOut();
114     _jspx_out = out;
115
116     out.write("\r\n");
117     out.write("\r\n");
118
119 /**
120 String age=request.getParameter("age");
121 */
122
123     out.write(" \r\n");
124     out.write("<!DOCTYPE html>\r\n");
125     out.write("<!-- HTML 주석문입니다. -->\r\n");
126     out.write("<html>\r\n");
127     out.write("<head>\r\n");
128     out.write("   <title>주석문 연습</title>\r\n");
129     out.write("</head>\r\n");
130     out.write("<body>\r\n");
131     out.write("   <h1>주석문 예제입니다!!</h1>\r\n");

```

서블릿에 자바 주석문으로 표시됩니다.

하지만 JSP 주석문은 JSP 자체의 주석문이기 때문에 서블릿 코드로 변환되지 않습니다.

12.6

스크립트 요소 이용해 실습하기

HTML 태그 기반의 화면에 스크립트 요소를 어떻게 사용하는지 살펴봤으니 이제 배운 내용을 활용해 보겠습니다. 로그인, 학점 변환 계산기, 구구단 출력, 이미지 리스트 출력 예제를 차례로 실습해 보겠습니다.

12.6.1 로그인 예제

- 로그인창에서 ID와 비밀번호를 입력한 후 JSP로 전송하여 출력하는 예제입니다. 다음과 같이 실습 파일 login.html, result.jsp, result2.jsp, result3.jsp를 준비합니다.

▼ 그림 12-17 실습 파일 위치



- login.html을 다음과 같이 작성합니다. 로그인창에서 ID와 비밀번호를 입력한 후 action의 result.jsp로 전송합니다.

코드 12-5 pro12/WebContent/login.html

```
<!DOCTYPE html>
<html>...</head>
<body>
<form name="frmLogin" method="post" action="result.jsp" encType="utf-8">
    아이디 :<input type="text" name="user_id"><br>
    비밀번호:<input type="password" name="user_pw"><br>
    <input type="submit" value="로그인">
```

입력한 ID와 비밀번호를 result.jsp로 전송합니다.

```
<input type="reset" value="다시 입력">  
</form>  
</body>  
</html>
```



제공하는 예제 파일에서는 login.html을 중복해서 사용하므로 각 로그인 예제 실습 시 action의 값을 변경해서 실행하기 바랍니다.

3. result.jsp를 다음과 같이 작성합니다. 스크립트릿을 이용해 전송된 ID와 비밀번호를 가져온 후 표현식을 이용해 변수의 값을 출력합니다.

코드 12-6 pro12/WebContent/result.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"  
pageEncoding="UTF-8"%>  
  
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="UTF-8">  
    <title>결과출력창</title>  
</head>  
<body>  
    <h1>결과 출력</h1>  
    <%  
        request.setCharacterEncoding("utf-8");  
        String user_id=request.getParameter("user_id");  
        String user_pw=request.getParameter("user_pw");  
    %>  
    <h1>아이디 : <%= user_id %></h1>  
    <h1>비밀번호: <%= user_pw %></h1>  
</body>  
</html>
```

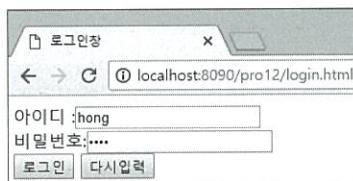
getParameter() 메서드를 이용해
입력 정보를 가져옵니다.

ID를 표현식으로 출력합니다.

비밀번호를 표현식으로 출력합니다.

4. <http://localhost:8090/pro12/login.html>로 요청한 후 ID와 비밀번호를 입력하여 로그인합니다.

▼ 그림 12-18 로그인창에서 로그인



5. 로그인 정보가 출력됩니다.

▼ 그림 12-19 로그인 정보 출력



6. 이번에는 한 걸음 더 나아가 스크립트릿 안에 자바 코드를 사용해 ID가 정상적으로 입력되었는지 체크한 후 정상 입력 여부에 따라 동적으로 다른 결과를 출력하도록 구현해 보겠습니다. result2.jsp를 다음과 같이 작성합니다.

코드 12-7 pro12/WebContent/result2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding( "utf-8" );
    String user_id = request.getParameter("user_id");
    String user_pw = request.getParameter("user_pw");
%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>결과출력창</title>
</head>
<body>
```

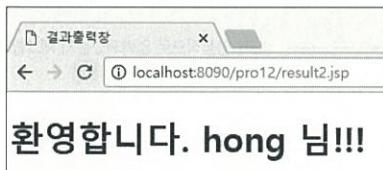
```

<%
    if(user_id==null || user_id.length()==0){           ← ID가 정상적으로 입력되었는지 체크합니다.
%>
    아이디를 입력하세요.<br>
    <a href="/pro12/login.html">로그인하기</a>          ← ID를 입력하지 않았을 경우 다시
                                                               로그인창으로 이동합니다.
<%
    }else{                                              ← ID를 정상적으로 입력했을 경우
%>
    <h1> 환영합니다. <%=user_id %> 님!!!</h1>          ← 메시지를 표시합니다.
    <%
    }
%>
</body>
</html>

```

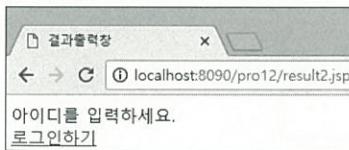
7. login.html의 action 속성을 result2.jsp로 수정 후, 로그인창에서 먼저 ID를 정상적으로 입력한 후 전송했을 때의 결과를 확인합니다.

▼ 그림 12-20 ID를 정상적으로 입력했을 경우



8. 다음은 ID를 입력하지 않고 전송한 경우입니다.

▼ 그림 12-21 ID를 입력하지 않았을 경우



서블릿에서는 자바 코드로 화면을 구현하듯이 JSP에서는 스크립트릿 안에 자바 코드를 사용해서 다양한 기능을 구현한다는 점이 이제 어느 정도 이해가 될 것입니다.

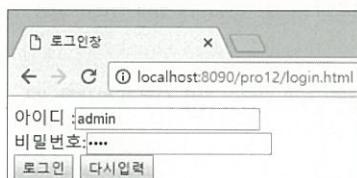
9. 로그인 예제를 조금 더 응용해 보겠습니다. 다음과 같이 result3.jsp를 작성합니다. 첫 번째 if문에서 먼저 ID가 입력되었는지 체크한 후 정상적으로 입력되었으면 다시 내부 if문을 수행하여 ID가 admin인지 체크합니다.

코드 12-8 pro12/WebContent/result3.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding( "utf-8" );
    String user_id = request.getParameter("user_id");
    String user_pw = request.getParameter("user_pw");
%>
<!DOCTYPE html>
<html>
<head>
    <title>결과출력창</title>
    <meta charset="UTF-8">
</head>
<body>
<%
    if(user_id == null || user_id.length()==0){ ────────── ID가 정상적으로 입력되었는지 체크합니다.
%>
        아이디를 입력하세요.<br>
        <a href="/pro12 /login.html">로그인하기</a>
<%
}else{
    if(user_id.equals("admin")){ ────────── ID를 입력한 경우 ID가 admin인지 다시 체크합니다.
%>
        <h1>관리자로 로그인 했습니다.</h1>
        <form>
            <input type=button value="회원정보 삭제하기" />
            <input type=button value="회원정보 수정하기" />
        </form>
<%
}else{
%>
        <h1> 환영합니다. <%=user_id %> 님!!!</h1>
<%
}
%>
</body>
</html>
```

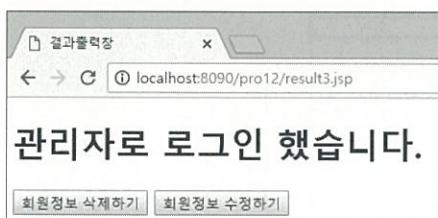
10. 다음은 admin으로 로그인했을 때의 실행 결과입니다.

▼ 그림 12-22 ID를 admin으로 입력하고 로그인



11. 관리자창이 나타납니다.

▼ 그림 12-23 관리자창이 나타남



12. 다른 ID로 로그인 시 “환영합니다. lee 님!!!”이라는 메시지가 나타납니다.

▼ 그림 12-24 일반 사용자창이 나타남





Caution 스크립트릿의 자바 코드 작성 시 주의하세요!

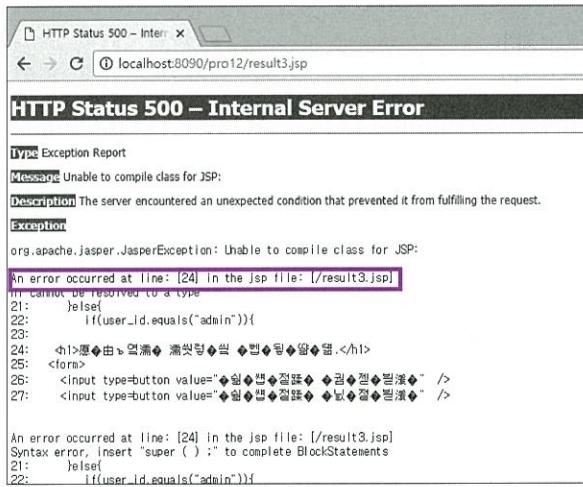
JSP 페이지의 화면 기능이 복잡해질수록 스크립트릿의 자바 코드와 HTML 태그가 같이 표시되므로 코드가 복잡해질 수 있습니다. 따라서 들여쓰기를 습관화해서 스크립트릿의 여닫는 부분이나 자바 코드의 괄호 여닫는 부분이 틀리지 않도록 주의해서 작성해야 합니다.

만약 스크립트릿의 닫는 기호(%)를 빠트리고 로그인창에서 요청할 경우 다음과 같은 오류가 발생합니다.

▼ 그림 12-25 23행의 %)가 누락된 경우

```
14<body>
15<%
16 if(user_id == null || user_id.length()==0){
17 %}
18 아이디를 입력하세요.<br>
19 <a href="/pro12/login.html">로그인하기</a>
20<%
21 }else{
22     if(user_id.equals("admin")){
23         <h1>관리자로 로그인 했습니다.</h1>
24         <form>
25             <input type=button value="회원정보 삭제하기" />
26             <input type=button value="회원정보 수정하기" />
27         </form>
28     <%
29     }else{
30     }
31 %}
32     <h1> 환영합니다. <%=user_id %> 님!!!</h1>
33<%
34 }
35 }
36 %}
37 </body>
```

▼ 그림 12-26 JSP 실행 시 스크립트릿 오류 발생



12.6.2 학점 변환 예제

이번에는 시험 점수를 입력 받은 후 학점으로 변환하는 예제를 실습해 보겠습니다.

1. 다음과 같이 scoreTest.html, scoreTest.jsp 파일을 준비합니다.

▼ 그림 12-27 실습 파일 위치



2. scoreTest.html을 다음과 같이 작성합니다. 사용자로부터 시험 점수를 입력 받아 scoreTest.jsp로 전송합니다.

코드 12-9 pro12/WebContent(scoreTest.html)

```
<!DOCTYPE html>
<html>
<head>...</head>
<body>
    <h1>시험 점수를 입력해 주세요</h1>
    <form method=get action="scoreTest.jsp">—————  
입력한 시험 점수를 scoreTest.jsp로  
시험점수 :<input type=text name="score" /> <br> 전송합니다.  
        <input type="submit" value="변환하기">
    </form>
</body>
</html>
```

3. scoreTest.jsp를 다음과 같이 작성합니다. scoreTest.html로부터 받은 점수를 다중 if~else if문을 이용해 학점으로 변환합니다.

코드 12-10 pro12/WebContent(scoreTest.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%
request.setCharacterEncoding("utf-8");
int score=Integer.parseInt(request.getParameter("score")); ← 전송된 시험 점수를
%> 가져옵니다.

<!DOCTYPE html>
<html>
<head>
<title>점수 출력창</title>
<meta charset="UTF-8">
</head>
<body>
<h1>시험점수 <%=score %>점</h1><br>
<%
if(score>=90){ ← 90점 이상이면 A를 출력합니다.
%>
<h1>A학점입니다.</h1>
<%
}else if(score>=80 && score<90){ ← 80~90점 사이면 B를 출력합니다.
%>
<h1> B학점입니다.</h1>
<%
}else if(score>=70 && score<80){ ← 70~80점 사이면 C를 출력합니다.
%>
<h1> C학점입니다.</h1>
<%
}else if(score>=60 && score<70){ ← 60~70점 사이면 D를 출력합니다.
%>
<h1> D학점입니다.</h1>
<%
}else{
%>
<h1> F학점입니다.</h1> ← 그 외 점수는 F를 출력합니다.
%>
}
<br>
<a href="scoreTest.html">시험점수입력</a>
</body>
</html>
```

4. <http://localhost:8090/pro12/scoreTest.html>로 요청하여 시험점수 입력창에 시험 점수를 입력한 후 **변환하기**를 클릭합니다.

▼ 그림 12-28 시험 점수 입력 후 **변환하기** 클릭

시험점수입력창

localhost:8090/pro12/scoreTest.html

시험 점수를 입력해 주세요

시험점수: 78

변환하기

5. 시험 점수를 학점으로 변환하여 출력합니다.

▼ 그림 12-29 시험 점수를 학점으로 변환 후 출력

점수 출력창

localhost:8090/pro12/scoreTest.jsp?score=78

시험점수 78점

C학점입니다.

시험점수입력

시험 점수는 보통 0~100점 사이이므로 만약 이 범위를 벗어나는 점수를 입력하면 오류 메시지를 나타내고 다시 scoreTest.html 입력창으로 이동하는 기능도 추가해 보세요.

12.6.3 구구단 출력 예제

이번에는 구구단의 단수를 전송 받은 후 구구단을 자바 for문과 <table> 태그의 <tr> 태그를 이용해 리스트로 출력하는 예제를 실습해 보겠습니다.

1. 구구단 예제 실습 파일인 gugu.html, gugu.jsp, gugu2.jsp를 준비합니다.

▼ 그림 12-30 실습 파일 위치



2. gugu.html을 다음과 같이 작성합니다. 출력할 구구단의 단수를 입력 받아 gugu.jsp로 포워딩합니다.

코드 12-11 pro12/WebContent/gugu.html

```
<!DOCTYPE html>
<html>
<head>...</head>
<body>
<h1> 구구단의 단수를 입력하세요.</h1>           구구단의 단수를 gugu.jsp로 전송합니다.
<form method=get action="gugu.jsp">
    출력할 구구단 : <input type='text' name='dan' /> <br>
    <input type ='submit' value='출력하기'>
</form>
</body>
</html>
```

3. gugu.jsp를 다음과 같이 작성합니다. 스크립트릿 안에서 자바 for문을 이용해 <table> 태그의 행을 나타내는 <tr> 태그를 연속해서 브라우저로 출력합니다.

코드 12-12 pro12/WebContent/gugu.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%
request.setCharacterEncoding("utf-8");
```

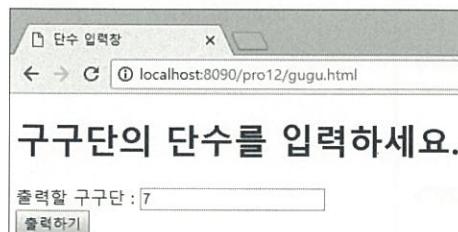
```

int dan=Integer.parseInt(request.getParameter("dan")); ----- 전송된 단수를 구합니다.
%>
<!DOCTYPE html>
<html>
<head>...</head>
<body>
    <table border='1' width='800' >
        <tr align='center' bgcolor='#FFFF66'>
            <td colspan='2'><%= dan %>단 출력 </td>
        </tr> ----- 전송된 단수를 출력합니다.
<%>
    for(int i=1; i<10;i++){
%>
        <tr align='center'>
            <td width='400'>
                <%=dan %> * <%=i %>
            </td>
            <td width='400'>
                <%=i*dan %>
            </td>
        </tr>
%>
    }
%>
</table>
</body>
</html>

```

4. <http://localhost:8090/pro12/gugu.html>로 요청하여 입력창에서 단수를 입력한 후 전송합니다.

▼ 그림 12-31 단수 입력 후 전송



5. for문을 이용해 구구단을 리스트로 출력합니다.

▼ 그림 12-32 전송된 구구단을 리스트로 출력

7단 출력	
7 * 1	7
7 * 2	14
7 * 3	21
7 * 4	28
7 * 5	35
7 * 6	42
7 * 7	49
7 * 8	56
7 * 9	63

결과가 잘 나왔나요? 이번에는 구구단을 리스트로 출력할 때 홀수 행과 짝수 행이 구분되도록 배경색을 지정하는 효과를 추가해 보겠습니다. 스크립트릿을 이용해 배경색을 교대로 출력하는 기능입니다.

6. 다음과 같이 gugu2.jsp를 작성합니다. if문에서 for 반복문의 반복 변수 i를 사용해 홀수인지 짝수인지를 체크합니다. 그런 다음 <tr> 태그의 bgcolor 속성 값을 다르게 설정하여 브라우저로 출력합니다.

코드 12-13 pro12/WebContent/gugu2.jsp

```
...
<%
    for(int i=1; i<10;i++){
%>
<%
    if(i%2==1){
%>
        <tr align=center bgcolor="#CCFF66">
<%
    }else{
%>
        <tr align=center bgcolor="#CCCCFF">
<%
    }
%>
...
    
```

테이블의 홀수 행과 짝수 행의 색깔을 다르게 표시합니다.

7. 브라우저에서 실행하면 홀수 행과 짝수 행의 배경색이 다르게 출력됩니다.

▼ 그림 12-33 실행 결과

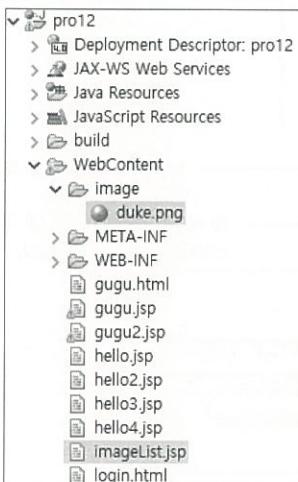
7 단 출력	
7 * 1	7
7 * 2	14
7 * 3	21
7 * 4	28
7 * 5	35
7 * 6	42
7 * 7	49
7 * 8	56
7 * 9	63

12.6.4 이미지 리스트 출력 예제

이번에는 스크립트릿을 이용해 image 폴더의 이미지를 가져와서 리스트로 출력하는 예제를 실습해 보겠습니다.

1. imageList.jsp를 생성하고 실습 이미지인 duke.png를 추가합니다(예제 파일로 제공).

▼ 그림 12-34 실습 파일 위치



2. imageList.jsp를 다음과 같이 작성합니다. for 반복문을 이용해 태그 안에 태그를 연속적으로 출력해서 이미지를 나타냅니다.

코드 12-14 pro12/WebContent/imageList.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<style>
.lst_type{overflow:hidden;width:80%;padding:0 10px 10px; margin:0 auto}
.lst_type li{overflow:hidden;clear:both;margin:10px 0;color:#2d2c2d;
font-family:'돋움',Dotum;font-size:12px;line-height:100px;
list-style:none ; border-bottom: 2px solid lightgray;position:relative; }
.lst_type li img{display:inline;float:left;position:absolute; }
.lst_type li a{color:#2d2c2d;text-decoration:none; margin-left:340px}
.lst_type li a:hover{text-decoration:underline}
.lst_type li span{color:blue; margin-left:330px;font-family:'돋움',Dotum;font-
size:14px; }
</style>

<meta charset="UTF-8">
<title>이미지리스트창</title>
</head>
<body>
<ul class="lst_type">
<li>
<span style='margin-left:50px'>이미지 </span>
<span>이미지 이름</span>
<span>선택하기</span>
</li>
<%
for(int i=0 ; i<10; i++){
%>
<li>
<a href="#" style='margin-left:50px'>
<img src='image/duke.png' width='90' height='90' alt=''/></a>
<a href="#"><strong>이미지 이름: 듀크<%=i %> </strong></a>
<a href="#"> <input name='chk<%=i %>' type='checkbox' /></a>
</li>
<%
}
%>
</ul>

```

리스트의 헤더를 표시합니다.

for 반복문을 이용해 태그를 연속해서 출력합니다.

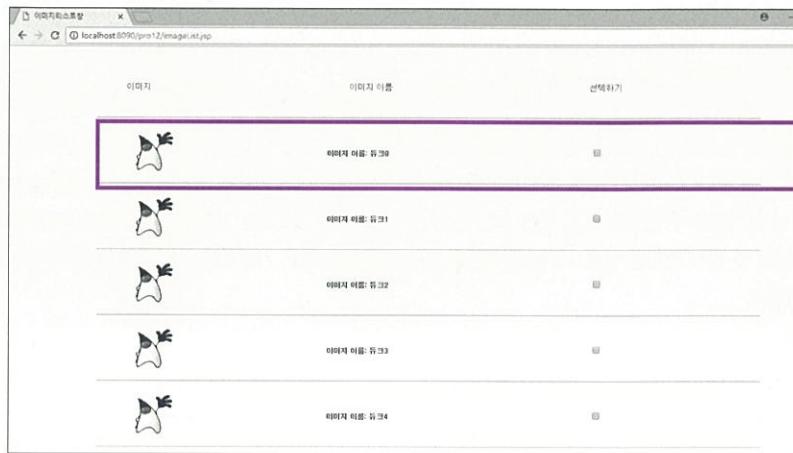
 태그를 이용해 한 행에 <a> 태그의 이미지와 텍스트를 나타냅니다.

image 폴더의 이미지를 나타냅니다.

```
</body>
</html>
```

3. `http://localhost:8090/pro12/imageList.jsp`로 요청하면 다음과 같이 출력됩니다.

▼ 그림 12-35 실행 결과



리스트로 출력하는 기능은 웹 페이지에서 많이 사용하는 기능입니다. 그림 12-35처럼 쇼핑몰에서 상품 검색 결과를 리스트로 출력하는 기능도 `` 태그 안의 `` 태그를 `for` 반복문을 이용해서 나타냅니다.

▼ 그림 12-36 상품 검색 결과 리스트 출력 예시

상품명	가격	할인 및 판매처 정보
삼성 컴퓨터 모음전	199,000원	AJ본사판 판권자: 삼성전자
삼성 사무용 컴퓨터	155,000원	IDC MALL 판권자: 삼성전자
LG 애스拓 PC Z71EV-AX7P26	419,000원	Glory Mall 판권자: LG전자
비쁜부팅신제품SSD넉넉한저장공간 DB-Z600 슬림PC	299,000원	Microsoft리퍼비시 판권자: 삼성전자

지금까지 JSP 페이지에서 스크립트릿에 자바 코드를 추가하여 여러 가지 화면을 구현해 보았습니다. JSP 페이지의 기능이 복잡해질수록 HTML 태그와 스크립트릿 요소도 복잡해지므로 충분히 실습해 보기 바랍니다. 6.8절에서도 서블릿을 이용해 화면에 동일한 결과를 나타내는 실습을 해 봤습니다. 서로 다른 두 방법이 같은 결과를 얻기까지 접근하는 과정이 어떻게 다른지 잘 이해해 두기 바랍니다.

Note ≡ JSP 프리컴파일(Precompile) 기능

브라우저에서 서블릿으로 최초 요청을 보내면 먼저 톰캣이 컴파일을 한 후 실행을 합니다. 따라서 톰캣은 시작 시 미리 서블릿을 메모리에 로드해서 사용하는 방법을 제공합니다(8.6절 참고). JSP도 최초 요청 시 변환 과정을 거치기 때문에 실행이 늦어지게 됩니다. 따라서 톰캣 컨테이너에서는 JSP Precompile 기능을 제공해 미리 JSP를 컴파일함으로써 요청 시 바로 처리할 수 있도록 하고 있습니다. 웹 애플리케이션 개발 시에는 JSP에 변경 사항이 자주 발생하므로 그다지 필요할 것 같지 않지만 실제 서비스를 제공할 때는 사용하면 좋은 기능입니다. 자세한 내용은 톰캣 홈페이지에서 확인하세요.

12.7

내장 객체(내장 변수) 기능

JAVA WEB

JSP 페이지의 내장 객체(내장 변수)란 JSP가 서블릿으로 변환될 때 컨테이너가 자동으로 생성시키는 서블릿 멤버 변수를 말합니다. 즉, 서블릿으로 구현 시 자주 사용했던 객체를 개발자가 일일이 만드는 것이 아니라 서블릿으로 변환 시 컨테이너가 자동으로 생성하여 사용하게끔 제공하는 것입니다.

그림 12-37은 JSP 파일이 서블릿으로 변환되었을 때 `_jspService()` 메서드에 생성된 내장 객체를 저장하는 내장 변수가 선언된 코드입니다.

▼ 그림 12-37 서블릿에 선언된 내장 객체(내장 변수)

```

79  public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.se
80      throws java.io.IOException, javax.servlet.ServletException {
81
82  if (!javax.servlet.DispatcherType.ERROR.equals(request.getDispatcherType())) {
83      final java.lang.String _jspx_method = request.getMethod();
84      if ("OPTIONS".equals(_jspx_method)) {
85          response.setHeader("Allow","GET, HEAD, POST, OPTIONS");
86          return;
87      }
88      if (!"GET".equals(_jspx_method) && !"POST".equals(_jspx_method) && !"HEAD".equals(_jspx
89          response.setHeader("Allow","GET, HEAD, POST, OPTIONS");
90          response.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED, "JSPs only permit GET,
91          return;
92      }
93  }
94
95  final javax.servlet.jsp.PageContext pageContext;
96  javax.servlet.http.HttpSession session = null;
97  final javax.servlet.ServletContext application;
98  final javax.servlet.ServletConfig config;
99  javax.servlet.jsp.JspWriter out = null;
100 final java.lang.Object page = this;
101 javax.servlet.jsp.JspWriter _jspx_out = null;
102 javax.servlet.jsp.PageContext _jspx_page_context = null;
103

```

표 12-1은 JSP 페이지에서 제공하는 여러 가지 내장 객체를 정리한 것입니다.

▼ 표 12-1 JSP에서 제공하는 내장 객체들

내장 객체	서블릿 타입	설명
request	javax.servlet.http.HttpServletRequest	클라이언트의 요청 정보를 저장합니다.
response	javax.servlet.http.HttpServletResponse	응답 정보를 저장합니다.
out	javax.servlet.jsp.JspWriter	JSP 페이지에서 결과를 출력합니다.
session	javax.servlet.http.HttpSession	세션 정보를 저장합니다.
application	javax.servlet.ServletContext	컨텍스트 정보를 저장합니다.
pageContext	javax.servlet.jsp.PageContext	JSP 페이지에 대한 정보를 저장합니다.
page	java.lang.Object	JSP 페이지의 서블릿 인스턴스를 저장합니다.
config	javax.servlet.ServletConfig	JSP 페이지에 대한 설정 정보를 저장합니다.
exception	java.lang.Exception	예외 발생 시 예외를 처리합니다.

이 중 application, request, response, session은 이미 서블릿에서 사용해본 객체들입니다. 따라서 앞에서 배웠던 동일한 기능을 제공합니다.

표 12-2는 자주 사용되는 내장 객체들의 스코프를 정리한 것입니다. `request`, `session`, `application`은 서블릿의 스코프와 같습니다. `page`는 요청하는 해당 JSP 페이지에 대해서만 공유할 수 있습니다.

▼ 표 12-2 내장 객체들의 스코프

내장 객체	서블릿	스코프
page	this	한 번의 요청에 대해 하나의 JSP 페이지를 공유합니다.
request	HttpServletRequest	한 번의 요청에 대해 같은 요청을 공유하는 JSP 페이지를 공유합니다.
session	HttpSession	같은 브라우저에서 공유합니다.
application	ServletContext	같은 애플리케이션에서 공유합니다.

먼저 `session` 내장 객체의 바인딩 기능을 사용해 보겠습니다.

12.7.1 session 내장 객체에 데이터 바인딩 실습

1. JSP 파일이 많아지므로 `test01` 폴더를 만든 후 `session1.jsp`, `session2.jsp` 등 실습 파일들을 생성합니다.

▼ 그림 12-38 실습 파일 위치



2. SessionTest 클래스를 다음과 같이 작성합니다. 서블릿에서 getSession() 메서드를 이용해 session 객체를 얻은 후 name을 바인딩합니다.

코드 12-15 pro12/src/sec01/ex01/SessionTest.java

```
package sec01.ex01;
...
@WebServlet("/sess")
public class SessionTest extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter pw = response.getWriter();
        HttpSession session = request.getSession(); ────────── session 객체를 가져옵니다.
        session.setAttribute("name", "이순신"); ────────── session 객체에 name을 바인딩합니다.
        pw.println("<html><body>");
        pw.println("<h1>세션에 이름을 바인딩합니다.</h1>");
        pw.println("<a href='/pro12/test01/session1.jsp'>첫 번째 페이지로 이동하기 </a>");
        pw.println("</body></html>");
    }
}
```

3. session1.jsp 파일을 다음과 같이 작성합니다. session 객체의 사용법은 서블릿에서 배운 HttpSession 사용법과 같습니다. 차이점은 JSP에서는 자동으로 세션 객체를 생성해 주므로 굳이 getSession() 메서드를 호출해서 세션을 얻을 필요가 없다는 것입니다. getAttribute() 메서드를 이용해 첫 번째 JSP에서 session에 바인딩된 name 값을 가져온 후 setAttribute() 메서드를 이용해 session에 address를 바인딩합니다.

코드 12-16 pro12/WebContent/test01/session1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<% String name=(String)session.getAttribute("name"); ────────── session 객체에 바인딩된 name
    session.setAttribute("address","서울시 강남구"); ────────── 값을 가져옵니다.
%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>session 내장 객체 테스트2</title>
</head>
```

```
<body>
    이름은 <%=name %>입니다. <br>
    <a href=session3.jsp>두 번째 페이지로 이동</a>
</body>
</html>
```

4. session2.jsp에서는 `getAttribute()`를 이용해 서블릿과 JSP에서 session에 바인딩된 name과 address 값을 가져옵니다.

코드 12-17 pro12/WebContent/test01/session2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

```
<%
    String name=(String)session.getAttribute("name");
    String address = (String)session.getAttribute("address");
%>
```

session 객체에 바인딩된
name 값과 address 값을
가져옵니다.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>session 내장 객체 테스트3</title>
</head>
```

```
<body>
    이름은 <%=name %>입니다.<br>
    주소는 <%=address %>입니다. <br>
</body>
</html>
```

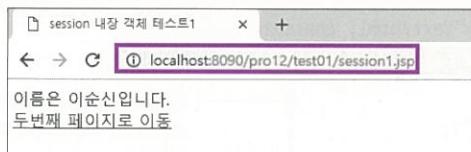
5. 다음은 최초 서블릿에 요청한 결과입니다. 서블릿 요청 시 session 객체에 name을 바인딩합니다.

▼ 그림 12-39 최초 서블릿에 요청 결과



6. 첫번째 페이지로 이동하기 클릭 시 서블릿에서 바인딩한 name을 출력합니다. 두번째 페이지로 이동을 클릭합니다.

▼ 그림 12-40 첫 번째 JSP 페이지 출력 결과



7. 서블릿과 첫 번째 JSP에서 바인딩한 이름(name)과 주소(address)를 출력합니다.

▼ 그림 12-41 두 번째 JSP 출력 결과

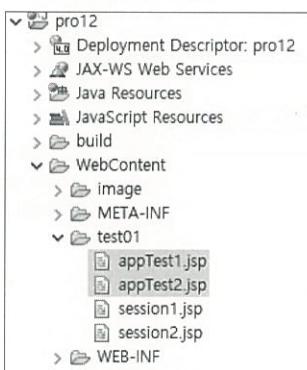


다음에는 session과 application 내장 객체의 스크립트에 대해 알아보겠습니다.

12.7.2 application 내장 객체에 데이터 바인딩 실습

1. 다음과 같이 appTest1.jsp, appTest2.jsp 실습 파일을 준비합니다.

▼ 그림 12-42 실습 파일 위치



2. appTest1.jsp를 다음과 같이 작성합니다. 첫 번째 JSP에서 session과 application 내장 객체에 name과 address 값을 바인딩합니다.

코드 12-18 pro12/WebContent/test01/appTest1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
    session.setAttribute("name", "이순신");
    application.setAttribute("address", "서울시 성동구");
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>내장 객체 스크립트 테스트1</title>
</head>
<body>
<h1>이름과 주소를 저장합니다.</h1>
<a href=appTest2.jsp>두 번째 웹 페이지로 이동</a>
</body>
</html>
```

이름과 주소를 session과 application 내장 객체에 바인딩합니다.

3. appTest2.jsp를 다음과 같이 작성합니다. 첫 번째 JSP에서 session과 application 내장 객체에 바인딩한 값을 가져옵니다.

코드 12-19 pro12/WebContent/test01/appTest2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
    String name=(String)session.getAttribute("name");
    String address=(String )application.getAttribute("address");
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>내장 객체 스크립트 테스트2</title>
</head>
<body>
<h1>이름은 <%=name %>입니다.</h1>
<h1>주소는 <%=address %>입니다.</h1>
</body>
</html>
```

첫 번째 웹 페이지에서 저장한 데이터를 session과 application 내장 객체에서 가져옵니다.

4. http://localhost:8090/pro12/test01/appTest1.jsp로 요청합니다. 첫 번째 JSP에서 name과 address를 session과 application에 바인딩합니다.

▼ 그림 12-43 첫 번째 브라우저에서 요청 시



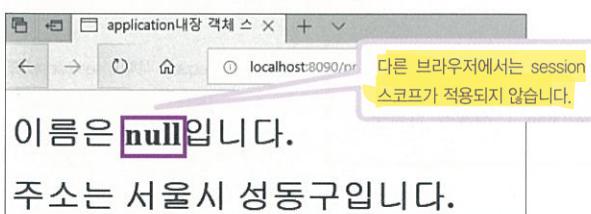
5. 같은 브라우저에서 요청할 경우 두 번째 JSP에서 session과 application에 접근할 수 있습니다.

▼ 그림 12-44 같은 브라우저에서 요청 시



하지만 익스플로리에서는 application의 값에만 접근할 수 있습니다.

▼ 그림 12-45 다른 세션의 브라우저에서 요청 시



같은 브라우저에서 appTest2.jsp를 요청하면 session과 application에 저장된 값을 그대로 출력합니다. 그러나 다른 브라우저로 요청할 경우 session 내장 객체의 스코프는 접근할 수 없으므로 null을 출력합니다. 이를 통해 application 내장 객체의 스코프는 애플리케이션 전체이고, session 내장 객체의 스코프는 같은 브라우저임을 알 수 있습니다.

다음으로 가장 널리 사용되는 내장 객체인 request에 대해 알아보겠습니다.

12.7.3 request 내장 객체에 데이터 바인딩 실습

- request 내장 객체 실습 파일인 request1.jsp, request2.jsp를 준비합니다.

▼ 그림 12-46 실습 파일 위치



- 첫 번째 JSP인 request1.jsp를 다음과 같이 작성합니다. 브라우저의 요청에 대한 request 객체에 name과 address를 바인딩합니다. 그리고 RequestDispatcher를 이용해 request 객체를 두 번째 JSP로 전송합니다.

코드 12-20 pro12/WebContent/request1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   import="javax.servlet.RequestDispatcher"
   pageEncoding="UTF-8"
%>
<%
    request.setAttribute("name", "이순신");
    request.setAttribute("address", "서울시 강남구");
%>
<!DOCTYPE html>
<html>
<head>...</head>
<body>

<%
    RequestDispatcher dispatch = request.getRequestDispatcher("request2.jsp");
    dispatch.forward(request, response);
%>
</body>
</html>
```

request 객체에 setAttribute()를 이용해 name과 address를 바인딩합니다.

request 객체를 다른 JSP로 포워딩합니다.

3. 두 번째 JSP인 request2.jsp를 다음과 같이 작성합니다. 첫 번째 JSP에서 전송된 request 객체에서 바인딩된 name과 address를 가져옵니다.

코드 12-21 pro12/WebContent/request2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
String name=(String)request.getAttribute("name");
String address=(String )request.getAttribute("address");
%>

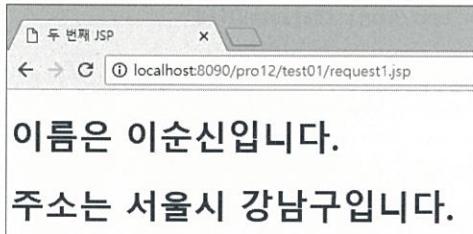
<!DOCTYPE html>
<html>
<head>...</head>
<body>
<h1>이름은 <%=name %>입니다.</h1>
<h1>주소는 <%=address %>입니다.</h1>
</body>
</html>
```

첫 번째 JSP 페이지에서 포워딩된 request 객체에서 getAttribute()를 이용해 정보를 가져옵니다.

이전 JSP에서 전송된 정보를 출력합니다.

4. 브라우저에서 request1.jsp로 요청하면 request 객체에 바인딩한 후 request2.jsp로 포워딩하여 이름과 주소를 출력합니다.

▼ 그림 12-47 실행 결과

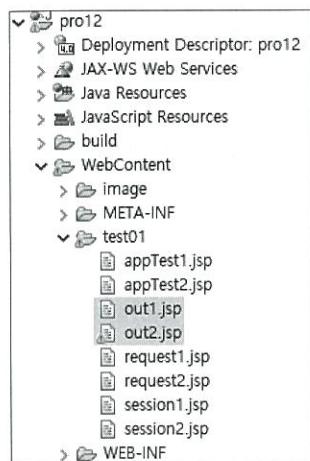


12.7.4 out 내장 객체 이용해 데이터 출력하기

이번에는 결과를 출력할 때 사용하는 out 내장 객체를 사용해 보겠습니다.

1. 다음과 같이 실습 파일 out1.jsp, out2.jsp를 준비합니다.

▼ 그림 12-48 실습 파일 위치



2. 첫 번째 JSP 페이지인 out1.jsp를 작성합니다. 이름과 나이를 두 번째 JSP로 전송합니다.

코드 12-22 pro12/WebContent/test01/out1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>...</head>
<body>
    <form method="post" action="out2.jsp">
        이름: <input type="text" name="name"><br>
        나이: <input type="text" name="age"><br>
        <input type ="submit" value="전송">
    </form>
</body>
</html>
```

3. 두 번째 JSP 페이지인 out2.jsp를 작성합니다. 전송된 이름과 나이를 표현식과 out 내장 객체를 이용해 출력합니다.

코드 12-23 pro12/WebContent/test01/out2.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding( "utf-8" );
    String name=request.getParameter("name");
    String age=request.getParameter("age");
%>

<!DOCTYPE html>
<html>
<head>...</head>
<body>
<%
    if(name!=null &&name.length()!=0){
%>
    <h1><%=name %> ,<%=age %> </h1> ─────────── name과 age의 값을 표현식으로 출력합니다.
%>
}else{
%>
    <h1>이름을 입력하세요</h1>
%>
}
%>

<%
    if(name!=null &&name.length()!=0){
%>
    <h1><% out.println(name+ " , "+age); %></h1> ─────────── name과 age의 값을 out 내장 객체로 출력합니다.
%>
}else{
%>
    <h1>이름을 입력하세요</h1>
%>
}
%>
</body>
</html>

```

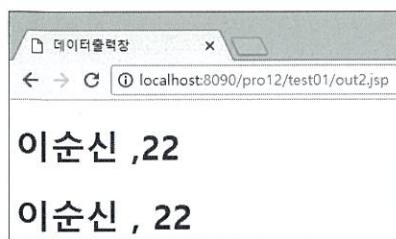
4. 브라우저에서 요청하여 이름과 나이를 입력한 후 전송합니다.

▼ 그림 12-49 회원 정보 전송



5. 전달받은 정보를 표현식과 out 내장 객체로 출력합니다.

▼ 그림 12-50 표현식과 out 내장 객체로 회원 정보 출력



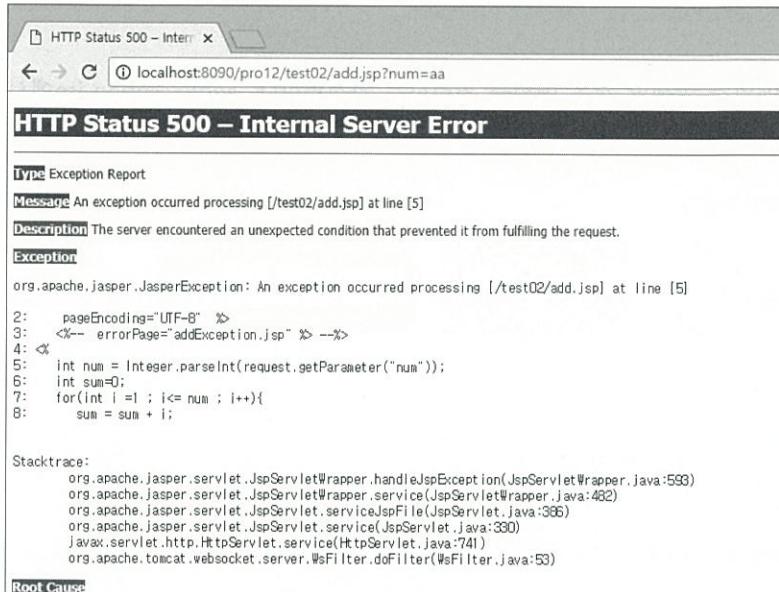
out 내장 객체를 이용해 스크립트릿으로 출력하면 복잡한 코드를 상대적으로 간단하게 출력할 수 있습니다.

12.8

JSP 페이지 예외 처리하기

JSP 페이지를 실행하다 보면 종종 실행 중에 오류가 발생합니다. 예를 들어 은행 사이트에서 송금을 하려고 하는데 그림 12-51 같은 오류 메시지가 브라우저에 나타났다고 생각해 보세요.

▼ 그림 12-51 JSP 예외 발생 화면



사용자 입장에서는 큰 문제가 발생한 것으로 인식하겠지요. 그러면 사이트에 대한 신뢰도 떨어질 수밖에 없습니다. 따라서 프로그램 실행 시 예외나 오류가 발생할 경우 이를 안내하는 페이지, 즉 전용 예외 처리 페이지가 나타나게 하여 좀 더 신뢰 있고 사용자 친화적인 웹 페이지를 만들 수 있습니다.

12.8.1 JSP 페이지 예외 처리 과정

JSP 페이지에서 오류가 발생하면 예외 처리 페이지를 이용해 예외 처리를 할 수 있습니다.

JSP 예외 처리 페이지는 어떻게 만들까요? 먼저 예외 처리 JSP를 만든 후 디렉티브 태그 속성 중 isErrorPage 속성을 true로 설정합니다. 그리고 일반 JSP 페이지의 디렉티브 태그 속성 중 errorPage 속성을 예외 처리 페이지 이름으로 지정합니다.

▼ 그림 12-52 JSP 예외 처리 페이지 만드는 과정

① 예외 처리 담당 JSP를 만듭니다.

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



② 예외 발생 시 예외 처리 담당 JSP 파일을 지정합니다.

```
<%@ page errorPage='addException.jsp' %>
```

다음은 add.jsp에서 예외가 발생할 경우 예외를 처리하는 과정입니다. addException.jsp에서 exception 내장 객체를 사용해 예외 처리를 합니다.

▼ 그림 12-53 실습 예제 예외 처리 과정



12.8.2 JSP 페이지 예외 처리 실습

그럼 add.jsp와 addException.jsp를 이용해 예외 처리를 실습해 보겠습니다.

1. 실습을 위해 WebContent 아래 test02 폴더를 만들고 다음과 같이 add.html, add.jsp, addException.jsp 파일들을 준비합니다.

▼ 그림 12-54 실습 파일 위치



2. add.html을 다음과 같이 작성합니다. 입력창에서 숫자를 입력 받아 action에 지정한 add.jsp로 전송합니다.

```
코드 12-24 pro12/WebContent/test02/add.html
<!DOCTYPE html>
<html>
<head>
    <title>합계</title>
</head>
<body>
    자연수를 입력하세요.
    <form action='add.jsp'>————— 입력한 값을 add.jsp로 전송합니다.
        1부터 <input type='text' name='num'>
        <input type='submit' value='계산하기'>
    </form>
</body>
</html>
```

3. add.jsp를 다음과 같이 작성합니다. 페이지 디렉티브 태그의 errorPage 속성에 예외 처리 페이지인 addException.jsp를 지정하여 오류가 발생하면 예외 처리를 합니다.

```
코드 12-25 pro12/WebContent/test02/add.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    errorPage="addException.jsp" %>————— 예외 발생 시 예외를 처리할 JSP
<%
    int num = Integer.parseInt(request.getParameter("num"));
    int sum=0;
    for(int i =1 ; i<= num ; i++){
        sum = sum + i;
    }
%>

<!DOCTYPE html>
<html>
<head>
    <title>합계 구하기</title>
</head>
<body>
    <h2>합계 구하기</h2>
    <h1>1부터 <%=num %>까지의 합은 <%=sum %>입니다</h1>
</body>
</html>
```

4. 또 다른 JSP 페이지인 addException.jsp를 다음과 같이 작성합니다. 페이지 디렉티브 태그의 isErrorPage 속성을 true로 설정해 exception 내장 객체를 이용해서 발생한 예외를 처리하도록 합니다. 이때 exception 내장 객체는 자바의 Exception 클래스의 인스턴스입니다.

코드 12-26 pro12/WebContent/test02/addException.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <title>에러 페이지</title>
</head>
<body>
    ====== toString() 내용 ====== <br>
    <h1><%= exception.toString() %></h1>
    ====== getMessage() 내용 ======<br>
    <h1><%=exception.getMessage()%></h1>
    ====== printStackTrace() 내용 ======<br>
    <h1><% exception.printStackTrace(); %></h1>
    <h3>
        숫자만 입력 가능합니다. 다시 시도하세요.
        <a href='add.html'>다시 하기</a>
    </h3>
</body>
</html>
```

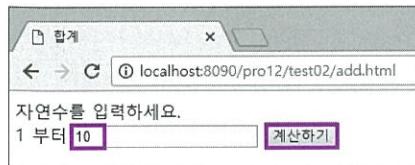
다른 JSP 페이지에서 예외 발생 시 예외를 처리하는 예외 페이지로 지정합니다.

exception 내장 객체를 사용해 예외 처리를 합니다.

이클립스 콘솔로 예외 메시지를 출력합니다.

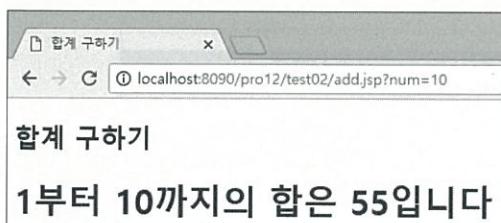
5. <http://localhost:8090/pro12/test02/add.html>로 요청하여 입력창에 정상적인 숫자를 입력 한 후 계산하기를 클릭합니다.

▼ 그림 12-55 숫자 입력 후 전송



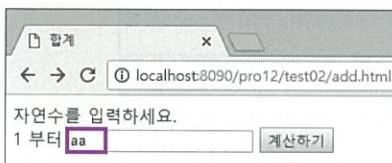
6. 정상적인 결과가 출력됩니다.

▼ 그림 12-56 숫자 전송 시 정상적인 결과 출력



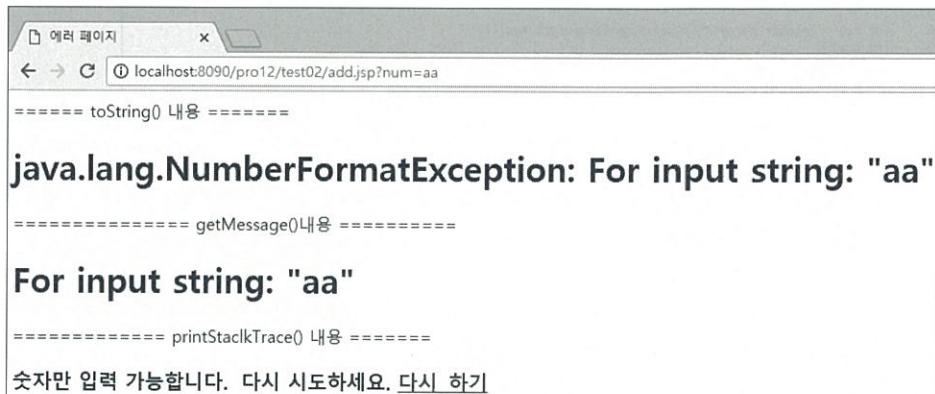
7. 이번에는 문자를 입력해 볼까요?

▼ 그림 12-57 문자 입력 후 전송



8. 문자는 처리 시 예외가 발생합니다. 다음과 같이 예외 처리 페이지에서 예외를 처리합니다.

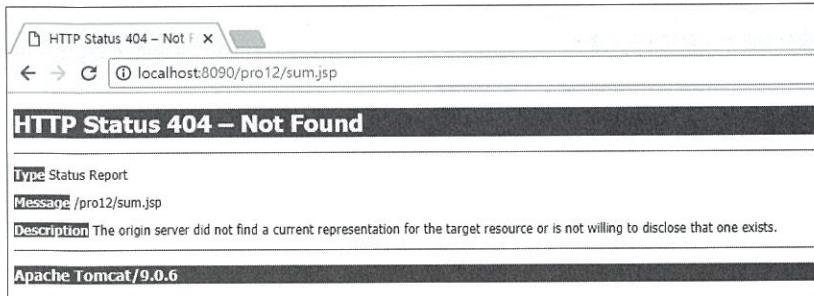
▼ 그림 12-58 문자 전송 시 예외 메시지 출력



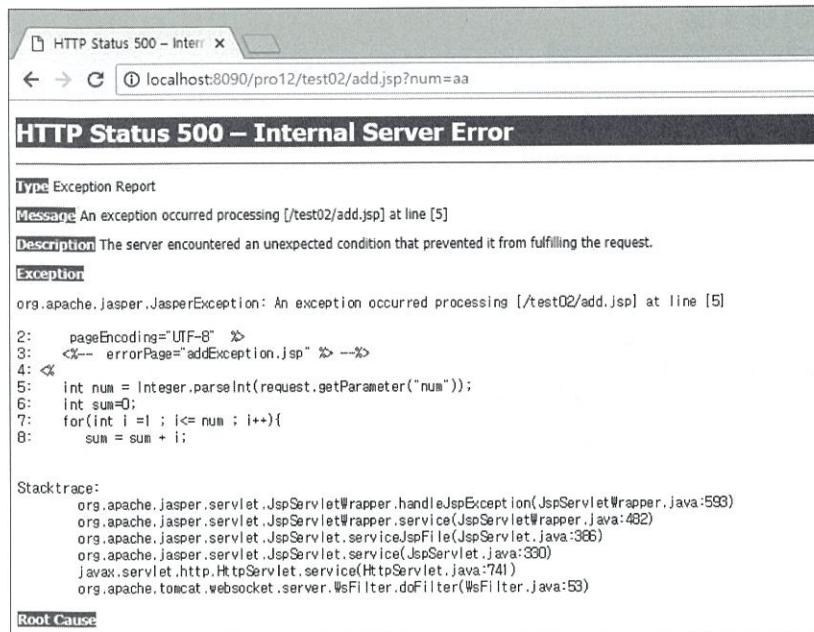
12.8.3 JSP 페이지의 오류 페이지 종류

JSP 실행 시 자주 발생하는 오류들이 있습니다. 이 책의 실습 과정에서도 자주 보았던 404 오류는 요청한 JSP 페이지가 없을 때 발생하는 오류이고, 500 오류는 컨테이너에서 JSP 페이지 처리 중에 오류가 발생할 때 표시되는 오류입니다.

▼ 그림 12-59 존재하지 않는 sum.jsp 요청 시 표시되는 404 오류



▼ 그림 12-60 컨테이너 처리 중 JSP에서 오류 발생 시 표시되는 500 오류



그런데 JSP 페이지가 많을 경우 이런 오류 처리를 일일이 JSP 페이지에서 설정해야 한다면 불편하겠죠. 전체 JSP 페이지에 대해 발생하는 오류에 따라서 화면에 표시되는 각각의 예외 처리 JSP 페이지를 적용할 수 있습니다.

12.8.4 에러 코드에 따른 예외 페이지 지정

다음은 web.xml에서 xml로 각각의 에러 코드에 대한 예외 처리 페이지를 지정하는 방법입니다.

코드 web.xml

```
<error-page>
    <error-code>오류코드</error-code>
    <location>오류 페이지 위치</location>
</error-page>
```

실제로 web.xml에 오류 페이지를 지정하여 실습해 보겠습니다.

- WebContent 하위에 오류 페이지들이 위치할 err 폴더를 만들고 error_404.jsp, error_500.jsp 파일을 준비합니다.

▼ 그림 12-61 실습 파일 위치



- web.xml에 <error-page> 태그를 이용해 각각의 에러 코드에 대해 처리할 오류 페이지가 있는 경로를 지정합니다.

코드 12-27 pro12/WebContent/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
```

```
<error-page>
    <error-code>404</error-code>
    <location>/err/error_404.jsp</location>
</error-page>

<error-page>
    <error-code>500</error-code>
    <location>/err/error_500.jsp</location>
</error-page>
</web-app>
```

404와 500 오류 발생 시 예외 처리를 할 페이지를 지정합니다.

3. 404 오류를 처리하는 JSP 페이지인 error_404.jsp를 다음과 같이 작성합니다.

코드 12-28 pro12/WebContent/err/error_404.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>404 예외 처리 페이지</title>
</head>
<body>
    <h1>요청한 페이지는 존재하지 않습니다.</h1>
</body>
</html>
```

4. 500 오류를 처리하는 JSP 페이지인 error_500.jsp를 다음과 같이 작성합니다.

코드 12-29 pro12/WebContent/err/error_500.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>500 예외 처리 페이지</title>
</head>
<body>
    <br>
    <h1>죄송합니다. 서비스 실행 중 오류가 발생했습니다.</h1>
    <h1>잠시 후 다시 시도해 보세요.</h1>
</body>
</html>
```

한 단계 위에 있는 image 폴더의 이미지를 표시합니다.

5. 브라우저 요청 시 예외를 발생시키는 number.jsp를 다음과 같이 작성합니다.

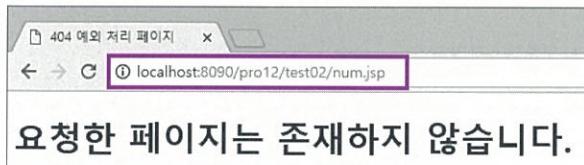
코드 12-30 pro12/WebContent/test02/number.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%
int num = Integer.parseInt(request.getParameter("num"));
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>테스트 페이지</title>
</head>

<body>
<h1>쇼핑몰 중심 JSP 입니다!!!!</h1>
</body>
</html>
```

6. 이제 각각의 예외를 고의로 발생시켜 볼까요? 먼저 존재하지 않는 <http://localhost:8090/pro12/test02/num.jsp>를 요청한 결과를 확인해 봅시다.

▼ 그림 12-62 존재하지 않는 페이지를 요청한 경우



7. 실행 중 예외를 발생시키는 <http://localhost:8090/pro12/test02/number.jsp>를 요청합니다.

▼ 그림 12-63 JSP 페이지 처리 중 500 오류가 발생한 경우



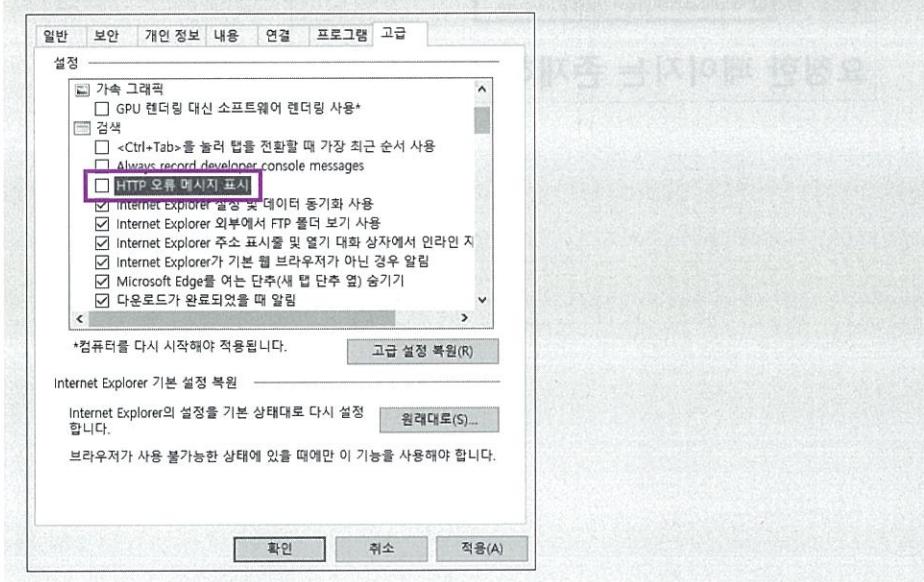
만약 한 개의 JSP 페이지에 페이지 디렉티브의 `errorPage` 속성과 `web.xml`이 같이 지정되어 있으면 페이지 디렉티브의 `errorPage`가 우선적으로 나타납니다.

Note

인터넷 익스플로러에서 사용자 오류 메시지 표시하기

익스플로러의 메뉴에서 도구 > 인터넷 옵션을 선택한 후 고급 탭에서 HTTP 오류 메시지 표시의 체크를 해제 합니다.

▼ 그림 12-64 익스플로러에서 사용자 오류 메시지 표시



12.9

JSP welcome 파일 지정하기



지금까지는 JSP나 서블릿을 일일이 브라우저에서 요청하여 화면을 표시했습니다. 그런데 웹 애플리케이션 첫 화면에 해당하는 홈페이지를 다음과 같이 web.xml에 등록해 두면 브라우저에서는 컨텍스트 이름만으로 요청하여 간단하게 표시할 수 있습니다.

코드 web.xml

```
<welcome-file-list>
    <welcome-file>jsp 또는 html 파일 이름1</welcome-file>
    <welcome-file>jsp 또는 html 파일 이름2</welcome-file>
    ...
</welcome-file-list>
```

12

홈페이지로 사용되는 welcome 페이지는 JSP나 HTML 파일이 될 수도 있고 여러 개를 등록해서 사용할 수도 있겠죠. 그러면 요청 시 첫 번째로 지정한 welcome 파일부터 차례로 찾아 홈페이지로 보여줍니다. 직접 web.xml에 설정해서 요청해 보겠습니다.

1. 다음과 같이 test02 폴더 하위에 main.jsp 파일과 web.xml 파일을 준비합니다.

▼ 그림 12-65 실습 파일 위치



2. web.xml에 <welcome-file-list> 태그 경로를 포함하여 홈페이지에 해당하는 파일들을 나열합니다.

코드 12-31 pro12/WebContent/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
    <welcome-file-list>
        <welcome-file>/test02/main.jsp</welcome-file> ────────── 여러 개의 welcome 파일을 지정합니다.
        <welcome-file>/test02/add.jsp</welcome-file>
        <welcome-file>/test02/add.html</welcome-file>
    </welcome-file-list>
</web-app>
```

3. 첫 번째 홈페이지인 main.jsp 페이지를 다음과 같이 작성합니다.

코드 12-32 pro12/WebContent/test02/main.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>홈페이지</title>
</head>
<body>
    <br>
    <h1>안녕하세요</h1>
    <h1>쇼핑몰 중심 JSP 홈페이지 입니다!!!</h1>
</body>
</html>
```

4. 톰캣을 다시 실행한 후 브라우저에서 컨텍스트 이름(/pro12)으로 요청합니다.

▼ 그림 12-66 실행 결과



Tip ☆

개발을 모두 마치고 실제 서비스를 제공할 때는 웹 사이트에 대한 도메인 이름을 구한 후 웹 호스팅 업체에서 제공하는 방법으로 브라우저에서 도메인 이름으로 요청해야 합니다. 그리고 다시 컨텍스트 이름으로 재요청하도록 설정하면 됩니다.

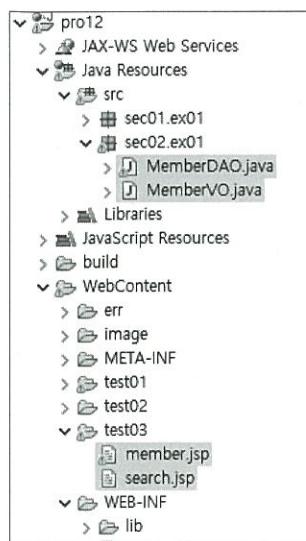
12.10

스크립트 요소 이용해 회원 정보 조회하기

이번에는 앞에서 배운 스크립트 요소를 이용해 데이터베이스의 회원 정보를 조회한 후 JSP 페이지에 출력하는 예제를 실습해 보겠습니다. 실습하기 전에 우선 데이터베이스를 연동하는 데 필요한 라이브러리를 반드시 설치합니다.

- sec02.ex01 패키지를 생성한 후 MemberVO, MemberDAO 클래스를 복사해 붙여 넣습니다. 그리고 test03 폴더에 member.jsp, search.jsp 파일을 추가합니다.

▼ 그림 12-67 실습 파일 위치



- 데이터베이스의 회원을 조회하는 JSP 페이지인 search.jsp를 다음과 같이 작성합니다. 찾고자 하는 이름을 입력하면 member.jsp로 전송합니다.

코드 12-33 pro12/WebContent/test03/search.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
```

```

<title>회원 검색창</title>
</head>
<body>
    <form method="post" action="member.jsp">———— 이름을 member.jsp로 전송합니다.
        이름:<input type="text" name="name"><br>
        <input type="submit" value="조회하기">
    </form>
</body>
</html>

```

3. member.jsp를 다음과 같이 작성합니다. 전송된 name 값을 가져온 후 스크립트릿에서 MemberDAO 객체를 생성하고 listMembers() 메서드를 호출해 이름에 대한 회원 정보를 조회합니다. 그리고 조회한 회원 정보를 for 반복문을 이용해 출력합니다.

코드 12-34 pro12\WebContent\test03\member.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    import="java.util.*"
    import="sec02.ex01.*"
    pageEncoding="UTF-8"
%>
<!DOCTYPE html>
<html>
<head>
    <style>
        h1 {
            text-align: center;
        }
    </style>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
</head>

<body>
    <h1>회원 정보 출력</h1>
<%
    request.setCharacterEncoding( "utf-8" );
    String _name = request.getParameter("name");———— 전송된 이름을 가져옵니다.
    MemberVO memberVO = new MemberVO();
    memberVO.setName(_name);
    MemberDAO dao=new MemberDAO();
    List membersList=dao.listMembers(memberVO);———— memberVO를 listMembers() 메서드로 전
    달하여 조회 조건에 해당되는 회원 정보를 조회합니다.
%>

```

```

<table border=1 width=800 align=center>
    <tr align=center bgcolor="#FFFF66">
        <td>아이디</td>
        <td>비밀번호</td>
        <td>이름</td>
        <td>이메일</td>
        <td>가입일자</td>
    </tr>

<%
for (int i=0; i < membersList.size(); i++){
    MemberVO vo=(MemberVO) membersList.get(i);
    String id=vo.getId();
    String pwd=vo.getPwd();
    String name=vo.getName();
    String email=vo.getEmail();
    Date joinDate=vo.getJoinDate();
%>

    <tr align=center>
        <td><%= id %></td>
        <td><%= pwd %></td>
        <td><%= name %></td>
        <td><%= email %></td>
        <td><%= joinDate %></td>
    </tr>

<%
}
%>
</table>
</body>
</html>

```

MemberDAO에서 조회한 회원 정보를
for 반복문을 이용해 테이블의 행으로
출력합니다.

- MemberDAO 클래스를 다음과 같이 작성합니다. 메서드로 전달된 이름에 대해 값이 존재하는 경우와 존재하지 않는 경우에 대해 SQL문을 동적으로 만들어서 조회합니다.

코드 12-35 pro12/src/sec02/ex01/MemberDAO.java

```

package sec02.ex01;
...
public class MemberDAO {
    private Connection con;
    private PreparedStatement pstmt;
    private DataSource dataFactory;

```

```

public MemberDAO() {
    try {
        Context ctx = new InitialContext();
        Context envContext = (Context) ctx.lookup("java:/comp/env");
        dataFactory = (DataSource) envContext.lookup("jdbc/oracle");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public List listMembers(MemberVO memberVO) {
    List membersList = new ArrayList();
    String _name=memberVO.getName(); •———— 조회할 이름을 가져옵니다.
    try {
        con = dataFactory.getConnection();
        String query = "select * from t_member ";
        if(_name!=null && _name.length()!=0){ •———— _name 값이 존재하면 SQL문에 where절을
            query+=" where name=?"; 추가하여 해당 이름으로 조회합니다.
            pstmt = con.PreparedStatement(query);
            pstmt.setString(1, _name); •———— 첫 번째 '?'에 전달된 이름을 지정합니다.
        }else { •———— _name 값이 없으면 모든 회원 정보를
            pstmt = con.PreparedStatement(query); 조회합니다.
        }
        System.out.println("prepareStatement: " + query);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            String id = rs.getString("id");
            String pwd = rs.getString("pwd");
            String name = rs.getString("name");
            String email = rs.getString("email");
            Date joinDate = rs.getDate("joinDate");
            MemberVO vo = new MemberVO();
            vo.setId(id);
            vo.setPwd(pwd);
            vo.setName(name);
            vo.setEmail(email);
            vo.setJoinDate(joinDate);
            membersList.add(vo);
        }
        rs.close();
        pstmt.close();
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```
        return membersList;
    }
}
```

5. <http://localhost:8090/pro12/test03/search.jsp>로 요청한 다음 조회할 이름을 입력하고 [member.jsp](http://localhost:8090/pro12/test03/member.jsp)로 전송합니다.

▼ 그림 12-68 검색창에 이름 입력



6. 그러면 조회한 회원 정보가 출력됩니다.

▼ 그림 12-69 회원 이름으로 조회한 결과 출력

회원 정보 출력				
아이디	비밀번호	이름	이메일	가입일자
hong	1212	홍길동	hong@gmail.com	2018-09-04

만약 이름을 입력하지 않고 조회할 경우에는 모든 회원 정보가 출력됩니다.

▼ 그림 12-70 이름을 입력하지 않으면 모든 회원 정보가 출력

회원 정보 출력				
아이디	비밀번호	이름	이메일	가입일자
hong	1212	홍길동	hong@gmail.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
kim	1212	김유신	kim@jweb.com	2018-09-04
park	1234	박찬호	park@test.com	2018-09-04

13

장

자바 코드를 없애는 액션 태그

13.1 인클루드 액션 태그 사용하기

13.2 포워드 액션 태그 사용하기

13.3 useBean, setProperty, getProperty 액션 태그 사용하기

JSP가 등장하게 된 배경은 디자이너가 자바 코드를 사용하지 않고도 쉽게 화면을 구현할 수 있도록 하기 위함이었다고 앞서 설명했습니다. 하지만 화면이 점차 복잡해지면서 디자이너들은 상황에 따라 HTML 태그에 자바 코드를 같이 써야 하는 문제로 어려움을 겪게 되었습니다. 따라서 JSP는 스크립트릿의 자바 코드를 제거하고 디자이너 입장에서 더 쉽고 편리하게 작업할 수 있는 태그 형태로 기능을 제공하게 되었고, 다음과 같은 액션 태그들로 자바 코드를 대신하게 되었습니다.

▼ 표 13-1 JSP의 여러 가지 액션 태그

이름	형식	설명
인클루드 액션 태그	<jsp:include>	이미 있는 JSP를 현재 JSP에 포함하는 태그
포워드 액션 태그	<jsp:forward>	서블릿에서 RequestDispatcher 클래스의 포워딩 기능을 대신하는 태그
유즈빈 액션 태그	<jsp:useBean>	객체를 생성하기 위한 new 연산자를 대신하는 태그
셋프로퍼티 액션 태그	<jsp:setProperty>	setter를 대신하는 태그
겟프로퍼티 액션 태그	<jsp:getProperty>	getter를 대신하는 태그

그럼 이 중 인클루드 액션 태그부터 알아보겠습니다.

13.1

인클루드 액션 태그 사용하기

JAVA WEB

인클루드 액션 태그(Include Action Tag)는 12장의 인클루드 디렉티브 태그처럼 화면을 분할해서 관리할 때 사용합니다. 우선 다음 예시를 통해 알아보겠습니다. 처음 메인 페이지를 요청하면 그림 13-1처럼 상단 화면과 왼쪽 메뉴 화면이 포함되어 나타납니다.

▼ 그림 13-1 메인 페이지

The screenshot shows the homepage of BookTopia, a Korean online bookstore. At the top, there's a navigation bar with links for '로그인' (Login), '회원가입' (Sign Up), and '고객센터' (Customer Service). A search bar is located next to a '검색' (Search) button. On the left side, there's a sidebar with a '국내외 도시' (Domestic and International Cities) dropdown menu and a 'Book ShoppingMall' section featuring a 'Try! helloworld 파이썬 ...' (Python) banner. Below this, there's a '베스트셀러' (Best Seller) section displaying several book covers with their titles and prices. A 'Pay 11월 이벤트' (November Pay Event) box is also visible. The right side of the page has social media links for Facebook, Twitter, and RSS, along with a note about regional shipping.

제목	가격
모두의 딥러닝	24,000원
마인크래프트 개발 대작	14,000원
판테크 가계부	30,000원
부동산 상식 시전	20,000원
Try! helloworld 자바스크립트	25,000원
직강만 위한 앤솔 실무	25,000원
시나공 워드 프로세서 살기	25,000원
컴퓨터 활용능력 2급 살기	25,000원

그리고 상품을 클릭하면 그림 13-2처럼 상세 페이지가 나타나면서 상단 화면과 왼쪽 메뉴 화면은 그대로 재사용합니다.

▼ 그림 13-2 상세 페이지 표시

The screenshot shows a product detail page for 'Java Programming'. The page includes a sidebar menu for '국내외 도서' categories like IT/인터넷, 경제/경영, 대학교재, 자기계발, 자연과학/공학, 역사/민문학, and more. A 'Pay 11월 이벤트' banner is visible. The main content area displays the book cover, title, and detailed product information:

정가	30,000원
판매가	27,000원(10% 할인)
포인트적립	2000P(10% 적립)
포인트 추가적립	만점이상 구매시 1,000P, 5만점이상 구매시 2,000P 추가적립 편의점 배송 이용 시 300P 추가적립
발행일	2018-10-02
페이지 수	996쪽
ISBN	923454566778
배송료	무료
배송안내	[당일배송] 당일배송 서비스 시작! [당일배송] 휴일에도 배송받는 Bookshop
도착예정일	지금 주문 시 내일 도착 예정
수량	1

Below the product details are buttons for '구매하기', '경매구니', and '위시리스트'. At the bottom of the page are navigation links: 책소개, 저자소개, 책목차, 출판사서평, 추천사, and 리뷰.

인클루드 액션 태그를 이용하면 이처럼 공통적으로 사용하는 홈페이지의 상단 화면과 왼쪽 메뉴 화면을 재사용할 수 있습니다.

인클루드 액션 태그의 형식은 다음과 같습니다.

```
<jsp:include page="jsp페이지" flush="true 또는 false">  
...  
</jsp:include>
```

여기서 **page**는 포함할 JSP 페이지를 의미합니다. 그리고 **flush**는 지정한 JSP를 실행하기 전 출력 버퍼 비움 여부를 지정합니다.

그럼 인클루드 액션 태그와 인클루드 디렉티브 태그의 차이점은 무엇일까요? 표 13-2에 인클루드 디렉티브 태그와의 공통점과 차이점을 정리했습니다.

▼ 표 13-2 인클루드 액션 태그와 인클루드 디렉티브 태그 비교

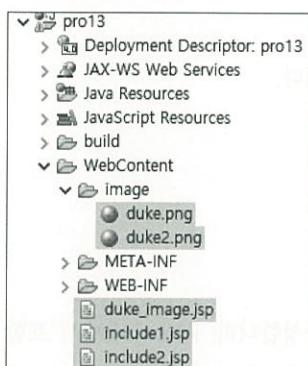
항목	인클루드 액션 태그	인클루드 디렉티브 태그
기능	JSP 레이아웃 모듈화	JSP 레이아웃 모듈화
처리 시간	요청 시간에 처리	JSP를 자바 코드로 변환 시 처리
데이터 처리 방법	param 액션 태그를 이용해 동적 처리 가능	정적 처리만 가능
포함된 JSP 자바 파일 변환 여부	포함되는 JSP가 각각 자바 파일로 생성	포함되는 JSP가 포함하는 JSP에 합쳐진 후 한 개의 자바 파일로 생성

13.1.1 JSP 페이지에 이미지 포함 실습

인클루드 액션 태그를 사용하여 실습해 보겠습니다.

- 새 프로젝트 pro13을 만들고 다음과 같이 실습에 필요한 이미지 파일(duke.png, duke2.png)과 duke_image.jsp, include1.jsp, include2.jsp 파일을 추가합니다.

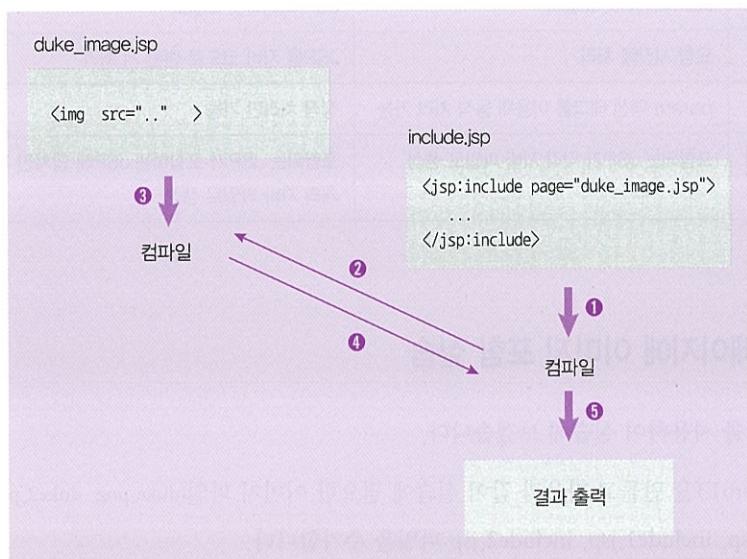
▼ 그림 13-3 실습 파일 위치



이 책과 함께 제공하는 예제 파일의 image 폴더에서 실습에 필요한 이미지 파일들을 내려 받을 수 있습니다.

인클루드 액션 태그의 실행 과정은 다음과 같습니다.

▼ 그림 13-4 인클루드 액션 태그 처리 과정



- ① 브라우저 요청 시 JSP 파일을 컴파일합니다.
 - ② 컴파일 시 `<jsp:include>`가 지시하는 JSP를 요청합니다.
 - ③ 요청된 JSP를 컴파일합니다.
 - ④ 컴파일된 JSP가 응답을 보냅니다.
 - ⑤ JSP는 브라우저에서 요청한 응답 결과를 출력합니다.
2. 자식 JSP에 해당하는 `duke_image.jsp`를 다음과 같이 작성합니다. 부모 JSP에서 포함 요청 시 전달되는 이름과 이미지 파일을 `getParameter()` 메서드를 이용해 가져온 후 이름과 해당 이미지를 출력합니다.

코드 13-1 pro13\WebContent\duke_image.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding("utf-8");
    String name = request.getParameter("name");
    String imgName = request.getParameter("imgName");
%>
<!DOCTYPE html>
```

param 액션 태그로 전달된 매개변수를
getParameter() 메서드를 이용해 가져
옵니다.

```

<html>
<head>
<meta charset="UTF-8">
<title>듀크이미지</title>
</head>
<body>
<br><br>
<h1>이름은 <%=name %>입니다. </h1> <br><br>

</body>
</html>

```

3. 부모 JSP인 include1.jsp를 다음과 같이 작성합니다. <jsp:include> 태그의 page 속성에 포함할 자식 JSP인 duke_image.jsp를 지정합니다. 그리고 <jsp:param> 태그(param 액션 태그)를 이용해 이름과 이미지 파일 이름을 동적으로 자식 JSP인 duke_image.jsp로 포워딩합니다.

코드 13-2 pro13/WebContent/include1.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
<%
request.setCharacterEncoding("utf-8");
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>include1.jsp</title>
</head>
<body>
안녕하세요. 쇼핑몰 중심 JSP 시작입니다!!!
<br>
<jsp:include page="duke_image.jsp" flush="true" >
<jsp:param name="name" value="듀크" />
<jsp:param name="imgName" value="duke.png" />
</jsp:include>
<br>
안녕하세요. 쇼핑몰 중심 JSP 끝 부분입니다.!!!
</body>
</html>

```

duke_image.jsp를 동적으로 포워딩합니다.

param 액션 태그를 이용해 duke_image.jsp로 이름과 파일 이름을 전달합니다.

4. 이번에는 다른 부모 JSP인 include2.jsp를 다음과 같이 작성합니다. 자식 JSP로 다른 이름과 이미지 파일 이름을 전달합니다.

코드 13-3 pro13/WebContent/include2.jsp

```
...
<body>
    안녕하세요. 쇼핑몰 중심 JSP 시작입니다!!!
    <br>
    <jsp:include page="duke_image.jsp" flush="true" >
        <jsp:param name="name" value="듀크2"/>
        <jsp:param name="imgName" value="duke2.png"/>
    </jsp:include>
    <br>
    안녕하세요. 쇼핑몰 중심 JSP 끝 부분입니다.!!!
</body>
</html>
```

param 액션 태그를 이용해 duke_image.jsp로 이름과 파일 이름을 전달합니다.

Note 지면의 한계로 중복되는 코드는 생략하고 주요 부분만 수록하였습니다. 전체 소스는 예제 파일을 참고하기 바랍니다.

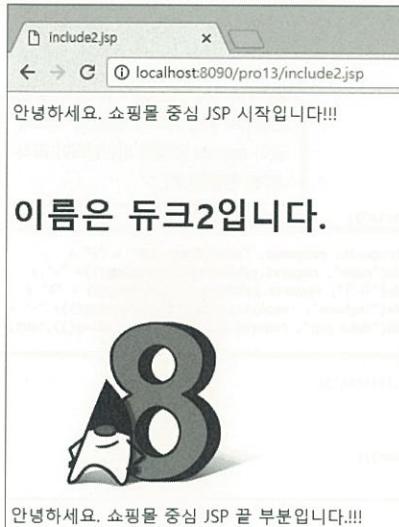
5. 브라우저에서 JSP 파일을 요청하면 각각 다른 이미지와 이름이 출력됩니다. 먼저 `http://localhost:8090/pro13/include1.jsp`로 요청합니다.

▼ 그림 13-5 include1.jsp 요청 시 결과 출력



6. <http://localhost:8090/pro13/include2.jsp>로 요청하면 앞에서와는 다른 이미지와 이름이 출력되는 것을 볼 수 있습니다.

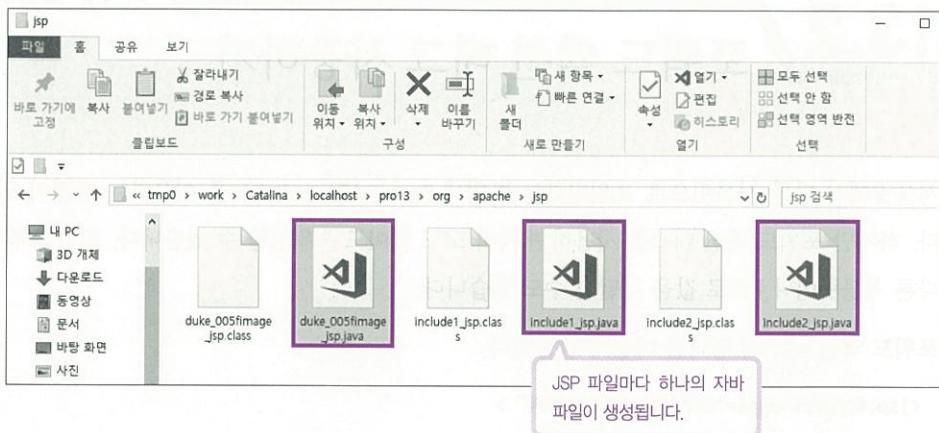
▼ 그림 13-6 include2.jsp 요청 시 결과 출력



이처럼 인클루드 액션 태그는 포함되는 자식 JSP에 데이터를 동적으로 전달해서 원하는 결과를 출력할 수 있습니다.

7. 다음 경로의 윈도 탐색기로 가면 JSP 파일이 자바 파일로 변환된 것을 볼 수 있습니다.

▼ 그림 13-7 자바 파일로 변환된 상태



8. 인클루드 액션 태그로 요청한 include1.jsp의 자바 파일을 열어볼까요? <jsp:param> 태그로 전달한 매개변수들이 자식 JSP로 전달됩니다.

▼ 그림 13-8 include 액션 태그가 포함된 JSP 파일 변환 결과(include1.jsp.java)

```

120  request.setCharacterEncoding("utf-8");
121
122  out.write("\r\n");
123  out.write("\r\n");
124  out.write("<!DOCTYPE html>\r\n");
125  out.write("<html>\r\n");
126  out.write("<head>\r\n");
127  out.write("<meta charset=\"UTF-8\">\r\n");
128  out.write("<title>include1.jsp</title>\r\n");
129  out.write("</head>\r\n");
130  out.write("<body>\r\n");
131  out.write(" 안녕하세요, 쇼핑몰 중심 JSP 시작입니다!!!\r\n");
132  out.write("<br>\r\n");
133  org.apache.jasper.runtime.JspRuntimeLibrary.include(request, response, "duke_image.jsp" + "?"
134  org.apache.jasper.runtime.JspRuntimeLibrary.URLEncode("name", request.getCharacterEncoding())+ "=" +
135  org.apache.jasper.runtime.JspRuntimeLibrary.URLEncode("듀크", request.getCharacterEncoding()) + "&" +
136  org.apache.jasper.runtime.JspRuntimeLibrary.URLEncode("imgName", request.getCharacterEncoding())+ "=" +
137  org.apache.jasper.runtime.JspRuntimeLibrary.URLEncode("duke.png", request.getCharacterEncoding()), out, true);
138  out.write("\r\n");
139  out.write("<br>\r\n");
140  out.write(" 안녕하세요, 쇼핑몰 중심 JSP 끝 부분입니다.!!!\r\n");
141  out.write("</body>\r\n");
142  out.write("</html>\r\n");
143  } catch (java.lang.Throwable t) {
144      if (!(t instanceof javax.servlet.jsp.SkipPageException)){
145          out = _jspx_out;
146          if (out != null && out.getBufferSize() != 0)

```

인클루드 액션 태그의 <param> 태그의
값이 request 객체에 바인딩되어 자식
JSP로 전달됩니다.

인클루드 디렉티브 태그와 인클루드 액션 태그의 실행 결과는 같지만 두 기능의 차이점은 어떤 것들인지도 잘 알아두세요.

13.2 포워드 액션 태그 사용하기

J A V A W E B

서블릿에서 다른 서블릿으로 포워딩하는 방법에 RequestDispatcher를 이용하는 방법이 있습니다. 하지만 포워드 액션 태그를 사용하면 자바 코드 없이도 포워딩할 수 있습니다. 또한 포워딩 시 다른 서블릿이나 JSP로 값을 전달할 수도 있습니다.

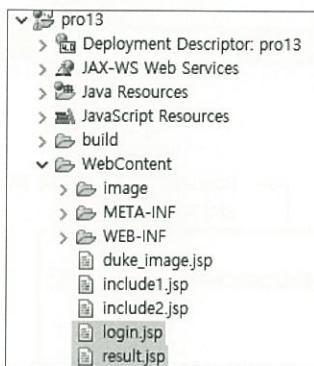
포워드 액션 태그의 형식은 다음과 같습니다.

```
<jsp:forward page="포워딩할 JSP 페이지" >
  ..
</jsp:forward>
```

그럼 포워드 액션 태그를 다음 예제를 통해 실습해 보겠습니다.

1. 다음과 같이 실습 파일 login.jsp, result.jsp를 생성합니다.

▼ 그림 13-9 실습 파일 위치



2. 로그인창에서 ID와 비밀번호를 입력한 후 action의 result.jsp로 전달하도록 login.jsp를 작성합니다.

코드 13-4 pro13/WebContent/login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding("utf-8");
%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>로그인창</title>
</head>
<body>
    <h1>아이디를 입력하지 않았습니다. 아이디를 입력해 주세요. </h1>
    <form action="result.jsp" method="post">
        아이디: <input type="text" name="userID"><br>
        비밀번호: <input type="password" name="userPw"><br>
        <input type="submit" value="로그인">
        <input type="reset" value="다시입력">
    </form>
</body>
</html>
```

3. ID를 입력하지 않은 경우 자바의 RequestDispatcher를 사용하지 않고 포워드 액션 태그를 사용해 다시 로그인창으로 이동하도록 result.jsp를 작성합니다.

코드 13-5 pro13/WebContent/result.jsp

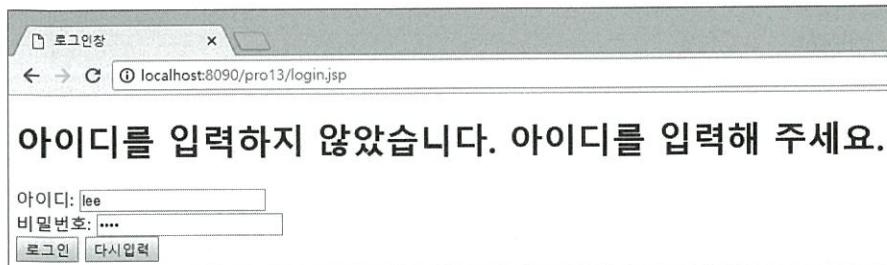
```
...
<body>
<%
String userID = request.getParameter("userID" );
if(userID.length()==0) {
/*
RequestDispatcher dispatch = request.getRequestDispatcher("login.jsp");
dispatch.forward(request, response);
*/
%>
<jsp:forward page="login.jsp" />
<%
}
%>
<h1> 환영합니다 <%= userID %>님!! </h1>
</body>
</html>
```

RequestDispatcher를 사용해 포워딩
하지 않아도 됩니다.

ID를 입력하지 않았으면 다시 <jsp:forward>
태그를 사용해 로그인창으로 포워딩합니다.

4. <http://localhost:8090/pro13/login.jsp>로 요청하여 ID와 비밀번호를 입력하고 로그인합니다.

▼ 그림 13-10 로그인창에서 로그인



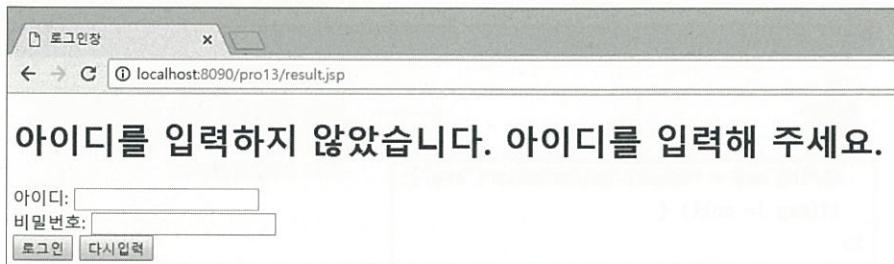
5. ID를 입력하면 정상적인 메시지를 출력합니다.

▼ 그림 13-11 ID를 입력한 경우



6. 하지만 ID를 입력하지 않고 로그인을 시도하면 로그인창으로 포워딩하여 다음과 같은 메시지를 출력합니다.

▼ 그림 13-12 ID를 입력하지 않은 경우



그런데 이 로그인 예제는 약간의 문제가 있습니다. 최초 login.jsp로 접속하면 로그인창에 오류 메시지("아이디를 입력하지 않았습니다. 아이디를 입력해 주세요.")가 나타난다는 것입니다(그림 13-10). 폐이지에 처음 접속했을 때는 ID와 비밀번호 입력창만 나타나게 하고, 오류 시에만 오류 메시지를 나타나게 하는 것이 사용자에게 더 익숙하겠지요?

이 과정을 <jsp:forward> 태그 안에 param 액션 태그를 이용해서 처리해 보겠습니다.

7. login2.jsp와 result2.jsp 파일을 새로 만듭니다.

▼ 그림 13-13 실습 파일 위치



8. login2.jsp를 다음과 같이 작성합니다. 로그인창에 접속 시에는 getParameter() 메서드를 이용해 msg 값을 가져와서 표시하도록 구현합니다. 최초 요청 시에는 msg 값이 null이므로 아무것도 표시하지 않습니다.

코드 13-6 pro13/WebContent/login2.jsp

```
...
<body>
<%
String msg = request.getParameter("msg");
if(msg != null) {
%>
<h1> <%=msg %> </h1>
<%
}
%>
...
...
```

브라우저에서 접속 시에는 msg 값을 가져와서 표시하고, 최초 접속 시에는 null이므로 아무것도 표시하지 않습니다.

9. ID를 입력하지 않았을 경우 다시 로그인창으로 포워딩하면서 이번에는 <jsp:param> 태그를 이용해 msg 값을 전달합니다.

코드 13-7 pro13/WebContent/result2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
request.setCharacterEncoding("utf-8");
%>
<%
String msg = "아이디를 입력하지 않았습니다. 아이디를 입력해 주세요." ;
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>결과창</title>
</head>
<body>
<%
String userID = request.getParameter("userID" );
if(userID.length()==0) {
%>
<jsp:forward  page="login2.jsp" >
```

login.jsp로 전달할 오류 메시지를 선언합니다.

로그인 시 입력한 이름을 가져옵니다.

이름을 입력하지 않았을 경우 <jsp:param> 액션 태그를 이용해 오류 메시지를 login.jsp로 전달합니다.

```

<jsp:param name="msg" value="<% msg %>" />
</jsp:forward>

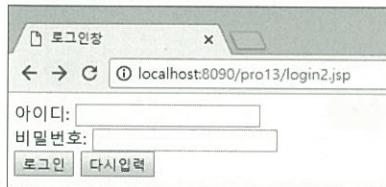
<%
}

%>
<h1>환영합니다. <%=userID %>님!!! </h1>
</body>
</html>

```

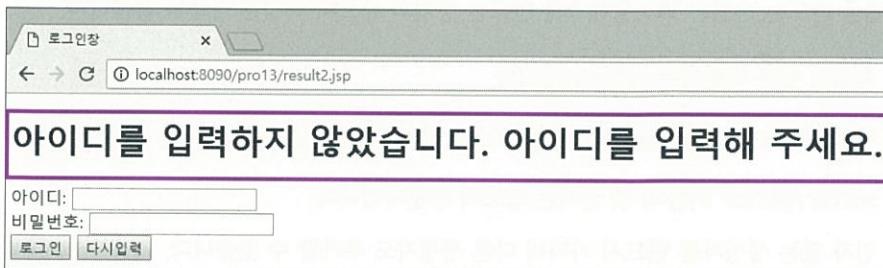
10. <http://localhost:8090/pro13/login2.jsp>로 요청합니다. 최초 로그인창 접속 시 앞에서와는 달리 어떤 메시지도 나타나지 않습니다.

▼ 그림 13-14 최초 로그인창에 접속 시



11. ID를 입력하지 않고 로그인하면 로그인창으로 다시 포워딩되면서 오류 메시지가 나타납니다.

▼ 그림 13-15 ID를 입력하지 않고 로그인한 경우



13.3

useBean, setProperty, getProperty 액션 태그 사용하기

화면 작업을 하는 디자이너 입장에서는 클래스 객체의 속성에 접근할 때 자바의 getter나 setter를 사용하는 것보다는 태그를 사용하는 것이 더 쉽습니다.

이 절에서는 useBean, setProperty, getProperty 액션 태그를 사용해 객체 생성부터 속성에 값을 저장하거나 가져오는 방법에 대해 알아보겠습니다. 본격적으로 useBean 액션 태그를 살펴보기 전에 먼저 자바 빙(Java bean)의 개념부터 실습을 통해 알아보겠습니다.

13.3.1 자바 빙을 이용한 회원 정보 조회 실습

자바 빙은 웹 프로그램, 즉 Java EE 프로그래밍 시 여러 객체를 거치면서 만들어지는 데이터를 저장하거나 전달하는 데 사용합니다. 자바의 DTO(Data Transfer Object, 데이터 전송 객체)¹ 클래스, VO(Value Object, 값 객체) 클래스와 같은 개념이라고 할 수 있습니다.

자바 빙을 만드는 방법은 VO 클래스를 만드는 방법과 같으며 특징은 다음과 같습니다.

- 속성의 접근 제한자는 private입니다.
- 각 속성(attribute, property)은 각각의 setter/getter를 가집니다.
- setter/getter 이름의 첫 글자는 반드시 소문자입니다.
- 인자 없는 생성자를 반드시 가지며 다른 생성자도 추가할 수 있습니다.

그럼 자바 빙을 이용해 회원 테이블의 회원 정보를 조회한 후 출력해 보겠습니다.

¹ 객체와 객체 사이에 데이터를 전달하는 용도로 사용하는 값 객체

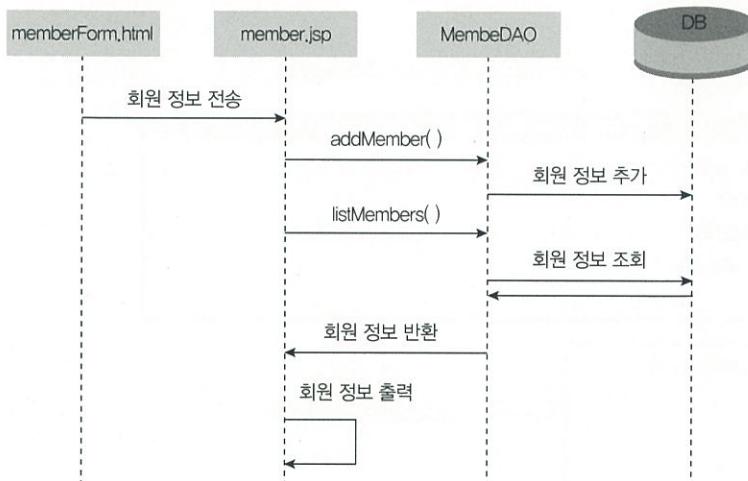
1. sec01.ex01 패키지를 생성하고 MemberBean, MemberDAO 클래스를 추가합니다. 그리고 실습 파일 member.jsp와 memberForm.html을 생성합니다.

▼ 그림 13-16 실습 파일 위치



다음은 MemberBean을 이용해 회원 정보를 등록, 조회하고 출력하는 과정입니다.

▼ 그림 13-17 회원 정보 등록 및 조회 과정



2. 회원 정보를 저장하는 MemberBean 클래스를 만들기 전에 다음의 회원 테이블을 봅시다.

▼ 그림 13-18 회원 정보 저장 테이블의 각 컬럼 이름

```
1 --회원 테이블 생성
2 create table t_member(
3     id varchar2(10) primary key,
4     pwd varchar2(10),
5     name varchar2(50),
6     email varchar2(50),
7     joinDate date default sysdate
8 );
```

3. 회원 테이블을 참고해 MemberBean 클래스를 작성합니다. MemberBean 클래스의 속성은 회원 테이블의 각 컬럼 이름을 그대로 사용하며 자료형도 컬럼 이름의 자료형과 동일하게 선언합니다.

코드 13-8 pro13/src/sec01/ex01/MemberBean.java

```
package sec01.ex01;
...
public class MemberBean {
    private String id;           ← 회원 테이블의 컬럼 이름과 동일하게
    private String pwd;          ← 이름과 자료형을 선언합니다.
    private String name;
    private String email;
    private Date joinDate;

    public MemberBean() {         ← 인자가 네 개인 생성자를 추가합니다.
    }

    public MemberBean(String id, String pwd, String name, String email) {
        this.id = id;
        this.pwd = pwd;
        this.name = name;
        this.email = email;
    }

    public String getId() {
        return id;
    }                           ← getter/setter를 추가합니다.

    public void setId(String id) {
        this.id = id;
    }

    public String getPwd() {
```

```

        return pwd;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}

```

4. 회원가입창에서 회원 정보를 입력한 후 member.jsp로 전송하도록 memberForm.html을 작성합니다.

코드 13-9 pro13/WebContent/memberForm.html

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>회원 가입창</title>
<body>
    <form method="post" action="member.jsp">
        <h1 style="text-align:center">회원 가입창</h1>
        <table align="center">
            <tr>
                <td width="200">
                    <p align="right">아이디</p>
                </td>
                <td width="400"><input type="text" name="id"></td>
            </tr>
            <tr>

```

```

<td width="200">
    <p align="right">비밀번호
</td>
<td width="400"><input type="password" name="pwd"></td>
</tr>
<tr>
    <td width="200">
        <p align="right">이름
    </td>
    <td width="400">
        <p><input type="text" name="name">
    </td>
</tr>
<tr>
    <td width="200">
        <p align="right">이메일
    </td>
    <td width="400">
        <p><input type="text" name="email">
    </td>
</tr>
<tr>
    <td width="200">
        <p>&ampnbsp</p>
    </td>
    <td width="400">
        <input type="submit" value="가입하기">
        <input type="reset" value="다시입력">
    </td>
</tr>
</table>
</form>
</body>
</html>

```

5. member.jsp를 다음과 같이 작성합니다. 전송된 회원 정보를 getParameter() 메서드를 이용해 가져온 후 MemberBean 객체를 생성하여 각 회원 정보를 속성에 설정합니다. 그런 다음 MemberDAO의 addMember() 메서드를 호출해 인자로 전달합니다. 새 회원을 추가한 후에는 다시 MemberDAO의 listMembers() 메서드를 호출해 모든 회원 정보를 조회하고 목록으로 출력합니다.

코드 13-10 pro13\WebContent\member.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
           import="java.util.* , sec01.ex01.*" pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding("UTF-8"); •———— 한글 인코딩을 설정합니다.
%>
<%
    String id=request.getParameter("id");
    String pwd = request.getParameter("pwd");
    String name = request.getParameter("name");
    String email = request.getParameter("email");
    MemberBean m = new MemberBean(id, pwd, name, email);
    MemberDAO memberDAO=new MemberDAO();
    memberDAO.addMember(m); •———— 회원 정보를 테이블에 추가합니다.
    List membersList = memberDAO.listMembers(); •———— 전체 회원 정보를 조회합니다.
%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>회원 목록창</title>
</head>
<body>
    <table align="center" width="100%">
        <tr align="center" bgcolor="#99ccff">
            <td width="7%">아이디</td>
            <td width="7%">비밀번호</td>
            <td width="5%">이름</td>
            <td width="11%">이메일</td>
            <td width="5%">가입일</td>
        </tr>
        <%
            if(membersList.size()==0) {
        %>
        <tr>
            <td colspan="5">
                <p align="center"><b><span style="font-size:9pt;">
                    등록된 회원이 없습니다.</span></b></p>
            </td>
        </tr>
        <%
    }else{
        for( int i = 0; i < membersList.size(); i++ ) {
            MemberBean bean = (MemberBean) membersList.get(i);
        %>
    }
}

```

전송된 회원 정보를 가져옵니다.

MemberBean 객체를 생성한 후 회원 정보를 속성에 설정합니다.

회원 정보를 테이블에 추가합니다.

전체 회원 정보를 조회합니다.

for 반복문을 이용해 membersList의 저장된 MemberBean 객체를 한 개씩 가져온 후 각 속성의 회원 정보를 다시 getter를 이용해 출력합니다.

```

<tr align="center">
    <td>
        <%= bean.getId() %>
    </td>
    <td>
        <%= bean.getPwd() %>
    </td>
    <td>
        <%= bean.getName() %>
    </td>
    <td>
        <%= bean.getEmail() %>
    </td>
    <td>
        <%= bean.getJoinDate() %>
    </td>
</tr>
<%
    } // end for
} // end if
%>
<tr height="1" bgcolor="#99ccff">
    <td colspan="5"></td>
</tr>
</table>
</body>
</html>

```

6. MemberDAO를 다음과 같이 작성합니다. addMember() 메서드 호출 시 MemberBean 객체로 전달된 회원 정보를 getter를 이용해 가져온 후 insert문을 이용해 추가합니다.

코드 13-11 pro13/src/sec01/ex01/MemberDAO.java

```

package sec01.ex01;

...
public class MemberDAO
{
    private Connection con;
    private PreparedStatement pstmt;
    private DataSource dataFactory;

    public MemberDAO()
    {
        try

```

```

    {
        Context ctx = new InitialContext();
        Context envContext = (Context) ctx.lookup("java:/comp/env");
        dataFactory = (DataSource) envContext.lookup("jdbc/oracle");
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

public List listMembers()
{
    List list = new ArrayList();
    try
    {
        con = dataFactory.getConnection();
        String query = "select * from t_member order by joinDate desc ";
        System.out.println("prepareStatement: " + query);
        pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next())
        {
            String id = rs.getString("id");
            String pwd = rs.getString("pwd");
            String name = rs.getString("name");
            String email = rs.getString("email");
            Date joinDate = rs.getDate("joinDate");
            MemberBean vo = new MemberBean(); ← 조회한 회원 정보를 MemberBean 객체의 속성에
            vo.setId(id);                    저장한 후 다시 ArrayList에 저장합니다.
            vo.setPwd(pwd);
            vo.setName(name);
            vo.setEmail(email);
            vo.setJoinDate(joinDate);
            list.add(vo);
        }
        rs.close();
        pstmt.close();
        con.close();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    return list;
}

```

회원 정보를 최근 가입일 순으로 조회합니다.

```

public void addMember(MemberBean memberBean) {
    try
    {
        Connection con = dataFactory.getConnection();
        String id = memberBean.getId();           ← MemberBean 객체에 저장된
                                                회원 정보를 전달합니다.
        String pwd = memberBean.getPwd();
        String name = memberBean.getName();
        String email = memberBean.getEmail();
        String query = "insert into t_member";
        query += " (id,pwd,name,email)";
        query += " values(?, ?, ?, ?)";
        System.out.println("prepareStatement: " + query);
        pstmt = con.prepareStatement(query);
        pstmt.setString(1, id);
        pstmt.setString(2, pwd);
        pstmt.setString(3, name);
        pstmt.setString(4, email);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

7. <http://localhost:8090/pro13/memberForm.html>로 요청하여 회원 정보를 입력한 후 **가입하기**를 클릭합니다.

▼ 그림 13-19 회원 정보 입력 후 가입하기 클릭

회원 가입창	
아이디	<input type="text" value="park2"/>
비밀번호	<input type="password" value="..."/>
이름	<input type="text" value="박지성"/>
이메일	<input type="text" value="park2@test.com"/>
<input type="button" value="가입하기"/> <input type="button" value="다시입력"/>	

8. 새 회원이 추가된 후 다시 회원 정보를 조회하여 목록으로 출력하는 것을 볼 수 있습니다.

▼ 그림 13-20 새 회원 등록 후 회원 목록 출력



A screenshot of a web browser window titled "회원 목록화". The address bar shows "localhost:8090/pro13/member.jsp". The page displays a table with the following data:

아이디	비밀번호	이름	이메일	가입일
park2	1234	박지성	park2@test.com	2018-09-10
park	1234	박찬호	park@test.com	2018-09-04
kim	1212	김유신	kim@jweb.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
hong	1212	홍길동	hong@gmail.com	2018-09-04

13.3.2 유즈빈 액션 태그를 이용한 회원 정보 조회 실습

앞에서 자바 빈을 사용해 회원 추가와 조회를 어떻게 하는지 살펴봤습니다. 그런데 이처럼 자바 코드로 이루어진 자바 빈을 자주 사용할 경우 화면이 복잡해진다는 단점이 있습니다. 이러한 단점을 보완하기 위해 나온 것이 유즈빈 액션 태그입니다.

유즈빈 액션 태그는 JSP 페이지에서 자바 빈을 대체하기 위한 태그로, 사용 형식은 다음과 같습니다.

```
<jsp:useBean id="빈 이름" class="패키지 이름을 포함한 자바 빈 클래스 [scope="접근범위"]/>
```

여기서 `id`는 JSP 페이지에서 자바 빈 객체에 접근할 때 사용할 이름을 의미합니다. `class`는 패키지 이름을 포함한 자바 빈 이름을, `scope`는 자바 빈에 대한 접근 범위를 지정하는 역할을 합니다 (`page`, `request`, `session`, `application`를 가지며 기본값은 `page`입니다).

앞의 예제를 기반으로 이번에는 유즈빈 액션 태그를 이용해 회원을 등록하고 조회해 보겠습니다. 계속해서 같은 예제로 살펴보는 이유는 두 태그의 차이를 올바로 이해하기 위해서입니다.

1. 다음과 같이 실습 파일 member2.jsp를 추가합니다.

▼ 그림 13-21 실습 파일 위치



2. member2.jsp를 다음과 같이 작성합니다. 회원 가입 및 조회 시 MemberBean 클래스에 대한
유즈빈 액션 태그를 사용하려면 먼저 <jsp:useBean> 액션 태그를 이용하여 MemberBean 클
래스에 대해 id가 m인 빈을 생성합니다. 이는 직접 자바 코드로 MemberBean 객체를 생성하
는 것과 같은 역할을 합니다. 그런 다음 자바 코드에서 빈 id인 m을 이용해 회원 가입창에서
전달된 회원 정보를 setter를 통해 빈 속성에 설정합니다.

코드 13-12 pro13/WebContent/member2.jsp

```
<%@ page language="java"    contentType="text/html; charset=UTF-8"
       import="java.util.*, sec01.ex01.*"  pageEncoding="UTF-8"%>
<%
request.setCharacterEncoding("UTF-8");           useBean에 속성 값을 설정하기 전에 한글
%>                                         인코딩 작업을 합니다.
<jsp:useBean id="m" class="sec01.ex01.MemberBean" scope="page" />           유즈빈 액션 태그로 id가 m인 MemberBean
                                                                           객체를 만듭니다.
<%
String  id=request.getParameter("id");
String  pwd = request.getParameter("pwd");
String  name = request.getParameter("name");
String  email = request.getParameter("email");           자바 코드에서 MemberBean 객체를
                                                                           생성하지 않습니다.
// MemberBean  m = new MemberBean(id, pwd, name, email);
```

```

m.setId(id);
m.setPwd(pwd);
m.setName(name);
m.setEmail(email);

MemberDAO memberDAO=new MemberDAO();
memberDAO.addMember(m);
List membersList = memberDAO.listMembers();

%>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원 목록창</title>
</head>
<body>
<table align="center" width="100%">
<tr align="center" bgcolor="#99ccff">
<td width="7%">아이디</td>
<td width="7%">비밀번호</td>
<td width="5%">이름</td>
<td width="11%">이메일</td>
<td width="5%">가입일</td>
</tr>
<%
    if( membersList.size()==0) {
%>
    ...

```

useBean에 전송된 회원 정보를 설정합니다.

회원 정보를 추가한 후
목록으로 출력합니다.

3. 마찬가지로 <http://localhost:8090/pro13/memberForm.html>로 요청하여 회원가입창에 회원 정보를 입력하고 **가입하기**를 클릭합니다.

▼ 그림 13-22 회원 정보 입력 후 가입하기 클릭

회원 가입창

아이디

비밀번호

이름

이메일

4. 자바빈을 사용했을 때와 마찬가지로 추가된 새 회원과 함께 회원 목록을 출력합니다.

▼ 그림 13-23 회원 등록 후 회원 목록 출력

아이디	비밀번호	이름	이메일	가입일
cha	1234	차범근	cha@test.com	2018-09-10
park2	1234	박지성	park2@test.com	2018-09-10
park	1234	박찬호	park@test.com	2018-09-04
kim	1212	김유신	kim@jweb.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
hong	1212	홍길동	hong@gmail.com	2018-09-04

눈으로 보는 출력 결과는 같지만 유즈빈 액션 태그를 사용하면 굳이 자바 코드를 사용하지 않고도 JSP 페이지에서 처리할 수 있다는 점에서 효율적입니다.

13.3.3 setProperty/getProperty 액션 태그를 이용한 회원 정보 조회 실습

앞 절에서는 `useBean` 액션 태그를 사용해 자바 코드를 사용하지 않고 자바빈을 생성했습니다. 그러나 여전히 빙의 속성에 값을 설정할 때는 자바 코드에서 `setter`를 사용하고 있습니다. `useBean`에 접근해 속성 값을 설정하거나 가져오는 `<jsp:setProperty>` 액션 태그와 `<jsp:getProperty>` 액션 태그를 사용하는 방법을 표 13-2에 정리해 두었으니 참고하기 바랍니다.

▼ 표 13-3 setProperty와 getProperty 태그의 특징

이름	정의	형식
setProperty	useBean의 속성에 값을 설정하는 태그	<jsp:setProperty name="자바 빈 이름" property="속성 이름" value="값" /> - name: <jsp:useBean> 액션 태그의 id 속성에 지정한 이름 - property: 값을 설정할 속성 이름 - value: 속성에 설정할 속성 값
getProperty	useBean의 속성 값을 얻는 태그	<jsp:getProperty name="자바 빈 이름" property="속성 이름" /> - name: <jsp:useBean> 액션 태그의 id 속성에 지정한 이름 - property: 값을 얻을 속성 이름

이번에는 자바의 **setter**를 사용하지 않고 빈 속성을 설정해 보겠습니다.

1. 다음과 같이 실습 파일 member3~7.jsp를 준비합니다.

▼ 그림 13-24 실습 파일 위치



2. member3.jsp에서 <jsp:useBean> 액션 태그로 생성된 빈에 대해 <jsp:setProperty> 액션 태그를 이용해 빈의 속성을 설정합니다. 이번에는 회원 가입창에서 전송한 회원 정보를 자바 코드, 즉 setter를 사용해 일일이 설정하지 않았다는 것이 큰 차이입니다.

코드 13-13 pro13/WebContent/member3.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
           import="java.util.*, sec01.ex01.*" pageEncoding="UTF-8"%>
<% request.setCharacterEncoding("UTF-8"); %>
<jsp:useBean id="m" class="sec01.ex01.MemberBean" scope="page"/>
<jsp:setProperty name="m" property="id"
                  value='<%= request.getParameter("id") %>' />
<jsp:setProperty name="m" property="pwd"
                  value='<%= request.getParameter("pwd") %>' />
<jsp:setProperty name="m" property="name"
                  value='<%= request.getParameter("name") %>' />
<jsp:setProperty name="m" property="email"
                  value='<%= request.getParameter("email") %>' />
<%
    String id=request.getParameter("id");
    String pwd = request.getParameter("pwd");
    String name = request.getParameter("name");
    String email = request.getParameter("email");

    m.setId(id);
    m.setPwd(pwd);
    m.setName(name);
    m.setEmail(email)
%>
MemberDAO memberDAO=new MemberDAO();
memberDAO.addMember(m);
List membersList = memberDAO.listMembers();
%>

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>회원 목록창</title>
</head>
```

```

<body>
    <table align="center" width="100%">
        <tr align="center" bgcolor="#99ccff">
            <td width="7%">아이디</td>
            <td width="7%">비밀번호</td>
            <td width="5%">이름</td>
            <td width="11%">이메일</td>
            <td width="5%">가입일</td>
        </tr>
        <%
            if( membersList.size()==0) {
        %>
            ...

```

3. 실행 결과는 자바 빈을 사용했을 때와 같습니다.

4. 이번에는 회원 가입창에서 전달된 회원 정보를 `<jsp:setProperty>` 액션 태그를 이용해 유즈빈의 속성에 좀 더 편리하게 설정하는 방법을 알아보겠습니다.

먼저 회원 가입창의 각 입력창의 매개변수 이름을 자바 빈 속성 이름과 동일하게 설정합니다.

▼ 그림 13-25 회원 가입창의 각 매개변수 이름

```

5  <table align="center">
6      <tr>
7          <td width="200"><p align="right">아이디</td>
8          <td width="400"><input type="text" name="id"></td>
9      </tr>
10     <tr>
11         <td width="200"><p align="right">비밀번호</td>
12         <td width="400"><input type="password" name="pwd"></td>
13     </tr>
14     <tr>
15         <td width="200"><p align="right">이름</td>
16         <td width="400"><p><input type="text" name="name"></td>
17     </tr>
18     <tr>
19         <td width="200"><p align="right">이메일</td>
20         <td width="400"><p><input type="text" name="email"></td>
21     </tr>
22     <tr>
23         <td width="200"><p>&ampnbsp</p></td>
24         <td width="400">
25             <input type="submit" value="가입하기"/>
26             <input type="reset" value="다시입력"/>
27         </td>
28     </tr>
29 
```

MemberBean의 속성 이름과
동일하게 설정합니다.

5. member4.jsp를 다음과 같이 작성합니다. <jsp:setProperty> 액션 태그의 param 속성을 이용해 회원가입창에서 전달된 매개변수 이름으로 해당 useBean의 속성에 자동으로 값을 설정합니다.

코드 13-14 pro13/WebContent/member4.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
       import="java.util.* , sec03.ex02.*"    pageEncoding="UTF-8"%>
<%
request.setCharacterEncoding("UTF-8");
%>
<jsp:useBean id="m" class="sec01.ex01.MemberBean" scope="page"/>
<jsp:setProperty name="m" property="id" param="id" />
<jsp:setProperty name="m" property="pwd" param="pwd" />
<jsp:setProperty name="m" property="name" param="name"/>
<jsp:setProperty name="m" property="email" param="email" />
<%
MemberDAO memberDAO=new MemberDAO();
memberDAO.addMember(m);
List membersList = memberDAO.listMembers();
%>
...
...
```

회원 가입창에서 전달된 매개변수 이름과 속성 이름이 같으면 같은 이름으로 값을 설정합니다

6. member5.jsp를 다음과 같이 작성합니다. <jsp:setProperty> 액션 태그에 param 속성을 생략하고 property 속성 이름만 지정하면 회원가입창에서 전달받은 매개변수 중 같은 매개변수 값을 자동으로 설정해 줍니다.

코드 13-15 pro13/WebContent/member5.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
       import="java.util.* , sec01.ex01.*"    pageEncoding="UTF-8"%>
<%
request.setCharacterEncoding("UTF-8");
%>
<jsp:useBean id="m" class="sec01.ex01.MemberBean" scope="page"/>
<jsp:setProperty name="m" property="id" />
<jsp:setProperty name="m" property="pwd" />
<jsp:setProperty name="m" property="name" />
<jsp:setProperty name="m" property="email" />
<%
MemberDAO memberDAO=new MemberDAO();
memberDAO.addMember(m);
List membersList = memberDAO.listMembers();
%>
...
...
```

회원 가입창에서 전달받은 매개변수 이름이 일치하는 useBean 속성에 자동으로 값을 설정해 줍니다.

7. member6.jsp를 다음과 같이 작성합니다. <jsp:setProperty> 액션 태그의 property 속성에 *를 지정하면 JSP 페이지에서 자동으로 매개변수 이름과 속성 이름을 비교한 후 같은 이름의 속성 이름에 전달된 값을 알아서 설정해 줍니다. 따라서 JSP나 HTML 페이지에서 전달된 데이터를 처리할 때 미리 매개변수 이름과 속성 이름을 동일하게 설정하여 편리하게 사용할 수 있습니다.

코드 13-16 pro13/WebContent/member6.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   import="java.util.* , sec01.ex01.*"
   pageEncoding="UTF-8"%>
<%
   request.setCharacterEncoding("UTF-8");
%>

<jsp:useBean id="m" class="sec01.ex01.MemberBean" scope="page"/>
<jsp:setProperty name="m" property="*" /> •————— 전송된 매개변수 이름과 빈 속성을 비교한
   후 동일한 빈에 값을 자동으로 설정합니다.

<%
   MemberDAO memberDAO=new MemberDAO();
   memberDAO.addMember(m);
   List membersList = memberDAO.listMembers();
%>
...
...
```

8. 마지막으로 member7.jsp를 다음과 같이 작성합니다. 회원가입장에서 전달받은 회원 정보를 일단 <jsp:setProperty> 액션 태그를 이용해 useBean 속성에 저장한 후 <jsp:getProperty> 액션 태그를 이용해 useBean의 속성에 접근하여 값을 출력합니다.

코드 13-17 pro13/WebContent/member7.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   import="java.util.* , sec01.ex01.*" pageEncoding="UTF-8"%>
<%
   request.setCharacterEncoding("UTF-8");
%>
<jsp:useBean id="m" class="sec01.ex01.MemberBean" scope="page"/>
<jsp:setProperty name="m" property="*" />
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>회원 목록창</title>
</head>
```

```

<body>
<table align="center" width="100%">
  <tr align="center" bgcolor="#99ccff">
    <td width="7%">아이디</td>
    <td width="7%">비밀번호</td>
    <td width="5%">이름</td>
    <td width="11%">이메일</td>
  </tr>
  <tr align="center">
    <td>
      <jsp:getProperty name="m" property="id" />
    </td>
    <td>
      <jsp:getProperty name="m" property="pwd" />
    </td>
    <td>
      <jsp:getProperty name="m" property="name" />
    </td>
    <td>
      <jsp:getProperty name="m" property="email" />
    </td>
  </tr>
  <tr height="1" bgcolor="#99ccff">
    <td colspan="5"></td>
  </tr>
</table>
</body>
</html>

```

→ 〈jsp:getProperty〉 태그를 이용해
useBean 속성 값에 접근합니다.

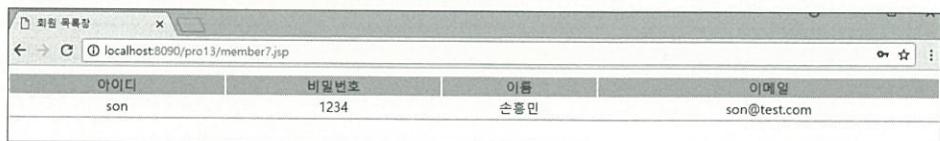
9. <http://localhost:8090/pro13/memberForm.html>로 요청하여 회원 가입창에 회원 정보를 입력한 후 가입하기를 클릭합니다.

▼ 그림 13-26 회원 가입창에 회원 정보 입력 후 가입하기 클릭

The screenshot shows a registration form titled "회원 가입창". It contains four input fields: "아이디" (ID) with value "son", "비밀번호" (Password) with value "....", "이름" (Name) with value "손흥민", and "이메일" (Email) with value "son@test.com". Below the inputs are two buttons: "가입하기" (Register) and "다시입력" (Re-enter). The "가입하기" button is highlighted with a purple border.

10. 그러면 전송된 회원 정보를 <jsp:getProperty> 액션 태그를 이용해 출력합니다.

▼ 그림 13-27 <jsp:getProperty> 태그 이용해 회원 정보 출력



A screenshot of a web browser window titled "회원 목록장". The address bar shows "localhost:8090/pro13/member7.jsp". The page displays a table with four columns: 아이디, 비밀번호, 이름, and 이메일. The data is as follows:

아이디	비밀번호	이름	이메일
son	1234	손흥민	son@test.com

마지막으로 <jsp:param> 액션 태그는 <include> 액션 태그와 <forward> 액션 태그 사용 시 다른 JSP로 매개변수 값을 전송할 때 사용합니다.

the first time. I am very excited about this opportunity to learn more about the
culture and history of Japan. I am looking forward to experiencing the traditional
ways of life and meeting new people. I am also interested in learning about the
modern aspects of Japanese society, such as technology and business. I am
hoping to gain a better understanding of the Japanese way of thinking and how it
influences their daily lives. I am also looking forward to trying new foods and
experiencing different customs and traditions. I am excited to be part of this
program and I am grateful for the chance to travel and learn.

14 장

표현 언어와 JSTL

- 14.1 표현 언어란?
- 14.2 표현 언어 내장 객체(내장 변수)
- 14.3 표현 언어로 바인딩 속성 출력하기
- 14.4 커스텀 태그
- 14.5 JSP 표준 태그 라이브러리(JSTL)
- 14.6 Core 태그 라이브러리 사용하기
- 14.7 Core 태그 라이브러리 실습 예제
- 14.8 다국어 태그 라이브러리 사용하기
- 14.9 한글을 아스키 코드로 변환하기
- 14.10 포매팅 태그 라이브러리 사용하기
- 14.11 문자열 처리 함수 사용하기
- 14.12 표현 언어와 JSTL을 이용한 회원 관리 실습

14.1 표현 언어란?

JSP의 발전 과정을 한 번 더 정리해 보겠습니다. 초기에는 HTML 태그를 중심으로 자바를 이용해 화면을 구현했으나 화면에 대한 요구 사항이 복잡해지면서 자바 코드를 대체하는 액션 태그가 등장했습니다. 이어서 JSP 2.0에서는 페이지 안에서 복잡한 자바 코드를 제거하는 쪽으로 발전했습니다. 디자이너 입장에서는 JSP 페이지 안에 복잡한 자바 코드가 있으면 화면 작업을 하기가 어려웠기 때문입니다. 그리고 현재 JSP 페이지는 스크립트 요소보다는 **표현 언어**(EL, Expression Language)와 JSTL(JSP Standard Tag Library, JSP 표준 태그 라이브러리)¹을 사용해서 구현합니다.

표현 언어는 자바 코드가 들어가는 표현식을 좀 더 편리하게 사용하기 위해 JSP 2.0부터 도입된 데이터 출력 기능입니다. 표현식에는 자바 변수나 여러 가지 자바 코드로 된 식을 사용하는데, 표현식의 자바 코드가 복잡해짐에 따라 JSP 2.0부터는 자바 코드로 출력하는 표현식을 대체하기 위해 표현 언어라는 것이 등장했습니다.

그림 14-1은 JSP에서 표현식을 사용해 회원 정보를 출력하는 코드입니다.



JSP 페이지에서 표현 언어를 사용하려면 페이지 디렉티브 태그의 속성인 `isELIgnored`를 `false`로 설정해야 합니다.

▼ 그림 14-1 JSP 페이지에서 표현식을 이용한 값 출력

```

18<table border="1" align="center">
19  <tr align="center" bgcolor="#99ccff">
20    <td width="20%"><b>아이디</b></td>
21    <td width="20%"><b>비밀번호</b></td>
22    <td width="20%"><b>이름</b></td>
23    <td width="20%"><b>이메일</b></td>
24  </tr>
25  <tr align="center">
26    <td>%=id %</td>
27    <td>%=pwd%</td>
28    <td>%=name %</td>
29    <td>%=email %</td>
30  </tr>

```

¹ JSP 페이지에서 일반적인 핵심 기능을 캡슐화하여 제공하는 JSP 태그 컬렉션을 의미합니다.

표현 언어의 특징은 다음과 같습니다.

- 기존 표현식보다 편리하게 값을 출력합니다.
- 변수와 여러 가지 연산자를 포함할 수 있습니다.
- JSP의 내장 객체에 저장된 속성 및 자바의 빈 속성도 표현 언어에서 출력할 수 있습니다.
- 표현 언어 자체 내장 객체도 제공됩니다.
- JSP 페이지 생성 시 기본 설정은 표현 언어를 사용할 수 없습니다.
- 페이지 디렉티브 태그에서는 반드시 `isELIgnored=false`로 설정해야 합니다.

다음은 표현 언어의 형식입니다.

`${표현식 or 값}`

14.1.1 표현 언어에서 사용되는 자료형과 연산자

이번에는 표현 언어에서 다루는 자료형과 연산자에 대해 알아보겠습니다.

표 14-1에 표현 언어에서 사용되는 여러 가지 자료형을 정리했습니다.

▼ 표 14-1 표현 언어에서 사용되는 자료형

자료형	설명
불	true와 false 값을 가집니다.
정수	0~9로 이루어진 값을 가지고 음수인 경우 마이너스(−)가 붙습니다.
실수	소수점(.)을 사용할 수 있고, 1.4e5와 같이 지수형으로 표현할 수 있습니다.
문자열	따옴표('hello'나 "hello")와 같이 사용됩니다.
널	null을 가집니다.

그리고 표 14-2에는 표현 언어에서 사용되는 여러 가지 연산자의 기능을 정리했습니다.

▼ 표 14-2 표현 언어의 여러 가지 연산자

연산자 종류	연산자	설명
산술 연산자	+	덧셈
	-	뺄셈
	*	곱셈
	/ 또는 div	나눗셈
	% 또는 mod	나머지
비교 연산자	== 또는 eq	두 값이 같은지 비교합니다.
	!= 또는 ne	두 값이 다른지 비교합니다.
	< 또는 lt	값이 다른 값보다 작은지 비교합니다.
	> 또는 gt	값이 다른 값보다 큰지 비교합니다.
	<= 또는 le	값이 다른 값보다 작거나 같은지 비교합니다.
	>= 또는 ge	값이 다른 값보다 크거나 같은지 비교합니다.
논리 연산자	&& 또는 and	논리곱 연산을 합니다.
	또는 or	논리합 연산을 합니다.
	! 또는 not	부정 연산을 합니다.
empty 연산자	empty <값>	<값>이 null이거나 빈 문자열이면 true를 반환합니다.
조건 연산자	<수식> ? <값1> : <값2>	<수식>의 결괏값이 true이면 <값1>을 반환하고, false이면 <값2>를 반환합니다.

14.1.2 JSP에서 표현 언어 사용 실습

JSP에서 표현 언어를 사용해 여러 가지 데이터를 출력해 보겠습니다.

1. 다음과 같이 실습 파일 elTest1.jsp를 준비합니다.

▼ 그림 14-2 실습 파일 위치



2. 다음과 같이 elTest1.jsp를 작성합니다. 문자열과 숫자를 더하면 자동으로 숫자로 변환해 합을 구하고, null과 숫자를 더하면 null을 0으로 인식하도록 합니다.

코드 14-1 pro14/WebContent/elTest1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    isELIgnored="false" %>

<html>
<head>
    <meta charset="UTF-8">
    <title>표현 언어에서 사용되는 데이터들</title>
</head>
<body>
    <h1>표현 언어로 여러 가지 데이터 출력하기</h1>
    <h1>
        ${100}: ${100}<br>
        ${"안녕하세요"}: ${"안녕하세요"}<br>
        ${10+1}: ${10+1}<br>           문자형 문자열과 실제 숫자를 더하면 문자
                                         열을 자동으로 숫자로 변환하여 더합니다.
        ${"10"+1} : ${"10"+1 }<br>   null과 10을 더하면 10이 됩니다.
        <%-- ${null+10 }: ${null+10 }<br> --%>
        <%-- ${"안녕"+11 }: ${"안녕"+11 }<br> --%>
        <%-- ${"hello"+"world"}:${"hello"+"world"}<br> --%>
    </h1>
</body>
</html>
```

문자형 문자열과 실제 숫자를 더하면 문자

열을 자동으로 숫자로 변환하여 더합니다.

null과 10을 더하면 10이 됩니다.

문자열끼리는 더할 수 없습니다.

문자열과 숫자는 더할 수 없습니다.

3. <http://localhost:8090/pro14/elTest1.jsp>로 요청하여 실행 결과를 확인합니다.

▼ 그림 14-3 실행 결과



지금부터는 표현 언어의 여러 가지 연산자들을 사용해 몇 가지 예제를 실습해 보겠습니다.

14.1.3 표현 언어의 산술 연산자

표현 언어에서는 사칙 연산자를 어떻게 사용하는지 알아보겠습니다.

1. 실습 파일 MemberBean.java와 elTest2~5.jsp를 준비합니다.

▼ 그림 14-4 실습 파일 위치



2. elTest2.jsp를 다음과 같이 작성합니다. 나누기 연산을 하려면 div를 사용하고, 나머지 연산을 하려면 mod를 사용하면 됩니다.

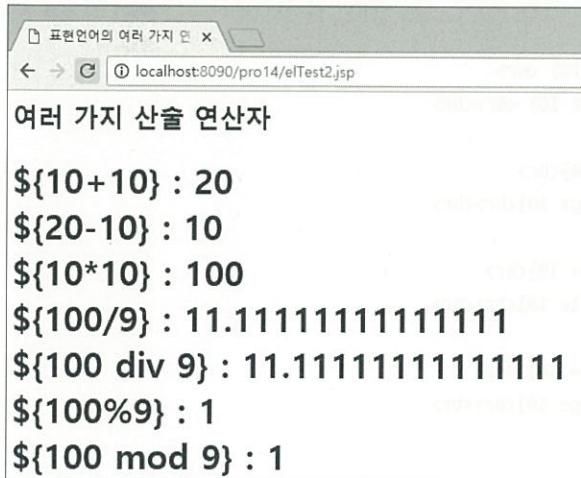
코드 14-2 pro14/WebContent/elTest2.jsp

```
<body>
<h2>여러 가지 산술 연산자</h2>
<h1>
    \${10+10} : \${10+10} <br>
    \${20-10} : \${20-10} <br>
    \${10*10} : \${10*10} <br>
    \${100/9} : \${100/9} <br>           div를 사용해 나누기 연산을 합니다.
    \${100 div 9} : \${100 div 9} <br>
    \${100%9} : \${100%9}<br>
    \${100 mod 9} : \${100 mod 9}<br>
</h1>
</html>
```

mod를 사용해 나머지를 구합니다.

3. <http://localhost:8090/pro14/elTest2.jsp>로 요청합니다. 표현 언어에서 나누기 연산 시 나누어지지 않으면 소수점 이하까지 표시합니다.

▼ 그림 14-5 실행 결과



14.1.4 표현 언어의 비교 연산자

이번에는 표현 어어에서 비교 연사자를 사용해 보겠습니다.

- 값이 같은지 비교할 때는 == 또는 eq 연산자를 사용합니다.
 - 값이 같지 않은지 비교할 때는 != 또는 ne 연산자를 사용합니다.
 - 대소 비교 시 >와 < 연산자 그리고 gt와 lt도 각각 연산자로 사용할 수 있습니다.
 - 대소 및 동등 비교를 동시에 할 때는 >=와 <= 연산자 그리고 ge와 le도 각각 연산자로 사용할 수 있습니다.

- ### 1. 다음과 같이 elTest3.jsp를 작성합니다.

코드 14-3 pro14\WebContent\elTest3.jsp

```
<body>
  <h2>여러 가지 비교 연산자</h2>
  <h3>
    \${10==10} : \${10==10} <br>
    \${10 eq 10} : \${10 eq 10} <br><br>
    \${"hello"]=="hello"} : \${"hello"]=="hello"} <br> ━━━━━━━━━━ 문장열이 서로 같은지 비교할 때는
    \${"hello" eq "hello"} : \${"hello" eq "hello"} <br><br> ==나 eq를 연산자로 사용합니다.
```

```

\${20!=10} : ${20!=10}<br>
\${20 ne 10} : ${20 ne 10}<br><br>

\${"hello"!="apple"} : ${"hello"!="apple"} <br>
\${"hello" ne "apple"} : ${"hello" ne "apple"} <br><br> 문자열이 서로 다른지 비교할 때는 !=나 ne를 연산자로 사용합니다.

\${10 < 10} : ${10 < 10} <br>
\${10 lt 10} : ${10 lt 10} <br><br>

\${100>10} : ${100 > 10}<br>
\${100 gt 10} : ${100 gt 10}<br><br>

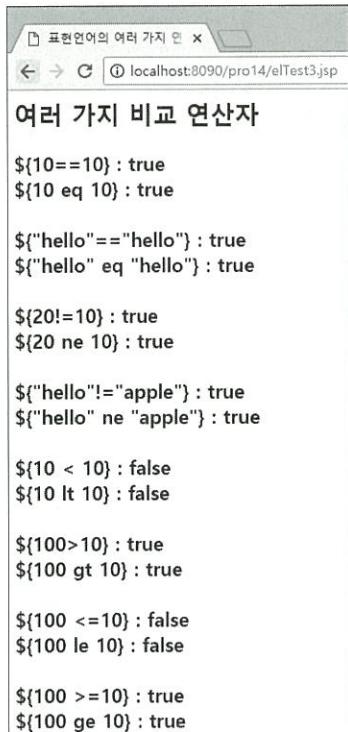
\${100 <=10} : ${100 <= 10}<br>
\${100 le 10} : ${100 le 10}<br><br>

\${100 >=10} : ${100 >= 10}<br>
\${100 ge 10} : ${100 ge 10}<br><br>
</h3>
</body>

```

2. <http://localhost:8090/pro14/elTest3.jsp>로 요청하여 실행 결과를 확인합니다.

▼ 그림 14-6 실행 결과



The screenshot shows a browser window with the URL localhost:8090/pro14/elTest3.jsp. The page title is "여러 가지 비교 연산자". The content displays the results of various EL expressions:

- `\${10==10}` : true
- `\${10 eq 10}` : true
- `\${"hello"=="hello"}` : true
- `\${"hello" eq "hello"}` : true
- `\${20!=10}` : true
- `\${20 ne 10}` : true
- `\${"hello"!="apple"}` : true
- `\${"hello" ne "apple"}` : true
- `\${10 < 10}` : false
- `\${10 lt 10}` : false
- `\${100>10}` : true
- `\${100 gt 10}` : true
- `\${100 <=10}` : false
- `\${100 le 10}` : false
- `\${100 >=10}` : true
- `\${100 ge 10}` : true

14.1.5 표현 언어의 논리 연산자

이번에는 논리 연산자를 알아보겠습니다.

- `&&` 연산자나 `and` 연산자는 논리곱 연산을 합니다.
- `||` 연산자나 `or` 연산자는 논리합 연산을 합니다.
- `!` 연산자나 `not` 연산자는 반대의 결과를 출력합니다.

1. 다음과 같이 `elTest4.jsp`를 작성합니다.

코드 14-4 pro14/WebContent/elTest4.jsp

```
<body>
    <h2>여러 가지 논리연산자</h2>
    <h2>
        \${(10==10) && (20==20)} : ${(10==10)&&(20==20)} <br>
        \${(10==10) and (20!=20)} : ${(10==10) and (20!=20)} <br><br>
        \${(10==10) || (20!=30)} : ${(10==10)|| (20!=30)} <br>
        \${(10!=10) or (20!=20)} : ${(10!=10) or (20!=20)} <br><br>
        \${!(20==10)} : ${!(20==10)}.br>
        \${not (20==10)} : ${not (20==10)}<br><br>
        \${!(20!=10)} : ${!(20!=10)}.br>
        \${not(20!=10)} : ${not(20!=10)}<br><br>
    </h2>
</body>
```

연산자 양쪽 값이 true일 때만 true를 반환합니다.

연산자 양쪽 값 중 하나라도 true면 true를 반환합니다.

비교 연산자의 결과값이 false이므로 true를 출력합니다.

비교 연산자의 결과값이 true이므로 false를 출력합니다.

2. <http://localhost:8090/pro14/elTest4.jsp>로 요청하여 실행 결과를 확인합니다.

▼ 그림 14-7 실행 결과

The screenshot shows a browser window with the URL localhost:8090/pro14/elTest4.jsp. The page title is "여러가지 논리연산자". The content displays several EL expressions and their evaluated results:

- `$(10==10) && (20==20)` : true
- `$(10==10) and (20!=20)` : false
- `$(10==10) || (20!=30)` : true
- `$(10!=10) or (20!=20)` : false
- `$(!(20==10))` : true
- `$not (20==10)` : true
- `$(!(20!=10))` : false
- `$not(20!=10)` : false

14.1.6 표현 언어의 empty 연산자

empty 연산자는 자바 빈의 속성이 값으로 설정되었는지 또는 List, Map 같은 저장 객체에 값(객체)이 존재하는지를 판단하는 연산자입니다.

1. elTest5.jsp를 다음과 같이 작성합니다. <useBean> 액션 태그로 생성한 빈 m1은 생성 후 name 속성에 값을 설정했기 때문에 empty 연산자를 적용하면 false를 반환합니다. <useBean> 액션 태그로 생성한 빈 m2는 생성 후 아무 값도 저장하지 않았기 때문에 empty 연산자를 적용하면 true를 반환합니다.

코드 14-5 pro14/WebContent/elTest5.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    import="java.util.*"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<jsp:useBean id="m1" class="sec01.ex01.MemberBean" scope="page" />
<jsp:setProperty name="m1" property="name" value="이순신"/>
<jsp:useBean id="m2" class="java.util.ArrayList" scope="page" />
<html>
```

유즈빈(useBean)을 생성합니다.

빈의 name 속성에 값을 설정합니다.

ArrayList 객체를 빈으로 생성합니다.

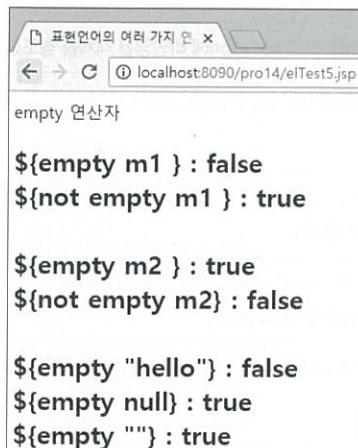
```

<head>
    <meta charset="UTF-8">
    <title>표현 언어의 여러 가지 연산자들</title>
</head>
<body>
    empty 연산자
    <h2>
        \${empty m1} : ${empty m1} <br>           m1의 name 속성에 값이 설정되어 있으므로 false를 반환합니다.
        \${not empty m1} : ${not empty m1} <br><br> true를 반환합니다.
        \${empty m2} : ${empty m2} <br>           ArrayList 객체인 m2는 비어 있으므로 true를 반환합니다.
        \${not empty m2} : ${not empty m2} <br><br> false를 반환합니다.
        \${empty "hello"} : ${empty "hello"} <br>   문자열에 대해 false를 반환합니다.
        \${empty null} : ${empty null} <br>          null은 true를 반환합니다.
        \${empty ""} : ${empty ""} <br>
    </h2>
</body>
</html>

```

2. <http://localhost:8090/pro14/elTest5.jsp>로 요청하여 실행 결과를 확인합니다.

▼ 그림 14-8 실행 결과



14.2

표현 언어 내장 객체(내장 변수)



이번에는 표현 언어에서 제공하는 여러 가지 내장 객체에 대해 알아보겠습니다.

JSP는 기본적으로 내장 객체들을 제공하지만 이 객체들은 표현식에서만 사용할 수 있습니다. 따라서 표현 언어에서는 따로 내장 객체들을 제공합니다. 표현 언어에서 제공하는 내장 객체들은 \${} 안에서만 사용할 수 있습니다.

14.2.1 표현 언어에서 제공하는 내장 객체의 종류와 기능

표현 언어에서 제공하는 여러 가지 내장 객체들은 표 14-3과 같습니다.

▼ 표 14-3 표현 언어에서 제공하는 여러 가지 내장 객체

구분	내장 객체	설명
스코프	pageScope	JSP의 page와 같은 기능을 하고 page 영역에 바인딩된 객체를 참조합니다.
	requestScope	JSP의 request와 같은 기능을 하고 request에 바인딩된 객체를 참조합니다.
	sessionScope	JSP의 session과 같은 기능을 하고 session에 바인딩된 객체를 참조합니다.
	applicationScope	JSP의 application과 같은 기능을 하고 application에 바인딩된 객체를 참조합니다.
요청 매개변수	param	request.getParameter() 메서드를 호출한 것과 같으며 한 개의 값을 전달하는 요청 매개변수를 처리합니다.
	paramValues	request.getParameterValues() 메서드를 호출한 것과 같으며 여러 개의 값을 전달하는 요청 매개변수를 처리합니다.
헤더 값	header	request.getHeader() 메서드를 호출한 것과 같으며 요청 헤더 이름의 정보를 단일 값으로 반환합니다.
	headerValues	request.getHeader() 메서드를 호출한 것과 같으며 요청 헤더 이름의 정보를 배열로 반환합니다.
쿠키 값	Cookies	쿠키 이름의 값을 반환합니다.
JSP 내용	pageContext	pageContext 객체를 참조할 때 사용합니다.
초기 매개변수	initParam	컨텍스트의 초기화 매개변수 이름의 값을 반환합니다.

14.2.2 param 내장 객체 사용 실습

회원 가입창에서 회원 정보를 입력하고 JSP로 전송하면 `getParameter()` 메서드를 이용하지 않고 `param` 내장 객체를 이용해 전송된 회원 정보를 출력하는 예제를 살펴보겠습니다.

1. WebContent 폴더 하위에 test01 폴더를 생성한 후 다음과 같이 여러 개의 JSP 파일을 준비합니다.

▼ 그림 14-9 실습 파일 위치



2. memberForm.jsp를 다음과 같이 작성합니다. 회원 가입창에서 회원 정보를 입력하고 member1.jsp로 전송합니다.

코드 14-6 pro14/WebContent/test01/memberForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>회원 가입창</title>
</head>
<body>
  <form method="post" action="member1.jsp">———— 실습 파일에서 이 부분을 member1.jsp로 수정합니다.
    <h1 style="text-align:center">회원 가입창</h1>———— 이 부분은 계속 고쳐가면서 진행하겠습니다.
```

실습 파일에서 이 부분을 member1.jsp로 수정합니다.
이 부분은 계속 고쳐가면서 진행하겠습니다.

```
<table align="center">
<tr>
    <td width="200">
        <p align="right">아이디
    </td>
    <td width="400"><input type="text" name="id"></td>
</tr>
<tr>
    <td width="200">
        <p align="right">비밀번호
    </td>
    <td width="400"><input type="password" name="pwd"></td>
</tr>
<tr>
    <td width="200">
        <p align="right">이름
    </td>
    <td width="400">
        <p><input type="text" name="name">
    </td>
</tr>
<tr>
    <td width="200">
        <p align="right">이메일
    </td>
    <td width="400">
        <p><input type="text" name="email">
    </td>
</tr>
<tr>
    <td width="200">
        <p>&ampnbsp</p>
    </td>
    <td width="400">
        <input type="submit" value="가입하기">
        <input type="reset" value="다시입력">
    </td>
</tr>
</table>
</form>
</body>
</html>
```

3. member1.jsp를 다음과 같이 작성합니다. 첫 번째 방법은 전송된 회원 정보를 getParameter() 메서드를 이용해 출력합니다. 두 번째 방법은 param 내장 객체를 이용해 전송된 매개변수 이름으로 바로 회원 정보를 출력합니다.

코드 14-7 pro14/WebContent/test01/member1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
isELIgnored="false" %>
<%
request.setCharacterEncoding("UTF-8");
String id=request.getParameter("id");
String pwd=request.getParameter("pwd");
String name= request.getParameter("name");
String email= request.getParameter("email");
%>

<html>
<head>
<meta charset="UTF-8">
<title>회원 정보 출력창</title>
</head>
<body>
<table border="1" align="center" >
<tr align="center" bgcolor="#99ccff">
<td width="20%"><b>아이디</b></td>
<td width="20%"><b>비밀번호</b></td>
<td width="20%" ><b>이름</b></td>
<td width="20%"><b>이메일</b></td>
</tr>
<tr align=center>
<td><%=id %> </td>
<td><%=pwd%> </td>
<td><%=name %> </td>
<td><%=email %> </td>
</tr>
<tr align=center>
<td>${param.id } </td>
<td>${param.pwd } </td>
<td>${param.name } </td>
<td>${param.email }</td>
</tr>
</table>
</body>
</html>
```

회원 정보를 표시하기 전에 한글 인코딩을 설정합니다.

표현식으로 출력하기 위해 getParameter() 메서드를 이용해 회원 정보를 가져옵니다.

getParameter()로 가져온 회원 정보를 표현식으로 출력합니다.

param 객체를 이용해 getParameter() 메서드를 이용하지 않고 바로 회원 정보를 출력합니다.

4. <http://localhost:8090/pro14/test01/memberForm.jsp>로 요청하여 회원 정보를 입력하고 가입하기를 클릭합니다.

▼ 그림 14-10 회원가입창에서 회원 정보 입력 후 가입하기 클릭

회원가입창

아이디 lee

비밀번호 ...

이름 이순신

이메일 lee@test.com

가입하기 다시입력

5. 실행 결과를 보면 회원 정보가 두 번 출력된 것을 알 수 있습니다. 첫 번째 회원 정보는 `getParameter()` 메서드로 가져온 후 출력한 것이고, 두 번째 회원 정보는 `param` 내장 객체로 출력한 결과입니다.

▼ 그림 14-11 회원 정보 출력

아이디	비밀번호	이름	이메일
lee	1234	이순신	lee@test.com
lee	1234	이순신	lee@test.com

따라서 `param` 내장 객체를 사용하면 굳이 전송된 매개변수를 `getParameter()` 메서드를 이용하지 않고 바로 매개변수 이름으로 접근해서 값을 얻을 수 있습니다.

14.2.3 requestScope 사용 실습

이번에는 `request` 객체와 동일한 기능을 하는 `requestScope`를 사용해 보겠습니다.

1. 회원가입창인 `memberForm.jsp`의 `action` 속성을 `forward.jsp`로 수정하고 회원 정보를 입력한 후 `forward.jsp`로 전송합니다.

```
<form method="post" action="forward.jsp">
```



실습 파일에서 forward.jsp 부분은 계속 고쳐가면서 진행하겠습니다.

- forward.jsp를 다음과 같이 작성합니다. 회원가입창의 request 객체에 setAttribute() 메서드를 이용해 address를 바인딩한 후 다시 member2.jsp로 포워딩합니다.

코드 14-8 pro14/WebContent/test01/forward.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding("utf-8");
    request.setAttribute("address", "서울시 강남구"); •———— 회원 가입창의 request에 대해 다시 주소 정보를 바인딩합니다.
%>

<html>
<head>
    <meta charset="UTF-8">
    <title>forward</title>
</head>
<body>
    <jsp:forward page="member2.jsp"></jsp:forward> •———— member2.jsp로 포워딩합니다.
</body>
</html>
```

- member2.jsp를 다음과 같이 작성합니다. requestScope를 이용해 바인딩된 address에 접근해서 주소를 출력합니다.

코드 14-9 pro14/WebContent/test01/member2.jsp

```
...
<body>
    <table border="1" align="center" >
        <tr align="center" bgcolor="#99ccff">
            <td width="20%"><b>아이디</b></td>
            <td width="20%"><b>비밀번호</b></td>
            <td width="20%"><b>이름</b></td>
            <td width="20%"><b>이메일</b></td>
            <td width="20%"><b>주소</b></td>
        </tr>
        <tr align=center>
            <td>${param.id } </td>
```

```

<td>${param.pwd} </td>
<td>${param.name } </td>
<td>${param.email }</td>
<td>${requestScope.address}</td>
</tr>
</table>
</body>

```

requestScope를 이용해 바인딩된 주소 정보를 출력합니다.

4. <http://localhost:8090/pro14/test01/memberForm.jsp>로 다시 요청하여 회원 가입창에 회원 정보를 입력하고 가입하기를 클릭합니다.

▼ 그림 14-12 회원 가입창에서 회원 정보 입력 후 가입하기 클릭

회원 가입창

아이디	cha
비밀번호
이름	차범근
이메일	cha@test.com
<input type="button" value="가입하기"/> <input type="button" value="다시입력"/>	

5. 회원 가입창에서 회원 정보와 함께 forward.jsp에서 request 객체에 바인딩한 주소도 출력합니다.

▼ 그림 14-13 주소와 함께 회원 정보 출력

/test01/forward.jsp

아이디	비밀번호	이름	이메일	주소
cha	1234	차범근	cha@test.com	서울시 강남구

requestScope를 이용하면 request 객체에 바인딩된 데이터에 접근할 수 있습니다. 마찬가지로 session이나 application 객체에 바인딩된 데이터는 sessionScope나 applicationScope로 접근할 수 있습니다.

14.2.4 pageContext 객체 사용 실습

pageContext 객체는 javax.servlet.jsp.PageContext 클래스를 상속해 웹 컨테이너가 JSP 실행 시 자동으로 생성해서 제공하는 내장 객체입니다. 이번에는 pageContext 객체의 편리한 기능을 사용해 보겠습니다.

<a> 태그를 이용해 다른 서블릿이나 JSP를 요청하는 방법은 다음의 두 가지입니다.

첫 번째는 컨텍스트 이름(pro14)을 직접 입력하는 방법입니다.

```
<a href="/pro14/test01/memberForm.jsp"> 회원 가입하기</a>
```

두 번째는 getContextPath() 메서드를 이용해 컨텍스트 이름을 가져오는 방법입니다.

```
<a href="<%=request.getContextPath()%>/test01/memberForm.jsp">회원 가입하기</a>
```

그런데 첫 번째 방법은 컨텍스트 이름(pro14)이 바뀌면 일일이 찾아서 수정해야 한다는 단점이 있고, 두 번째 방법은 자바 코드가 사용되므로 화면 작업이 복잡해진다는 단점이 있습니다. 그러나 pageContext 객체의 속성인 request의 contextPath 속성을 이용하면 쉽게 컨텍스트 이름을 가져올 수 있습니다.

1. 다음과 같이 login.jsp를 작성합니다.

코드 14-10 pro14/WebContent/test01/login.jsp

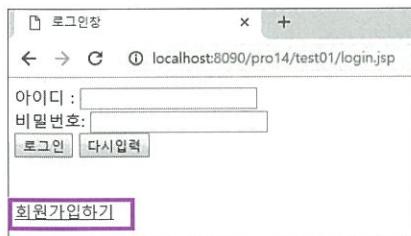
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
isELIgnored="false" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>로그인창</title>
</head>
<body>
<form action="result.jsp">
    아이디 : <input type="text" size=20 /><br>
    비밀번호: <input type="password" size=20 /><br>
    <input type="submit" value="로그인" /> <input type="reset" value="다시입력" />
</form>
<br><br>
<!-- &lt;a href="http://localhost:8090/pro14/test01/memberForm.jsp"&gt;회원가입하기&lt;/a&gt; --&gt;
&lt;%--&gt;
&lt;a href="<%=request.getContextPath()%>/test01/memberForm.jsp">회원가입하기</a>
```

```
--%>
<a href="${pageContext.request.contextPath}/test01/memberForm.jsp">회원가입하기</a>
</body>
```

자바 코드를 사용하지 않고 pageContext의 속성인 request
하위의 contextPath 속성으로 컨텍스트 이름을 가져옵니다.

2. <http://localhost:8090/pro14/test01/login.jsp>로 로그인창을 요청하여 회원가입하기를 클릭합니다.

▼ 그림 14-14 로그인창에서 회원가입하기 클릭



3. 회원가입창으로 이동합니다.

▼ 그림 14-15 회원가입창으로 이동

14.2.5 빈 사용 실습

표현 언어에서 빈 속성에 접근하는 방법을 알아보겠습니다. 빈의 속성에 접근할 때는 다음과 같은 형식을 사용합니다.

`${빈이름.속성이름}`

그럼 빈에 회원 정보를 저장한 후 표현 언어를 이용해 빈의 회원 정보를 출력해 보겠습니다.

1. memberForm.jsp의 action 값을 member3.jsp로 수정합니다.
2. 다음과 같이 member3.jsp를 작성합니다. 표현 언어에서는 getter를 사용하지 않고, 바로 빈 id 다음에 .(마침표) 연산자를 사용하여 속성에 바로 접근할 수 있습니다.

코드 14-11 pro14/WebContent/test01/member3.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<%
    request.setCharacterEncoding("UTF-8");
%>
<jsp:useBean id="m" class="sec01.ex01.MemberBean" /> ────────── 회원 정보를 저장할 빈을 생성합니다.
<jsp:setProperty name="m" property="*" /> ────────── 전송된 회원 정보를 빈의 속성에 설정합니다.

<meta charset="UTF-8">
<html>
<head>
    <title>회원 정보 출력창</title>
</head>
<body>
    <table align="center" border="1">
        <tr align="center" bgcolor="#99ccff">
            <td width="20%"><b>아이디</b></td>
            <td width="20%"><b>비밀번호</b></td>
            <td width="20%"><b>이름</b></td>
            <td width="20%"><b>이메일</b></td>
        </tr>
        <tr align="center">
            <td><%=m.getId() %></td> ────────── 표현식을 이용해 회원 정보를 출력합니다
            <td><%=m.getPwd() %></td>
            <td><%=m.getName() %></td>
            <td><%=m.getEmail() %></td>
        </tr>
        <tr align="center">
            <td>${m.id }</td> ────────── 빈 id와 속성 이름으로 접근해 회원 정보를 출력합니다.
            <td>${m.pwd}</td>
            <td>${m.name }</td>
            <td>${m.email }</td>
        </tr>
    </table>
</body>
</html>
```

3. <http://localhost:8090/pro14/test01/login.jsp>로 요청하여 회원가입창에서 회원 정보를 입력한 후 **가입하기**를 클릭합니다.

▼ 그림 14-16 회원가입창에서 회원 정보 입력 후 **가입하기** 클릭

회원가입창

아이디: hong

비밀번호:

이름: 홍길동

이메일: hong@test.com

[가입하기] [다시입력]

4. 빈에 저장된 회원 정보를 출력합니다.

▼ 그림 14-17 회원 정보 출력

14/test01/member3.jsp

아이디	비밀번호	이름	이메일
hong	1234	홍길동	hong@test.com
hong	1234	홍길동	hong@test.com

이처럼 표현 언어에서는 자바 코드를 사용하지 않고 바로 빈 id로 속성에 접근해 값을 출력할 수 있습니다.

14.2.6 Collection 객체 사용 실습

표현 언어에서 Collection 객체에 접근하는 방법을 알아보겠습니다. Collection 객체에 접근할 때는 다음과 같은 형식을 사용합니다.

`${Collection객체이름[index].속성이름 }`



여기서 index는 Collection에 저장된 순서를 의미합니다.

지금부터 Collection 객체 중 가장 많이 사용되는 ArrayList에 회원 정보 빈을 저장한 후 다시 출력해 보겠습니다.

1. memberForm.jsp의 action 값을 member4.jsp로 수정합니다.
2. 다음과 같이 member4.jsp를 작성합니다. 회원 가입창에서 전송된 회원 정보를 빈 m1에 저장한 후 다시 ArrayList에 저장합니다. 그리고 자바 코드로 두 번째 MemberBean 객체를 생성한 후 회원 정보를 설정하여 ArrayList에 저장합니다. 그리고 인덱스로 각 속성에 순차적으로 접근해서 ArrayList의 값을 출력합니다.

코드 14-12 pro14/WebContent/test01/member4.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   import="java.util.* , sec01.ex01.*" pageEncoding="UTF-8"
   isELIgnored="false" %>
<
request.setCharacterEncoding("UTF-8");
%>
<jsp:useBean id="m1" class="sec01.ex01.MemberBean"/>
<jsp:setProperty name="m1" property="*" /> 회원 가입창에서 전송된 회원 정보를 빈
속성에 설정합니다.
<jsp:useBean id="membersList" class="java.util.ArrayList" /> membersList로 ArrayList 객체를 생성합니다.
<
MemberBean m2 = new MemberBean("son", "1234", "손흥민", "son@test.com");
membersList.add(m1); 자바 코드로 새로운 회원 정보를 저장하는
membersList.add(m2); MemberBean 객체를 생성합니다.
%> 두 개의 MemberBean 객체를 ArrayList에
<html> 저장합니다.
<head>
<meta charset="UTF-8">
<title>회원 정보 출력창</title>
</head>
<body>
<table border=1 align="center" >
  <tr align="center" bgcolor="#99ccff">
    <td width="20%"><b>아이디</b></td>
    <td width="20%"><b>비밀번호</b></td>
    <td width="20%"><b>이름</b></td>
    <td width="20%"><b>이메일</b></td>
  </tr>
  <tr align="center">
    <td>${membersList[0].id}</td> 인덱스가 0이므로 첫 번째 회원
    <td>${membersList[0].pwd}</td> 정보를 출력합니다.
    <td>${membersList[0].name}</td>
    <td>${membersList[0].email}</td>
  </tr>
</table>
```

```

<tr align="center">
    <td>${membersList[1].id}</td>
    <td>${membersList[1].pwd}</td>
    <td>${membersList[1].name}</td>
    <td>${membersList[1].email}</td>
</tr>
</table>
</body>
</html>

```

3. <http://localhost:8090/pro14/test01/memberForm.jsp>로 요청하여 회원 정보를 입력한 후 전송합니다.

▼ 그림 14-18 회원 가입창에서 회원 정보 입력

회원 가입창

아이디	cha
비밀번호
이름	차범근
이메일	cha@test.com
<input type="button" value="가입하기"/> <input type="button" value="다시입력"/>	

4. `ArrayList`에 저장된 회원 정보를 출력합니다.

▼ 그림 14-19 회원 정보 출력

t01/member4.jsp

아이디	비밀번호	이름	이메일
cha	1234	차범근	cha@test.com
son	1234	손흥민	son@test.com

14.2.7 HashMap 사용 실습

다음은 표현 언어에서 자바 HashMap에 저장된 객체에 접근하는 방법입니다.

```
 ${HashMap객체이름.키이름}
```

HashMap에 객체를 저장한 후 다시 출력해 보겠습니다.

1. memberForm.jsp의 action 값을 member5.jsp로 수정합니다.
2. member5.jsp를 다음과 같이 작성합니다. 전송된 회원 정보를 첫 번째 빈 m1 속성에 설정합니다. <useBean> 태그를 이용해 HashMap 객체인 membersMap을 생성하고 membersMap에 회원 정보를 key/value로 저장합니다. memberMap에 ArrayList를 저장한 다음 membersMap에 key로 접근하여 value를 출력합니다.

코드 14-13 pro14/WebContent/test01/member5.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    import="java.util.*, sec01.ex01.*"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<
    request.setCharacterEncoding("UTF-8");
%>
<jsp:useBean id="m1" class="sec01.ex01.MemberBean"/>
<jsp:setProperty name="m1" property="*" />
<jsp:useBean id="membersList" class="java.util.ArrayList" />
<jsp:useBean id="membersMap" class="java.util.HashMap" />
<
    membersMap.put("id", "park2");
    membersMap.put("pwd", "4321");
    membersMap.put("name", ",박지성");
    membersMap.put("email", "park2@test.com");
    MemberBean 객체를 저장할
    ArrayList 객체를 생성합니다.

    회원 정보를 저장할 HashMap 객체를
    <useBean> 액션 태그를 이용해 생성합니다.

    HashMap에 key/value 쌍으로 회원
    정보를 저장합니다.

MemberBean m2 = new MemberBean("son", "1234", "손흥민", "son@test.com");
membersList.add(m1);      전송된 회원 정보와 자바 코드로 생성한
membersList.add(m2);      회원 정보를 ArrayList에 저장합니다.
membersMap.put("membersList", membersList);
    회원 정보를 저장하는
    MemberBean 객체를
    생성합니다.

    회원 정보가 저장된 membersList를 memberList라는
    key로 HashMap에 저장합니다.

%>
<html>
<head>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
</head>
<body>
```

```

<table border=1 align="center" >
  <tr align=center bgcolor="#99ccff">
    <td width="20%"><b>아이디</b></td>
    <td width="20%"><b>비밀번호</b></td>
    <td width="20%" ><b>이름</b></td>
    <td width="20%"><b>이메일</b></td>
  </tr>
  <tr align=center>
    <td>${membersMap.id}</td>
    <td>${membersMap.pwd}</td>
    <td>${membersMap.name}</td>
    <td>${membersMap.email }</td>
  </tr>
  <tr align=center>
    <td>${membersMap.membersList[0].id}</td>
    <td>${membersMap.membersList[0].pwd}</td>
    <td>${membersMap.membersList[0].name}</td>
    <td>${membersMap.membersList[0].email}</td>
  </tr>
  <tr align=center>
    <td>${membersMap.membersList[1].id}</td>
    <td>${membersMap.membersList[1].pwd}</td>
    <td>${membersMap.membersList[1].name}</td>
    <td>${membersMap.membersList[1].email}</td>
  </tr>
</table>
</body>

```

HashMap 이름 뒤에 ,(마침표) 연산자로 저장 시 사용한 key를 사용하여 value를 가져옵니다.

HashMap에 저장된 ArrayList에 ,(마침표)로 접근한 후 다시 각각의 속성에 ,(마침표)를 이용해 접근하여 첫 번째 회원 정보를 출력합니다.

ArrayList에 저장된 두 번째 회원 정보를 출력합니다.

3. <http://localhost:8090/pro14/test01/memberForm.jsp>로 요청하여 회원 정보를 입력한 후 가입하기를 클릭합니다.

▼ 그림 14-20 회원 가입창에서 회원 정보 입력 후 가입하기 클릭

회원 가입창

아이디

비밀번호

이름

이메일

가입하기

4. `HashMap`에 저장된 회원 정보를 출력합니다.

▼ 그림 14-21 회원 정보 출력

아이디	비밀번호	이름	이메일
park2	4321	박지성	park2@test.com
cha	1234	차범근	cha@test.com
son	1234	손흥민	son@test.com

14.2.8 has-a 관계 빈 사용 실습

이번에는 표현 언어에서 `has-a` 관계를 가지는 빈의 자식 빈 속성에 접근하는 방법을 알아보겠습니다.

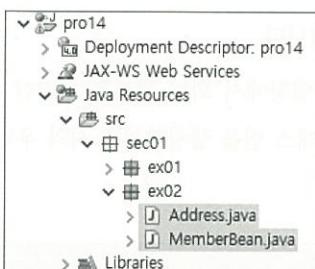
객체가 다른 객체를 속성으로 가지는 경우를 `has-a` 관계라고 합니다. 사용 형식은 다음과 같이 ‘속성 이름’과 `.`(마침표) 연산자로 자식 속성에 접근하면 됩니다.

`${부모빈이름.자식속성이름.속성이름}`

그리면 `has-a` 관계를 가지는 빈의 자식 속성에 접근하여 값을 출력하는 예제를 실습해 보겠습니다.

1. `sec01.ex02` 패키지를 만들고 `MemberBean` 클래스와 `Address` 클래스를 준비합니다.

▼ 그림 14-22 MemberBean과 Address 클래스 위치



2. `MemberBean` 클래스를 다음과 같이 작성합니다. 이번에는 회원의 주소를 저장하는 `Address` 클래스 타입으로 선언된 `addr`을 속성으로 가집니다. 이처럼 속성으로 다른 자바 빈을 가지는 경우를 `has-a` 관계라고 합니다.

코드 14-14 pro14/src/sec01/ex02/MemberBean.java

```
package sec01.ex02;
```

```
...
```

```
public class MemberBean {  
    private String id;  
    private String pwd;  
    private String name;  
    private String email;  
    private Date joinDate;  
    private Address addr; ────────── 주소 정보를 저장하는 Address 클래스 타입  
                           속성을 선언합니다.  
    public MemberBean() {  
  
    }  
    // 속성에 대한 getter/setter  
}
```

3. 회원의 거주 도시와 우편번호를 저장하는 자식 클래스 Address를 다음과 같이 작성합니다.

코드 14-15 pro14/src/sec01/ex02/Address.java

```
public class Address {  
    private String city; ────────── 회원의 거주 도시와 우편번호를 저장합니다.  
    private String zipcode;  
  
    public Address() {  
    }  
    // 속성에 대한 getter/setter  
}
```

4. memberForm.jsp의 action 값을 member6.jsp로 설정합니다.

5. member6.jsp를 다음과 같이 작성합니다. 먼저 회원 가입창에서 회원 정보를 입력한 후 전달받아 빈 속성에 설정합니다. 그리고 다시 Address 클래스 빈을 생성하여 도시와 우편번호 정보를 설정합니다.

코드 14-16 pro14/WebContent/test01/member6.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"  
pageEncoding="UTF-8"  
isELIgnored="false" %>  
<%  
    request.setCharacterEncoding("UTF-8");  
%>  
<jsp:useBean id="m" class="sec01.ex02.MemberBean" />  
<jsp:setProperty name="m" property="*" />  
<jsp:useBean id="addr" class="sec01.ex02.Address"/>
```

```

<jsp:setProperty name="addr" property="city" value="서울"/>
<jsp:setProperty name="addr" property="zipcode" value="07654"/>
<%
    m.setAddr(addr);
%>                               MemberBean의 addr 속성에
                               Address 빈을 생성한 후 도시(city)와 우편번호
                               (zipcode)를 설정합니다.

                               Addres 빈을 설정합니다.

<html>
<head>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
</head>
<body>
    <table border=1 align="center" >
        <tr align="center" bgcolor="#99ccff" >
            <td width="7%"><b>아이디</b></td>
            <td width="7%"><b>비밀번호</b></td>
            <td width="5%" ><b>이름</b></td>
            <td width="5%"><b>이메일</b></td>
            <td width="5%" ><b>도시</b></td>
            <td width="5%" ><b>우편번호</b></td>
        </tr>
        <tr align="center">
            <td>${m.id } </td>
            <td>${m.pwd } </td>
            <td>${m.name } </td>
            <td>${m.email}</td>
            <td><%=m.getAddr().getCity() %></td>
            <td><%=m.getAddr().getZipcode() %></td>
        </tr>
        <tr align="center">
            <td>${m.id } </td>
            <td>${m.pwd } </td>
            <td>${m.name} </td>
            <td>${m.email}</td>
            <td>${m.addr.city}</td>
            <td>${m.addr.zipcode}</td>
        </tr>
    </table>
</body>
</html>

```

① 속성들의 getter를 두 번 호출해서 주소를 출력합니다.

② 자바 빈의 속성 이름과 .(마침표) 연산자를 이용해 주소를 출력합니다.

- ①에서는 표현식을 이용해 getter를 두 번 호출해서 표시했는데 이 방법은 불편합니다. 반면에
 ②에서는 빈 이름만을 이용해 .(마침표) 연산자로 주소 정보를 표시했습니다.

6. 브라우저에 요청하여 회원가입창에서 회원 정보를 입력하고 **가입하기**를 클릭합니다.

▼ 그림 14-23 회원가입창에서 회원 정보 입력 후 **가입하기** 클릭

회원가입

아이디

비밀번호

이름

이메일

가입하기 **다시입력**

7. 출력창에서 has-a 관계의 속성 값인 주소 정보를 출력합니다.

▼ 그림 14-24 .(마침표) 연산자로 자식 속성에 접근해 주소 정보 출력

아이디	비밀번호	이름	이메일	도시	우편번호
hong	1234	홍길동	hong@test.com	서울	07654
hong	1234	홍길동	hong@test.com	서울	07654

14.3 표현 언어로 바인딩 속성 출력하기

J A V A W E B

`request`, `session`, `application` 내장 객체에 속성을 바인딩한 후 다른 서블릿이나 JSP에 전달할 수 있습니다. 표현 언어를 사용하면 자바 코드를 사용하지 않고 바인딩된 속성 이름으로 바로 값을 출력할 수 있습니다.

14.3.1 내장 객체 속성 값 출력 실습

먼저 request, session, application 내장 객체에 바인딩된 속성 값을 표현 언어를 이용해 JSP에서 출력해 보겠습니다.

- 첫 번째 JSP인 forward1.jsp를 다음과 같이 작성합니다. 브라우저에서 요청 시 request, session, application 내장 객체에 회원 정보를 바인딩한 후 다시 member1.jsp로 포워딩합니다.

코드 14-17 pro14/WebContent/test02/forward1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding("utf-8");
    request.setAttribute("id", "hong");           ← request 내장 객체에 바인딩합니다.
    request.setAttribute("pwd", "1234");
    session.setAttribute("name", "홍길동");        ← session 내장 객체에 바인딩합니다.
    application.setAttribute("email", "hong@test.com"); ← application 내장 객체에
%>
<html>
<head>
    <meta charset="UTF-8">
    <title>forward1</title>
</head>
<body>
    <jsp:forward page="member1.jsp" />           ← member1.jsp로 포워딩합니다.
</html>
```

- 두 번째 JSP인 member1.jsp를 다음과 같이 작성합니다. 우선 첫 번째 방법으로 getAttribute() 메서드에 속성 이름을 인자로 하여 값을 가져옵니다. 그리고 두 번째 방법으로 표현 언어에서 자바 코드를 사용하지 않고 바로 속성 이름으로 회원 정보를 가져와 출력합니다.

코드 14-18 pro14/WebContent/test02/member1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"  isELIgnored="false" %>
<%
    request.setCharacterEncoding("UTF-8");
    String id= (String)request.getAttribute("id");
    String pwd= (String)request.getAttribute("pwd");
    String name= (String)session.getAttribute("name");
    String email= (String)application.getAttribute("email");           ← 각 내장 객체에 바인딩된
%>
```

속성 값을 getAttribute() 메서드를 이용해 가져옵니다.

```

%>
<html>
<head>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
</head>
<body>
    <table border="1" align="center" >
        <tr align="center" bgcolor="#99ccff">
            <td width="20%"><b>아이디</b></td>
            <td width="20%"><b>비밀번호</b></td>
            <td width="20%"><b>이름</b></td>
            <td width="20%"><b>이메일</b></td>
        </tr>
        <tr align="center">
            <td><%=id %></td> ← 표현식으로 회원 정보를 출력합니다.
            <td><%=pwd%></td>
            <td><%=name %></td>
            <td><%=email %></td>
        </tr>
        <tr align="center">
            <td>${id}</td> ← 자바 코드 없이 바로 바인딩된 속성
            <td>${pwd}</td>
            <td>${name}</td>
            <td>${email}</td>
        </tr>
    </table>
</body>
</html>

```

3. <http://localhost:8090/pro14/test02/forward1.jsp>로 요청합니다. 첫 번째 회원 정보는 `getAttribute()` 메서드를 이용해 출력하고, 두 번째 회원 정보는 표현 언어에서 속성 이름으로 바로 출력합니다.

▼ 그림 14-25 실행 결과

아이디	비밀번호	이름	이메일
hong	1234	홍길동	hong@test.com
hong	1234	홍길동	hong@test.com

이번에는 `request`에 회원 정보를 저장한 `MemberBean` 객체를 바인딩한 후 다시 출력해 보겠습니다.

4. 다음과 같이 forward2.jsp에서 MemberBean 객체를 생성하고 속성에 회원 정보를 설정합니다. 그리고 request 내장 객체에 속성 이름 member로 MemberBean 객체를 바인딩한 후 member2.jsp로 포워딩합니다.

코드 14-19 pro14/WebContent/test02/forward2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   import="sec01.ex01.*" pageEncoding="UTF-8"%>
<%>
    request.setCharacterEncoding("utf-8");
    MemberBean member = new MemberBean("lee", "1234", "이순신", "lee@test.com");
    request.setAttribute("member", member);
%>
<html>
<head>
    <meta charset="UTF-8">
    <title>forward2</title>
</head>
<body>
    <jsp:forward page="member2.jsp" />
</html>
```

MemberBean 객체 생성 후 회원 정보를 속성에 설정합니다.

속성 이름 member로 MemberBean 객체를 바인딩합니다.

5. member2.jsp를 다음과 같이 작성합니다. request 내장 객체에 속성 이름 member로 접근한 후 MemberBean 속성 값을 출력합니다.

코드 14-20 pro14/WebContent/test02/member2.jsp

```
...
<body>
    <table border="1" align="center" >
        <tr align="center" bgcolor="#99ccff">
            <td width="20%"><b>아이디</b></td>
            <td width="20%"><b>비밀번호</b></td>
            <td width="20%"><b>이름</b></td>
            <td width="20%"><b>이메일</b></td>
        </tr>
        <tr align="center">
            <td>${member.id} </td> ←
            <td>${member.pwd} </td> ←
            <td>${member.name} </td> ←
            <td>${member.email}</td> ←
        </tr>
    </table>
</body>
```

바인딩 시 속성 이름으로 각각의 MemberBean 속성에 접근하여 회원 정보를 출력합니다.

6. <http://localhost:8090/pro14/test02/forward2.jsp>로 요청하여 실행 결과를 확인합니다.

▼ 그림 14-26 실행 결과

아이디	비밀번호	이름	이메일
lee	1234	이순신	lee@test.com

7. 이번에는 `request`에 회원 정보를 저장한 `ArrayList`를 바인딩하고 다시 출력해 보겠습니다. `forward3.jsp`에서 다음과 같이 `ArrayList` 객체를 생성하고 `MemberBean` 객체를 저장합니다. 그리고 `request` 내장 객체에 `ArrayList` 객체를 다시 `membersList` 속성 이름으로 바인딩한 후 두 번째 JSP로 포워딩합니다.

코드 14-21 pro14/WebContent/test02/forward3.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
import="java.util.* , sec01.ex01.*" pageEncoding="UTF-8"
isELIgnored="false" %>
<%
    request.setCharacterEncoding("UTF-8");
    List membersList = new ArrayList(); •————— ArrayList 객체를 생성합니다.
    MemberBean m1 = new MemberBean("lee", "1234", "이순신", "lee@test.com");
    MemberBean m2 = new MemberBean("son", "1234", "손흥민", "son@test.com");
    membersList.add(m1); •————— 두 개의 MemberBean 객체를
    membersList.add(m2); •————— ArrayList에 저장합니다. •————— MemberBean 객체를 생성한 후
    request.setAttribute("membersList", membersList); •————— 두 명의 회원 정보를 저장합니다.
%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>forward3</title>
</head>
<body>
    <jsp:forward page="member3.jsp" />
</body>
</html>
```

8. member3.jsp를 다음과 같이 작성합니다. 바인딩 시 속성 이름 `membersList`로 바로 `ArrayList` 객체에 접근합니다. 그런 다음 저장 순서인 인덱스를 이용해 각각의 `MemberBean`에 접근한 후 속성 이름으로 회원 정보를 출력합니다.

코드 14-22 pro14/WebContent/test02/member3.jsp

```

...
<body>
    <table border=1 align="center" >
        <tr align="center" bgcolor="#99ccff">
            <td width="20%"><b>아이디</b></td>
            <td width="20%"><b>비밀번호</b></td>
            <td width="20%"><b>이름</b></td>
            <td width="20%"><b>이메일</b></td>
        </tr>
        <tr align="center">
            <td>${membersList[0].id}</td> •———— 표현 언어에서 속성 이름으로 ArrayList에
            <td>${membersList[0].pwd}</td> 접근한 후 인덱스를 이용해 첫 번째 회원
            <td>${membersList[0].name}</td> 정보를 출력합니다.
            <td>${membersList[0].email}</td>
        </tr>
        <tr align="center">
            <td>${membersList[1].id}</td> •———— 표현 언어에서 속성 이름으로 ArrayList에
            <td>${membersList[1].pwd}</td> 접근한 후 인덱스를 이용해 두 번째 회원
            <td>${membersList[1].name}</td> 정보를 출력합니다.
            <td>${membersList[1].email}</td>
        </tr>
    </table>
</body>

```

9. `http://localhost:8090/pro14/test02/forwar3.jsp`로 요청합니다. 복잡한 자바 코드를 사용하지 않고 바로 속성 이름과 인덱스만으로 회원 정보가 출력된 결과를 확인할 수 있습니다.

▼ 그림 14-27 실행 결과

14/test02/forward3.jsp

아이디	비밀번호	이름	이메일
lee	1234	이순신	lee@test.com
son	1234	손흥민	son@test.com

14.3.2 스코프 우선순위

`request`, `session`, `application` 내장 객체에서는 데이터를 바인딩해서 다른 JSP로 전달합니다. 그런데 각 내장 객체에 바인딩하는 속성 이름이 같은 경우 JSP에서는 각 내장 객체에 지정된 출력 우선순위에 따라 순서대로 속성에 접근합니다. 이번에는 각 내장 객체에 같은 속성 이름으로 바인딩할 때의 출력 우선순위를 알아보겠습니다.

1. `forward4.jsp`를 다음과 같이 작성합니다. `request`에 `address`를 바인딩한 후 다시 `member4.jsp`로 포워딩합니다.

코드 14-23 pro14/WebContent/test02/forward4.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isELIgnored="false"%>
<%
    request.setCharacterEncoding("utf-8");
    request.setAttribute("id", "hong");
    request.setAttribute("pwd", "1234");
    session.setAttribute("name", "홍길동");
    application.setAttribute("email", "hong@test.com");
    request.setAttribute("address", "서울시 강남구"); ────────── request에 address 속성 이름으로
%>                                         바인딩합니다.

<html>
<head>
    <meta charset="UTF-8">
    <title>forward4</title>
</head>
<body>
    <jsp:forward page="member4.jsp" />
</html>
```

2. `member4.jsp`를 다음과 같이 작성합니다. `session`에 다시 동일한 속성 이름 `address`로 바인딩합니다. 만약 표현 언어로 `address` 값을 출력하면 `session`보다 `request`가 우선순위가 높으므로 `request`의 `address` 값이 출력됩니다.

코드 14-24 pro14/WebContent/test02/member4.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isELIgnored="false" %>
<%
    session.setAttribute("address", "수원시 팔달구"); ────────── session에 address 속성 이름으로
%>                                         바인딩합니다.
```

```

<html>
<head>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
</head>
<body>
    <table border="1" align="center" >
        <tr align="center" bgcolor="#99ccff">
            <td width="7%"><b>아이디</b></td>
            <td width="7%"><b>비밀번호</b></td>
            <td width="5%"><b>이름</b></td>
            <td width="5%"><b>이메일</b></td>
            <td width="5%"><b>주소</b></td>
        <tr>
        <tr align="center">
            <td>${id } </td>
            <td>${pwd } </td>
            <td>${name } </td>
            <td>${email }</td>
            <td>${address }</td> ━━━━ request에서 바인딩된 address 값이 출력됩니다.
        </tr>
    </table>
</html>

```

3. <http://localhost:8090/pro14/test02/forward4.jsp>로 요청합니다. 주소를 보면 request에 바인딩된 값이 출력된 것을 알 수 있습니다.

▼ 그림 14-28 실행 결과

아이디	비밀번호	이름	이메일	주소
hong	1234	홍길동	hong@test.com	서울시 강남구

4. 이번에는 forward4.jsp의 request에 바인딩하는 부분을 주석 처리합니다.

▼ 그림 14-29 request에 바인딩 부분 주석 처리

```

1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%
4     request.setCharacterEncoding("utf-8");
5     request.setAttribute("id", "hong");
6     request.setAttribute("pwd", "1234");
7     session.setAttribute("name", "홍길동");
8     application.setAttribute("email", "hong@test.com");
9     //request.setAttribute("address", "서울시 강남구");
10 %>

```

5. 다시 회원가입창에서 가입하기를 클릭하면 다음과 같이 session에서 바인딩한 주소가 출력되는 것을 확인할 수 있습니다.

▼ 그림 14-30 session의 바인딩 값 출력

아이디	비밀번호	이름	이메일	주소
hong	1234	홍길동	hong@test.com	수원시 팔달구

6. 표현 언어에서는 동일한 속성 이름에 접근할 경우 page 객체의 속성이 우선순위가 가장 높습니다. 표현 언어에서 같은 속성에 대한 우선순위는 다음과 같습니다.

page > request > session > application

14.4 커스텀 태그

JAVA WEB

앞에서 구현한 JSP 페이지의 기능을 보면 액션 태그나 표현 언어를 사용하더라도 조건식이나 반복문에서는 여전히 자바 코드를 사용하고 있습니다. 이러한 자바 코드를 제거하기 위해 JSTL이나 커스텀 태그가 등장했습니다. 커스텀 태그란 JSP 페이지에서 자주 사용하는 자바 코드를 대체하기 위해 만든 태그입니다.

커스텀 태그의 종류는 다음 두 가지입니다.

- **JSTL(JSP Standard Tag Library)**: JSP 페이지에서 가장 많이 사용하는 기능을 태그로 제공하며, JSTL 라이브러리를 따로 설치해서 사용합니다.
- **개발자가 만든 커스텀 태그**: 개발자가 필요에 의해 만든 태그로, 스트리즈나 스프링 프레임워크에서 미리 만들어서 제공합니다.

JSP에서는 개발자가 필요할 때 태그를 만들어 사용할 수 있지만 스트리즈나 스프링 프레임워크에서는 프레임워크 기능과 편리하게 연동할 수 있도록 미리 태그를 만들어서 제공하기도 합니다.

먼저 JSTL부터 알아봅시다.

14.5

JSP 표준 태그 라이브러리(JSTL)

JSTL(JSP Standard Tag Library)이란 커스텀 태그 중 가장 많이 사용되는 태그를 표준화하여 라이브러리로 제공하는 것을 말합니다. JSTL에서는 여러 가지 태그를 지원하는데, 이를 표 14-4에 정리했습니다.

▼ 표 14-4 여러 가지 JSTL 태그 종류

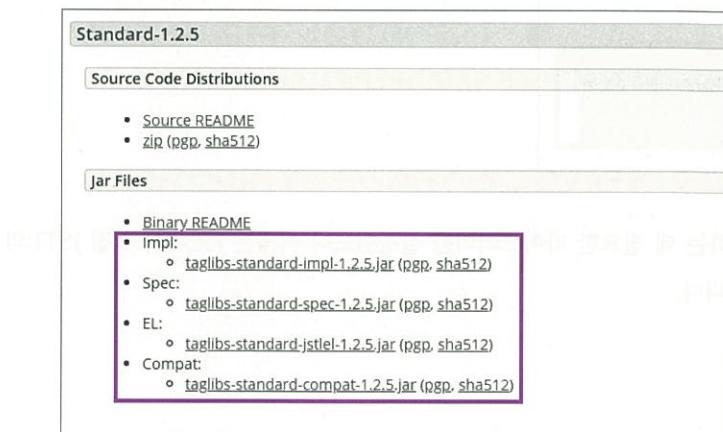
라이브러리	세부 기능	접두어	관련 URI
코어	변수 지원, 흐름 제어, 반복문 처리, URL 처리	c	http://java.sun.com/jsp/jstl/core
국제화	지역, 메시지 형식, 숫자 및 날짜 형식	fmt	http://java.sun.com/jsp/jstl/fmt
XML	XML 코어, 흐름 제어, XML 변환	x	http://java.sun.com/jsp/jstl/xml
데이터베이스	SQL	sql	http://java.sun.com/jsp/jstl/sql
함수	컬렉션 처리, 문자열 처리	fn	http://java.sun.com/jsp/jstl/functions

JSTL은 JSP 2.0 규약부터 추가된 기능이므로 현재는 톰캣에서 기본으로 제공되지 않습니다. 따라서 다음 사이트에서 라이브러리를 다운로드해 설치해야 합니다.

- <http://tomcat.apache.org/download-taglibs.cgi>

1. 사이트에 접속한 후 네 개의 jar 파일을 각각 다운로드합니다.

▼ 그림 14-31 JSTL 관련 네 개의 라이브러리 다운로드



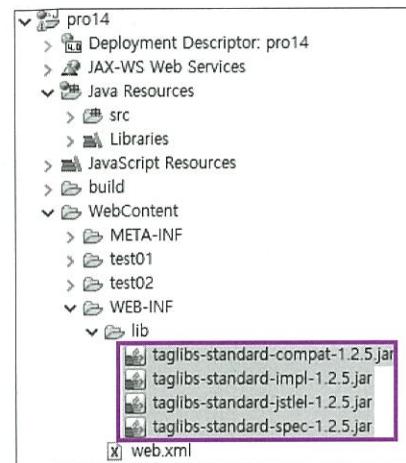
2. 네 개의 파일을 모두 로컬 PC에 저장합니다.

▼ 그림 14-32 JSTL 관련 라이브러리 다운로드 확인



3. 이 파일들을 복사해 프로젝트의 lib 폴더에 붙여 넣습니다.

▼ 그림 14-33 JSTL 라이브러리 lib 폴더에 복사 & 붙여 넣기



이렇게 해서 JSTL을 사용하는 데 필요한 라이브러리를 설정했으니 이제는 JSP에서 직접 JSTL의 기능을 하나씩 알아보겠습니다.

14.6

Core 태그 라이브러리 사용하기

이번 절에서는 기본 기능을 제공하는 코어 라이브러리를 사용해 보겠습니다. 아직 JSP에서는 변수 선언, 조건식, 반복문 기능은 자바 코드를 이용해서 구현합니다. 코어 라이브러리를 사용하면 이런 자바 기능을 태그로 대체할 수 있습니다. 톰캣에서는 JSTL 라이브러리를 기본으로 제공하지 않고 외부 라이브러리에서 가져와 기능을 수행합니다.

따라서 자바의 `import`문처럼 코어 태그 라이브러리를 사용하려면 반드시 JSP 페이지 상단에 다음과 같이 `taglib` 디렉티브 태그를 추가해서 톰캣에게 알려주어야 합니다. 만약 선언하지 않으면 JSP 실행 시 오류가 발생합니다.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

표 14-5에 Core 태그 라이브러리의 기능을 수행하는 태그의 종류와 각각의 기능에 대해 정리했습니다.

▼ 표 14-5 Core 태그 라이브러리 기능

기능	태그	설명
변수 지원	<code><c:set></code>	JSP 페이지에서 변수를 지정합니다.
	<code><c:remove></code>	지정된 변수를 제거합니다.
흐름 제어	<code><c:if></code>	조건문을 사용합니다.
	<code><c:choose></code>	<code>switch</code> 문을 사용합니다. <code><c:when></code> 과 <code><c:otherwise></code> 서브 태그를 갖습니다.
	<code><c:forEach></code>	반복문을 사용합니다.
	<code><c:forTokens></code>	구분자로 분리된 각각의 토큰을 처리할 때 사용합니다.
URL 처리	<code><c:import></code>	URL을 이용해 다른 자원을 JSP 페이지에 추가합니다.
	<code><c:redirect></code>	<code>response.sendRedirect()</code> 기능을 수행합니다.
	<code><c:url></code>	요청 매개변수로부터 URL을 생성합니다.
기타 태그	<code><c:catch></code>	예외 처리에 사용합니다.
	<code><c:out></code>	<code>JspWriter</code> 에 내용을 처리한 후 출력합니다.

14.6.1 <c:set> 태그를 이용한 실습

JSP에서 변수를 사용하려면 자바 코드에서 선언합니다. 그러나 <c:set> 태그를 이용하면 변수를 대체할 수 있습니다. 변수 선언 형식은 다음과 같습니다.

```
<c:set var="변수 이름" value="변수값" [scope="scope 속성 중 하나"] />
```

여기서 var은 변수 이름을, value는 변수에 저장할 값을, scope는 변수 스코프를 지정합니다(page, request, session, application 중 하나).

그럼 <c:set> 태그로 변수를 선언한 후 값을 출력해 보겠습니다.

1. 프로젝트의 WebContext 디렉터리 하위에 test03 디렉터리를 만들고 실습에 관련된 JSP 파일을 만듭니다.

▼ 그림 14-34 실습 파일 위치



2. 먼저 member1.jsp를 작성합니다. 상단에 taglib 디렉티브 태그를 선언하고 <c:set> 태그를 이용해 회원 정보를 저장하는 변수를 선언한 후 값을 초기화합니다. 이때 <c:set> 태그의 value 속성은 표현 언어로 값을 설정할 수 있습니다. 그리고 표현 언어에서 변수 이름을 사용해 값을 출력합니다.

코드 14-25 pro14\WebContent\test03\member1.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    request.setCharacterEncoding("UTF-8");
%>
<c:set var="id" value="hong" scope="page" />
<c:set var="pwd" value="1234" scope="page" />
<c:set var="name" value="${'홍길동'}" scope="page" />
<c:set var="age" value="${22}" scope="page" />
<c:set var="height" value="${177}" scope="page" />
<head>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
</head>
<html>
<body>
    <table align="center" border=1 >
        <tr align="center" bgcolor="lightgreen" >
            <td width="7%" ><b>아이디</b></td>
            <td width="7%" ><b>비밀번호</b></td>
            <td width="7%" ><b>이름</b></td>
            <td width="7%" ><b>나이</b></td>
            <td width="7%" ><b>키</b></td>
        </tr>
        <tr align="center">
            <td>${id}</td>
            <td>${pwd}</td>
            <td>${name}</td>
            <td>${age}</td>
            <td>${height}</td>
        </tr>
    </table>
</body>
</html>

```

core 태그 라이브러리를 사용하기 위해 반드시 선언해야 합니다.

<c:set> 태그를 이용해 변수를 선언합니다. value 속성에는 표현 언어를 사용해서 초기화할 수 있습니다.

표현 언어로 변수에 바로 접근하여 값을 출력합니다.

3. <http://localhost:8090/pro14/test03/member1.jsp>로 요청합니다. 표현 언어로 변수의 값을 출력합니다.

▼ 그림 14-35 실행 결과

아이디	비밀번호	이름	나이	키
hong	1234	홍길동	22	177

이번에는 `<c:set>` 태그를 이용해 너무 길어서 사용하기 불편한 변수나 속성 이름을 간결하게 만들어 보겠습니다. 먼저 JSP에서 `<a>` 태그를 이용해 다른 페이지로 이동하는 방법입니다. 지금 까지는 표현 언어로 `pageContext.request.contextPath` 같은 긴 속성을 그대로 사용했는데, `<c:set>` 태그를 이용하면 긴 이름의 속성이나 변수를 줄여서 사용할 수 있습니다.

```
<a href="${pageContext.request.contextPath}/memberForm.jsp">회원가입하기</a>
```

로그인창에서 회원 가입창으로 이동할 때 미리 `<c:set>` 태그를 이용해 `pageContext.request.contextPath` 속성 이름을 `contextPath`로 줄여서 사용하고 있습니다. 복잡한 웹 페이지에서 속성 이름을 짧게 줄이면 코드의 가독성이 좋아집니다.

코드 14-26 pro14/WebContent/test03/login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="contextPath" value="${pageContext.request.contextPath}" />
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>로그인창</title>
</head>
<body>
    <form action="result.jsp">
        아이디 : <input type="text" size=20 /><br>
        비밀번호: <input type="password" size=20 /><br>
        <input type="submit" value="로그인" /> <input type="reset" value="다시입력" />
    </form>
    <br><br>
<%-->
<a href="${pageContext.request.contextPath}/memberForm.jsp">회원가입하기</a>
```

〈c:set〉 태그 이용해 `pageContext` 내장 객체의 컨텍스트 이름을 변수 `contextPath`에 미리 설정 합니다.

```
--%>
</body>          긴 내장 객체의 속성을 사용할 필요 없이 간단한
</html>          변수 이름으로 컨텍스트 이름을 설정합니다.
```

이번에는 `<c:set>` 태그를 이용해 바인딩된 속성 이름이 긴 경우 더 짧은 변수로 대체해서 사용하는 방법을 알아보겠습니다.

다음은 앞 절에서 실습한 코드 14-14에서 `HashMap`에 저장된 `ArrayList`의 `MemberBean` 속성을 출력하는 표현 언어입니다.

```
 ${membersMap.membersList[0].id}
```

속성 이름이 길면 사용하기가 불편하고 가독성도 떨어집니다. 그래서 미리 `<c:set>` 태그를 이용해 사용하기 편리한 이름인 `membersList`로 설정한 후 인덱스를 이용해 회원 정보를 출력했습니다.

코드 14-27 pro14/WebContent/test03/member2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    import="java.util.*", sec01.ex01.*"
    pageEncoding="UTF-8" isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    request.setCharacterEncoding("UTF-8");
%>
<jsp:useBean id="membersList" class="java.util.ArrayList" />
<jsp:useBean id="membersMap" class="java.util.HashMap" />
<%
    membersMap.put("id", "park2");
    membersMap.put("pwd", "4321");
    membersMap.put("name", "박지성");
    membersMap.put("email", "park2@test.com");
    MemberBean m1 = new MemberBean("son", "1234", "손흥민", "son@test.com");
    MemberBean m2 = new MemberBean("ki", "4321", "기성용", "ki@test.com");
    membersList.add(m1);
    membersList.add(m2);
    membersMap.put("membersList", membersList);
%>
<c:set var="membersList" value="${membersMap.membersList}" />
<html>
<head>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
```

`<c:set>` 태그를 이용해 `HashMap`에 저장된 `ArrayList`에 접근하기 위해 사용하기 편리한 이름으로 설정합니다.

```

</head>
<body>
  <table border="1" align="center" >
    <tr align=center bgcolor="#99ccff">
      <td width="20%"><b>아이디</b></td>
      <td width="20%"><b>비밀번호</b></td>
      <td width="20%"><b>이름</b></td>
      <td width="20%"><b>이메일</b></td>
    </tr>
    <tr align="center">
      <td>${membersMap.id}</td>
      <td>${membersMap.pwd}</td>
      <td>${membersMap.name}</td>
      <td>${membersMap.email }</td>
    </tr>
    <tr align="center">
      <td>${membersList[0].id}</td> ← <c:set> 태그로 설정한 변수 이름으로 접근하여 출력합니다.
      <td>${membersList[0].pwd}</td>
      <td>${membersList[0].name}</td>
      <td>${membersList[0].email}</td>
    </tr>
    <tr align="center">
      <td>${membersList[1].id}</td> ← <c:set> 태그로 설정한 변수 이름으로 접근하여 출력합니다.
      <td>${membersList[1].pwd}</td>
      <td>${membersList[1].name}</td>
      <td>${membersList[1].email}</td>
    </tr>
  </table>
</body>
</html>

```

다음은 실행 결과입니다.

▼ 그림 14-36 실행 결과

아이디	비밀번호	이름	이메일
park2	4321	박지성	park2@test.com
son	1234	손홍민	son@test.com
ki	4321	기성용	ki@test.com

14.6.2 <c:remove> 태그를 이용한 실습

JSP 페이지에서 변수를 선언했으면 <c:remove> 태그를 이용해 변수를 제거할 수도 있습니다.

<c:remove> 태그를 이용하는 형식은 다음과 같습니다.

```
<c:remove var="변수이름" [scope="scope 속성 중 하나"] />
```

여기서 var은 제거할 변수 이름을, scope는 변수 범위(scope)를 지정합니다(page, request, session, application 중 하나).

- member3.jsp를 다음과 같이 작성합니다. <c:remove> 태그를 이용해 <c:set>으로 선언한 변수를 삭제합니다.

코드 14-28 pro14/WebContent/test03/member3.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    request.setCharacterEncoding("UTF-8");
%>
<c:set var="id" value="hong" scope="page" />
<c:set var="pwd" value="1234" scope="page" />
<c:set var="name" value="${'홍길동'}" scope="page" />
<c:set var="age" value="${22}" scope="page" />
<c:set var="height" value="${177}" scope="page" />
<c:remove var="age" />                                ← 변수 age와 height를 삭제합니다.
<c:remove var="height" />

<html>
<head>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
</head>
<body>
    <table align="center" border="1" >
        <tr align="center" bgcolor="lightgreen" >
            <td width="7%" ><b>아이디</b></td>
            <td width="7%" ><b>비밀번호</b></td>
            <td width="7%" ><b>이름</b></td>
            <td width="7%" ><b>나이</b></td>
            <td width="7%" ><b>키</b></td>
        </tr>
```

```

<tr align="center">
    <td>${id}</td>
    <td>${pwd}</td>
    <td>${name}</td>
    <td>${age}</td>
    <td>${height}</td>
</tr>
</table>
</body>
</html>

```

2. <http://localhost:8090/pro14/test03/member3.jsp>로 요청합니다. <c:remove> 태그를 이용해 변수 age와 height를 삭제했기 때문에 아무 값도 출력되지 않습니다.

▼ 그림 14-37 실행 결과

The screenshot shows a browser window titled '회원 정보 출력 창'. The address bar contains the URL 'localhost:8090/pro14/test03/member3.jsp'. The page displays a table with five columns: 아이디, 비밀번호, 이름, 나이, and 키. The data row is: hong, 1234, 홍길동, (empty), (empty). The table has a light gray background and dark gray header rows.

14.6.3 <c:if> 태그를 이용한 실습

<c:if> 태그는 이름에서도 알 수 있듯이 JSP 페이지에서 조건문을 대체해 사용하는 태그이며, 사용 형식은 다음과 같습니다.

```

<c:if test="${조건식}" var="변수이름" [scope="scope 속성 중 하나"] />
..
</c:if>

```

여기서 test는 표현 언어를 이용해 수행할 조건식 위치를, var은 조건식의 결과값을 저장합니다. 또한 scope는 변수의 스코프를 지정(page, request, session, application 중 하나)합니다.

그럼 <c:if> 태그를 이용해 조건문을 사용해 보겠습니다.

1. 다음과 같이 member4.jsp를 작성합니다. <c:if> 태그의 test 속성에는 표현 언어 안에 비교 연산자나 논리 연산자로 조건식을 수행합니다.

코드 14-29 pro14/WebContent/test03/member4.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
```

```

pageEncoding="UTF-8"
isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    request.setCharacterEncoding("UTF-8");
%>
<c:set var="id" value="hong" scope="page" />
<c:set var="pwd" value="1234" scope="page" />
<c:set var="name" value="${'홍길동'}" scope="page" />
<c:set var="age" value="${22}" scope="page" />
<c:set var="height" value="${177}" scope="page" />

<html>
<head>
    <meta charset="UTF-8">
    <title>조건문 실습</title>
</head>
<body>
    <c:if test="${true}">           조건식이 true이므로 항상 참입니다.
        <h1>항상 참입니다.</h1>
    </c:if>                      조건식에 비교 연산자를 사용합니다.
                                    ↓
    <c:if test="${11==11}">
        <h1>두 값은 같습니다.</h1>
    </c:if>                      조건식에 비교 연산자를 사용합니다.
                                    ↓
    <c:if test="${11!=31}">
        <h1>두 값은 같지 않습니다.</h1>
    </c:if>                      조건식에 논리 연산자를 사용합니다.
                                    ↓
    <c:if test='${(id=='hong') && (name=='홍길동')}'>
        <h1>아이디는 ${id}이고, 이름은 ${name }입니다.</h1>
    </c:if>

    <c:if test="${age==22}">
        <h1>${name }의 나이는 ${age}살입니다.</h1>
    </c:if>

    <c:if test="${height>160}">
        <h1>${name }의 키는 160보다 큽니다.</h1>
    </c:if>
</body>
</html>

```

2. <http://localhost:8090/pro14/test03/member4.jsp>로 요청하여 실행 결과를 확인합니다.

▼ 그림 14-38 실행 결과

항상 참입니다.
두 값은 같습니다.
두 값은 같지 않습니다.
아이디는 hong이고, 이름은 홍길동입니다.
홍길동의 나이는 22살입니다.
홍길동의 키는 160보다 큽니다.

14.6.4 <c:choose> 태그를 이용한 실습

<c:choose> 태그는 JSP 페이지에서 switch문의 기능을 수행하며, 사용 형식은 다음과 같습니다

```
<c:choose>
  <c:when test="조건식1" >본문내용1</c:when>
  <c:when test="조건식2" >본문내용2</c:when>
  ..
  <c:otherwise>본문내용n</c:otherwise>
</c:choose>
```

첫 번째 <c:when> 태그의 조건식1을 체크해서 참이면 본문내용1을 수행하고 만약 거짓이면 다음 <c:when>의 조건식2를 체크해서 참이면 본문내용2를 수행합니다. 모든 조건이 거짓이면 <c:otherwise> 태그의 본문 내용을 수행합니다.

그럼 실습을 통해 알아보겠습니다.

1. 다음과 같이 member5.jsp를 작성합니다. <c:choose> 태그를 이용해 name 값의 유무에 따라 다른 결과를 표시합니다. 만약 name 값이 정상적이면 회원 정보를 출력하고 name이 null이거나 빈 문자열이면 오류 메시지를 출력합니다.

코드 14-30 pro14/WebContent/test03/member5.jsp

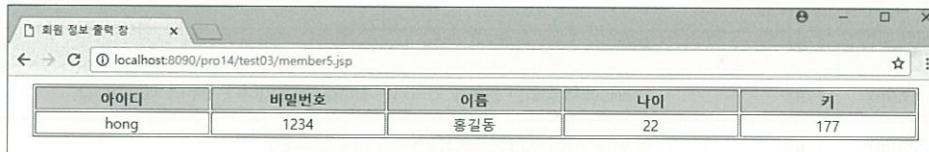
```

...
<body>
    <table align="center" border="1" >
        <tr align="center" bgcolor="lightgreen">
            <td width="7%"><b>아이디</b></td>
            <td width="7%"><b>비밀번호</b></td>
            <td width="7%"><b>이름</b></td>
            <td width="7%"><b>나이</b></td>
            <td width="7%"><b>키</b></td>
        </tr>
        <c:choose>
            <%-- <c:when test="${name==null}"> --%>
            <c:when test='${empty name}'>
                <tr align="center">
                    <td colspan=5>이름을 입력하세요!!</td>
                </tr>
            </c:when>
            <c:otherwise >
                <tr align="center">
                    <td>${id}</td>
                    <td>${pwd}</td>
                    <td>${name}</td>
                    <td>${age}</td>
                    <td>${height}</td>
                </tr>
            </c:otherwise>
        </c:choose>
    </table>
</html>

```

2. <http://localhost:8090/pro14/test03/member5.jsp>로 요청합니다. 먼저 name 변수를 정상적으로 선언한 후 브라우저에서 요청 시 회원 정보를 출력합니다.

▼ 그림 14-39 정상적인 회원 정보 출력 결과



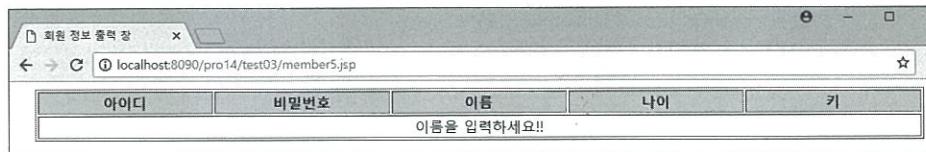
3. 이번에는 다음과 같이 name 변수를 주석 처리합니다.

▼ 그림 14-40 변수 name 주석 처리

```
8 <c:set var="id" value="hong" scope="page" />
9 <c:set var="pwd" value="1234" scope="page" />
10 <%-- <c:set var="name" value="${'홍길동'}" scope="page" /> --%>
11 <c:set var="age" value="${22}" scope="page" />
12 <c:set var="height" value="${177}" scope="page" />
13<html>
```

4. 브라우저에서 재요청 시 오류 메시지를 출력합니다.

▼ 그림 14-41 오류 메시지 출력!



14.6.5 <c:forEach> 태그를 이용한 실습

<c:forEach> 태그는 JSP 페이지에서 반복문을 수행하는 태그이며, 사용 형식은 다음과 같습니다.

```
<c:forEach var="변수이름" items="반복할객체이름" begin="시작값" end="마지막값"
           step="증가값" varStatus="반복상태변수이름">
...
</c:forEach>
```

여기서 var는 반복할 변수 이름을, items는 반복할 객체 이름을 지정합니다. begin과 end는 각각 반복 시작 및 종료 값을, step은 한 번 반복할 때마다 반복 변수를 증가시킬 값을, varStatus는 반복 상태 속성을 지정합니다.

표 14-6에 varStatus의 여러 가지 속성을 정리했습니다

▼ 표 14-6 varStatus의 속성

속성	값	설명
index	int	items에서 정의한 항목을 가리키는 index 번호입니다. 0부터 시작합니다.
count	int	몇 번째 반복인지 나타냅니다. 1부터 시작합니다.
first	boolean	첫 번째 반복인지 나타냅니다.
last	boolean	마지막 반복인지 나타냅니다.

<c:forEach>로 반복문을 만들어 사용해 보겠습니다.

- member6.jsp를 다음과 같이 작성합니다. 먼저 자바 코드로 ArrayList 객체를 생성하여 문자열을 저장한 후 <c:forEach> 태그에서 사용할 수 있도록 <c:set> 태그로 변수 list에 재할당합니다. 그리고 varStatus의 loop 속성을 이용해 반복 횟수를 출력합니다. <c:forEach> 태그의 items에 ArrayList를 설정한 후 반복문 수행 시 ArrayList에 저장된 문자열을 반복 변수 data에 한 개씩 가져와 출력합니다.

코드 14-31 pro14\WebContent\test03\member6.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    import="java.util.*"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    List dataList=new ArrayList();
    dataList.add("hello");
    dataList.add("world");
    dataList.add("안녕하세요!!");
%>
<c:set var="list" value="<%=dataList %>" /> ────────── 표현 언어에서 사용할 수 있도록 <c:set> 태그를
<html>           이용해 변수에 dataList를 할당합니다.
    <head>
        <meta charset="UTF-8">
        <title>반복문 실습</title>
    </head>
    <body>
        <c:forEach var="i" begin="1" end="10" step="1" varStatus="loop">
            i= ${i} &nbsp;&nbsp;&nbsp; 반복횟수: ${loop.count} <br>
        </c:forEach>
        <br>
        <c:forEach var="i" begin="1" end="10" step="2" >
            5 * ${i} = ${5*i}<br>
        </c:forEach>
        <br>
        <c:forEach var="data" items="${list}" >
            ${data } <br>
        </c:forEach>
        <br>
        <c:set var="fruits" value="사과,파인애플, 바나나, 망고, 귤" />
        <c:forTokens var="token" items="${fruits}" delims="," >
            ${token} <br>
        </c:forTokens>
    </body>
</html>

```

반복 변수 i를 1부터 10까지 1씩 증가시키면서 반복문을 수행합니다.

반복 변수 i를 1부터 10까지 2씩 증가시키면서 반복문을 수행합니다.

ArrayList 같은 컬렉션 객체에 저장된 객체(데이터)를 반복해서 반복 변수 data에 하나씩 가져와 처리합니다.

구분자 ,(콤마)를 이용해 문자열을 분리해서 출력합니다.

```
</body>  
</html>
```

2. <http://localhost:8090/pro14/test03/member6.jsp>로 요청하여 결과를 확인합니다.

▼ 그림 14-42 실행 결과



The screenshot shows a browser window titled "반복문 실습". The URL is "localhost:8090/pro14/test03/member6.jsp". The content of the page is as follows:

```
i= 1 반복횟수: 1  
i= 2 반복횟수: 2  
i= 3 반복횟수: 3  
i= 4 반복횟수: 4  
i= 5 반복횟수: 5  
i= 6 반복횟수: 6  
i= 7 반복횟수: 7  
i= 8 반복횟수: 8  
i= 9 반복횟수: 9  
i= 10 반복횟수: 10  
  
5 * 1 = 5  
5 * 3 = 15  
5 * 5 = 25  
5 * 7 = 35  
5 * 9 = 45  
  
hello  
world  
안녕하세요!!  
  
사과  
파인애플  
바나나  
망고  
귤
```

3. 이번에는 <c:forEach> 태그를 이용해 ArrayList에 저장된 회원 정보를 출력해 보겠습니다.

<c:forEach> 태그를 이용하면 ArrayList에 저장된 객체에 편리하게 접근할 수 있습니다.

다음과 같이 <c:forEach> 태그의 반복 변수 i를 ArrayList의 인덱스로 사용해서 저장된 회원 정보를 차례대로 출력하도록 member7.jsp를 작성합니다.

코드 14-32 pro14/WebContent/test03/member7.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"  
import="java.util.*, sec01.ex01.*"  
pageEncoding="UTF-8"  
isELIgnored="false" %>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%  
request.setCharacterEncoding("UTF-8");  
List membersList = new ArrayList();
```

```

MemberBean m1 = new MemberBean("son", "1234", "손흥민", "son@test.com");
MemberBean m2 = new MemberBean("ki", "4321", "기성용", "ki@test.com");
MemberBean m3 = new MemberBean("park", "1212", "박지성", "park@test.com");
membersList.add(m1); ← 세 명의 회원 정보를 MemberBean에
membersList.add(m2); 저장한 후 다시 ArrayList에 저장합니다.
membersList.add(m3);

%>
<c:set var="membersList" value="<%= membersList%>" />
<html>
<head>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
</head>
<body>
    <table border="1" align="center" >
        <tr align="center" bgcolor="lightgreen">
            <td width="7%"><b>아이디</b></td>
            <td width="7%"><b>비밀번호</b></td>
            <td width="5%"><b>이름</b></td>
            <td width="5%"><b>이메일</b></td>
        </tr>
        <c:forEach var="i" begin="0" end="2" step="1" >
            <tr align="center">
                <td>${membersList[i].id}</td> ← 반복 변수 i를 ArrayList의 인덱스로 사용해
                <td>${membersList[i].pwd}</td> 회원 정보를 차례대로 출력합니다.
                <td>${membersList[i].name}</td>
                <td>${membersList[i].email}</td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>

```

4. <http://localhost:8090/pro14/test03/member7.jsp>로 요청하여 결과를 확인합니다.

▼ 그림 14-43 실행 결과

아이디	비밀번호	이름	이메일
son	1234	손흥민	son@test.com
ki	4321	기성용	ki@test.com
park	1212	박지성	park@test.com

5. 이번에는 <c:forEach>문의 items 속성에 membersList를 할당한 후 실행하여 자동으로 var의 member에 membersList의 MemberBean 객체가 차례대로 할당되도록 member8.jsp를 작성합니다.

코드 14-33 pro14/WebContent/test03/member8.jsp

```
...
<body>
    <table border="1" align="center">
        <tr align="center" bgcolor="lightgreen">
            <td width="7%"><b>아이디</b></td>
            <td width="7%"><b>비밀번호</b></td>
            <td width="5%"><b>이름</b></td>
            <td width="5%"><b>이메일</b></td>
        </tr>
        <c:forEach var="member" items="${membersList}" >
            <tr align="center">
                <td>${member.id}</td>
                <td>${member.pwd}</td>
                <td>${member.name}</td>
                <td>${member.email}</td>
            </tr>
        </c:forEach>
    </table>
</body>
```

반복문을 수행하면서 memberList에 저장된 MemberBean 객체가 차례대로 member에 할당됩니다.

속성 이름으로 회원 정보를 차례대로 출력합니다.

6. <http://localhost:8090/pro14/test03/member8.jsp>로 요청하여 결과를 확인합니다.

▼ 그림 14-44 실행 결과

The screenshot shows a web browser window titled '회원 정보 출력 창'. The address bar displays the URL 'localhost:8090/pro14/test03/member8.jsp'. The page content is a table with four columns: '아이디', '비밀번호', '이름', and '이메일'. The table contains three rows of data:

아이디	비밀번호	이름	이메일
son	1234	손흥민	son@test.com
ki	4321	기성용	ki@test.com
park	1212	박지성	park@test.com

14.6.6 <c:url> 태그를 이용한 실습

<c:url> 태그는 JSP 페이지에서 URL 정보를 저장하는 역할을 하며, 사용 형식은 다음과 같습니다.

```
<c:url var="변수이름" value="URL경로" [scope="scope 속성 중 하나"]>
    [<c:param name="매개변수이름" value="전달값" />]
```

</c:url>

여기서 var은 생성된 URL이 저장될 변수를, value는 생성할 URL을, scope는 scope 속성의 값을 지정합니다.

1. urlTest.jsp를 다음과 같이 작성합니다. <c:url> 태그를 이용해 다른 페이지로 이동하면서 데이터를 전달합니다. 따라서 이동할 페이지로 전달할 데이터가 많을 경우에 사용하면 편리합니다.

코드 14-34 pro14\WebContent\test03\urlTest.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   import="java.util.*"
   pageEncoding="UTF-8"
   isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="contextPath" value="${pageContext.request.contextPath}" />
<c:url var="url1" value="/test01/member1.jsp" > ← (c:url) 태그로 이동할 페이지를
<c:param name="id" value="hong" /> 설정합니다.
<c:param name="pwd" value="1234" /> ← 이동할 페이지로 전달할 데이터를
<c:param name="name" value="홍길동" /> 설정합니다.
<c:param name="email" value="hong@test.com" />
</c:url>
<html>
<head>
  <meta charset="UTF-8">
  <title> c:url 태그 실습</title>
</head>
<body>
  <%-- <a href='${contextPath }/test01/member1.jsp'>회원정보출력</a> --%>
  <a href='${url1}'>회원정보출력</a>
</body>
</html>
```

2. <http://localhost:8090/pro14/test03/urlTest.jsp>로 요청한 후 회원정보출력을 클릭합니다.

▼ 그림 14-45 브라우저로 요청 후 회원정보출력 클릭



3. 매개변수로 전달된 회원 정보를 출력합니다(member1.jsp에서 \${속성}을 \${param.속성}으로 수정해 주세요.)

▼ 그림 14-46 실행 결과

4/test01/member1.jsp?id=hong&pwd=1234&name=홍길동&email=hong%40test.com			
아이디	비밀번호	이름	이메일
hong	1234	홍길동	hong@test.com
hong	1234	홍길동	hong@test.com

14.6.7 <c:redirect> 태그를 이용한 실습

<c:redirect> 태그는 지정된 JSP 페이지로 리다이렉트할 때 사용합니다. `response.sendRedirect()` 기능과 동일하며 <c:redirect> 태그로 리다이렉트할 때 매개변수를 전달할 수 있습니다. 사용 형식은 다음과 같습니다.

```
<c:redirect url="리다이렉트 할 URL">
  [ <c:param name="매개변수이름" value="전달값" /> ]
  ...
</c:redirect>
```

여기서 url은 리다이렉트 될 URL이 저장될 변수를 지정합니다.

1. <c:redirect> 태그를 이용해 회원 정보 출력창으로 리다이렉트합니다. 리다이렉트하면서 회원 정보를 매개변수로 전달합니다.

코드 14-35 pro14/WebContent/test03/redirectTest.jsp

```
...
<body>
  <c:redirect url="/test01/member1.jsp" >
    <c:param name="id" value="${'hong'}" />
    <c:param name="pwd" value="${'1234'}" />
    <c:param name="name" value="${'홍길동'}" />
    <c:param name="email" value="${'hong@test.com'}" />
  </c:redirect>
</body>
```

리다이렉트 할 페이지를 설정합니다.
리다이렉트 할 페이지로 전달할
매개변수를 설정합니다.

2. `http://localhost:8090/pro14/test03/redirectTest.jsp`로 요청하면 `test01/member1.jsp`로 리다이렉트됩니다. 그려면서 매개변수로 전달한 회원 정보를 출력합니다.

▼ 그림 14-47 실행 결과

아이디	비밀번호	이름	이메일
hong	1234	홍길동	hong@test.com
hong	1234	홍길동	hong@test.com

14.6.8 <c:out> 태그를 이용한 실습

<c:out> 태그는 화면에 지정한 값을 출력해 주는 태그입니다. 표현 언어와 기능은 거의 동일하지만 기본값 설정 기능 등을 제공하므로 더 편리하게 사용할 수 있습니다. 사용 형식은 다음과 같습니다.

```
<c:out value="출력값" default="기본값" [escapeXml="boolean값"] />
```

여기서 `value`는 출력할 값을, `default`는 `value` 속성에 지정된 값이 없을 때 출력할 기본값을, `escapeXml`은 `escape` 문자를 변환하는 역할을 합니다(생략할 수 있으며 기본값은 `true`)。

1. 다음은 회원 가입창에서 입력한 회원 정보를 전달받아 <c:out> 태그를 이용해 화면에 출력하는 예제입니다. 다음과 같이 `memberForm.jsp`를 작성하여 회원 가입창에서 회원 정보를 입력한 후 `member9.jsp`로 전달합니다.

코드 14-36 pro14/WebContent/test03/memberForm.jsp

```
...
<body>
    <form method="post" action="member9.jsp">
        <h1 style="text-align:center">회원 가입창</h1>
        <table align="center">
            <tr>
                <td width="200">
                    <p align="right">아이디
                </td>
                <td width="400"><input type="text" name="id"></td>
            </tr>
            <tr>
```

```
<td width="200">
    <p align="right">비밀번호
</td>
<td width="400"><input type="password" name="pwd"></td>
</tr>
....
```

2. member9.jsp를 다음과 같이 작성합니다. <c:out> 태그를 이용해 전송된 매개변수 값들을 출력합니다.

코드 14-37 pro14\WebContent\test03\member9.jsp

```
...
<body>
    <table align="center" border="1">
        <tr align="center" bgcolor="lightgreen">
            <td width="7%"><b>아이디</b></td>
            <td width="7%"><b>비밀번호</b></td>
            <td width="7%"><b>이름</b></td>
            <td width="7%"><b>이메일</b></td>
        </tr>
        <c:choose>
            <c:when test="${empty param.id}">
                <tr align="center">
                    <td colspan=5> 아이디를 입력하세요!!</td>
                </tr>
            </c:when>
            <c:otherwise>
                <tr align="center">
                    <td><c:out value="${param.id}" /></td>
                    <td><c:out value="${param.pwd}" /></td>
                    <td><c:out value="${param.name}" /></td>
                    <td><c:out value="${param.email}" /></td>
                </tr>
            </c:otherwise>
        </c:choose>
    </table>
</body>
```

3. <http://localhost:8090/pro14/test03/memberForm.jsp>로 요청하여 회원가입창에서 회원 정보를 입력한 후 **가입하기**를 클릭합니다.

▼ 그림 14-48 회원가입창에서 회원 정보 입력 후 **가입하기** 클릭

회원가입창

아이디: cha

비밀번호: (redacted)

이름: 차별근

이메일: cha@test.com

가입하기 다시입력

4. 그러면 <c:out> 태그를 이용해 전송된 회원 정보를 출력합니다.

▼ 그림 14-49 회원 정보 출력

아이디	비밀번호	이름	이메일
cha	1234	차별근	cha@test.com

프로그래밍을 하다 보면 > 또는 < 그리고 작은따옴표(')나 큰따옴표(") 같은 특수 문자를 출력해야 하는 경우가 있습니다. 그런데 이런 특수 문자들은 HTML 태그에도 사용되므로 각각의 특수 문자에 지정된 문자를 이용해서 브라우저에 출력해야 합니다.

표 14-7은 각 특수 문자가 어떤 문자로 변화되는지 보여줍니다.

▼ 표 14-7 escapeXml이 false일 때 변환되는 문자

특수 문자	변환된 문자
<	<
>	>
&	&
'	'
"	"
...	...

간단히 특수 문자 사용 예를 실습해 보겠습니다.

- escapeXml.jsp를 다음과 같이 작성합니다. <c:out> 태그의 escapeXml 속성을 이용해 변환된 문자를 특수 문자로 변환합니다.

코드 14-38 pro14/WebContent/test03/escapeXml.jsp

```
...
<body>
    <h2>escapeXml 변환하기</h2>
    <h2>
        <pre>
            <c:out value="&lt;" escapeXml="true" /> ←
            <c:out value="&lt;" escapeXml="false" /> ←
            <c:out value="&gt;" escapeXml="true" />
            <c:out value="&gt;" escapeXml="false" />
            <c:out value="&amp;" escapeXml="true" />
            <c:out value="&amp;" escapeXml="false" />
            <c:out value="'" escapeXml="true" />
            <c:out value="'" escapeXml="false" />
            <c:out value=""" escapeXml="true" />
            <c:out value=""" escapeXml="false" />
        </pre>
    </h2>
</body>
```

escapeXml 속성이 true이므로 value의 <는 그대로 화면에 출력됩니다.

escapeXml 속성이 false이므로 value의 <는 해당하는 특수 문자로 변환되어 화면에 출력됩니다.

- http://localhost:8090/pro14/test03/escapeXml.jsp로 요청하여 결과를 확인합니다.

▼ 그림 14-50 실행 결과



지금까지 일반적으로 많이 사용하는 코어 라이브러리에 대해 알아봤습니다. 그 외 `<c:import>` 태그는 `<jsp:include>`와 같은 기능을 수행합니다.

14.7

Core 태그 라이브러리 실습 예제

그럼 지금까지 배운 Core 태그 라이브러리에 좀 더 익숙해지기 위해 로그인, 학점 변환, 구구단 출력 기능을 Core 태그 라이브러리를 이용해서 구현해 보겠습니다. 12장에서 스크립트릿으로 구현했던 예제들과 결과는 같지만 구현 방법은 다릅니다. 그럼 스크립트릿의 구현 방법과 어떻게 다른지 비교하면서 실습해 보세요. 표준 태그 라이브러리 사용법을 금방 익힐 수 있을 것입니다.

14.7.1 로그인 예제

- 프로젝트의 WebContent 폴더에 실습 파일들을 저장할 test04 폴더를 만들고 다음과 같이 여러 개의 JSP 파일들을 준비합니다.

▼ 그림 14-51 실습 파일 위치



2. 로그인창에서 ID와 비밀번호를 입력한 후 로그인을 클릭할 수 있도록 login.jsp를 작성합니다.

코드 14-39 pro14/WebContent/test04/login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding("utf-8");
%>
<!DOCTYPE html>
<html>
<head>...</head>
<body>
    <form action="result.jsp" method="post">
        아이디: <input type="text" name="userID"><br>
        비밀번호: <input type="password" name="userPw"><br>
        <input type="submit" value="로그인">
        <input type="reset" value="다시입력">
    </form>
</body>
</html>
```

3. 이번에는 result.jsp를 다음과 같이 작성합니다. 로그인창에서 ID를 입력한 경우와 입력하지 않은 경우 <c:if> 태그를 이용해 각기 다른 화면을 출력하도록 설정합니다.

코드 14-40 pro14/WebContent/test04/result.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"
    isELIgnored="false"
%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    request.setCharacterEncoding("utf-8");
%>

<html>...</head>
<body>
    <c:if test="${empty param.userID }">
        아이디를 입력하세요.<br>
        <a href="login.jsp">로그인창 </a>
    </c:if>
    <c:if test="${not empty param.userID }">
        <h1> 환영합니다. <c:out value="${param.userID }" />님!!!</h1>
    </c:if>
```

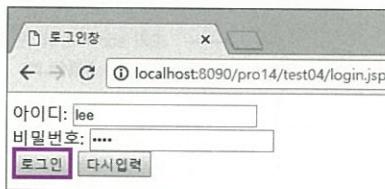
empty 연산자를 이용해 ID 값이 비었는지 체크합니다.

ID를 정상적으로 입력한 경우 로그인 메시지를 출력합니다.

```
</body>
</html>
```

4. http://localhost:8090/pro14/test04/login.jsp로 요청하여 ID와 비밀번호를 입력한 후 **로그인**을 클릭합니다.

▼ 그림 14-52 로그인창에서 **로그인** 클릭



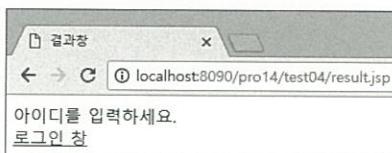
5. ID를 정상적으로 입력한 경우 로그인 메시지를 출력합니다.

▼ 그림 14-53 로그인 메시지 출력



6. ID를 입력하지 않고 로그인한 경우 다시 로그인하라는 메시지를 출력합니다.

▼ 그림 14-54 로그인창으로 이동 메시지 출력



7. 이번에는 <c:if>로 이중 조건문을 구현하도록 다음과 같이 result2.jsp를 작성합니다. 로그인 시 admin으로 로그인하면 관리자 화면을 출력합니다.

코드 14-41 pro14/WebContent/test04/result2.jsp

```
...
<body>
<c:if test="${empty param.userID}">
    아이디를 입력하세요.<br>
    <a href="login.jsp">로그인창 </a>
</c:if>
```

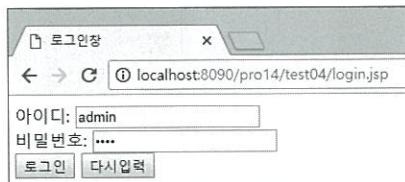
```

<c:if test="${not empty param.userID }"> ────────── ID가 null이 아님을 체크합니다.
    <c:if test="${param.userID =='admin' }">
        <h1>관리자로 로그인 했습니다.</h1>
        <form>
            <input type=button value="회원정보 삭제하기" />
            <input type=button value="회원정보 수정하기" />
        </form>
    </c:if>
    <c:if test="${param.userID !='admin' }">
        <h1> 환영합니다.
        <c:out value="${param.userID}" /> 님!!!</h1>
    </c:if>
</c:if>
</body>

```

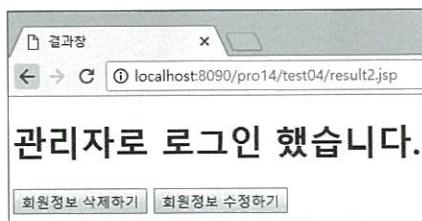
8. `http://localhost:8090/pro14/test04/login.jsp`로 요청하여 admin으로 로그인합니다.

▼ 그림 14-55 관리자로 로그인



9. 그러면 다음과 같이 관리자 화면을 출력합니다.

▼ 그림 14-56 관리자 화면 출력



10. 다른 ID로 로그인 시 로그인 메시지를 출력합니다.

▼ 그림 14-57 다른 ID로 로그인 시 로그인 메시지 출력



14.7.2 학점 변환기 예제

시험 점수를 입력하면 해당하는 학점으로 변환해 주는 프로그램을 Core 태그 라이브러리를 사용해 만들어 보겠습니다. 앞에서 구현했던 방법과 어떻게 다른지 비교하면서 실습하는 것도 도움이 될 것입니다.

1. 다음과 같이 scoreTest.jsp를 작성합니다. 학점으로 변환할 시험 점수를 입력한 후 scoreResult1.jsp로 전송합니다.

코드 14-42 pro14/WebContent/test04/scoreTest.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>시험 점수 입력 페이지</title>
</head>
<body>
    <h1>시험 점수를 입력해 주세요</h1>
    <form method=get action="scoreResult1.jsp">
        시험점수 : <input type=text name="score" /> <br>
        <input type="submit" value="학점변환">
    </form>
</body>
</html>
```

입력한 시험 점수를 scoreResult1.jsp로 전송합니다(제공하는 예제 파일의 매팅 이름을 확인한 후 실행하세요).

2. 다음과 같이 scoreResult1.jsp를 작성합니다. 조건이 여러 개인 경우 이번에는 <c:choose> 태그의 <c:when> 태그에 설정하여 학점을 변환합니다.

코드 14-43 pro14/WebContent/test04/scoreResult1.jsp

```
...
<body>
    <c:set var="score" value="${param.score }" /> • param.score를 score 변수에 할당해서 사용합니다.
    <h1>시험점수
        <c:out value="${score}" />
    </h1><br>
    <c:choose>
        <c:when test="${score}>=90 && score<=100 }"> • 조건이 여러 개인 경우 <c:choose> 안의 <c:when> 태그를 이용해 점수를 해당하는 학점으로 변환합니다
            <h1>A학점입니다.</h1>
        </c:when>
        <c:when test="${score}>=80 && score<90 }">
```

```

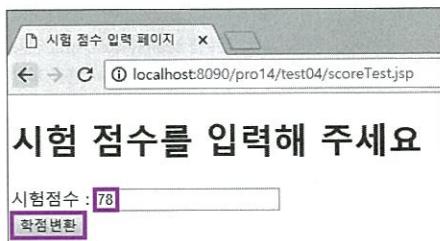
<h1>B학점입니다.</h1>
</c:when>
<c:when test="${score}>=70 && score<80 }">
    <h1>C학점입니다.</h1>
</c:when>
<c:when test="${score}>=60 && score<70 }">
    <h1>D학점입니다.</h1>
</c:when>
<c:otherwise>
    <h1>F학점입니다.</h1>
</c:otherwise>
</c:choose>

```

</body>

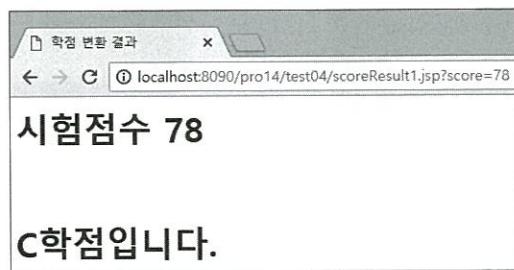
3. <http://localhost:8090/pro14/test04/scoreTest.jsp>로 요청하여 시험 점수 입력 페이지에서 점수를 입력하고 학점변환을 클릭합니다.

▼ 그림 14-58 시험 점수 입력 페이지



4. 변환된 학점을 출력합니다.

▼ 그림 14-59 시험 점수가 학점으로 변환



하지만 현재는 입력한 시험 점수가 0~100점 사이인지 유효성 검사를 하는 기능이 빠져 있으므로 학점 변환기 프로그램을 조금 수정해 보겠습니다.

5. 학점 변환 시 <c:choose> 태그를 이용해 시험 점수가 0~100점 사이인지를 먼저 체크한 후 전송된 시험 점수가 유효 범위이면 그 다음에 학점을 변환하도록 다음과 같이 scoreResult2.jsp를 작성합니다. 만약 유효 범위를 벗어나면 새로 점수를 입력하라는 메시지를 출력합니다.

코드 14-44 pro14/WebContent/test04/scoreResult2.jsp

```

...
<body>
<c:set var="score" value="${param.score }" />
<h1>시험점수
<c:out value="${score }" />
</h1><br>
<c:choose> ←———— <c:choose> 태그를 이용해 시험 점수의 유효성을 체크합니다.
<c:when test="${score}>=0 && score<=100 }">
<c:choose>
<c:when test="${score}>=90 && score<100 }">
<h1>A학점입니다.</h1>
</c:when>
<c:when test="${score}>=80 && score<90 }">
<h1>B학점입니다.</h1>
</c:when>
<c:when test="${score}>=70 && score<80 }">
<h1>C학점입니다.</h1>
</c:when>
<c:when test="${score}>=60 && score<70 }">
<h1>D학점입니다.</h1>
</c:when>
<c:otherwise>
<h1>F학점입니다.</h1>
</c:otherwise>
</c:choose>
</c:when>
<c:otherwise>
<h1>점수를 잘못 입력했습니다. 다시입력하세요</h1>
<a href="scoreTest.jsp">점수 입력 창으로 이동</a>
</c:otherwise>
</c:choose>
</body>

```

정상적인 점수이면 학점으로
변환됩니다.

시험 점수가 범위를 벗어났으면
입력 창으로 다시 이동합니다.

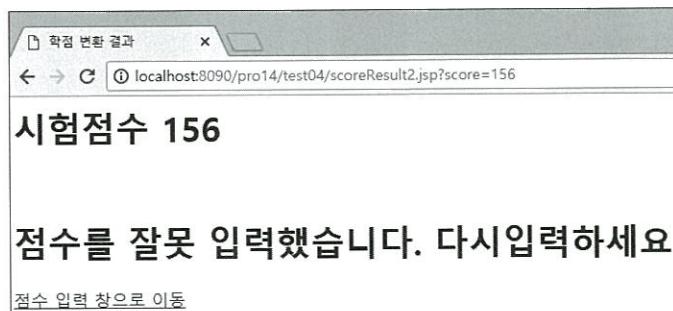
6. http://localhost:8090/pro14/test04/scoreTest.jsp로 다시 요청하여 정상적인 시험 점수를 입력합니다.

▼ 그림 14-60 정상적인 시험 점수의 학점 변환



7. 이번에는 시험 점수의 유효 범위인 0~100을 벗어나는 값을 일부러 입력합니다. 그러면 점수를 잘못 입력했으니 다시 입력하라는 메시지를 출력합니다.

▼ 그림 14-61 유효 범위를 벗어난 시험 점수를 입력했을 경우



14.7.3 구구단 출력 예제

이번에는 Core 태그 라이브러리를 사용한 구구단 출력 예제를 실습해 보겠습니다.

1. 다음과 같이 gugu.jsp를 작성합니다. 구구단 입력창에서 구구단 수를 입력한 후 입력한 단수를 guguResult1.jsp로 전송합니다.

코드 14-45 pro14/WebContent/test04/gugu.jsp

```
<body>
    <h1>출력할 구구단의 수를 지정해 주세요.</h1>
    <form method=get action='guguResult1.jsp'>———— 입력한 단수를 guguResult1.jsp로 전송합니다.
        출력할 구구단 : <input type=text name="dan" /> <br>
        <input type="submit" value="구구단 출력">
```

```
</form>  
</body>
```

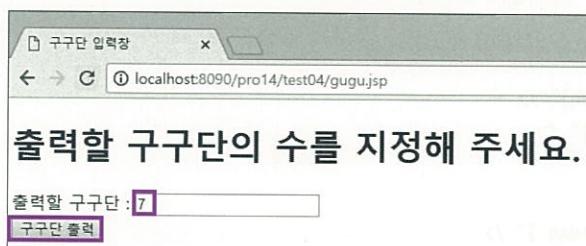
2. 전송된 단수를 가져와 <c:forEach> 태그를 이용해서 <tr> 태그에 연속적으로 구구단을 출력하도록 guguResult1.jsp를 작성합니다.

코드 14-46 pro14/WebContent/test04/guguResult1.jsp

```
...  
<body>  
    <c:set var="dan" value="${param.dan }" />  
    <table border="1" width="800" align="center">  
        <tr align="center" bgcolor="lightgreen">  
            <td colspan="2">  
                <c:out value="${dan}" />단 출력 </td>  
            </tr>  
  
            <c:forEach var="i" begin="1" end="9" step="1"> •—————<c:forEach> 태그를 이용해 구구단을  
                <tr align="center">  
                    <td width="400">  
                        <c:out value="${dan}" /> *————— 출력합니다.  
                        <c:out value="${i}" />  
                    </td>  
                    <td width="400">  
                        <c:out value="${i*dan }" />  
                    </td>  
                </tr>  
            </c:forEach>  
        </table>  
    </body>
```

3. <http://localhost:8090/pro14/test04/gugu.jsp>로 요청하여 구구단 입력창에서 단수를 입력한 후 구구단 출력을 클릭합니다.

▼ 그림 14-62 구구단 수 입력 후 구구단 출력 클릭



4. 전송된 단수를 이용해 구구단을 출력합니다.

▼ 그림 14-63 실행 결과

7단 출력	
7 * 1	7
7 * 2	14
7 * 3	21
7 * 4	28
7 * 5	35
7 * 6	42
7 * 7	49
7 * 8	56
7 * 9	63

5. 앞에서와 마찬가지로 이 예제를 응용해 보겠습니다. <c:if> 태그를 이용해 구구단을 출력하면서 테이블 각 행의 배경색을 교대로 출력하도록 수정해 보겠습니다. guguResult2.jsp를 다음과 같이 작성합니다.

코드 14-47 pro14/WebContent/test04/guguResult2.jsp

```
...
<body>
    <c:set var="dan" value="${param.dan }" />
    <table border="1" width="800" align="center">
        <tr align="center" bgcolor="lightgreen">
            <td colspan="2">
                <c:out value="${dan}" />단 출력 </td>
            </td>
        </tr>
        <c:forEach var="i" begin="1" end="9" step="1">
            <c:if test="${i%2==0 }">
                <tr align="center" bgcolor="#CCFF66">
            </c:if>
            <c:if test="${i%2==1 }">
                <tr align="center" bgcolor="#CCCCFF">
            </c:if>
            <td width="400">
                <c:out value="${dan}" /> *
                <c:out value="${i}" />
            </td>
            <td width="400">
                <c:out value="${i*dan }" />
            </td>
        </c:forEach>
    </table>
</body>
```

→ <c:forEach> 태그의 반복 변수 i가 짝수인지 짝수인지 체크하여 행의 배경색을 교대로 출력합니다.

```

</tr>
</c:forEach>
</table>
</body>

```

6. <http://localhost:8090/pro14/test04/gugu.jsp>로 요청하여 구구단 입력창에서 단수 입력 후 전송하면 행들의 색이 교대로 변경되어 출력됩니다.

▼ 그림 14-64 실행 결과: 행의 배경색이 교대로 출력

8단 출력	
8 * 1	8
8 * 2	16
8 * 3	24
8 * 4	32
8 * 5	40
8 * 6	48
8 * 7	56
8 * 8	64
8 * 9	72

14.7.4 이미지 리스트 출력 예제

이번에는 여러 가지 이미지와 체크박스를 연속적으로 출력하는 프로그램을 `<c:forEach>` 태그를 사용해 만들어 보겠습니다.

1. 다음과 같이 `imageList.jsp`를 작성합니다. `<c:forEach>` 태그를 이용해 `` 태그 안에 `` 태그를 연속해서 출력하여 이미지를 나타냅니다.

코드 14-48 pro14/WebContent/test04/imageList.jsp

```

...
<body>
<ul class="lst_type">
<li>
<span style='margin-left:50px'>이미지 </span>
<span>이미지 이름</span>
<span>선택하기</span>
</li>

```

```

<c:forEach var="i" begin="1" end="9" step="1">
    <li>
        <a href="#" style='margin-left:50px'>
            <img src='../image/duke.png' width='90' height='90' alt=''/></a>
            <a href="#">이미지 이름: 뒹크${i}</a>
            <a href="#">

```

〈c:forEach〉 태그를 이용해 이미지와 체크박스를
연속해서 나타냅니다.

2. <http://localhost:8090/pro14/test04/imageList.jsp>로 요청하여 실행 결과를 확인합니다.

▼ 그림 14-65 이미지와 체크박스가 연속해서 출력

이미지	이미지 이름	선택하기
	이미지 이름: 뒹크1	<input type="checkbox"/>
	이미지 이름: 뒹크2	<input type="checkbox"/>
	이미지 이름: 뒹크3	<input type="checkbox"/>
	이미지 이름: 뒹크4	<input type="checkbox"/>
	이미지 이름: 뒹크5	<input type="checkbox"/>

지금까지 Core 태그 라이브러리를 사용해 실습해 봤습니다. 현재 JSP 페이지는 JSTL로 구현하므로 라이브러리를 사용하는 방법에 익숙해지면 실제로 개발하는 데 많은 도움이 될 것입니다. 라이브러리를 사용해서 구현한 예제와 12장에서 스크립트릿으로 실습한 내용을 비교해 보세요.

14.8

다국어 태그 라이브러리 사용하기

7

여러 온라인 쇼핑몰을 이용하다 보면 간혹 영어나 일본어로 언어를 변환해서 표시해 주는 화면을 보았을 것입니다.

▼ 그림 14-66 다국어를 지원하는 온라인 쇼핑몰 사이트

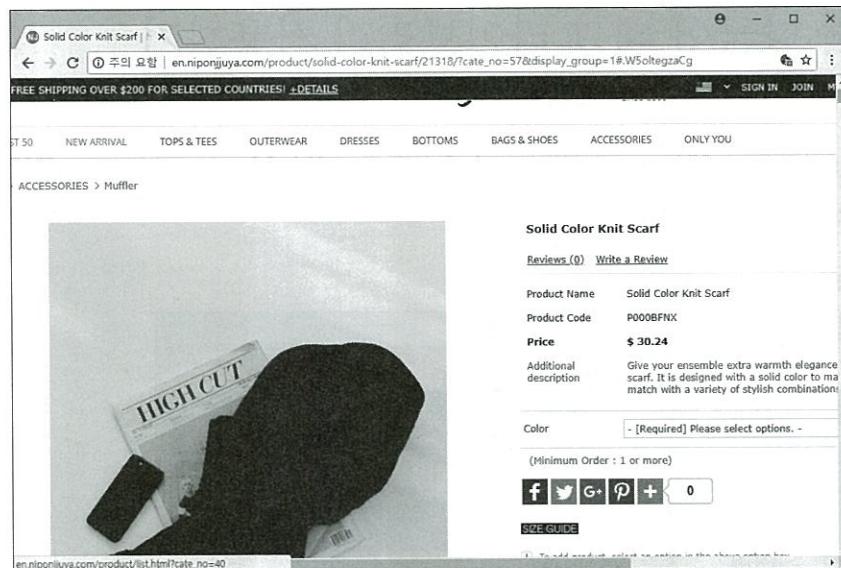


14

표현언어와 JSTL

이 쇼핑몰에서 해당 언어에 해당하는 국기를 클릭하면 다음과 같이 쇼핑몰 사이트가 해당 국가의 언어로 표시됩니다.

▼ 그림 14-67 해당 국가 언어로 변환된 모습



이러한 기능은 어떻게 구현하는 것일까요?

JSP에서 다국어 태그 라이브러리를 사용하면 다국어 기능을 쉽게 구현할 수 있습니다. 표 14-8은 JSP에서 다국어 기능을 구현하는 태그들입니다.

▼ 표 14-8 다국어 태그 라이브러리 종류

분류	태그	설명
다국어	<fmt:setLocale>	Locale(언어)을 지정합니다.
	<fmt:message>	지정한 언어에 해당하는 언어를 표시합니다.
	<fmt:setBundle>	사용할 번들을 지정합니다.
	<fmt:setParam>	전달할 매개변수를 지정합니다.
	<fmt:requestEncoding>	요청 매개변수의 문자 인코딩을 지정합니다.

실제로 다국어 태그 라이브러리를 어떻게 사용하는지는 14.9.3절에서 살펴보겠습니다.

14.9

한글을 아스키 코드로 변환하기

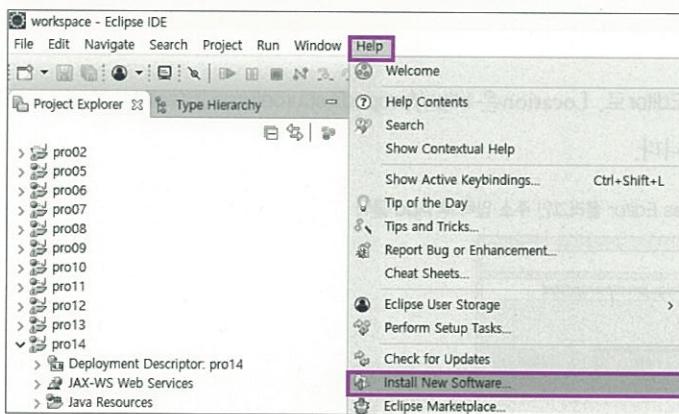
다국어 기능을 사용하려면 미리 한글을 아스키 코드로 변환한 형태로 저장하고 있다가 요청 시 이 아스키 코드를 다시 한글로 변환해서 표시합니다. 따라서 표시할 한글을 아스키 코드로 변환하는 방법부터 알아보겠습니다.

14.9.1 Properties Editor 설치하기

그럼 먼저 이클립스에 한글을 아스키 코드로 변환하는 기능을 제공하는 Properties Editor 플러그인을 설치합니다.

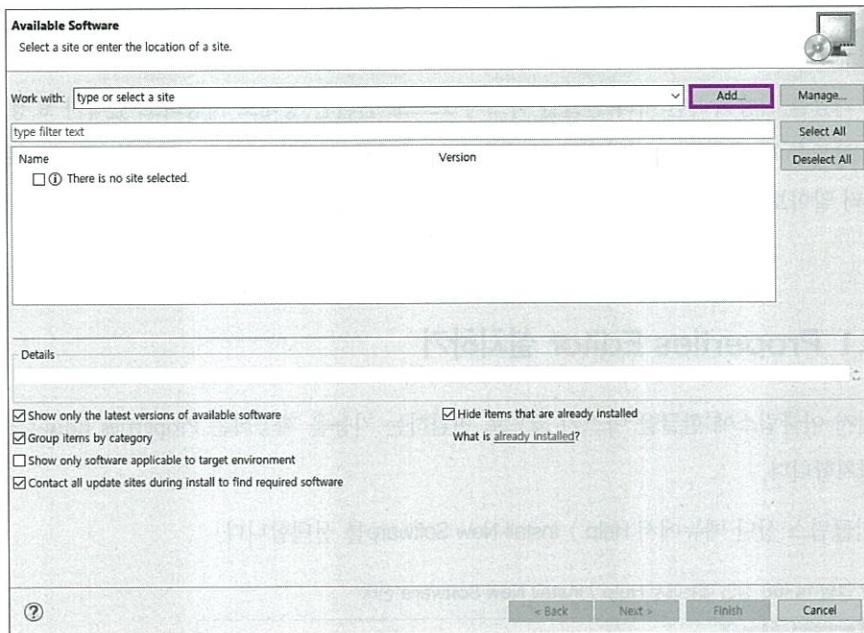
1. 이클립스 상단 메뉴에서 Help > Install New Software를 선택합니다.

▼ 그림 14-68 상단 메뉴에서 Help > Install New Software 선택



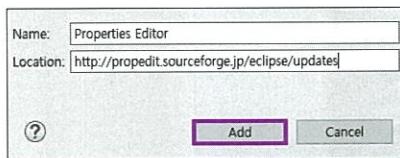
2. Add...를 클릭합니다.

▼ 그림 14-69 Add... 클릭



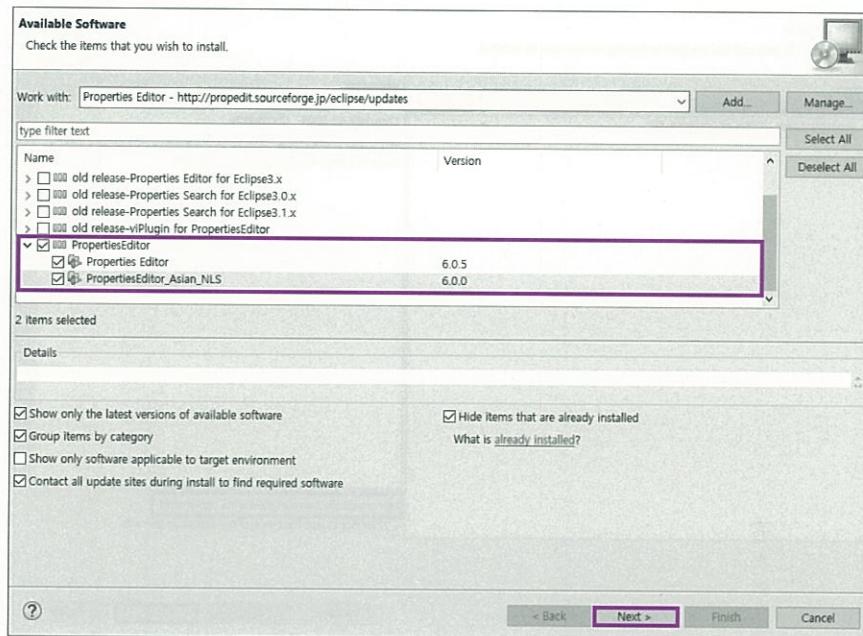
3. Name은 Properties Editor로, Location은 <http://propedit.sourceforge.jp/eclipse/updates>로 입력하고 Add를 클릭합니다.

▼ 그림 14-70 Properties Editor 플러그인 주소 입력 후 Add 클릭



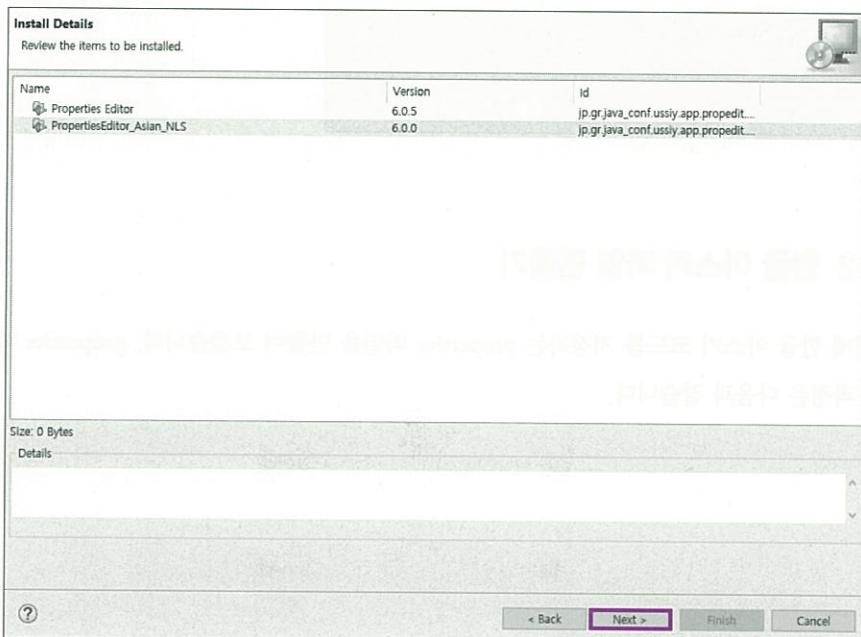
4. PropertiesEditor 항목을 선택한 후 Next를 클릭합니다.

▼ 그림 14-71 PropertiesEditor 선택 후 Next 클릭



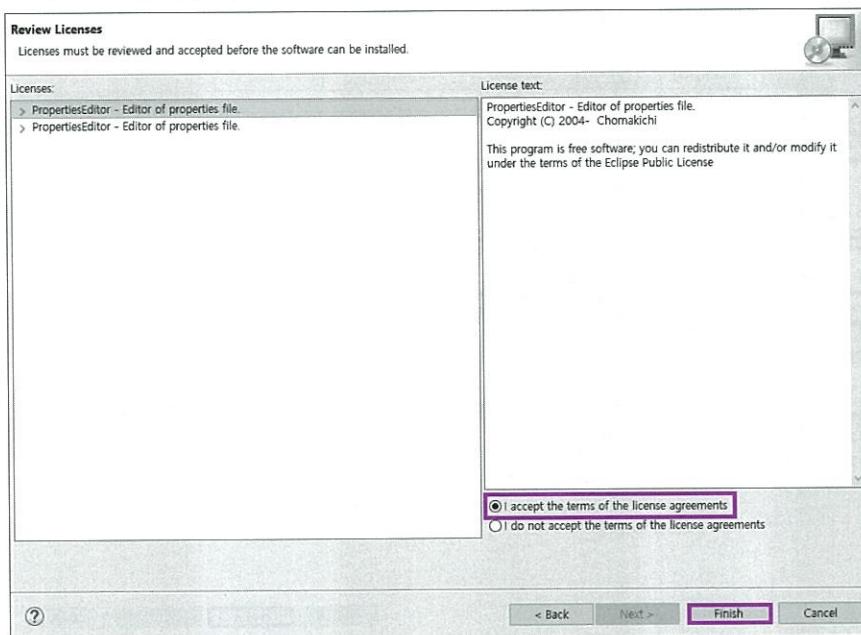
5. Install Details 화면이 나타나면 Next를 클릭합니다.

▼ 그림 14-72 Next 클릭



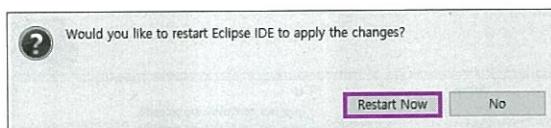
6. 라이선스 정책에 동의한 후 Finish를 클릭합니다.

▼ 그림 14-73 사용 동의에 체크 후 Finish 클릭



7. 설치 후 이클립스를 재실행할지 묻는 창이 나타나면 Restart Now를 클릭합니다

▼ 그림 14-74 이클립스 재실행 위해 Restart Now 클릭

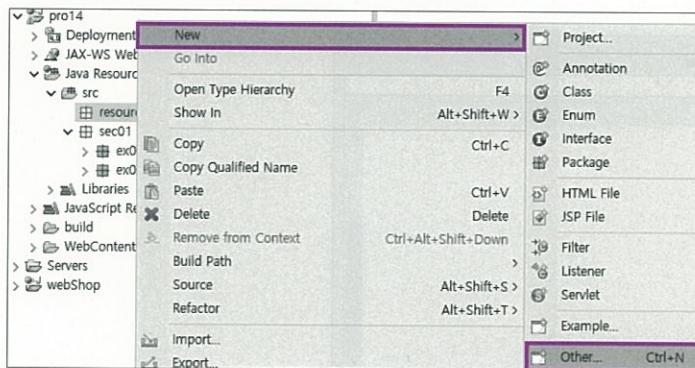


14.9.2 한글 아스키 파일 만들기

그럼 이제 한글 아스키 코드를 저장하는 properties 파일을 만들어 보겠습니다. properties 파일을 만드는 과정은 다음과 같습니다.

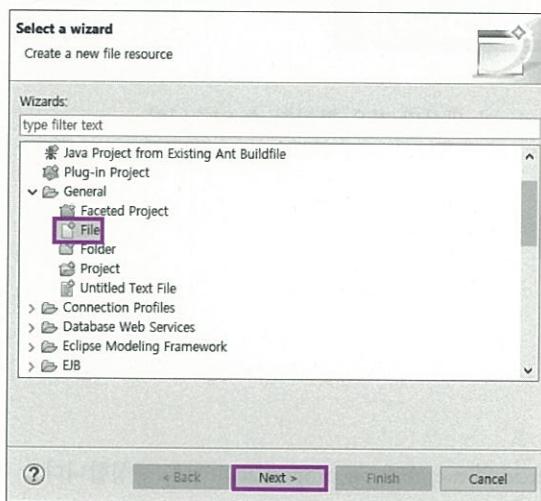
1. 프로젝트의 src 폴더에 resource 패키지를 생성하고 마우스 오른쪽 버튼을 클릭한 후 New > Other...를 선택합니다.

▼ 그림 14-75 resource 패키지 생성 후 New > Other... 선택



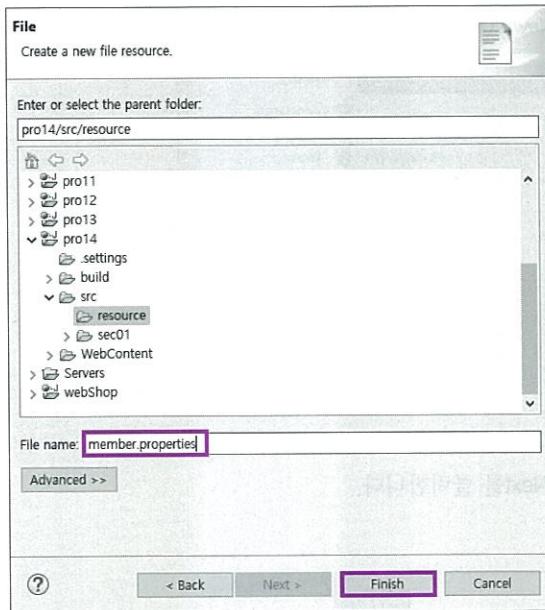
2. General 항목의 File을 선택한 후 Next를 클릭합니다.

▼ 그림 14-76 File 선택 후 Next 클릭



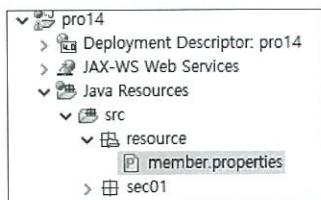
3. 파일 이름으로 member.properties를 입력한 후 Finish를 클릭합니다.

▼ 그림 14-77 파일 이름으로 member.properties 입력 후 Finish 클릭



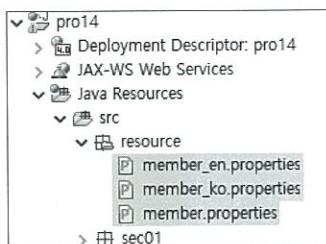
4. resource 패키지에 member.properties가 생성된 것을 확인할 수 있습니다.

▼ 그림 14-78 프로퍼티 파일 생성 확인



5. 같은 방법으로 member_ko.properties와 member_en.properties 파일을 생성합니다.

▼ 그림 14-79 다른 프로퍼티 파일 생성



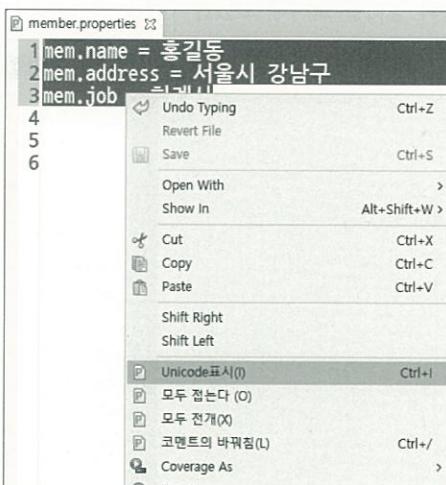
6. member.properties 파일을 열어 키/값 쌍으로 회원 정보를 한글로 작성한 후 저장합니다.

▼ 그림 14-80 member.properties에 한글로 정보 입력

```
1 mem.name = 홍길동
2 mem.address = 서울시 강남구
3 mem.job = 회계사
```

7. 작성한 회원 정보 전체를 마우스로 드래그한 후 오른쪽 버튼을 클릭해 Unicode표시를 선택합니다.

▼ 그림 14-81 전체 선택 후 Unicode표시 선택



8. 창에 표시된 아스키 코드 전체를 마우스로 드래그한 후 [Ctrl]+[C]를 눌러 복사합니다.

▼ 그림 14-82 한글이 변환된 아스키 코드 복사

```
mem.name = \ud64d\uae38\ub3d9
mem.address = \uc11c\uc6b8\uc2dc \uac15\ub0a8\ud6c
mem.job = \ud68c\uacc4\uc0ac
```

9. member_ko.properties 파일을 열어 붙여 넣은 후 저장합니다.

▼ 그림 14-83 member_ko.properties에 아스키 코드 붙여 넣기

```
1 mem.name = \ud64d\uae38\ub3d9
2 mem.address = \uc11c\uc6b8\uc2dc \uac15\ub0a8\ud6c
3 mem.job = \ud68c\uacc4\uc0ac
```

10. member_en.properties 파일을 열어 동일한 key에 대한 회원 정보를 영어로 따로 입력합니다.

▼ 그림 14-84 영어로 회원 정보 입력

```
member_en.properties
1 mem.name = hong kil-dong
2 mem.address = kang-name gu, seoul
3 mem.job = account
```

14.9.3 JSP 페이지에 다국어 표시하기

자, 그럼 앞에서 만든 아스키 코드를 이용해 한글과 영어를 표시해 보겠습니다.

1. 먼저 프로젝트의 WebContent 폴더에 test05 폴더를 만든 후 message1.jsp 파일을 저장합니다.

▼ 그림 14-85 다국어 관련 실습 파일 위치



2. message1.jsp 파일을 다음과 같이 작성합니다. <fmt:setLocale> 태그를 이용해 표시할 locale(언어)을 지정한 후 <fmt:bundle> 태그를 이용해 resource 패키지의 프로퍼티 파일을 읽어옵니다. 그리고 <fmt:message> 태그를 이용해 프로퍼티 파일의 키(key)에 대한 값을 각각 출력합니다.

코드 14-49 pro14/WebContent/test05/message1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
```

 〈fmt〉 태그를 이용하기 전에 반드시 설정해야 합니다.

```

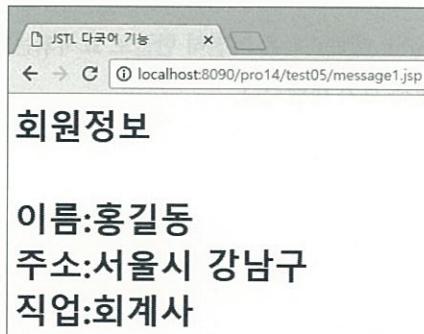
request.setCharacterEncoding("UTF-8");
%>

<html>
<head>
<meta charset="UTF-8">
<title>JSTL 다국어 기능</title>
</head>
<body>
<%-->
<fmt:setLocale value="en_US" /> ────────── locale을 영어로 지정합니다.
--%>
<fmt:setLocale value="ko_KR" /> ────────── locale을 한글로 지정합니다.
<h1>
회원정보<br><br>
<fmt:bundle basename="resource.member" > ────────── resource 패키지 아래 member
이름 <fmt:message key="mem.name" /><br> ────────── 프로퍼티 파일을 읽어옵니다.
주소:<fmt:message key="mem.address" /><br>
직업:<fmt:message key="mem.job" /> ────────── <fmt:message> 태그의 key 속성에 프로퍼티 파일의
</fmt:bundle> ────────── key를 지정하여 값(value)을 출력합니다.
</h1>
</body>
</html>

```

3. <http://localhost:8090/pro14/test05/message1.jsp>로 요청합니다. 최초 요청 시 한글로 회원 정보를 출력합니다.

▼ 그림 14-86 한글로 회원 정보 출력



4. message1.jsp에서 한글 locale을 주석 처리하고 영어 locale을 활성화합니다.

▼ 그림 14-87 영어 표시 설정

```
16 <fmt:setLocale value="en_US" />
17 <%-- <fmt:setLocale value="ko_KR" /> --%>
18
19
20 <h1>
21 회원정보<br><br>
22 <fmt:bundle basename="resource.member" >
23   이름:<fmt:message key="mem.name" /><br>
24   주소:<fmt:message key="mem.address" /><br>
25   직업:<fmt:message key="mem.job" />
26 </fmt:bundle>
```



제공하는 예제 파일에는 미리 설정되어 있으므로 해제하고 다시 테스트해 보세요!

5. 브라우저에서 다시 요청하면 다음과 같이 영어로 회원 정보를 출력합니다.

▼ 그림 14-88 영어로 회원 정보 출력



그런데 브라우저에 출력되는 ‘이름’, ‘주소’, ‘직업’과 JSP 제목(title)은 여전히 한글로 표시되는데, 이 부분은 여러분이 직접 다국어 기능을 이용해서 구현해 보기 바랍니다.

14.10

포매팅 태그 라이브러리 사용하기



쇼핑몰의 상품 가격은 숫자 데이터로 테이블에 저장됩니다. 그런데 쇼핑몰에서 상품 가격을 보면 보통 5,000원, 18,000원, 120,000원 등 세 자리마다 콤마(.)가 찍혀 있습니다. JSTL 포매팅 라이브러리를 사용하면 쉽게 원하는 형태로 숫자, 날짜, 문자열을 표시할 수 있습니다.

표 14-9는 숫자 또는 날짜와 관련된 포매팅 태그 라이브러리의 종류입니다.

▼ 표 14-9 포매팅 태그 라이브러리 종류

분류	태그	설명
포매팅	<fmt:timeZone>	둘 다 지정한 국가의 시간을 지정하는 태그입니다. 그러나 <fmt:timeZone> 태그의 경우 태그를 열고 닫는 영역 안에서만 적용된다는 차이점이 있습니다.
	<fmt:setTimeZone>	
	<fmt:formatNumber>	표시할 숫자의 형식을 지정합니다.
	<fmt:formatDate>	지정한 형식의 날짜를 표시합니다.

그리고 각각의 포매팅 태그 라이브러리들은 표 14-10, 표 14-11처럼 상세한 설정을 위해 여러 가지 속성을 가집니다.

▼ 표 14-10 <formatNumber> 태그의 여러 가지 속성

속성	설명
value	출력될 숫자를 지정합니다.
type	출력된 타입을 지정합니다. percent인 경우 %, number인 경우 숫자, currency인 경우 통화 형식으로 출력합니다.
dateStyle	날짜의 출력 형식을 지정합니다. DateFormat 클래스의 full, long, medium, short 등이 지정되어 있습니다.
groupingUsed	콤마(,)등 기호로 구분 여부를 지정합니다. 이 속성이 true이면 50000이 50,000으로 표시됩니다. 기본값은 true입니다.
currencyCode	통화 코드를 지정합니다. 한국 원화는 KRW입니다.
currentSymbol	통화를 표시할 때 사용할 기호를 표시합니다.
var	<formatNumber> 태그 결과를 저장할 변수의 이름을 지정합니다.
scope	변수의 접근 범위를 지정합니다.
pattern	숫자가 출력될 양식을 지정합니다. 자바의 DecimalFormat 클래스에 정의된 형식을 따릅니다.

▼ 표 14-11 <fmtDate> 태그의 여러 가지 속성

속성	설명
value	포맷될 날짜를 지정합니다.
type	포매팅할 타입을 지정합니다. date인 경우 날짜만, time인 경우 시간만, both인 경우 모두 지정합니다.
dateStyle	날짜의 출력 형식을 지정합니다. DateFormat 클래스의 full, long, medium, short 등이 지정되어 있습니다.
timeStyle	시간 출력 형식을 지정합니다. 자바 클래스 DateFormat에 정의된 형식을 사용합니다.
pattern	직접 출력 형식을 지정합니다. 자바 클래스 SimpleDateFormat에 지정된 패턴을 사용합니다.
timeZone	특정 나라 시간대로 시간을 설정합니다.

14.10.1 포매팅 태그 라이브러리 사용 실습

다음은 포매팅 태그 라이브러리를 사용하여 여러 가지 숫자와 날짜 정보를 표시해 보겠습니다.

1. 다음과 같이 formatTest.jsp 파일을 준비합니다.

▼ 그림 14-89 포매팅 태그 라이브러리 관련 실습 파일 위치



2. 다음과 같이 formatTest.jsp를 작성합니다. 변수 price를 <fmt:formatNumber> 태그를 이용해 숫자를 포매팅하고 price 값을 각각의 형식에 맞게 출력합니다. 이때 price의 값을 세 자리마다 콤마(,)로 구분해서 표시합니다. 단, groupingUsed를 false로 설정한 경우는 콤마(,)를 표시하지 않으며 <fmt:formatNumber> 태그의 var 속성에 설정한 priceNumber로 포매팅한 숫자를 표현 언어에서 출력합니다.

코드 14-50 pro14/WebContent/test05/formatTest.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    import="java.util.Date"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    request.setCharacterEncoding("UTF-8");
%>
<html>
<head>
    <meta charset="UTF-8">
    <title>포매팅 태그 라이브러리 예제</title>
</head>

<body>
    <h2>fmt의 number 태그를 이용한 숫자 포맷팅 예제.</h2>
    <c:set var="price" value="100000000" />
    <fmt:formatNumber value="${price}" type="number" var="priceNumber" />
    통화로 표현 시 : <fmt:formatNumber type="currency" currencySymbol="₩" value="${price}" groupingUsed="true" /><br>
    퍼센트로 표현 시 : <fmt:formatNumber value="${price}" type="percent" groupingUsed="false" /><br>
    일반 숫자로 표현 시 : ${priceNumber}<br>
    <h2>formatDate 예제</h2>
    <c:set var="now" value="<%=new Date() %>" />
    <fmt:formatDate value="${now }" type="date" dateStyle="full" /><br>
    <fmt:formatDate value="${now }" type="date" dateStyle="short" /><br>
    <fmt:formatDate value="${now }" type="time" /><br>
    <fmt:formatDate value="${now }" type="both" dateStyle="full" timeStyle="full" /><br>
    <fmt:formatDate value="${now }" pattern="YYYY-MM-dd :hh:mm:ss" /><br>
    <br><br>
    한국 현재 시간:

```

포매팅 태그 라이브러리를 사용하기 위해 반드시 선언해야 합니다.

숫자를 원화로 표시합니다.

세 자리마다 콤마(,)로 표시합니다. 설정하지 않으면 기본값이 true입니다.

groupingUsed가 false이므로 세 자리마다 콤마(,)가 표시되지 않습니다.

<fmt:formatDate> 태그에서 var 속성에 정한 변수 이름으로 표현 언어에서 출력합니다.

<fmt:formatDate> 태그의 pattern 속성에 출력한 날짜 포맷을 지정합니다.

```

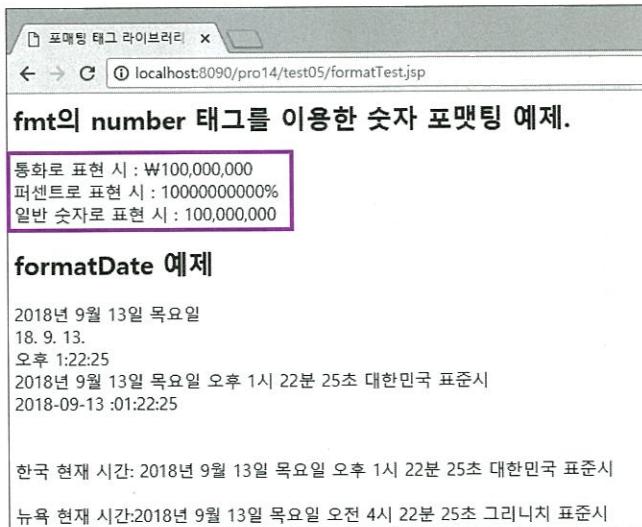
<fmt:formatDate value="${now }"
                 type="both" dateStyle="full" timeStyle="full" /><br><br>

<fmt:timeZone value="America/New York">———— 뉴욕 시간대로 변경합니다.
    뉴욕 현재 시간:
    <fmt:formatDate value="${now }"
                    type="both" dateStyle="full" timeStyle="full" /><br>
</fmt:timeZone>
</body>
</html>

```

3. <http://localhost:8090/pro14/test05/formatTest.jsp>로 요청하여 실행 결과를 확인합니다. 앞에서 설명했듯이 <fmt:formatNumber> 태그의 groupingUsed 속성을 false로 설정한 경우에는 콤마(,)가 표시되지 않습니다.

▼ 그림 14-90 실행 결과



14.11

문자열 처리 함수 사용하기

자바에서 문자열을 처리할 때 사용하는 문자열 관련 기능을 JSTL에서 제공하는 함수를 이용해 JSP에서도 사용할 수 있습니다.

표 14-12는 JSTL에서 제공하는 문자열 함수들입니다. 표에 나오지 않는 문자열 기능은 자바 String 클래스의 메서드 기능을 참고하기 바랍니다.

▼ 표 14-12 JSTL에서 제공하는 여러 가지 문자열 함수

함수	반환	설명
fn:contains(A,B)	boolean	문자열 A에 문자열 B가 포함되어 있는지 확인합니다.
fn:endsWith(A, B)	boolean	문자열 A의 끝이 B로 끝나는지 확인합니다.
fn:indexOf(A, B)	int	문자열 A에서 B가 처음으로 위치하는 인덱스(index)를 반환합니다.
fn:length(A)	int	문자열 A의 전체 길이를 반환합니다.
fn:replace(A, B, C)	String	문자열 A에서 B까지 해당되는 문자를 찾아 C로 변환합니다.
fn:toLowerCase(A)	String	A를 모두 소문자로 변환합니다.
fn:toUpperCase(A)	String	A를 모두 대문자로 변환합니다.
fn:substring(A, B, C)	String	A에서 인덱스 번호 B에서 C까지 해당하는 문자열을 반환합니다.
fn:split(A, B)	String[]	A에서 B에서 지정한 문자열로 나누어 배열로 반환합니다.
fn:trim(A)	String	문자열 A에서 앞뒤 공백을 제거합니다.

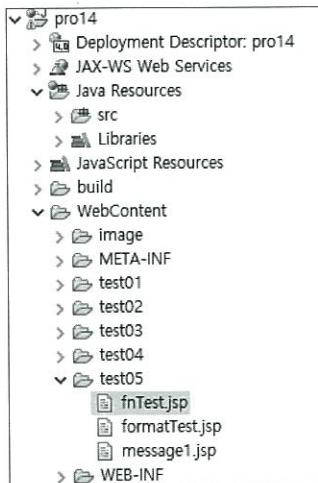
```

<%
    String[] arr = {"apple", "banana", "orange", "grape", "kiwi", "mango", "peach", "pear", "strawberry", "watermelon"};
    String str = "apple banana orange grape kiwi mango peach pear strawberry watermelon";
%>
<% for (String s : arr) { %>
    <div>${s}</div>
<% } %>
<% if (str.contains("apple")) { %>
    <div>${str}</div>
<% } %>
<% if (str.endsWith("apple")) { %>
    <div>${str}</div>
<% } %>
<% int index = str.indexOf("apple"); %>
<% if (index != -1) { %>
    <div>${index}</div>
<% } %>
<% String replaceStr = str.replace("apple", "orange"); %>
<% if (replaceStr.equals("orange banana orange orange kiwi orange peach orange pear orange strawberry orange watermelon")) { %>
    <div>${replaceStr}</div>
<% } %>
<% String lowerStr = str.toLowerCase(); %>
<% if (lowerStr.equals("apple banana orange grape kiwi mango peach pear strawberry watermelon")) { %>
    <div>${lowerStr}</div>
<% } %>
<% String upperStr = str.toUpperCase(); %>
<% if (upperStr.equals("APPLE BANANA ORANGE GRAPE KIWI MANGO PEACH PEAR STRAWBERRY WATERMELON")) { %>
    <div>${upperStr}</div>
<% } %>
<% String subStr = str.substring(0, 10); %>
<% if (subStr.equals("apple banana orange grape kiwi")) { %>
    <div>${subStr}</div>
<% } %>
<% String[] splitArr = str.split(" "); %>
<% if (splitArr.length == 10) { %>
    <div>${splitArr}</div>
<% } %>
<% String trimStr = str.trim(); %>
<% if (trimStr.equals("apple banana orange grape kiwi mango peach pear strawberry watermelon")) { %>
    <div>${trimStr}</div>
<% } %>
<% } %>

```

1. 그럼 JSTL의 문자열 함수를 사용하여 문자열을 출력하는 예제를 실습해 보겠습니다. 다음과 같이 실습 파일 fnTest.jsp를 준비합니다.

▼ 그림 14-91 JSTL의 문자열 함수 실습 파일 위치



2. fnTest.jsp를 다음과 같이 작성합니다. 문자열 함수를 사용하려면 먼저 taglib 디렉티브 태그를 선언해야 합니다. 문자열 관련 함수 기능은 자바의 String 클래스에서 제공하는 메서드 기능과 같습니다.

코드 14-51 pro14/WebContent/test05/fnTest.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"
   isELIgnored="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<
  request.setCharacterEncoding("utf-8");
%
<html>
<head>...</head>
<body>
  <c:set var="title1" value="hello world!" />
  <c:set var="title2" value="쇼핑몰 중심 JSP입니다!" />
  <c:set var="str1" value="중심" />
  <h2>여러 가지 문자열 함수 기능</h2>
  title1="hello world"<br>
  title2="쇼핑몰 중심 JSP 입니다.!"<br>
  str1="중심"<br><br>
```

함수를 사용하기 전에 반드시 선언합니다.

```

fn:length(title1) : ${fn:length(title1)} <br> 문자열 길이를 반환합니다.
fn:toUpperCase(title1) : ${fn:toUpperCase(title1)} <br> 문자열을 대문자로 변환합니다.
fn:toLowerCase(title1) : ${fn:toLowerCase(title1)} <br><br> 문자열을 소문자로 변환합니다.

fn:substring(title1,3,6) : ${fn:substring(title1,3,6)} <br>
fn:trim(title1) : ${fn:trim(title1)} <br> 문자열에서 4~5번째 문자열을 반환합니다.
fn:replace(title1, " ", "/") : ${fn:replace(title1, " ", "/")}<br><br>
fn:indexOf(title2,str1) : ${fn:indexOf(title2,str1)} <br> 문자열에서 str1의
fn:contains(title1,str1) : ${fn:contains(title1,str1)} <br> 위치를 구합니다.
fn:contains(title2,str1) : ${fn:contains(title2,str1)} <br>
</body> title1 문자열에 str1 문자열이 첫 번째 문자열이 두 번째 문자열을
</html> 있는지 판별합니다. 포함하는지 판별합니다.

```

3. <http://localhost:8090/pro14/test05/fnTest.jsp>로 요청하여 실행 결과를 확인합니다.

❖ 그림 14-92 실행 결과

```

여러 가지 문자열 함수 기능

title1="hello world"
title2="쇼핑몰 중심 JSP 입니다!"
str1="중심"

fn:length(title1)=12
fn:toUpperCase(title1)=HELLO WORLD!
fn:toLowerCase(title1)=hello world!

fn:substring(title1,3,6)=lo
fn:trim(title1)=hello world!
fn:replace(title1, " ","/")=hello/world!

fn:indexOf(title2,str1)=4
fn:contains(title1,str1)=false
fn:contains(title2,str1)=true

```

이처럼 JSTL의 문자열 함수를 이용하면 간단한 문자열은 바로 JSP에서 처리하여 사용할 수 있습니다. 다음은 표현 언어와 JSTL을 이용하여 회원 관리 프로그램을 실습해 보겠습니다.

14.12

표현 언어와 JSTL을 이용한 회원 관리 실습

표현 언어와 JSTL에 익숙해졌나요? 지금의 JSP는 표현 언어와 JSTL을 사용하여 개발합니다. 그럼 이번에는 좀 더 익숙해지기 위해 13장에서 실습한 회원 가입 및 조회 과정을 표현 언어와 JSTL을 이용하여 구현해 보겠습니다.

1. sec02.ex01 패키지를 만들고 13장에서 사용한 MemberBean 클래스와 MemberDAO 클래스를 복사해 붙여 넣습니다. 그리고 test06 폴더를 만들고 member_action.jsp, memberForm.jsp, memberList.jsp를 생성합니다.

▼ 그림 14-93 실습 파일 위치



2. memberForm.jsp를 다음과 같이 작성합니다. 회원 가입창에서 회원 정보를 입력한 후 action의 member_action.jsp로 전송합니다.

코드 14-52 pro14/WebContent/test06/memberForm.jsp

```
...
<body>
<form method="post" action="member_action.jsp">
<h1 style="text-align:center">회원 가입창</h1>
<table align="center">
<tr>
<td width="200">
<p align="right">아이디
</td>
<td width="400"><input type="text" name="id"></td>
</tr>
<tr>
<td width="200">
<p align="right">비밀번호
</td>
<td width="400"><input type="password" name="pwd"></td>
</tr>
...

```

회원 정보를 입력한 후 member_action.jsp로 전송합니다.

3. member_action.jsp를 다음과 같이 작성합니다. member_action.jsp는 화면 기능을 수행하지 않고 데이터베이스 연동 기능만 수행합니다. 회원 정보를 추가한 후 다시 회원 정보를 조회하고, 조회한 회원 정보를 request에 바인딩한 후 memberList.jsp로 포워딩합니다.

코드 14-53 pro14/WebContent/test06/member_action.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
import="java.util.* , sec02.ex01.*"
pageEncoding="UTF-8"
isELIgnored="false" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
request.setCharacterEncoding("UTF-8");
%>
<html>
<head>
<meta charset="UTF-8">
<jsp:useBean id="m" class="sec02.ex01.MemberBean" />
<jsp:setProperty name="m" property="*" />
```

```
<%  
    MemberDAO memDAO=new MemberDAO();  
    memDAO.addMember(m); • 회원 정보를 추가합니다.  
    List membersList =memDAO.listMembers(); • 회원 정보를 조회합니다.  
    request.setAttribute("membersList", membersList); • 조회한 회원 정보를 request에  
    바인딩합니다.  
%>  
</head>  
<body>  
    <jsp:forward page="membersList.jsp" /> • 다시 membersList.jsp로 포워딩합니다.  
</body>  
</html>
```

4. memberList.jsp를 다음과 같이 작성합니다. 자바 코드를 이용하지 않고 표현 언어와 JSTL만 사용하여 회원 정보를 표시합니다.

코드 14-54 pro14\WebContent\test06\memberList.jsp

```
<body>
<table align="center" border="1">
    <tr align="center" bgcolor="lightgreen">
        <td width="7%"><b>아이디</b></td>
        <td width="7%"><b>비밀번호</b></td>
        <td width="7%"><b>이름</b></td>
        <td width="7%"><b>이메일</b></td>
        <td width="7%"><b>가입일</b></td>
    </tr>
    <c:choose>
        <%-- 자바 코드를 사용하지 않습니다.
        ArrayList list =request.getAttribute("membersList");
        --%>
        <c:when test="${ membersList==null}">
            <tr>
                <td colspan=5>
                    <b>등록된 회원이 없습니다.</b>
                </td>
            </tr>
        </c:when>
        <c:when test="${membersList!= null}">
            <c:forEach var="mem" items="${membersList }">
                <tr align="center">
                    <td>${mem.id }</td>
                    <td>${mem.pwd}</td>
                    <td>${mem.name}</td>
                </tr>
            </c:forEach>
        </c:when>
    </c:choose>
</table>

```

```

<td>${mem.email}</td>
<td>${mem.joinDate}</td>
</tr>
</c:forEach>
</c:when>
</c:choose>
</table>
</body>

```

5. <http://localhost:8090/pro14/test06/memberForm.jsp>로 요청합니다. 회원가입창에서 회원 정보를 입력한 후 **가입하기**를 클릭합니다.

▼ 그림 14-94 회원가입창에서 회원 정보 입력 후 **가입하기** 클릭

▼ 그림 14-95 회원 목록 출력

아이디	비밀번호	이름	이메일	가입일
ki	1234	기성용	ki@test.com	2018-09-13
cha	1212	차명근	cha@test.com	2018-09-10
park2	1234	박지성	park2@test.com	2018-09-10
park	1234	박찬호	park@test.com	2018-09-04
kim	1212	김유신	kim@web.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
hong	1212	홍길동	hong@gmail.com	2018-09-04

지금까지 JSP에서 자바 코드를 사용하지 않고 표현 언어와 JSTL을 사용해 화면 기능을 어떻게 구현하는지 알아보았습니다. 우리가 13장에서 자바 코드를 사용해 실습한 회원 정보 출력 예제 코드와 비교해 보면 어떻게 다른지 더 잘 이해할 수 있을 것입니다.

1. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

2. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

3. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

4. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

5. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

6. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

7. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

8. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

9. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

10. 请根据以下提示，完成一篇短文。提示：你和你的家人、朋友一起度过了一个愉快的周末。星期六上午，你们去公园散步，下午去图书馆借书。星期天上午，你们去爬山，下午去电影院看电影。

15 장

JSP 페이지를 풍부하게 하는 오픈 소스 기능

15.1 JSP에서 파일 업로드

15.2 JSP에서 파일 다운로드

15.1 JSP에서 파일 업로드

1

지금까지 JSP와 직접 관련 있는 기능들에 대해 알아보았습니다. 15장에서는 오픈 소스 라이브러리로 제공되는 기능을 알아봅니다. JSP는 대부분의 기능을 오픈 소스로 제공합니다. 대표적인 기능이 파일 업로드와 파일 다운로드 기능이며, 이 외에도 이메일 등 수많은 오픈 소스 라이브러리를 제공하고 있습니다. 먼저 파일 업로드 기능부터 알아보겠습니다.

15.1.1 파일 업로드 라이브러리 설치

파일 업로드 기능을 사용하려면 오픈 소스 라이브러리를 설치해야 합니다. 파일 업로드 라이브러리를 설치하는 과정은 다음과 같습니다.

1. jakarta.apache.org로 접속한 후 왼쪽 메뉴에서 Commons를 클릭합니다.

▼ 그림 15-1 jakarta.apache.org로 접속 후 Commons 선택

The screenshot shows the homepage of The Apache Jakarta Project. At the top, there is a logo of a feather and the text "The Apache Jakarta Project" followed by the URL "http://jakarta.apache.org/". Below the logo, there are two main columns. The left column is titled "Support" and contains links to "License", "Mailing Lists", and "Jakarta Wiki". The right column is titled "Welcome to The Apache Jakarta™ Project" and contains a brief introduction about the project's history and structure. Further down, there is a "News" section with a heading "Latest Jakarta News" and a list of recent events. The link "Commons" under the "Support" column is highlighted with a purple rectangle, indicating it is the selected category.

Welcome to The Apache Jakarta™ Project	
Support	<ul style="list-style-type: none"> License Mailing Lists Jakarta Wiki
Ex-Jakarta	<ul style="list-style-type: none"> Ant Avalon BCEL BSF Commons DB Excalibur Gump HiveMind HttpComponents James JCS JMeter
News	<p><u>Latest Jakarta News</u></p> <ul style="list-style-type: none"> 21 December 2011 - Jakarta Retired 26 October 2011 - JMeter becomes a top level project 03 October 2011 - Apache JMeter 2.5.1 Released 11 September 2011 - BSF moves to Apache Commons 17 August 2011 - Apache JMeter 2.5 Released 05 August 2011 - Cactus moves to Apache Attic 25 June 2011 - JCS moves to Apache Commons 25 June 2011 - BCEL moves to Apache Commons 17 April 2011 - Reexpo is retired

2. 페이지 왼쪽 중간쯤에 위치한 FileUpload를 클릭합니다.

▼ 그림 15-2 FileUpload 클릭

DbUtils	JDBC helper library.
Digester	XML-to-Java-object mapping utility.
Email	Library for sending e-mail from Java.
Exec	API for dealing with external process execution and environment management in Java.
FileUpload	File upload capability for your servlets and web applications.
Functor	A functor is a function that can be manipulated as an object, or an object representing a single, generic function.
Geometry	Space and coordinates.
Imaging (previously called Sanselan)	A pure-Java image library.
IO	Collection of I/O utilities.

3. FileUpload 1.3.3 버전을 찾아서 here를 클릭합니다.

▼ 그림 15-3 다운로드 페이지에서 1.3.3 버전 선택

Downloading

Full Releases

FileUpload 1.3.3 - 13 June 2017

- Download the binary and source distributions from a mirror site [here](#)

FileUpload 1.3.2 - 26 May 2016

- Download the binary and source distributions from the archive site [here](#)

4. commons-fileupload-1.3.3-bin.zip을 클릭해 다운로드합니다.

▼ 그림 15-4 commons-fileupload-1.3.3-bin.zip 클릭

Apache Commons FileUpload 1.3.3 (requires Java 1.5+)

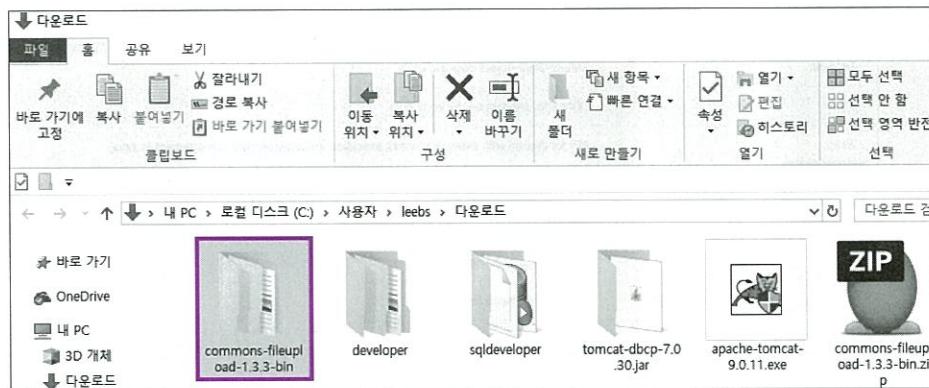
Binaries

[commons-fileupload-1.3.3-bin.tar.gz](#)

[commons-fileupload-1.3.3-bin.zip](#)

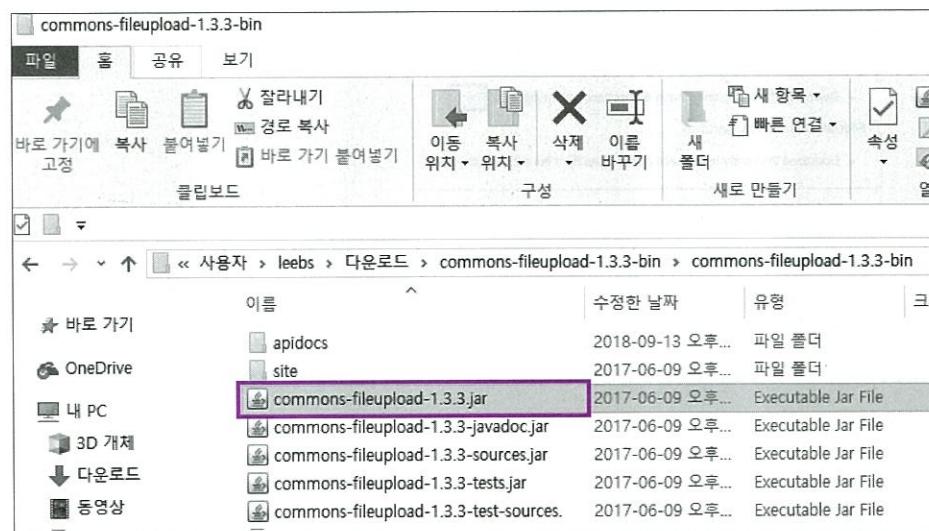
5. zip 파일의 압축을 풉니다.

▼ 그림 15-5 압축 파일 풀기



6. 압축을 푼 폴더의 하위 폴더인 commons-fileupload-1.3.3-bin에 위치한 commons-fileupload-1.3.3.jar 파일을 복사합니다.

▼ 그림 15-6 commons-fileupload-1.3.3.jar 파일 복사



7. 프로젝트 pro15의 WEB-INF 하위에 있는 lib 폴더에 붙여 넣습니다.

▼ 그림 15-7 commons-fileupload-1.3.3.jar 붙여 넣기

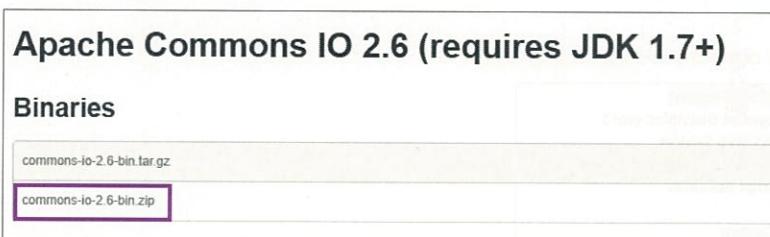


15.1.2 commons-io-2.6.jar 파일 설치

1. 다음 링크로 접속한 후 commons-io-2.6-bin.zip을 클릭해 다운로드합니다.

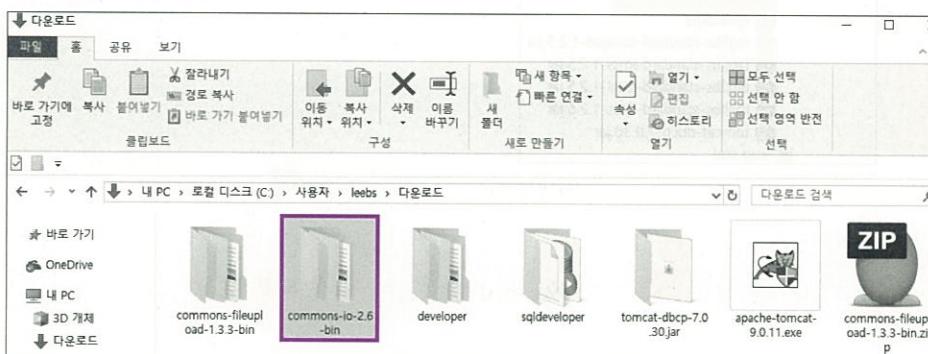
- https://commons.apache.org/proper/commons-io/download_io.cgi

▼ 그림 15-8 commons-io-2.6-bin.zip 클릭



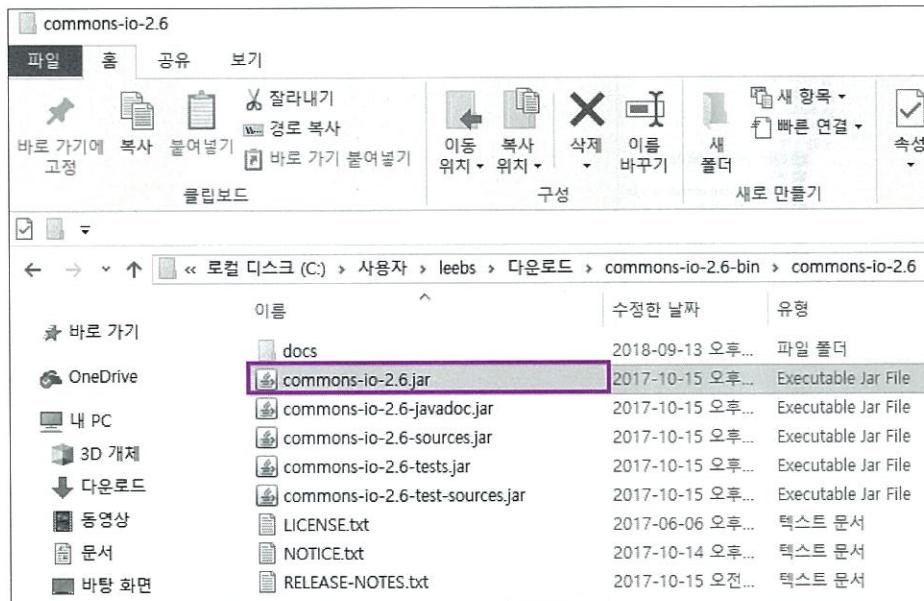
2. 로컬 PC의 여러분이 원하는 폴더에 zip 파일의 압축을 풉니다.

▼ 그림 15-9 압축 파일 풀기

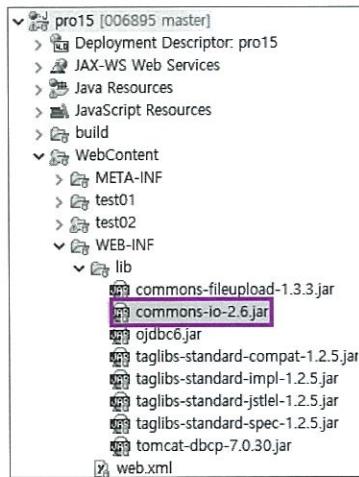


3. commons-io-2.6-bin 폴더로 이동한 후 commons-io-2.6.jar 파일을 복사해 이클립스 프로젝트의 WEB-INF/lib 폴더에 붙여 넣습니다.

▼ 그림 15-10 commons-io-2.6.jar 파일 복사



▼ 그림 15-11 commons-io-2.6.jar 파일 붙여 넣기



15.1.3 파일 업로드 관련 API

파일 업로드 라이브러리에서 제공하는 클래스에는 `DiskFileItemFactory`, `ServletFileUpload`가 있습니다. 각 클래스에서 제공하는 기능을 표 15-1, 15-2에 정리해 두었으니 참고하세요.

▼ 표 15-1 `DiskFileItemFactory` 클래스가 제공하는 메서드

메서드	기능
<code>setRepository()</code>	파일을 저장할 디렉터리를 설정합니다.
<code>setSizeThreadhold()</code>	최대 업로드 가능한 파일 크기를 설정합니다.

▼ 표 15-2 `ServletFileUpload` 클래스가 제공하는 메서드

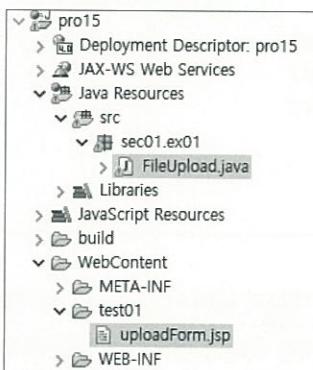
메서드	기능
<code>parseRequest()</code>	전송된 매개변수를 List 객체로 얻습니다.
<code>getItemIterator()</code>	전송된 매개변수를 Iterator 타입으로 얻습니다.

15.1.4 JSP 페이지에서 파일 업로드

이제 설치한 라이브러리를 이용해 파일을 업로드해 보겠습니다.

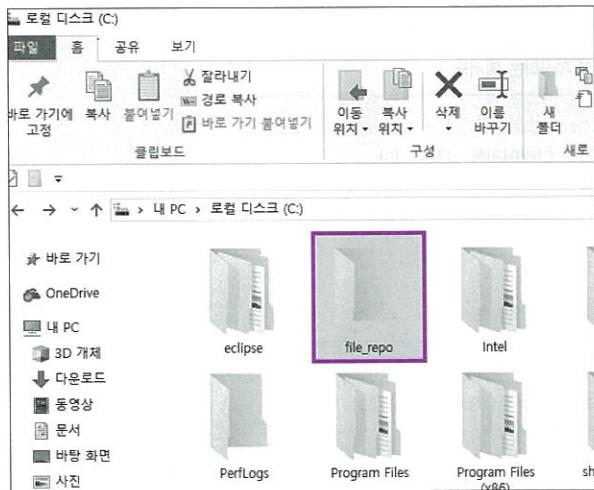
1. sec01.ex01 패키지를 만들고 `FileUpload` 클래스를 생성합니다. 또 test01 폴더를 생성하고 실습 파일 `uploadForm.jsp`를 추가합니다.

▼ 그림 15-12 실습 파일 위치



2. 파일을 업로드할 때 사용할 저장소를 다음과 같이 C 드라이브 아래에 만듭니다. 여기서는 폴더 이름을 file_repo로 하였습니다.

▼ 그림 15-13 업로드 파일 저장 폴더 생성



3. uploadForm.jsp를 다음과 같이 작성합니다. 파일 업로드창에서 파일을 업로드할 때 <form> 태그의 enctype 속성은 반드시 multipart/form-data로 지정해야 합니다.

코드 15-1 prot15\WebContent\test01\uploadForm.jsp

```
...
<body>
<form action="${contextPath}/upload.do"
      method="post" enctype="multipart/form-data">
    파일1: <input type="file" name="file1" ><br> 파일 업로드 시 반드시 enctype을 multipart/
    파일2: <input type="file" name="file2" > form-data로 설정해야 합니다.
    매개변수1: <input type="text" name="param1" > <br>
    매개변수2: <input type="text" name="param2" > <br>
    매개변수3: <input type="text" name="param3" > <br>
    <input type="submit" value="업로드" >
</form>
</body>
```

4. 파일 업로드를 처리하는 서블릿인 FileUpload 클래스를 다음과 같이 작성합니다. 라이브러리에서 제공하는 DiskFileItemFactory 클래스를 이용해 저장 위치와 업로드 가능한 최대 파일 크기를 설정합니다. 그리고 ServletFileUpload 클래스를 이용해 파일 업로드창에서 업로드된 파일과 매개변수에 대한 정보를 가져와 파일을 업로드하고 매개변수 값을 출력합니다.

코드 15-2 pro15/src/sec01/ex01/FileUpload.java

```

package sec01.ex01;
...
@WebServlet("/upload.do")
public class FileUpload extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doHandle(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doHandle(request, response);
    }
    private void doHandle(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        String encoding="utf-8";
        File currentDirPath =new File("C:\\file_repo"); ────────── 업로드할 파일 경로를 지정합니다.
        DiskFileItemFactory factory = new DiskFileItemFactory();
        factory.setRepository(currentDirPath); ────────── 파일 경로를 설정합니다.
        factory.setSizeThreshold(1024*1024); ────────── 최대 업로드 가능한 파일 크기를 설정합니다.
        ServletFileUpload upload=new ServletFileUpload(factory);
        try{
            List items = upload.parseRequest(request); ────────── request 객체에서 매개변수를 List로 가져옵니다.
            for(int i=0; i < items.size();i++) {
                FileItem fileItem = (FileItem) items.get(i); ────────── 파일 업로드창에서 업로드된 항목들을
                if(fileItem.isFormField()) { ────────── 하나씩 가져옵니다.
                    System.out.println(fileItem.getFieldName()+" = "+fileItem.getString(encoding));
                }else{
                    System.out.println("매개변수이름:"+fileItem.getFieldName());
                    System.out.println("파일이름:"+fileItem.getName());
                    System.out.println("파일크기:"+fileItem.getSize() + "bytes");
                    if(fileItem.getSize() > 0) {
                        int idx = fileItem.getName().lastIndexOf("\\");
                        if(idx ==-1) {
                            idx = fileItem.getName().lastIndexOf("/");
                        }
                        String fileName = fileItem.getName().substring(idx+1);
                        File uploadFile = new File(currentDirPath +"\\\" + fileName);
                        fileItem.write(uploadFile);
                    } //end if
                } //end if
            } //end for
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}

```

업로드할 파일 경로를 지정합니다.

파일 경로를 설정합니다.

최대 업로드 가능한 파일 크기를 설정합니다.

request 객체에서 매개변수를 List로 가져옵니다.

파일 업로드창에서 업로드된 항목들을 하나씩 가져옵니다.

폼 필드이면 전송된 매개변수 값을 출력합니다.

폼 필드가 아니면 파일 업로드 기능을 수행합니다.

업로드한 파일 이름을 가져옵니다.

업로드한 파일 이름으로 저장소에 파일을 업로드합니다.

```
    }  
}  
}
```

5. <http://localhost:8090/pro15/test01/uploadForm.jsp>로 요청하여 파일 업로드창을 엽니다.

그런 다음 이미지 파일(여기서는 duke.png와 duke2.jpg)을 첨부하고 해당하는 텍스트 필드 값 을 입력한 후 **업로드**를 클릭합니다.

▼ 그림 15-14 파일 업로드창에서 파일 첨부



6. 2번 과정에서 만든 파일 저장소(C:\file_repo)에 가면 업로드된 파일들을 볼 수 있습니다.

▼ 그림 15-15 파일 저장소에 업로드된 파일들



7. 또한 이클립스의 Console 탭을 보면 업로드한 매개변수 정보와 파일 정보가 출력된 것을 확인할 수 있습니다.

▼ 그림 15-16 업로드된 파일 정보와 매개변수 정보 출력

```
Tomcat v9.0 Server at localhost [Apache Tomcat]
매개변수명:file1
파일명:duke.png
파일크기:4437bytes
매개변수명:file2
파일명:duke2.png
파일크기:5103bytes
param1=홍길동
param2=1234
param3=hong@test.com
```

15.2 JSP에서 파일 다운로드

JAVA WEB

파일 업로드를 구현했으니 이번에는 서블릿 기능을 이용해 업로드한 파일을 다운로드하여 출력하는 예제를 실습해 보겠습니다.

1. 다음과 같이 sec01.ex02 패키지를 만들고 FileDownload 서블릿을 생성합니다. 이어서 test02 폴더를 만들고 실습 파일 first.jsp와 result.jsp를 추가합니다.

▼ 그림 15-17 실습 파일 위치



2. 첫 번째 JSP에서 다운로드할 이미지 파일 이름을 두 번째 JSP로 전달하도록 first.jsp를 작성합니다.

코드 15-3 pro15\WebContent\test02\first.jsp

```
...  
<body>  
  <form method="post" action="result.jsp" >  
    <input type=hidden name="param1" value="duke.png" /> <br>  
    <input type=hidden name="param2" value="duke2.jpg" /> <br>  
    <input type ="submit" value="이미지 다운로드">  
  </form>  
</body>
```

다운로드할 파일 이름을 매개변수로 전달합니다
(duke.png나 duke2.png가 아닌 다른 파일을 업로드
했다면 해당 파일 이름으로 수정하세요).

3. 두 번째 JSP인 result.jsp를 다음과 같이 작성합니다. 이미지 파일 표시창에서 태그의 src 속성에 다운로드를 요청할 서블릿 이름 download.do와 파일 이름을 GET 방식으로 전달합니다. 다운로드한 이미지 파일을 바로 태그에 표시하고, <a> 태그를 클릭해 서블릿에 다운로드를 요청하면 파일 전체를 로컬 PC에 다운로드합니다.

코드 15-4 pro15/WebContent/test02/result.jsp

```

<br>
</c:if>
<br>
<c:if test="${not empty file2 }">
    <br>
</c:if>
파일 내려받기 :<br>
<a href="${contextPath}/download.do?fileName=${file2}"> 파일 내려받기 </a><br>
</body>
</html>

```

4. 파일 다운로드 기능을 할 서블릿인 FileDownload 클래스를 다음과 같이 작성합니다. 파일 다운로드 기능은 자바 IO를 이용해 구현합니다. 먼저 response.getOutputStream();를 호출해 OutputStream을 가져옵니다. 그리고 배열로 버퍼를 만든 후 while 반복문을 이용해 파일에서 데이터를 한 번에 8KB씩 버퍼에 읽어옵니다. 이어서 OutputStream의 write() 메서드를 이용해 다시 브라우저로 출력합니다.

코드 15-5 pro15/src/sec01/ex02/FileDownload.java

```

package sec01.ex02;
...
@WebServlet("/download.do")
public class FileDownload extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{
        doHandle(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{
        doHandle(request, response);
    }
    private void doHandle(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html; charset=utf-8");
        String file_repo="C:\\file_repo";           매개변수로 전송된 파일 이름을 읽어옵니다.
        String fileName = (String)request.getParameter("fileName");
        System.out.println("fileName="+fileName);   response에서 OutputStream
        OutputStream out = response.getOutputStream(); 객체를 가져옵니다.
        String downFile=file_repo+"\\"+fileName;      파일을 다운로드할 수 있습니다.
        File f=new File(downFile);
        response.setHeader("Cache-Control", "no-cache");
        response.addHeader("Content-disposition", "attachment; fileName="+fileName);

```

```
FileInputStream in=new FileInputStream(f);
byte[] buffer=new byte[1024*8];
while(true) {
    int count=in.read(buffer);
    if(count==-1)
        break;
    out.write(buffer,0,count);
}
in.close();
out.close();
}
```

버퍼 기능을 이용해 파일에서 버퍼로 데이터를
읽어와 한꺼번에 출력합니다.

5. <http://localhost:8090/pro15/test02/first.jsp>로 요청한 후 이미지 다운로드를 클릭합니다.

▼ 그림 15-18 이미지 다운로드 클릭

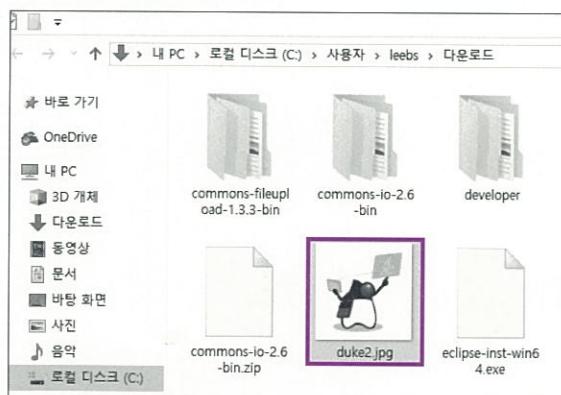


6. 업로드한 이미지가 브라우저에 출력되면 파일 내려받기를 클릭해 로컬 PC에 파일을 저장합니다.

▼ 그림 15-19 화면에 이미지가 출력되면 파일 내려받기 클릭



▼ 그림 15-20 로컬 PC에 파일 저장



JSP는 수많은 오픈 소스 라이브러리를 제공합니다. 15장에서 다루지 않은 그 외의 라이브러리들은 이 책 뒷부분에서 스프링 프레임워크를 배울 때 상세히 알아보겠습니다.

16장에서는 JSP의 화면을 구현하는 요소들의 추가된 기능(HTML 태그나 자바스크립트에 많은 변화가 있었습니다)에 대해 좀 더 알아보겠습니다.