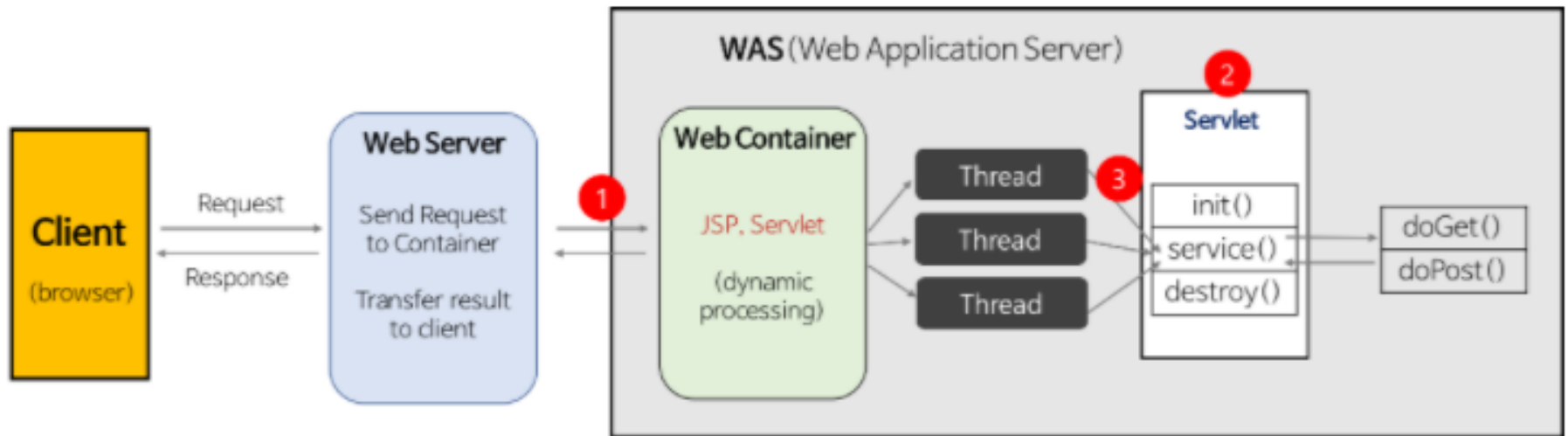


Servlet

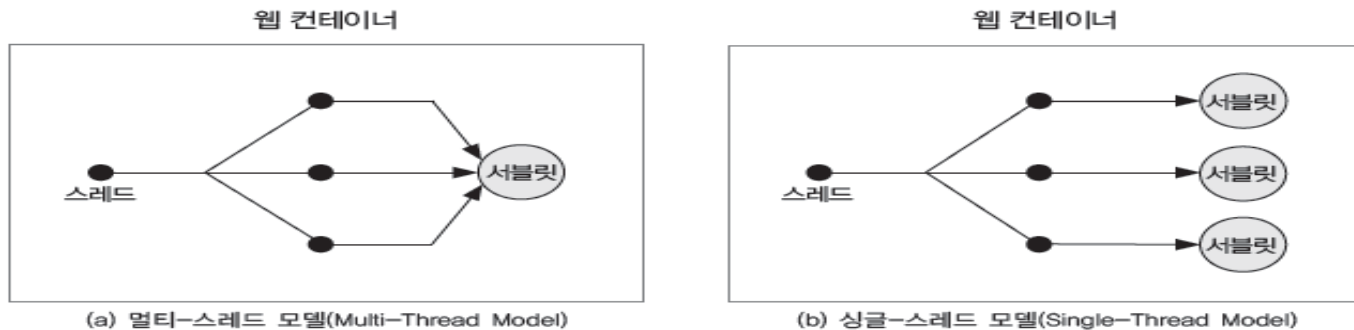
1. 서블릿이란?

- 클라이언트 요청을 처리하고 그 결과를 다시 클라이언트에게 전송하는 기능을 하는 클래스
- 서블릿 컨테이너
 - 서블릿 컨테이너는 서블릿의 생명주기를 관리하고 요청에 따른 스레드를 생성
 - 멀티 스레드로 동작



1. 서블릿이란?

- 멀티스레드 모델의 장점: 필요한 서블릿의 수가 적기 때문에 서블릿을 만들기 위해 필요한 시스템 자원과 서블릿이 차지하는 메모리를 절약할 수 있다.
단점: 여러 스레드가 동시에 한 서블릿을 사용하기 때문에 데이터 공유 문제에 신경을 써야 한다.



[그림 2-2] 멀티-스레드 모델과 싱글-스레드 모델

- 싱글-스레드 모델에서는 데이터 공유 문제를 걱정할 필요가 없지만 시스템 자원과 메모리가 더 많이 소모된다.

1. 서블릿이란?

- **서블릿과 서블릿 컨테이너**

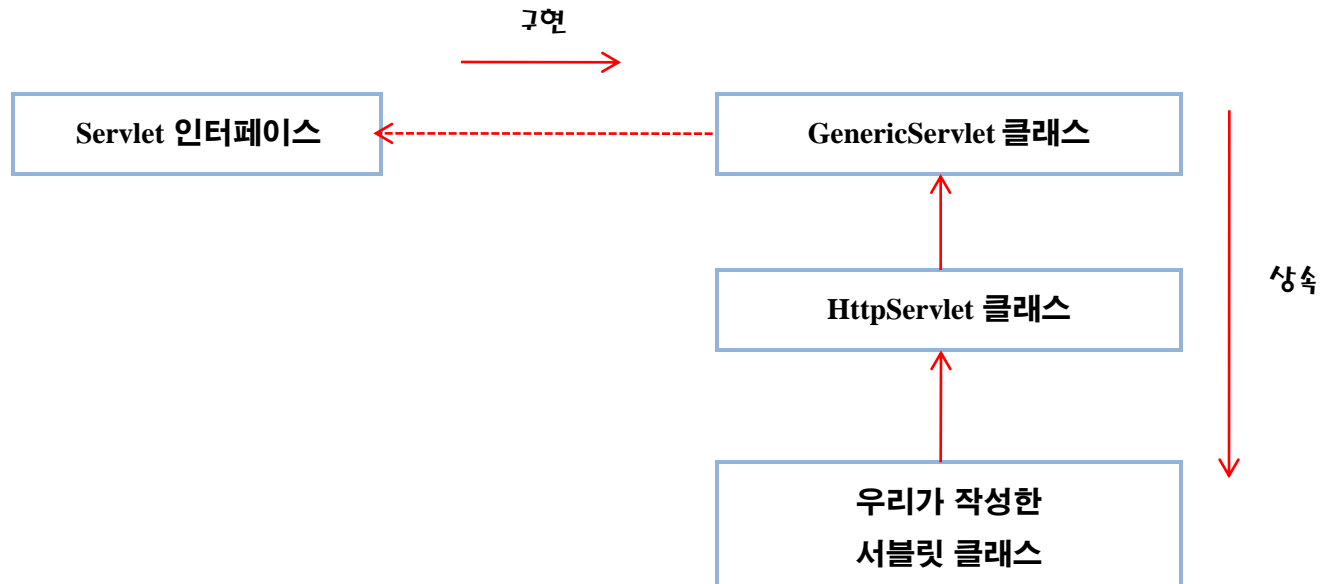
- 서블릿 컨테이너는 서블릿을 실행하기 위한 서버 소프트웨어를 말하는 것으로 JSP나 서블릿으로 만들어진 웹 프로그램을 개발하고 실행하기 위한 환경임.
- 아파치 톰캣이 대표적임.

[표 4-1] 웹 서버와 서블릿 컨테이너

구분	웹 서버	서블릿 컨테이너
사용목적	웹 서비스를 제공하기 위해 필요한 서버 기반의 소프트웨어다.	서블릿으로 개발된 자바 프로그램을 실행하고 처리하기 위한 서버 기반의 소프트웨어다.
처리 콘텐츠	HTML, CSS, 자바스크립트, 이미지 파일 등이다.	서블릿 클래스다.
실행 방법	콘텐츠가 위치한 URL 요청에 의해 실행하며 요청할 때마다 매번 디스크에서 읽어 처리한다.	서블릿 클래스 정보에 따라 서버에 매핑된 URL 정보에 따라 실행하며 컨테이너에 적재된 상태에서 처리한다.
JSP 실행	자체로 처리할 수 없다. 서블릿 컨테이너로 처리를 넘긴다.	JSP 자체로 처리할 수 있다.
특징	웹 서비스 제공을 위한 다양한 설정을 제공하기 때문에 서버를 유연하게 운영하려면 웹 서버를 사용해야 한다.	컨테이너에 따라 기본적인 웹 서버 기능을 내장하고 있으나 고급 설정이나 성능이 떨어지기 때문에 웹 서버와 병행해서 사용할 것을 권장한다.

2. 서블릿 클래스의 작성, 컴파일, 설치, 등록

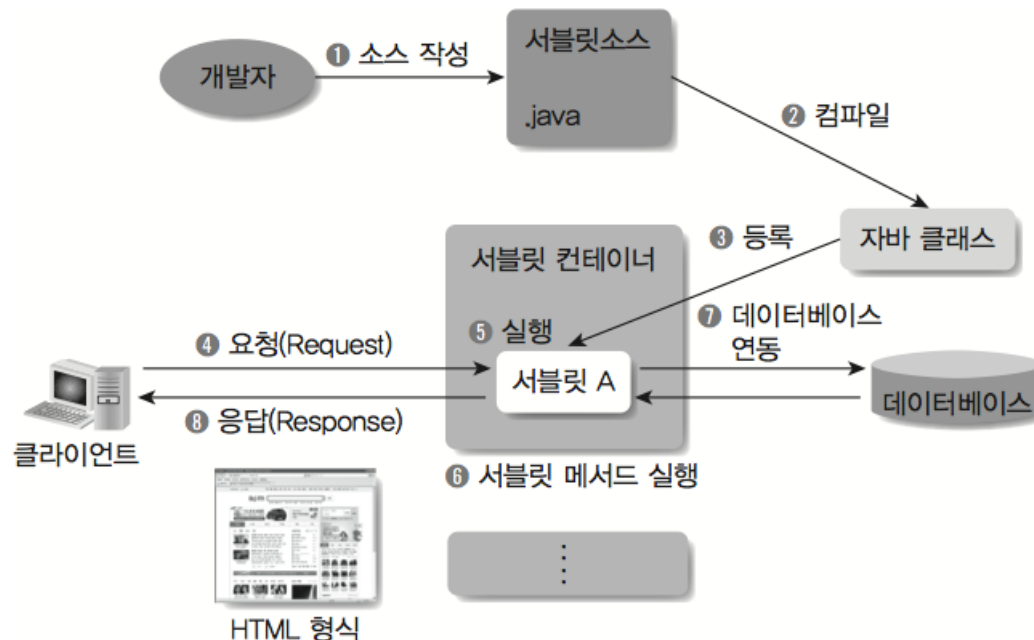
- 서블릿 클래스의 작성을 위한 준비
 - 서블릿 클래스를 작성할 때 지켜야 할 규칙 세 가지
 - 서블릿 클래스는 **javax.servlet.http.HttpServlet** 클래스를 상속하도록 만들어야 한다
 - **doGet** 또는 **doPost** 메서드 안에 웹 브라우저로부터 요청이 왔을 때 해야 할 일을 기술해야 한다
 - **HTML** 문서는 **doGet**, **doPost** 메서드의 두 번째 파라미터를 이용해서 출력해야 한다



[그림 2-3] 서블릿 클래스의 상속/구현 관계

3. 서블릿 동작 과정

- 서블릿은 개발자가 소스 작성 후 컴파일 과정을 거쳐 컨테이너에 배치(deploy)하게 되면 컨테이너에 의해 실행되어 관리된다.
- 이후 사용자 요청에 따라 스레드 단위로 실행되면서 데이터베이스 연동 등 필요한 작업을 수행하고 처리 결과를 사용자에게 HTML 형식으로 전달하는 구조로 동작한다.



[그림 4-1] 서블릿 개발과 실행 과정

2. 서블릿 생명주기

❖ 서블릿 초기화 : `init()` 메서드

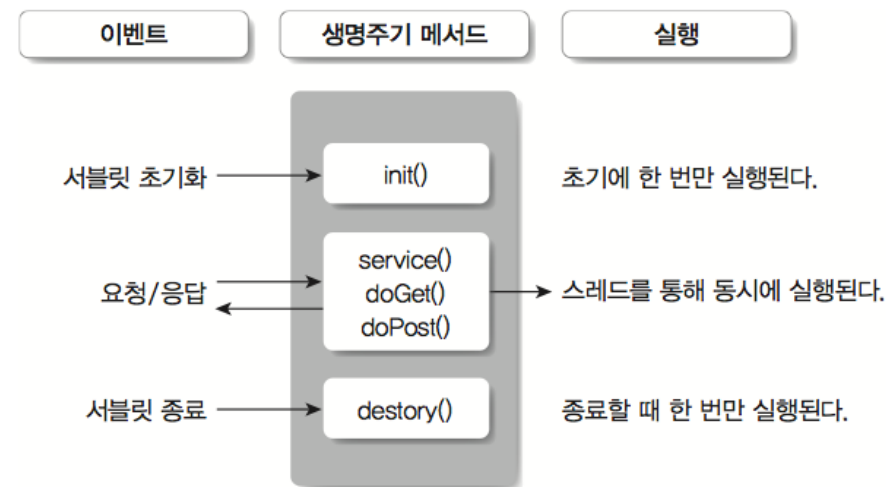
- 서블릿 실행시 호출되는 메서드로 초기에 한 번만 실행된다. 공통적으로 필요한 작업 등 수행

❖ 요청/응답 : `service()` 메서드

- 사용자 요청에 따라 스레드로 실행되는 메서드로 각각 `service()` 메서드를 통해 `doGet()` 혹은 `doPost()` 메서드가 호출된다.
- 파라미터인 `HttpServletRequest` 와 `HttpServletResponse` 를 통해 사용자 요청을 처리한다.

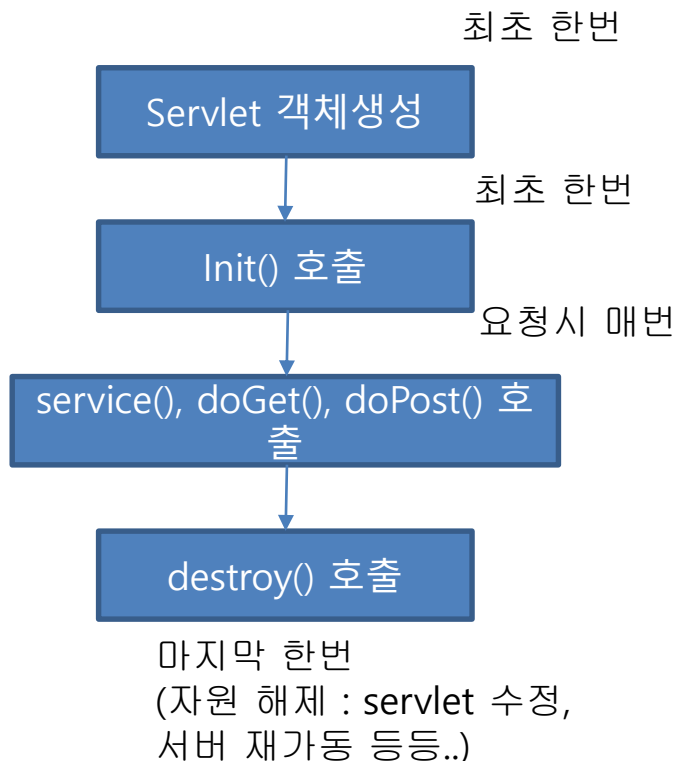
❖ 서블릿 종료 : `destroy()` 메서드

- 컨테이너로부터 서블릿 종료 요청이 있을 때 호출되는 메서드.
- `init()`와 마찬가지로 한 번만 실행되며, 서블릿이 종료되면서 정리할 작업이 있다면 `destroy()` 를 오버라이딩해서 구현함.

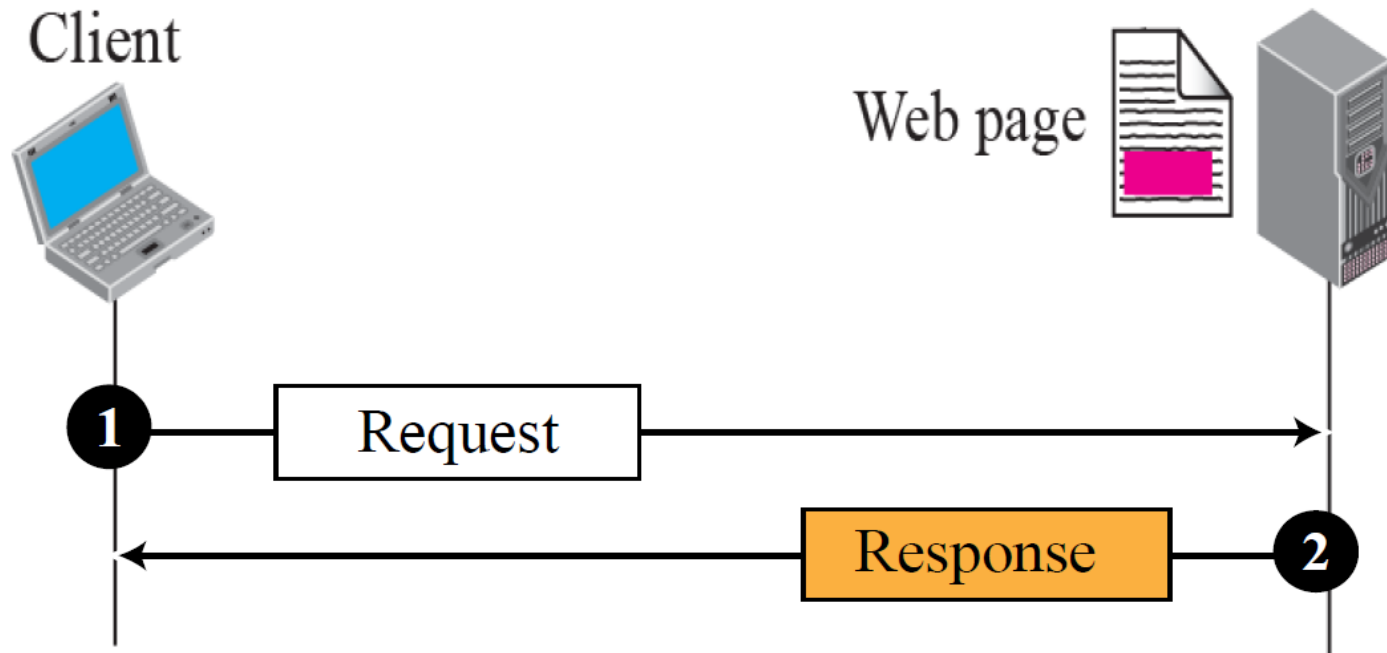


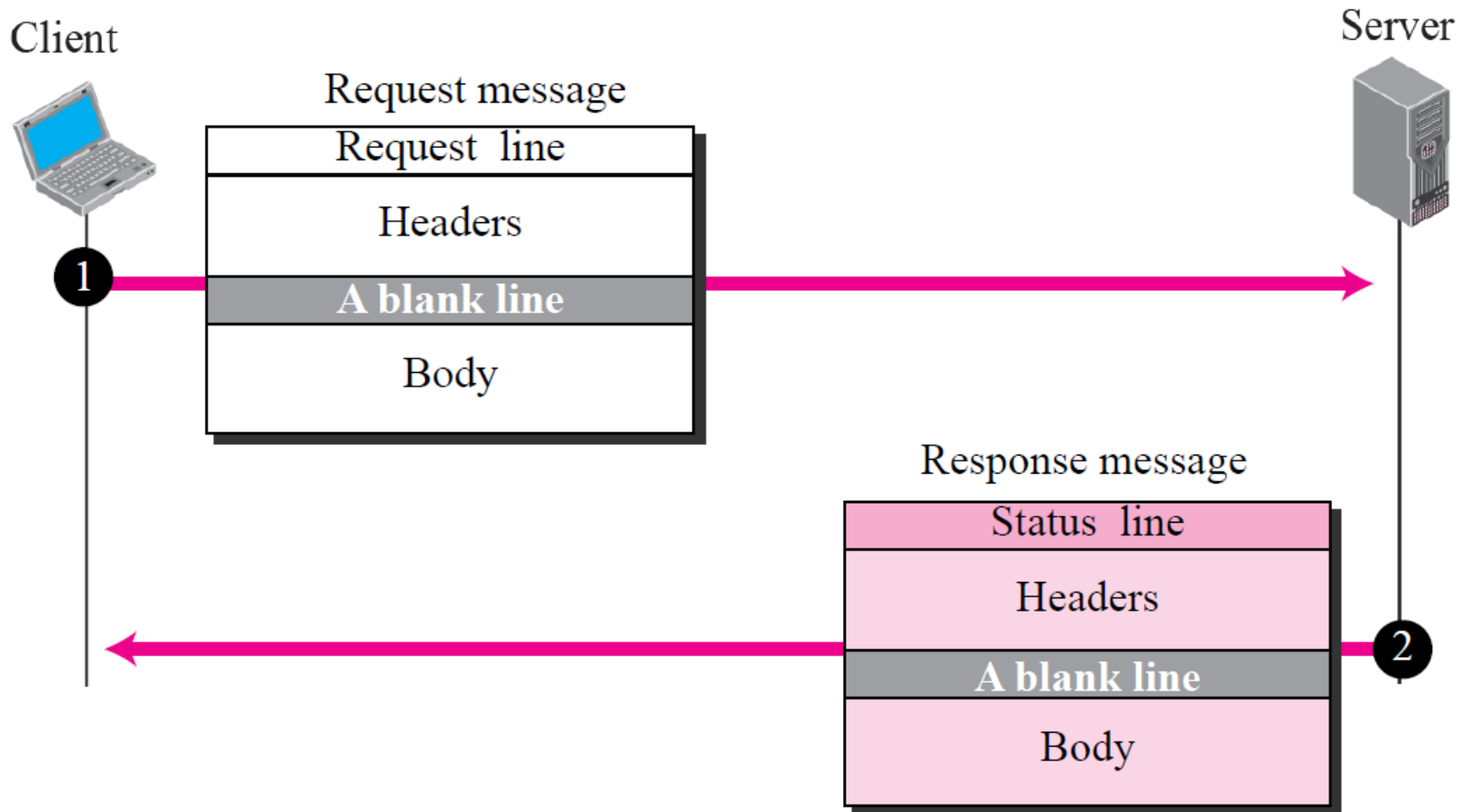
[그림 4-5] 서블릿 생명주기

- Servlet의 사용도가 높은 이유는 빠른 응답 속도 때문임
- Servlet은 최초 요청 시 객체가 만들어져 메모리에 로딩되고, 이후 요청 시에는 기존의 객체를 재활용하게 됨 -> 따라서 동작 속도가 빠름



```
public LifecycleEx() {  
    super();  
    // TODO Auto-generated constructor stub  
}  
  
@Override  
public void service(ServletRequest arg0, ServletResponse arg1)  
    throws ServletException, IOException {  
    // TODO Auto-generated method stub  
    System.out.println("service");  
}  
  
@Override  
public void init() throws ServletException {  
    // TODO Auto-generated method stub  
    System.out.println("init");  
}  
  
@Override  
public void destroy() {  
    // TODO Auto-generated method stub  
    System.out.println("destroy");  
}  
  
/**  
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  
 */  
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
    // TODO Auto-generated method stub  
    System.out.println("doGet");  
}
```



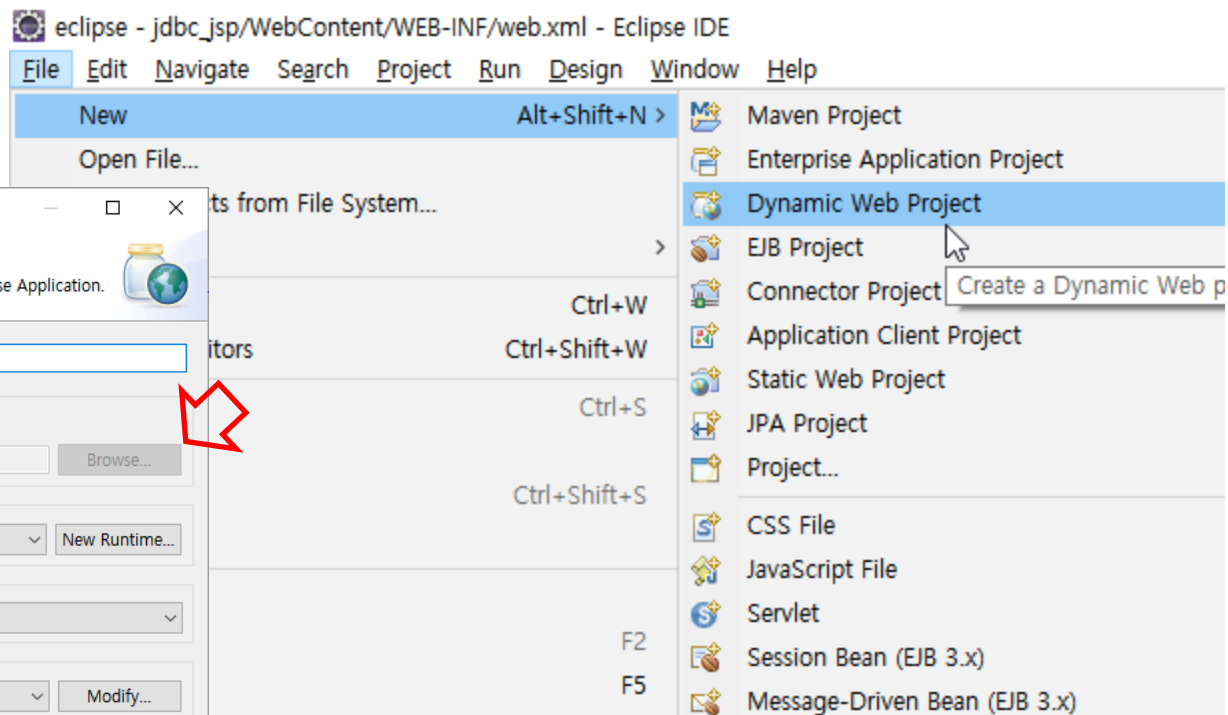
- GET 방식

- 서버에 있는 정보를 클라이언트로 가져오기 위한 방법이다. 예를 들어 HTML, 이미지 등을 웹 브라우저에서 보기 위한 요청.
- 서버에는 최대 240Byte까지 데이터를 전달할 수 있다.
- QUERY_STRING 환경변수를 통해서 서버로 전달되는데, 다음 형식을 따른다.
 - `http://www.xxx.co.kr/servlet/login?id=hj&name=hong`
- ‘?’ 이후의 값들은 서버에서 QUERY_STRING을 통해 전달된다. ‘속성=값’ 형태로 사용해야 하며 ‘&’는 여러 속성 값을 전달할 때 연결해주는 문자열이다.
- URL이 노출되기 때문에 보안에 문제가 생길 수 있다.

- POST 방식

- 서버로 정보를 올리기 위해 설계된 방법이다. 예를 들어 HTML 폼에 입력한 내용을 서버에 전달하기 위한 요청.
- 서버에 전달 할 수 있는 데이터 크기에는 제한이 없다.
- URL에는 매개변수가 표시되지 않는다.

- 프로젝트 생성



New Dynamic Web Project

Java
Configure project for building a Java application.

Source folders on build path:

src

Add Folder...
Edit...
Remove

Default output folder:
build\classes

< Back Next > Finish Cancel

New Dynamic Web Project

Web Module
Configure web module settings.

Context root: servlet_lecture

Content directory: WebContent

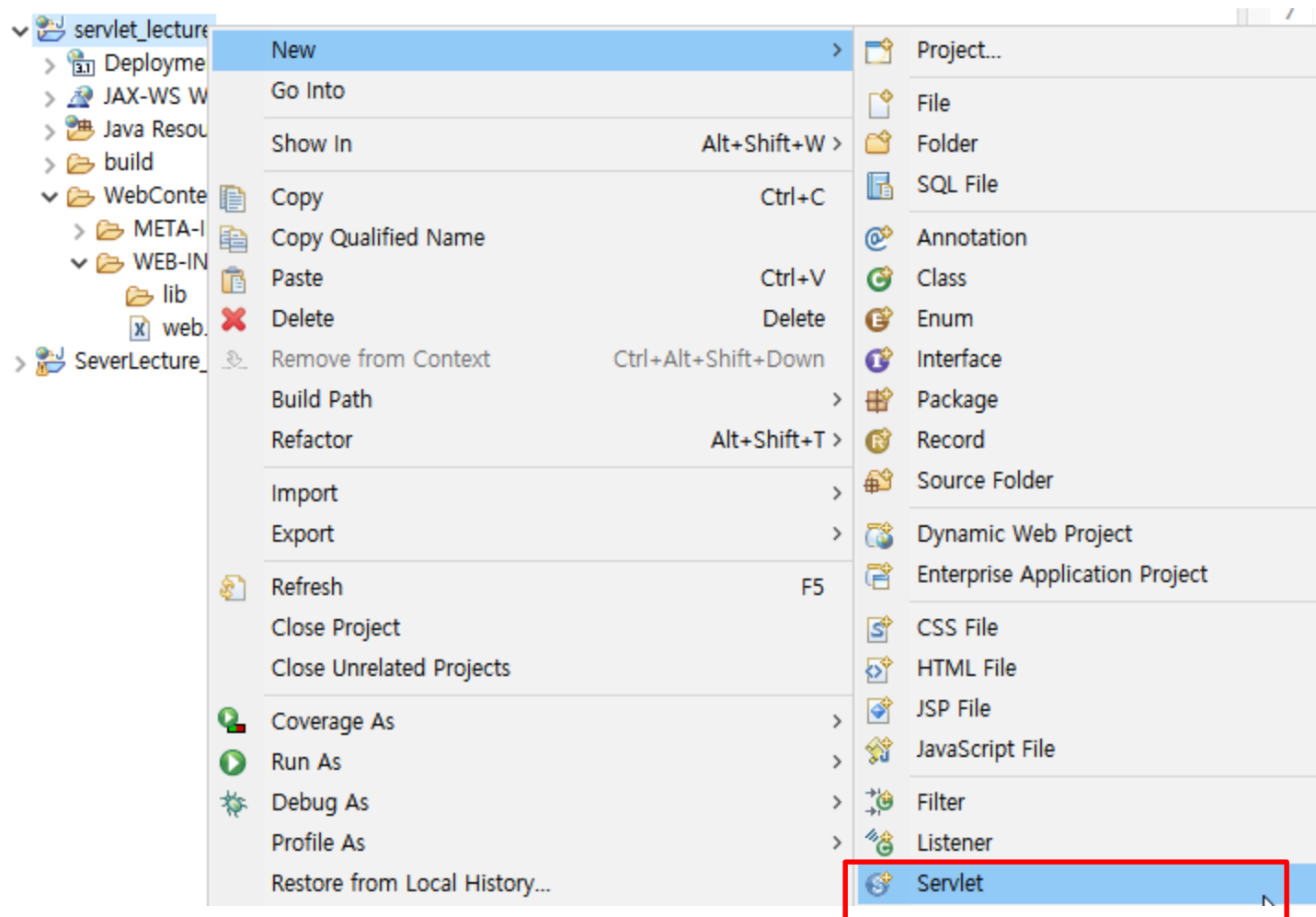
☒ Generate web.xml deployment descriptor

✓ 매핑방법

- web.xml
- 어노테이션

< Back Next > Finish Cancel

- 서블릿 클래스 생성



Create Servlet
Specify class file destination.

Project: servlet_lecture

Source folder: Wservlet_lectureWsrc [Browse...](#)

Java package: com.sw.controller [Browse...](#)

Class name: **ShoppingMallUserController**

Superclass: javax.servlet.http.HttpServlet [Browse...](#)

☐ Use an existing Servlet class or JSP

Class name: ShoppingMallUserController [Browse...](#)

[?](#) [< Back](#) **[Next >](#)** [Finish](#) [Cancel](#)

Create Servlet
Enter servlet deployment descriptor specific information.

Name: ShoppingMallUserController

Description:

Initialization parameters:

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

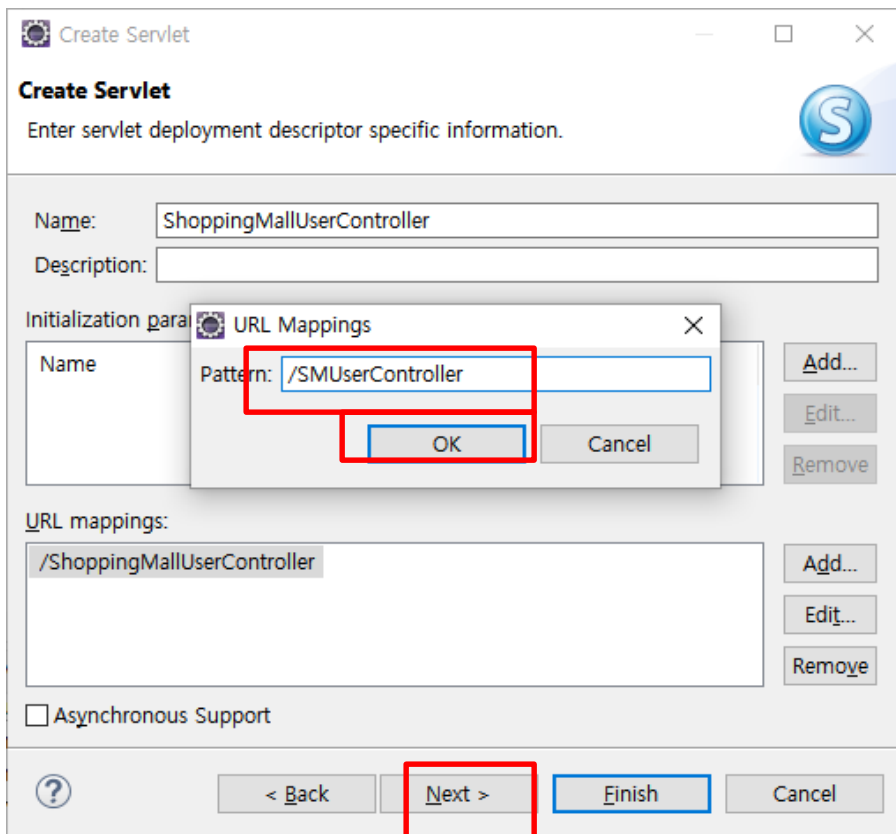
[Add...](#)
[Edit...](#)
[Remove](#)

URL mappings:

[/ShoppingMallUserController](#) [Add...](#)
[Edit...](#) [Remove](#)

☐ Asynchronous Support

[?](#) [< Back](#) [Next >](#) **[Finish](#)** [Cancel](#)



Create Servlet
Enter servlet deployment descriptor specific information.

Name:

Description:

Initialization parameters:

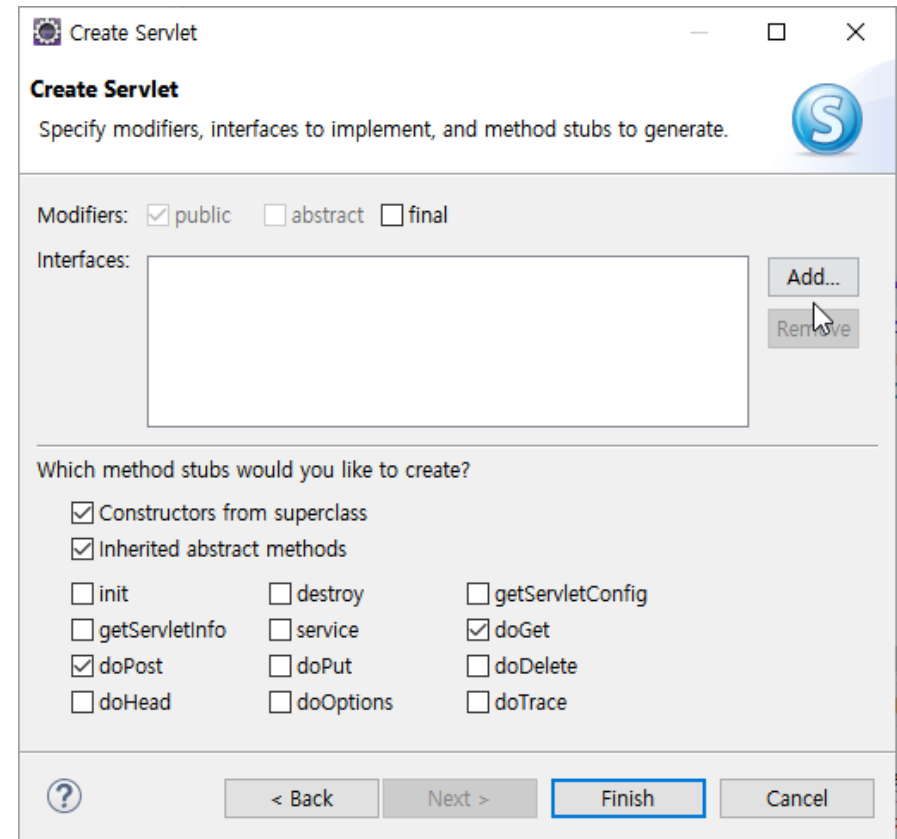
Name
URL Mappings
Pattern: <input type="text" value="/SMUserController"/>

URL mappings:

/ShoppingMallUserController

☐ Asynchronous Support

< Back **Next >** Finish Cancel



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

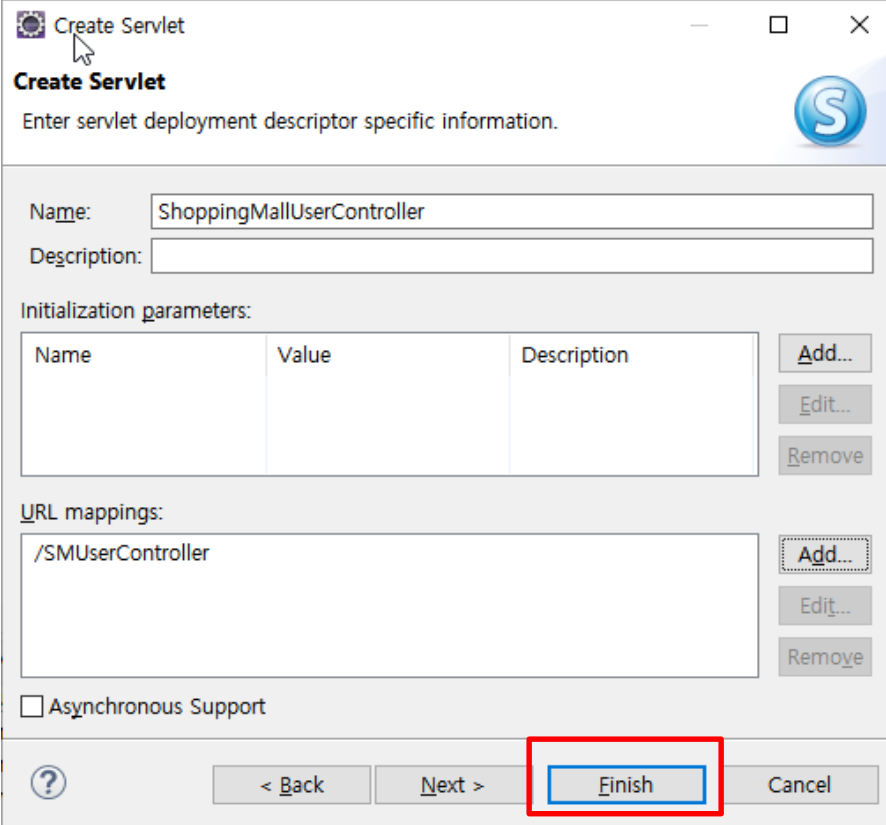
Modifiers: ☒ public ☐ abstract ☐ final

Interfaces:

Which method stubs would you like to create?

<input checked="" type="checkbox"/> Constructors from superclass		
<input checked="" type="checkbox"/> Inherited abstract methods		
<input type="checkbox"/> init	<input type="checkbox"/> destroy	<input type="checkbox"/> getServletConfig
<input type="checkbox"/> getServletInfo	<input type="checkbox"/> service	<input checked="" type="checkbox"/> doGet
<input checked="" type="checkbox"/> doPost	<input type="checkbox"/> doPut	<input type="checkbox"/> doDelete
<input type="checkbox"/> doHead	<input type="checkbox"/> doOptions	<input type="checkbox"/> doTrace

? < Back Next > **Finish** Cancel



The image shows a 'Create Servlet' dialog box from an IDE. It contains fields for Name (ShoppingMallUserController), Description, and URL mappings (/SMUserController). There are also buttons for Add, Edit, and Remove for both initialization parameters and URL mappings. The 'Finish' button is highlighted with a red rectangle.

Create Servlet

Enter servlet deployment descriptor specific information.

Name:

Description:

Initialization parameters:

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

URL mappings:

/SMUserController

☐ Asynchronous Support

- 생성된 서블릿 클래스

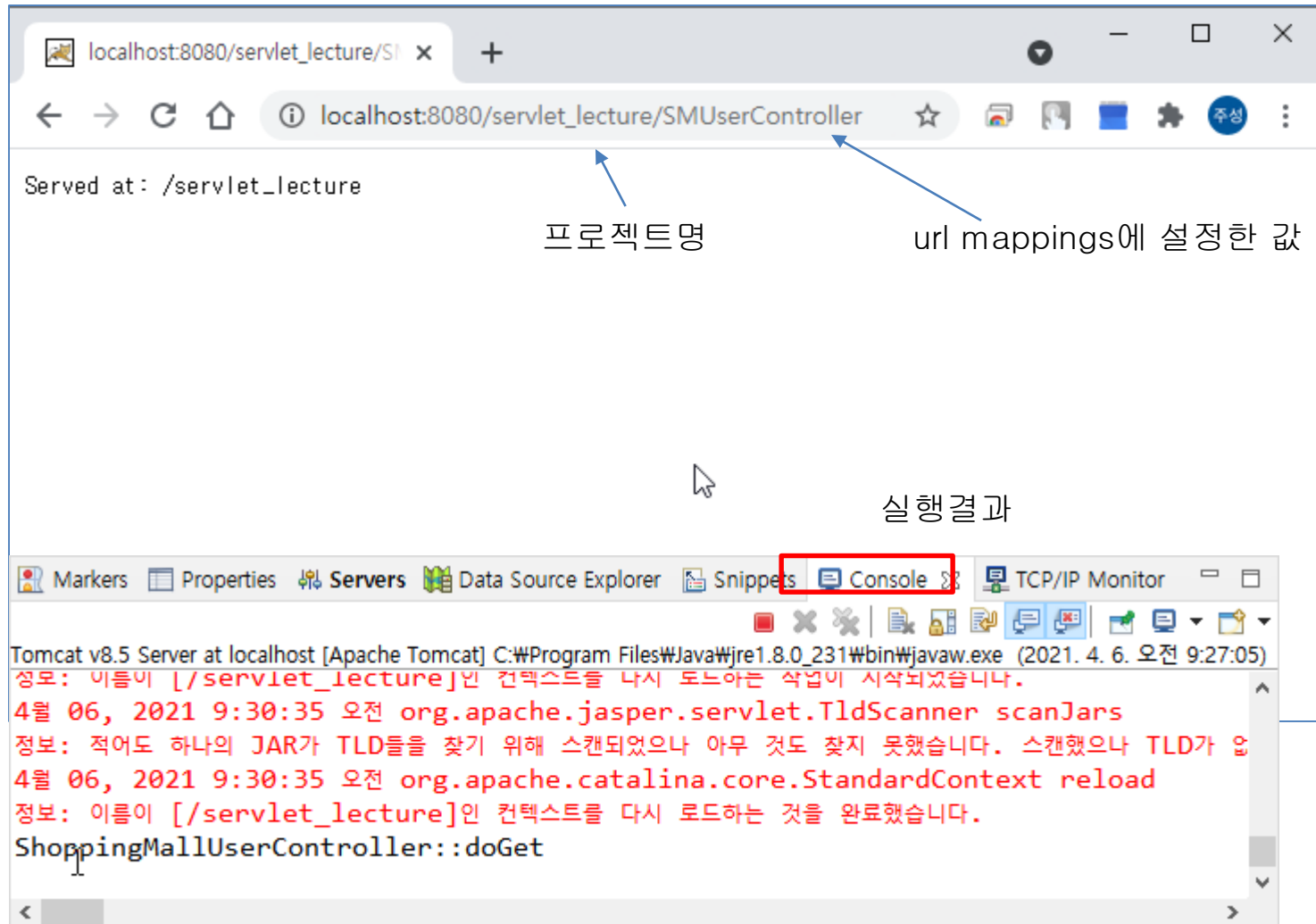
```
/**
 * Servlet implementation class ShoppingMallUserController
 */
@WebServlet("/SMUserController")
public class ShoppingMallUserController extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ShoppingMallUserController() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at: ").append(request.getContextPath());
        System.out.println("ShoppingMallUserController::doGet");
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
        System.out.println("ShoppingMallUserController::doPost");
    }
}
```

- 테스트
 - 브라우저에 아래와 같이 url 입력후 엔터



- **Get**

- context.xml 의 Connect 태그에 다음과 같이 추가

```
<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8088"  
protocol="HTTP/1.1" redirectPort="8443"/>
```

- **Post**

- doPost 메소드에 다음을 추가

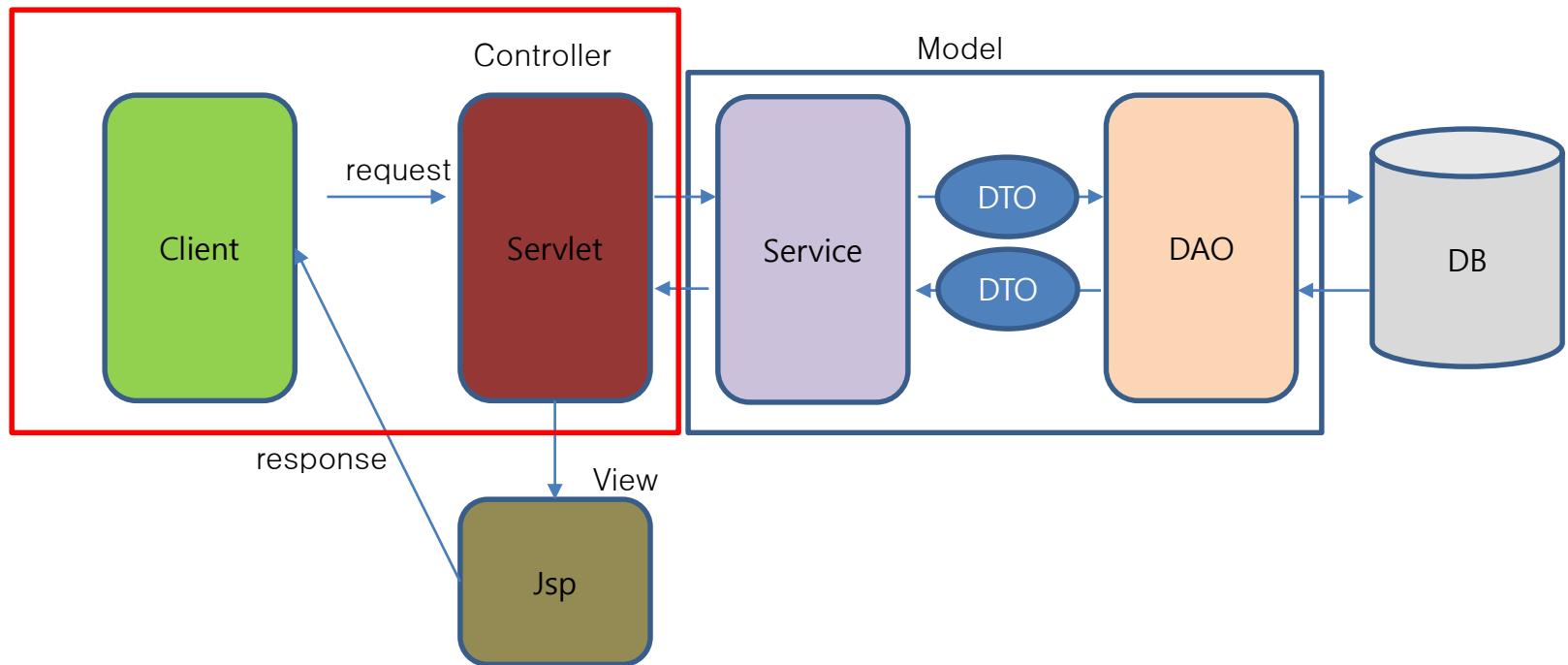
```
request.setCharacterEncoding("utf-8");
```

```
private void doPost(HttpServletRequest request, HttpServletResponse) {  
    System.out.println("actionDo");  
    request.setCharacterEncoding("utf-8");  
  
    String viewPage = null;  
    MemberDto dtos = null;  
  
}
```

- **jsp 파일 charset 을 utf-8로 변경**

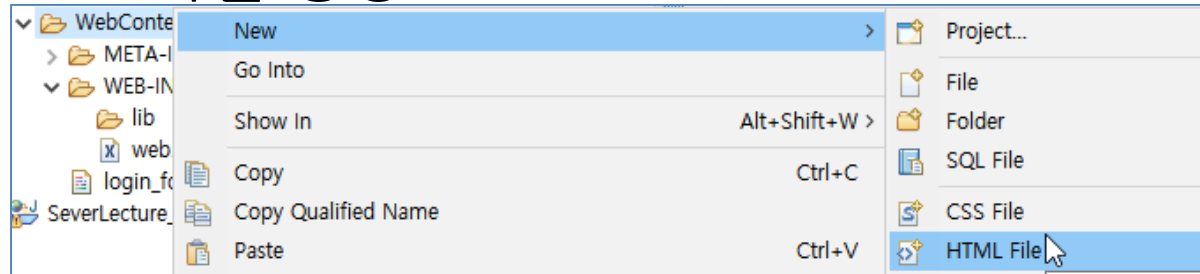
- 이클립스 preference -> web -> jsp Files -> Encoding을 utf-8 로 변경

- Form태그의 submit 버튼을 클릭하여 데이터를 서버로 전송하면, 해당파일(Servlet)에서는 HttpServletRequest객체를 이용하여 Parameter값을 얻을 수 있다.

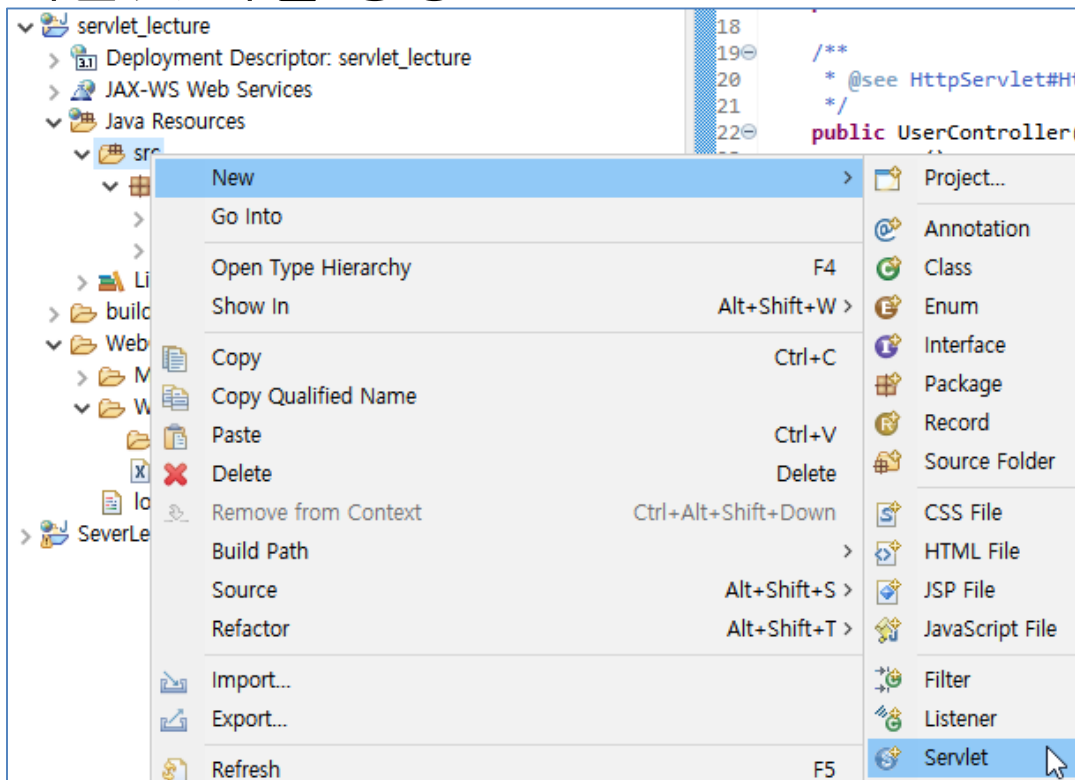


MVC 모델

- html 파일 생성



- 서블릿 파일 생성



login_form.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="EUC-KR">
<title>Insert title here</title>
</head>
<body>
```

1. 목적지

2. 방법

```
<form action= " ParameterController" method="post">
아이디 : <input type="text" name="id"><br/>
비밀번호 : <input type="password" name="pw"><br/>
<input type="submit" value="전송">
</form>
```

3. 파라미터

```
</body>
</html>
```

ParameterController.java

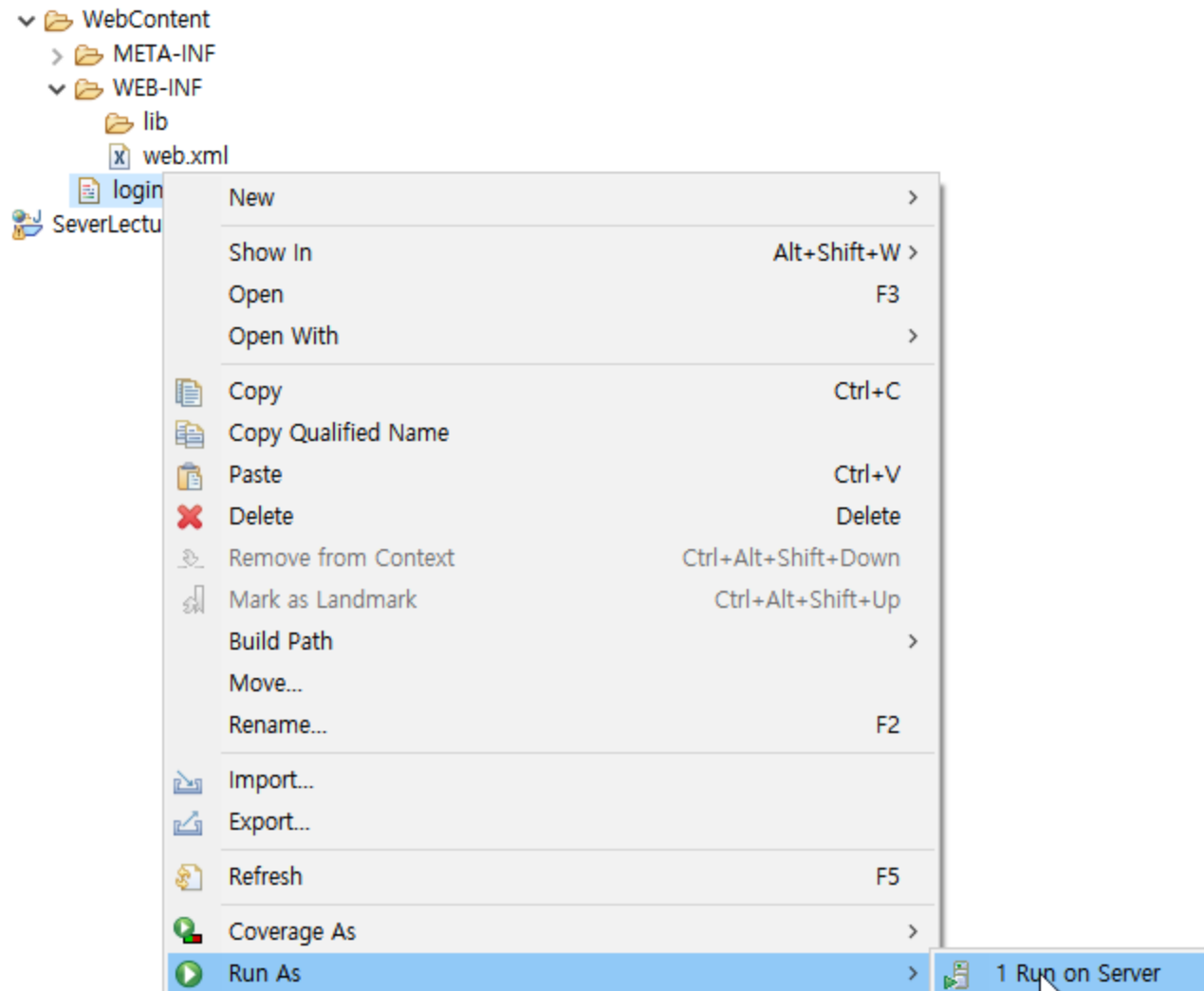
```
@WebServlet("/ParameterController")
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
    System.out.println("doPost");

    request.setCharacterEncoding("EUC-KR");
    String id = request.getParameter("id");
    String pw = request.getParameter("pw");

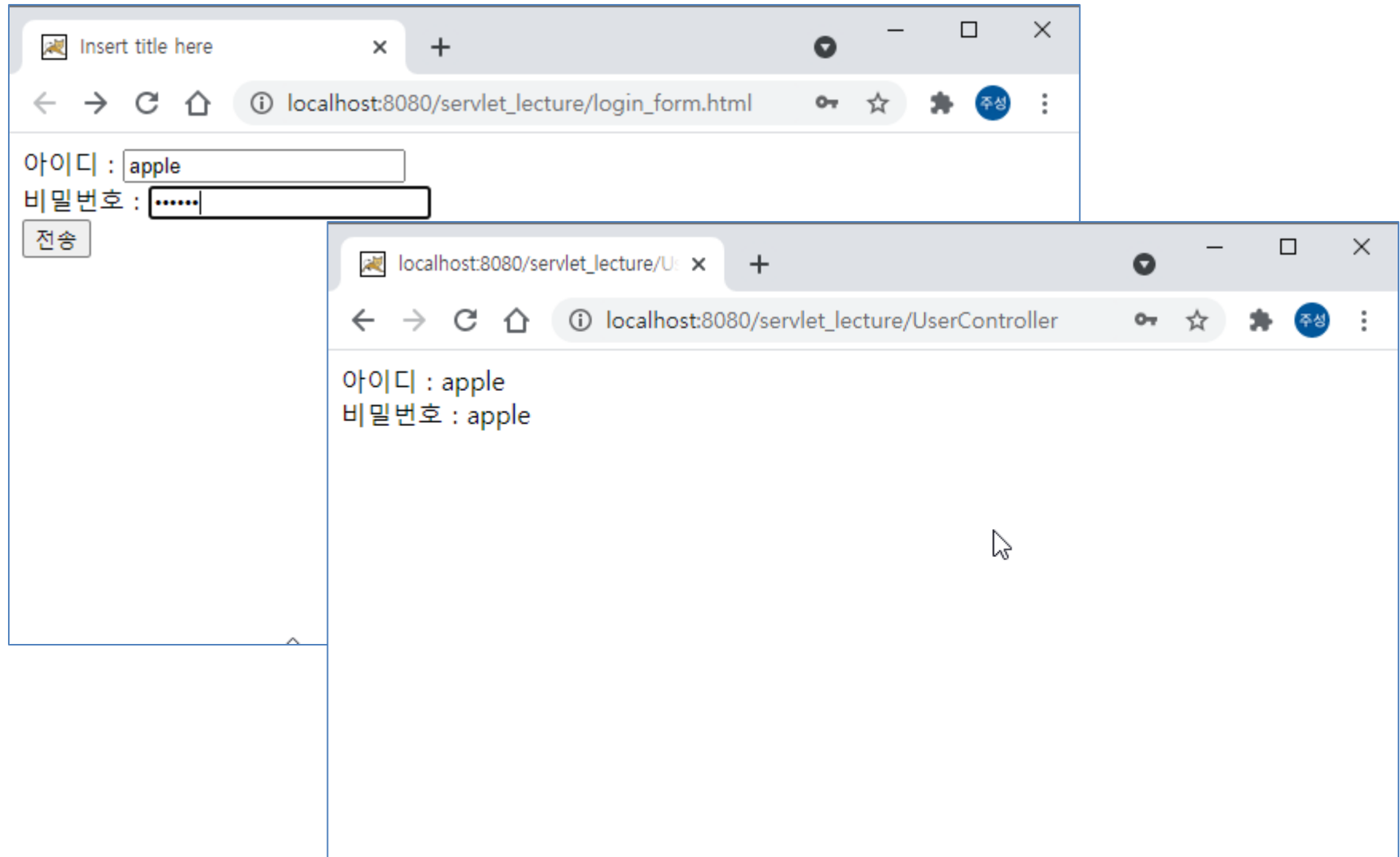
    System.out.println("id : "+id);
    System.out.println("pw : "+pw);
    response.setContentType("text/html; charset=EUC-KR");
    PrintWriter writer=response.getWriter();
    writer.println("<html><head></head><body>");
    writer.println("아이디 : "+id+"<br/>");
    writer.println("비밀번호 : "+ pw + "<br/>");

}
```

- 실행



- 실행결과



- 다음과 같은 입력폼에 입력을 받아서 서블릿에서 받은 정보를 출력하세요



도 :

시 :

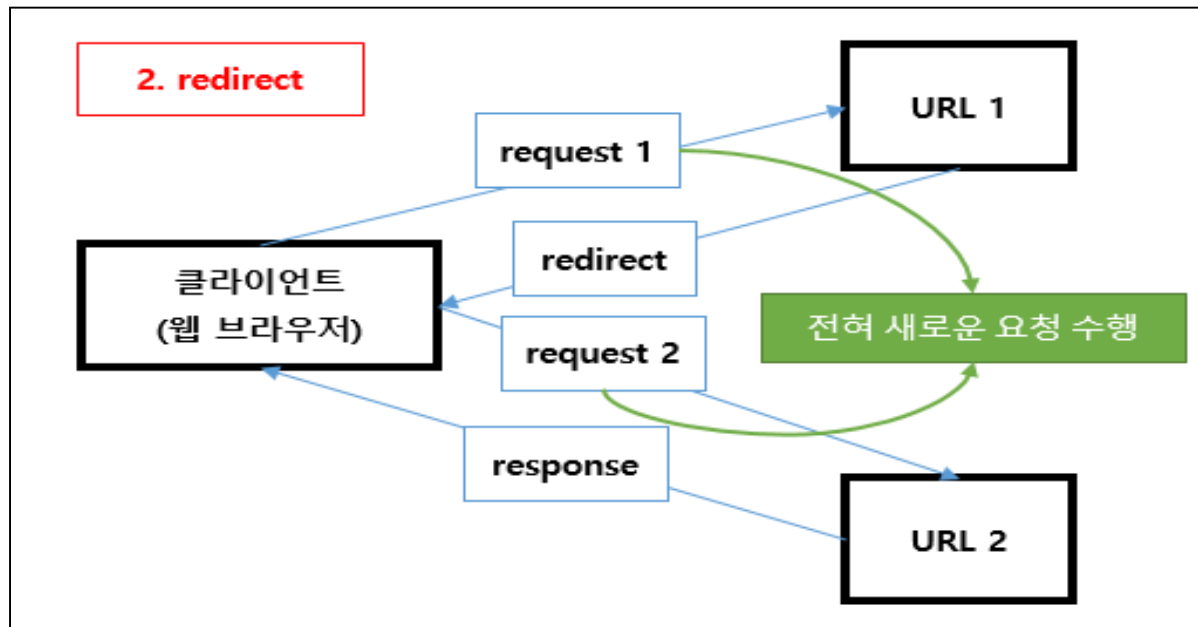
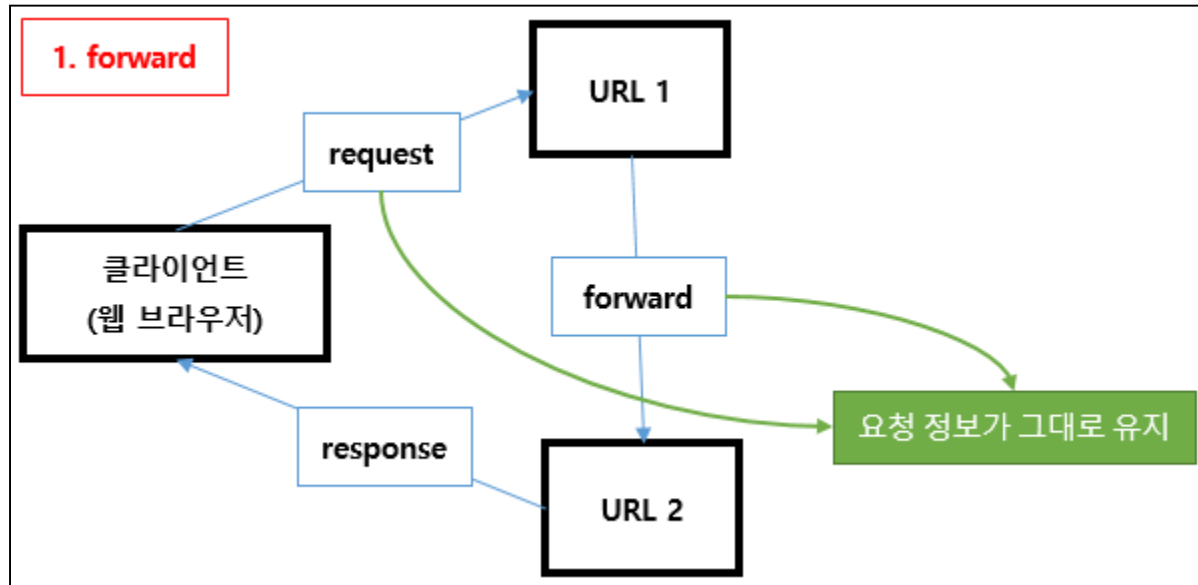
adress_form.html

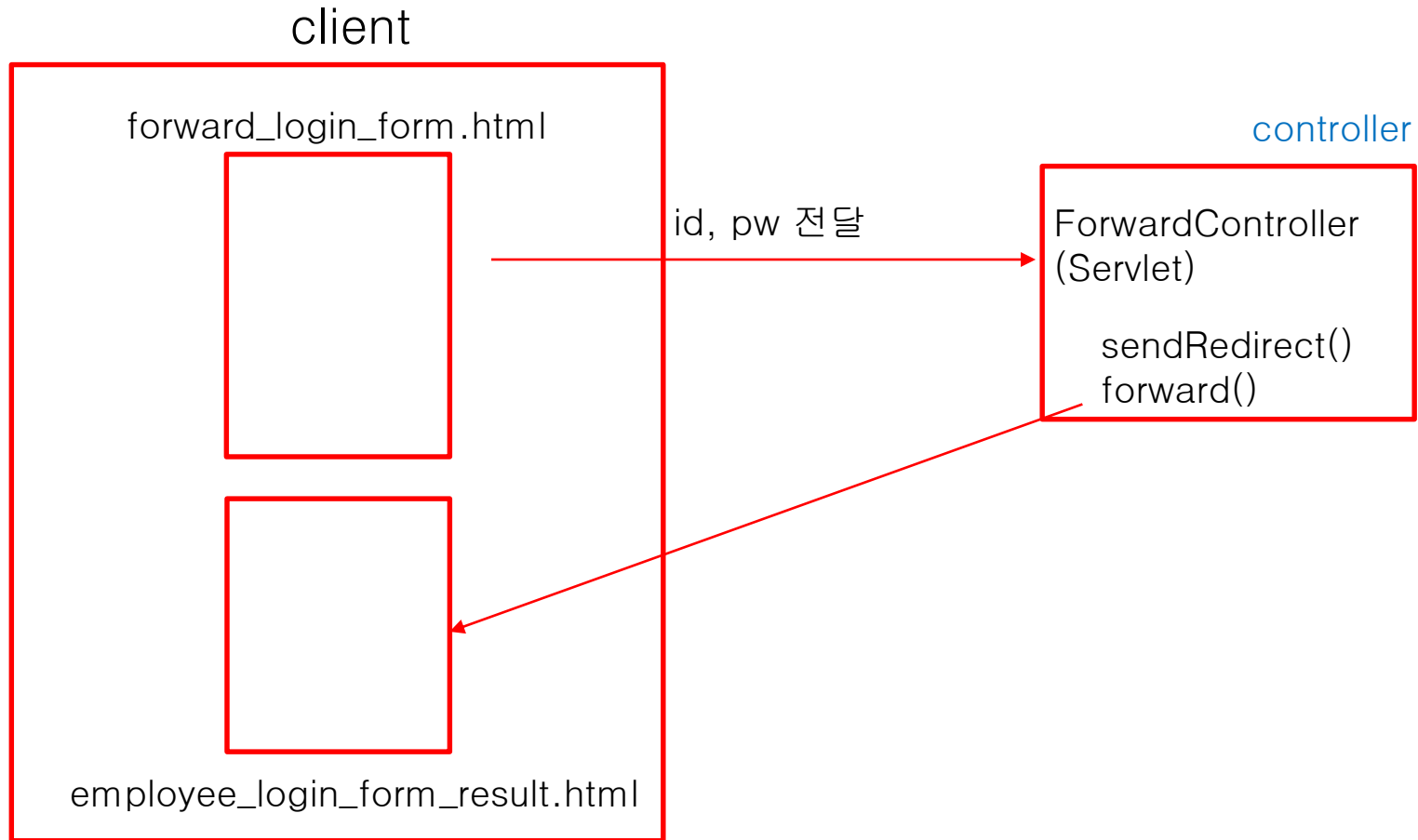
```
<body>

  <form action="PR1_AddressController" method="post">
    state : <input type="text" name="state"><br/>
    city : <input type="password" name="city"><br/>
    <input type="submit" value="전송">
  </form>

</body>
```

다른 뷰로 자동 연결하기





forward_login_form.html

```
<body>
  <form action="ForwardController" method="post">
    아이디 : <input type="text" name="id"><br/>
    비밀번호 : <input type="password" name="pw"><br/>
    <input type="submit" value="전송">
  </form>
</body>
```



forward_login_form_result.html

```
<body>
  forwarded page
</body>
```



ForwardController.java

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    System.out.println("doPost");
    request.setCharacterEncoding("EUC-KR");

    String id = request.getParameter("id");
    String pw = request.getParameter("pw");
    System.out.println("id : "+id);
    System.out.println("pw : "+pw);

    RequestDispatcher dispatcher = request.getRequestDispatcher("forward_login_form_result.html");
    dispatcher.forward(request, response);
}
```

다른 뷰로 자동 연결하기

redirected to login_form_extension_result



redirected to login_form_extension_result



```
3
orange
6
actionDo
uri : /servlet_lecture/login.do
conPath : /servlet_lecture
command : /login.do
login
-----
```

이클립스 콘솔창

- 전송을 누르면 StudentController를 거쳐
student_form_result.html 이 출력 되도록 만드세요
student_form.html

number :

name :



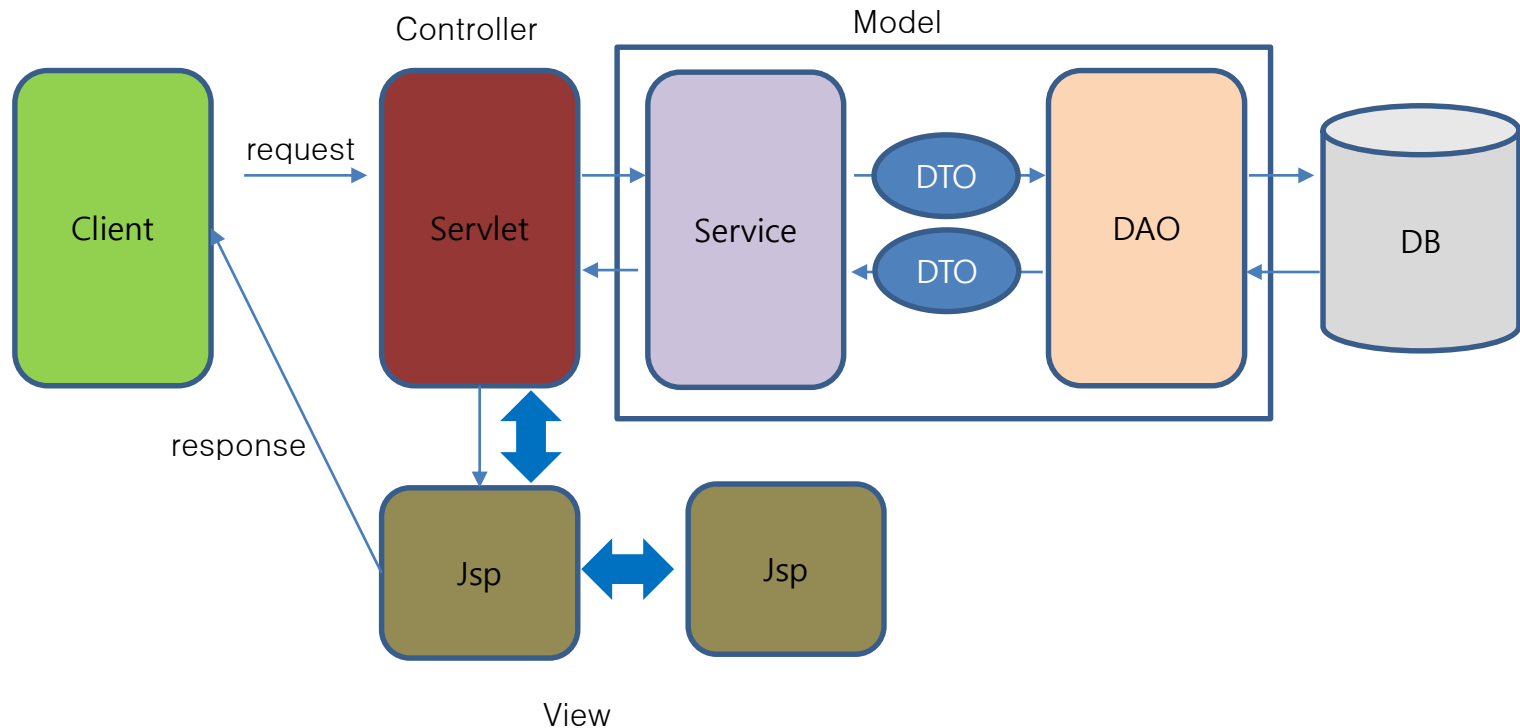
StudentController(서블릿)



student_form_result.html

```
<body>
forwarded page
</body>
```

- 세션이란?
 - 뷰에서 뷰, 서블릿에서 서블릿, 서블릿에서 뷰로의 데이터 전송에 사용됨
 - 예)
 - 쇼핑몰 장바구니
 - 다른 페이지시 로그인 상태 유지



- 서블릿 클래스에서 세션 기술을 사용하는 방법

- 1. HttpSession 객체 얻기

- doPost(), doGet()의 파라미터인 request 객체 이용

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

세션을 시작하는 메서드

- 2. 데이터를 세션에 저장

```
session.setAttribute("ID", "lee77");
```

attribute 이름 attribute 값

- 3. 세션에서 데이터 얻기 :

```
String str = (String) session.getAttribute("ID");
```

캐스트 - attribute 값의 타입

attribute 이름

■ 세션에서 데이터 삭제

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

setAttribute()에서 사용한 데이터 이름

■ 세션 닫기

```
session.invalidate();
```

controllerrk html에서 id, pw를 얻어서 jsp로 넘김

`http://localhost:8080/serverlet_lecture/session_login_form.html`

← HTML 문서

아이디 :
비밀번호 :

`http://localhost:8080/serverlet_lecture/SessionController`

← 서블릿 클래스

`http://localhost:8080/serverlet_lecture/session_login_form_reult.jsp`

← jsp

session_login_form_result
id : apple pw : orange

session_login_form.html

```
<body>
  <form action="SessionController" method="get">
    아이디 : <input type="text" name="id"><br/>
    비밀번호 : <input type="password" name="pw"><br/>
    <input type="submit" value="전송">
  </form>
</body>
```



SessionController.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    System.out.println("doPost");
    request.setCharacterEncoding("EUC-KR");

    String id = request.getParameter("id");
    String pw = request.getParameter("pw");
    System.out.println("id : "+id);
    System.out.println("pw : "+pw);

    response.setContentType("text/html; charset=EUC-KR");
    HttpSession session = request.getSession();
    session.setAttribute("id", id);
    session.setAttribute("pw", pw);

    RequestDispatcher dispatcher = request.getRequestDispatcher("session_login_form_result.jsp");
    dispatcher.forward(request, response);
}
```



session_login_form_result.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%
    request.setCharacterEncoding( "EUC-KR");
    String id = (String)session.getAttribute( "id");
    String pw = (String)session.getAttribute( "pw");
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="EUC-KR">
<title>Insert title here</title>
</head>
<body>
session_login_form_result
id : <%= id%>
pw : <%= pw%>
</body>
</html>
```

- 전송을 누르면 CarController에서 세션에 저장후
car_form_result.html 에서 정보가 출력 되도록 만드세요
car_form.html

car model :

year :



CarController(서블릿)



car_form_result.html

session_form_result

car model : sonata color : white

- EL(Expression Language)란, 세션에 저장된 값을 HTML상에 간단히 값을 표현하는 언어
- 형식

`${attribute이름}`

애트리뷰트란 `setAttribute`, `getAttribute`, `removeAttribute` 메서드를 통해 저장되고, 관리되는 데이터를 의미

`${cnt}`

익스프레션 언어의 식(EL 식)



`<%= cnt %>`

익스프레션의 식

jsp or servlet

```

httpSession.setAttribute('mdtos', mdtos);
httpSession.setAttribute('elStr', str);
httpSession.setAttribute('elInt', i);
  
```

객체

jsp

```

mdtos.name : ${mdtos.name}
elStr : ${elStr}
elint : ${elInt}
  
```

변수

el_start.html

```
<body>
  <a href="EIController">el</a>
</body>
```

EIController.java

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

```
    HttpSession httpSession = request.getSession();
```

```
    MemberDto mdto = new MemberDto();
    mdto.setName("mouse");
    mdto.setId("monitor");
    mdto.setPw("cup");
```

```
    String str = "apple";
    int i=10;
```

```
    httpSession.setAttribute("mdtos", mdto);
    httpSession.setAttribute("elStr", str);
    httpSession.setAttribute("elInt", i);
```

```
    response.sendRedirect("el_output.jsp");
```

```
}
```

el_output.jsp

```
<body>
mdtos.name : ${mdtos.name}
<br>
elStr : ${elStr}
<br>
elint : ${elInt}
</body>
```


MemberDto.java

```
public class MemberDto {  
    String name;  
    String id;  
    String pw;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getId() {  
        return id;  
    }  
  
    public void setId(String id) {  
        this.id = id;  
    }  
  
    public String getPw() {  
        return pw;  
    }  
  
    public void setPw(String pw) {  
        this.pw = pw;  
    }  
}
```

getter, setter 이 있어야 EL 사용 가능

- User class 생성
 - id, pw, name 세개의 필드를 가짐

user el test



User Controller

User 객체 생성후 아래와 같은 값 입력
id : para31
pw : jp3111
name : john



```
from User object  
id : para31  
pw : jpp3111  
name : john
```

- JSTL은 JSP 표준 태그 라이브러리(JSP Standard Tag Library)의 약어이다.
- 라이브러리란 여러 프로그램이 공통으로 사용하는 코드를 모아놓은 코드의 집합이다.
- JSTL을 가지고 할 수 있는 일
 - 간단한 프로그램 로직의 구사(자바의 변수 선언, if 문, for 문 등에 해당하는 로직)
 - 다른 JSP 페이지 호출(<c:redirect>, <c:import>)
 - 날짜, 시간, 숫자의 포맷
 - JSP 페이지 하나를 가지고 여러 가지 언어의 웹 페이지 생성
 - 데이터베이스로의 입력, 수정, 삭제, 조회
 - XML 문서의 처리
 - 문자열을 처리하는 함수 호출
- 문자열을 처리하는 함수 호출을 제외한 나머지 기능들은 모두 커스텀 액션 형태로 제공된다.

• JSTL이란?

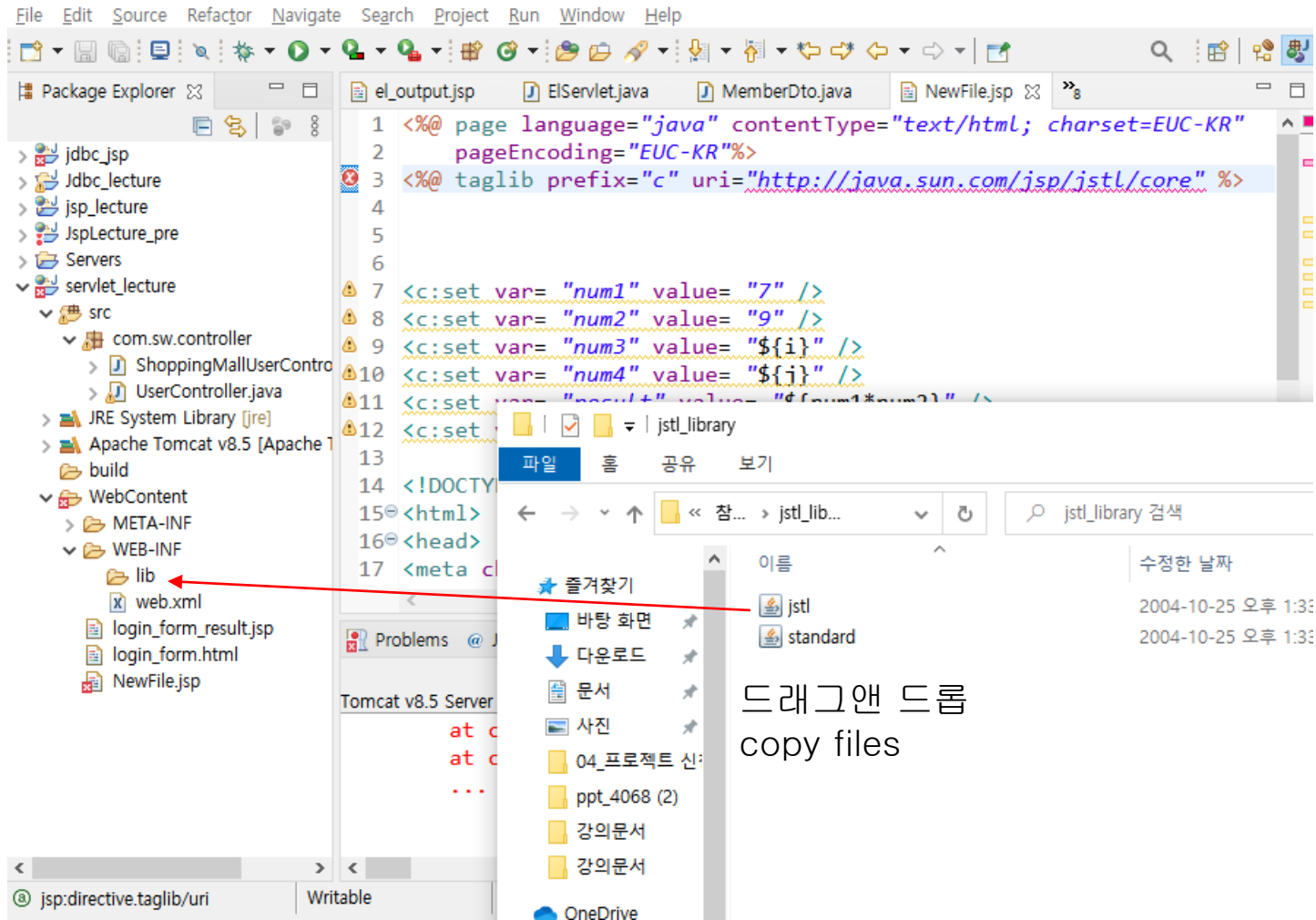
- JSP 페이지에서 앞 페이지의 접두어를 사용하기 위해서는 taglib 지시자를 이용해서 라이브러리의 URI 식별자와 접두어를 연결해야 한다.
- taglib 지시자는 다른 지시자와 마찬가지로 <%@으로 시작해서 %>로 끝난다.
- taglib 지시자에는 uri와 prefix라는 두 개의 애트리뷰트를 써야 하고, 이 두 애트리뷰트에 각각 URI 식별자와 접두어를 값으로 주어야 한다.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

↑
접두어

↑
라이브러리를 식별하는 URI

- 라이브러리 추가



File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- jdbc_jsp
- Jdbc_lecture
- jsp_lecture
- JspLecture_pre
- Servers
- ✓ servlet_lecture
 - src
 - com.sw.controller
 - ShoppingMallUserContro
 - UserController.java
 - JRE System Library [jre]
 - Apache Tomcat v8.5 [Apache T
 - build
 - WebContent
 - META-INF
 - WEB-INF
 - lib
 - web.xml
 - login_form_result.jsp
 - login_form.html
 - NewFile.jsp

el_output.jsp

```
1 <%@ page language="java" contentType="text/html; charset=EUC-KR"
2   pageEncoding="EUC-KR"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4
5
6
7 <c:set var="num1" value="7" />
8 <c:set var="num2" value="9" />
9 <c:set var="num3" value="${i}" />
10 <c:set var="num4" value="${j}" />
11 <c:set var="result" value="${num1*num2}" />
12 <c:set
13
14 <!DOCTYPE
15 <html>
16 <head>
17 <meta c
```

파일 홈 공유 보기

« 참... » jstl_lib...

jstl_library 검색

이름 수정한 날짜

- 즐거찾기
- 바탕 화면
- 다운로드
- 문서
- 사진
- 04_프로젝트 신:
- ppt_4068 (2)
- 강의문서
- 강의문서
- OneDrive

jstl

standard

2004-10-25 오후 1:30

2004-10-25 오후 1:30

드래그 앤 드롭
copy files

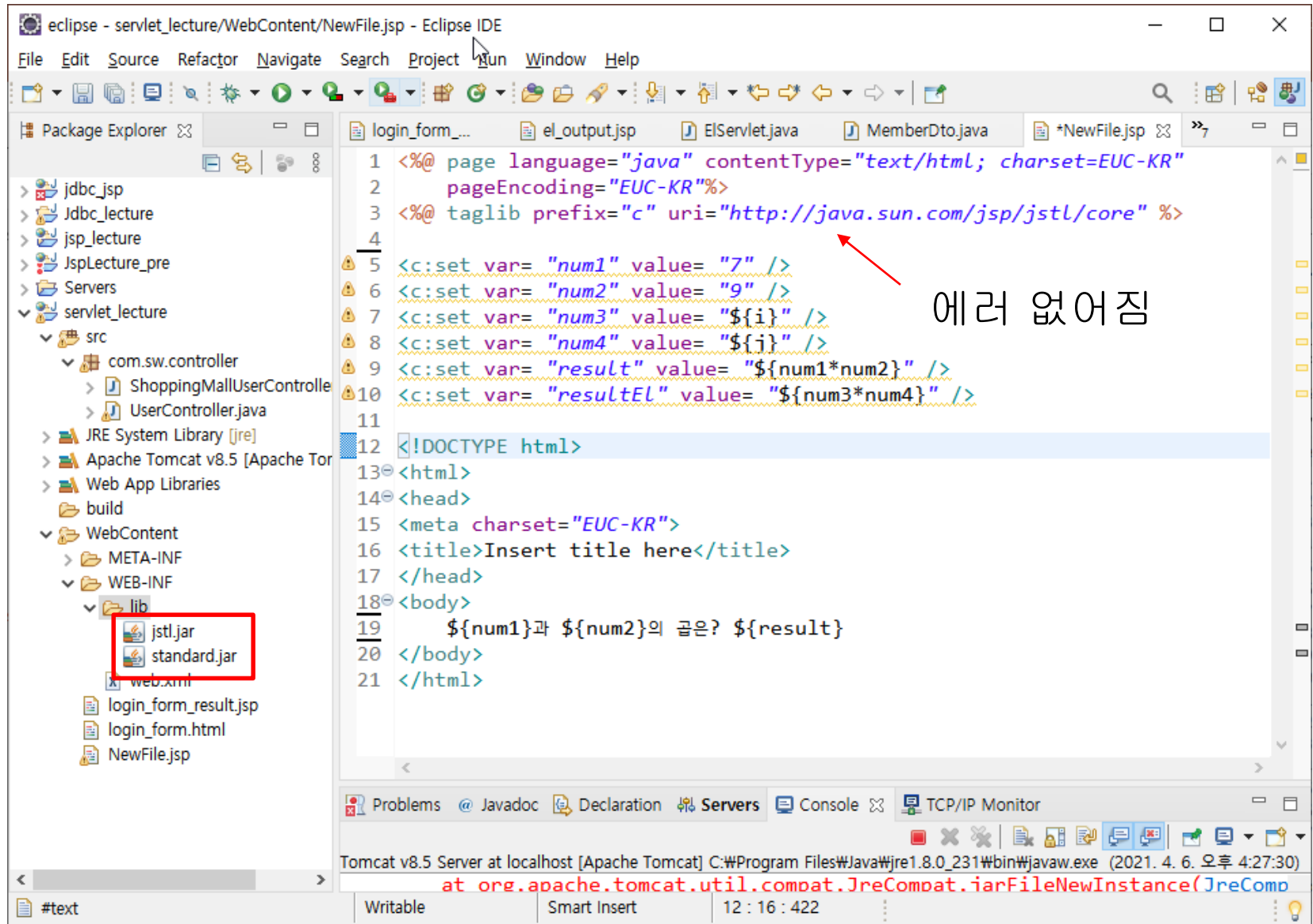
Tomcat v8.5 Server

at c

at c

...

jsp:directive.taglib/uri Writable



eclipse - servlet_lecture/WebContent/NewFile.jsp - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- jdbc.jsp
- Jdbc_lecture
- jsp_lecture
- JspLecture_pre
- Servers
- ▼ servlet_lecture
 - src
 - com.sw.controller
 - ShoppingMallUserController
 - UserController.java
 - JRE System Library [jre]
 - Apache Tomcat v8.5 [Apache Tor
 - Web App Libraries
 - build
 - WebContent
 - META-INF
 - WEB-INF
 - lib
 - jstl.jar
 - standard.jar
 - web.xml
 - login_form_result.jsp
 - login_form.html
 - NewFile.jsp

```
1 <%@ page language="java" contentType="text/html; charset=EUC-KR"
2   pageEncoding="EUC-KR"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4
5 <c:set var="num1" value="7" />
6 <c:set var="num2" value="9" />
7 <c:set var="num3" value="${i}" />
8 <c:set var="num4" value="${j}" />
9 <c:set var="result" value="${num1*num2}" />
10 <c:set var="resultEl" value="${num3*num4}" />
11
12 <!DOCTYPE html>
13 <html>
14 <head>
15 <meta charset="EUC-KR">
16 <title>Insert title here</title>
17 </head>
18 <body>
19   ${num1}과 ${num2}의 곱은? ${result}
20 </body>
21 </html>
```

에러 없어짐

Problems Javadoc Declaration Servers Console TCP/IP Monitor

Tomcat v8.5 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (2021. 4. 6. 오후 4:27:30)

at org.apache.tomcat.util.compat.JreCompat.iarFileNewInstance(JreCompo

#text Writable Smart Insert 12 : 16 : 422

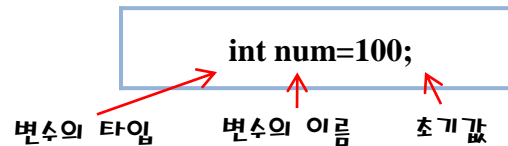
[표 9-1] JSTL을 구성하는 작은 라이브러리들

라이브러리	기 능	URI 식별자	접두어
코어	일반 프로그래밍 언어에서 제공하는 것과 유사한 변수 선언, 실행 흐름의 제어 기능을 제공하고, 다른 JSP 페이지로 제어를 이동하는 기능도 제공합니다.	http://java.sun.com/jsp/jstl/core	c
포매팅	숫자, 날짜, 시간을 포매팅하는 기능과 국제화, 다국어 지원 기능을 제공합니다	http://java.sun.com/jsp/jstl/fmt	fmt
데이터베이스	데이터베이스의 데이터를 입력/수정/삭제/조회하는 기능을 제공합니다.	http://java.sun.com/jsp/jstl/sql	sql
XML 처리	XML 문서를 처리할 때 필요한 기능을 제공합니다	http://java.sun.com/jsp/jstl/xml	x
함수	문자열을 처리하는 함수를 제공합니다.	http://java.sun.com/jsp/jstl/functions	fn

• 코어 라이브러리 사용하기

■ <c:set> 커스텀 액션의 사용 방법

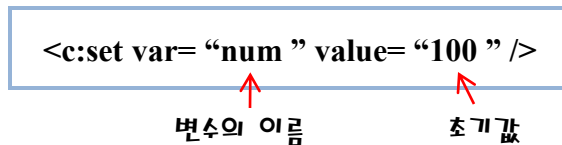
- <c:set>은 변수를 선언하고 초기값을 대입하는 커스텀 액션이다.
- 자바 프로그램에서 변수를 선언할 때는 기본적으로 변수의 타입과 이름을 기술하고, 선택적으로 초기값을 기술한다.



```
int num=100;
```

변수의 타입 변수의 이름 초기값

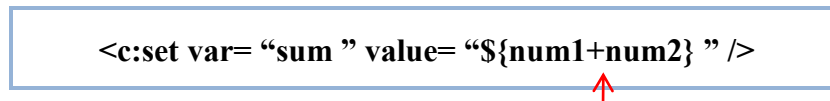
- <c:set> 커스텀 액션을 이용해서 변수를 선언할 때는 변수의 타입을 쓰지 않는다.



```
<c:set var= "num " value= "100 " />
```

변수의 이름 초기값

- value 애트리뷰트 값 위치에 EL 식을 쓸 수도 있다.



```
<c:set var= "sum " value= "${num1+num2} " />
```

value 애트리뷰트 값으로 EL 식을 쓸 수도 있습니다

- **<c:remove> 커스텀 액션의 사용 방법**

- <c:set> 액션을 이용해서 선언한 변수는 page, request, session, application 데이터 영역의 애트리뷰트로 저장되므로, 자바 변수와 달리 인위적으로 삭제해야 할 필요가 있다.
- <c:remove> 커스텀 액션은 이런 애트리뷰트를 삭제하는 기능을 한다.

```
<c:remove var= "num" />
```

↑
변수의 이름

- 위 코드는 page, request, session, application 데이터 영역에 저장되어 있는 num이라는 이름의 애트리뷰트를 모두 찾아서 제거한다. 특정 영역의 애트리뷰트만 제거하려면 scope 애트리뷰트를 사용하면 된다.

```
<c:remove var= "code" scope= "request" />
```

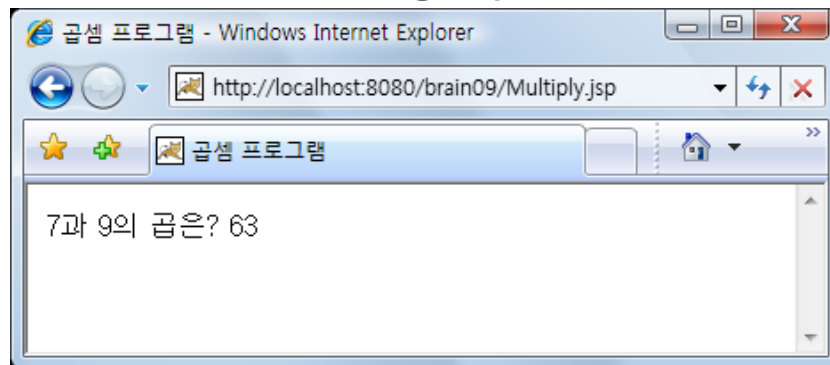
↑
request 데이터 영역에 있는 변수를 제거합니다

- <c:set> 커스텀 액션의 사용 방법

jstl_set1.jsp

```
<% @page contentType="text/html; charset=euc-kr"%>
<% @taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="num1" value="7" />
<c:set var="num2" value="9" />
<c:set var="result" value="${num1*num2}" />
<HTML>
  <HEAD><TITLE>곱셈 프로그램</TITLE></HEAD>
  <BODY>
    ${num1} 과 ${num2} 의 곱은? ${result}
  </BODY>
</HTML>

<c:remove var="num1" />
<c:remove var="num2" />
<c:remove var="result" />
```

실행 결과

- `<c:set>` scope

영역	영역객체	속성의 유효범위
page	pageContext	하나의 jsp 페이지 내에서만 객체를 공유하는 역할
request	request	클라이언트의 request부터 response 처리되는 동안 유효 (포워딩 또는 include를 이용하는 경우 여러개의 페이지에서도 요청 정보가 계속 유지되므로 request영역의 속성을 여러페이지에서 공유할수있다.)
session	session	세션이 유지되는 동안 유효 (하나의 브라우저에 1개의 세션이 생성되므로 같은 웹브라우저내에서 실행되는 페이지들이 속성을 공유할수있다.)
application	application	웹 애플리케이션이 실행되고 있는 동안 유효 (웹 컨테이너에서 해당 어플리케이션은 오직 하나만이 실행되므로 4가지 영역중 가장 큰 영역에 해당)

- <c:set> 액션을 이용해서 선언한 변수는 **page** 데이터 영역의 애트리뷰트가 된다.
- <c:set> 태그에 **scope** 애트리뷰트를 추가하고 **page, request, session, application** 중 한 값을 지정하면 선언된 변수가 **page, request, session, application** 데이터 영역의 애트리뷰트가 되도록 지정하는 것도 가능하다.

```
<c:set var= "PRICE " value= "15000 " scope= "request " />
```



변수가 저장될 데이터 영역

- **scope** 애트리뷰트에 **request**라는 값을 지정하고 나서 **forward** 메서드를 통해 다른 **JSP** 페이지를 호출하면 그 **JSP** 페이지 안에서도 선언된 변수를 사용할 수 있다.

3. 코어 라이브러리 사용하기

- <c:set> 커스텀 액션의 사용 방법

jstl_set2.jsp

```
<%@page contentType= "text/html; charset=euc-kr" %>
<%@taglib prefix= "c" uri= "http://java.sun.com/jsp/jstl/core" %>
<c:set var= "CODE" value= "80012" scope= "request" />
<c:set var= "NAME" value= "온습도계" scope= "request" />
<c:set var= "PRICE" value= "15000" scope= "request" />
<jsp:forward page= "ProductInfoView.jsp" />
```

request 데이터 영역에
데이터를 저장합니다

jstl_set2View.jsp

호출

```
<%@page contentType= "text/html; charset=euc-kr" %>
<HTML>
  <HEAD><TITLE>상품 정보</TITLE></HEAD>
  <BODY>
    <H3>상품 정보</H3>
    상품코드: ${CODE} <BR>
    상품명: ${NAME} <BR>
    단가: ${PRICE}원 <BR>
  </BODY>
</HTML>
```

request 데이터 영역에 있는
데이터 값을 가져다가 출력합니다

- **<c:if> 커스텀 액션의 사용 방법**

- <c:if> 커스텀 액션은 자바 프로그램의 if 문과 비슷한 역할을 한다.
- 자바 프로그램에서 if 문을 작성하는 방법은 다음과 같다.

조건식

```
if (num1 > num2) {  
    System.out.println( "num1이 더 큼니다. " );  
}
```

조건식의 결과가 true일 때만 실행되는 명령문

- <c:if> 커스텀 액션에서는 조건식을 괄호 안에 쓰는 것은 아니라, test라는 이름의 애트리뷰트 값으로 지정해야 한다.

조건식

```
<c:if test= "${num1 > num2}">  
    num1이 더 큼니다.  
</c:if>
```

조건식의 결과가 true일 때만 출력되는 코드

jstl_start.html

```
<body>
  <a href="JstlIfController">jstl-if</a>
</body>
```



JstlIfController.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    HttpSession session = request.getSession();
    int i=10, j=20;
    session.setAttribute("i", i);
    session.setAttribute("j", j);
    response.sendRedirect("jstl_if.jsp");
}
```



jstl_if.jsp

```
<body>
  if 문
  <br>
  <c:if test= "${i < j}">
    ${j}이 더 큼니다.
  </c:if>
</body>
```

실행결과

if 문
20이 더 큼니다.



- **<c:choose>** 커스텀 액션의 사용 방법
 - **<c:choose>** 커스텀 액션은 자바 프로그램의 **switch** 문과 비슷한 역할을 한다.
 - **<c:when>**, **<c:otherwise>**라는 커스텀 액션과 함께 사용되며, 두 커스텀 액션은 각각 **switch** 문의 **case**, **default** 절과 비슷한 역할을 한다.

- **<c:choose>** 커스텀 액션의 사용 방법
 - **<c:choose>** 커스텀 액션의 전체적인 구조는 **switch** 문과 비슷하지만, 변수의 이름이 아니라 조건식을 **<c:when>** 커스텀 액션을 **test** 애트리뷰트에 **EL** 식 형태로 지정해야 한다.

```
switch (num) {  
  case 0 :  
    System.out.println( “처음 뵙겠습니다.” );  
    break;  
  case 1 :  
    System.out.println( “반갑습니다.” );  
    break;  
  default :  
    System.out.println( “안녕하세요.” );  
    break;  
}
```

조건식을 직접
기술했습니다.

```
<c:choose>  
  <c:when test= “${num == 0} ”>  
    처음 뵙겠습니다. <BR>  
  </c:when>  
  <c:when test= “${num == 1} ”>  
    반갑습니다. <BR>  
  </c:when>  
  <c:otherwise>  
    안녕하세요. <BR>  
  </c:otherwise>  
</c:choose>
```

아무 조건도 만족하지 않을 때
출력할 코드

jstl_start.html

```
<body>
  <a href="JstlChooseController">jstl-choose</a>
</body>
```



JstlChooseController.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    HttpSession session = request.getSession();
    int i=10;
    session.setAttribute("i", i);
    response.sendRedirect("jstl_choose.jsp");
}
```



jstl_choose.jsp

```
<body>
  choose 문 <br>
  <c:choose>
    <c:when test= "${i} >= 10}">
      10보다 큰수 <BR>
    </c:when>
    <c:when test= "${i} >= 20}">
      11과 20 사이의 수 <BR>
    </c:when>
    <c:otherwise>
      그 이외 <BR>
    </c:otherwise>
  </c:choose>
</body>
```

실행결과

```
choose 문
10보다 큰수
```

- **<c:forEach> 커스텀 액션의 사용 방법**

- **<c:forEach> 커스텀 액션은 자바 프로그램의 for 문에 해당하는 기능을 제공한다. 이것을 이용하면 특정 HTML 코드를 지정된 횟수만큼 반복해서 출력할 수 있다.**

카운터의 초기값 반복 종료의 기준값 카운터를 증가시키는 식

```
for (int cnt = 0; cnt < 10; cnt++) {  
    System.out.println( "아호" );  
}
```

반복 실행할 명령문

- **<c:forEach> 액션을 사용할 때는 begin과 end라는 이름의 애트리뷰트를 쓰고, 거기에 각각 카운터 변수의 시작 값과 끝 값을 지정하면 된다.**

시작 값 끝 값

```
<c:forEach begin= "1 " end= "10 ">  
    아호<BR>  
</c:forEach>
```

반복 출력할 명령문

- **〈c:forEach〉 커스텀 액션의 사용 방법**

- 반복 출력할 코드 안에서 카운터 변수의 값을 사용해야 할 경우에는 〈c:forEach〉 태그 안에 var라는 애트리뷰트를 쓰고, 그 값으로 카운터 변수의 이름을 지정하면 된다.

카운터 변수



```
<c:forEach var= "cnt " begin= "1 " end= "10 ">  
    ${cnt} <BR>  
</c:forEach>
```

- 카운터 변수의 값은 기본적으로 1씩 증가하지만, 그 값을 바꾸려면 〈c:forEach〉 태그에 step이라는 애트리뷰트를 추가하고 증가치를 지정하면 된다.

증가치

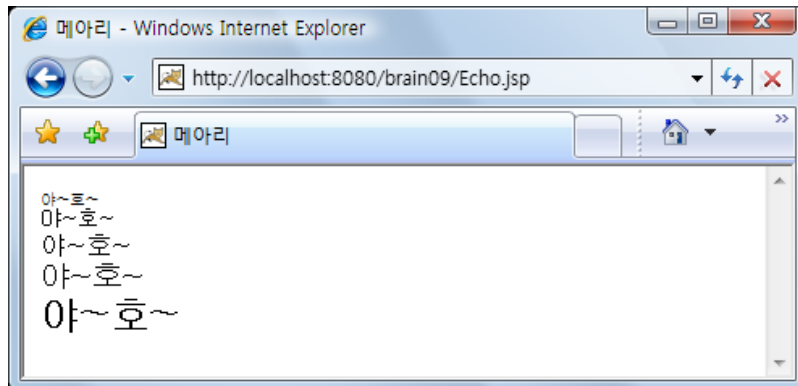


```
<c:forEach var= "cnt " begin= "1 " end= "10 " step= "2 ">  
    ${cnt} <BR>  
</c:forEach>
```

- <c:forEach> 커스텀 액션의 사용 방법

forEach1.jsp

```
<% @page contentType= "text/html; charset=euc-kr "%>
<%@taglib prefix= "c" uri= "http://java.sun.com/jsp/jstl/core " %>
<HTML>
  <HEAD><TITLE>메아리</TITLE></HEAD>
  <BODY>
    <c:forEach var= "cnt" begin= "1 " end= "5 ">
      <FONT size=${cnt} > 아~호~ </FONT> <BR>
    </c:forEach>
  </BODY>
</HTML>
```



[그림 9-13] 예제 9-5의 실행 결과

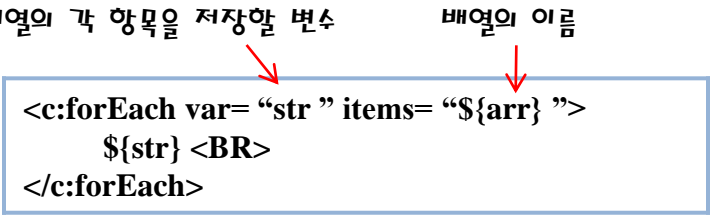
- foreach를 이용 1부터 10까지 출력하세요

- **<c:forEach> 커스텀 액션의 사용 방법**

- **<c:forEach> 커스텀 액션의 items 애트리뷰트를 이용하면 여러 개의 항목으로 구성된 데이터터를 순서대로 출력하는 일도 할 수 있다.**

배열의 각 항목을 저장할 변수

배열의 이름



```
<c:forEach var= "str" items= "${arr}">
    ${str} <BR>
</c:forEach>
```

- **<c:forEach> 액션의 items 애트리뷰트를 이용해서 처리할 수 있는 데이터**
 - 배열
 - java.util.Collection 객체
 - java.util.Iterator 객체
 - java.util Enumeration 객체
 - java.util.Map 객체
 - 콤마(,)로 구분된 항목들을 포함한 문자열

jstl_start.html

```
<body>
  <a href="JstlForEachControllerArr">jstl-foreach-arr</a>
</body>
```



JstlForEachControllerArr.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    HttpSession session = request.getSession();

    String arr[] = { "불고기 백반", "오므라이스", "콩국수" };
    session.setAttribute( "menu", arr);
    response.sendRedirect("jstl_foreach2.jsp");
}
```



jstl_foreach_arr.jsp

```
<body>

array<br>
<c:forEach var= "dish" items= "${menu}">
  ${dish}<br>
</c:forEach>

</body>
```

실행결과

```
array
불고기 백반
오므라이스
콩국수
```


MemberDTO.java

```
public class MemberDTO {  
    String name;  
    String id;  
    String pw;  
  
    public MemberDTO() {  
    }  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getId() {  
        return id;  
    }  
  
    public void setId(String id) {  
        this.id = id;  
    }  
  
    public String getPw() {  
        return pw;  
    }  
  
    public void setPw(String pw) {  
        this.pw = pw;  
    }  
}
```

jstl_start.html

```
<body>  
    <a href="JstlForEachControllerList">jstl-foreach-list</a>  
</body>
```

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

```
    System.out.println("doGet");
```

```
    ArrayList<MemberDTO> mdtoArrayList= new ArrayList <MemberDTO>();
```

```
    Map<String, String> mdtoStringMap= new HashMap<String, String>();
```

```
    Map<String, MemberDTO> mdtoObjMap= new HashMap<String, MemberDTO>();
```

```
    HttpSession httpSession = request.getSession();
```

```
    MemberDTO mdto = new MemberDTO();
```

```
    mdto.setName("mouse");
```

```
    mdto.setId("monitor");
```

```
    mdto.setPw("cup");
```

```
    mdtoArrayList.add(mdto);
```

```
    mdtoStringMap.put("monitor", "cup");
```

```
    mdtoObjMap.put("monitor", mdto);
```

```
    MemberDTO mdto1 = new MemberDTO();
```

```
    mdto1.setName("apple");
```

```
    mdto1.setId("orange");
```

```
    mdto1.setPw("desk");
```

```
    mdtoArrayList.add(mdto1);
```

```
    mdtoStringMap.put("orange", "desk");
```

```
    mdtoObjMap.put("orange", mdto1);
```

```
    httpSession.setAttribute("mdtosList", mdtoArrayList);
```

```
    httpSession.setAttribute("mdtosStringMap", mdtoStringMap);
```

```
    httpSession.setAttribute("mdtosObjectMap", mdtoObjMap);
```

```
    response.sendRedirect("jstl_foreach_list.jsp");
```

```
}
```

JstlForEachControllerList.java

jstl_foreach_list.jsp

```
<body>
---- for each List----<br>
<c:forEach var="mdtosVal" items="${mdtosList}">
    id : ${mdtosVal.id}, pw :${mdtosVal.pw}<br>
</c:forEach><br>

---- for each String map ----<br>
<c:forEach var="mdtosString" items="${mdtosStringMap}">
    id : ${mdtosString.key}, pw :${mdtosString.value}<br>
</c:forEach><br>

---- for each Object map ----<br>
<c:forEach var="mdtosObj" items="${mdtosObjectMap}">
    id : ${mdtosObj.key}, pw :${mdtosObj.value.pw}<br>
</c:forEach><br>
</body>
```

```
---- for each List----
id : monitor, pw :cup
id : orange, pw :desk
```

```
---- for each String map ----
id : orange, pw :desk
id : monitor, pw :cup
```

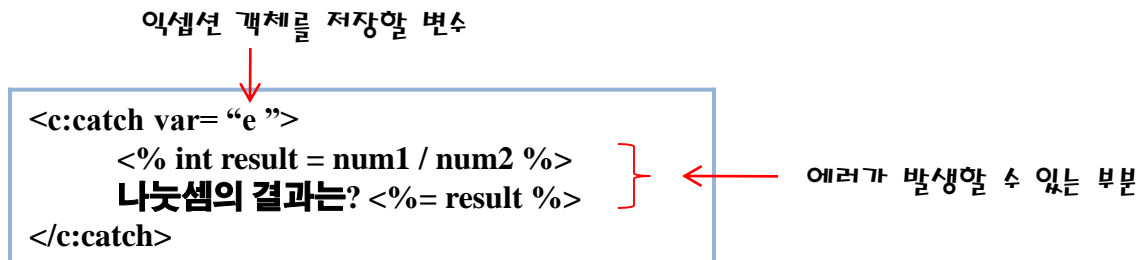
```
---- for each Object map ----
id : orange, pw :desk
id : monitor, pw :cup
```

실행결과

- 아래와 같은 dto list를 만들기
 - BookDTO
 - String name;
 - String isbn;
 - String price;
- dto list에 데이터 5개 추가
- view에서 dto list에 있는 데이터 5개 출력

- **<c:catch> 커스텀 액션의 사용 방법**

- <c:catch> 커스텀 액션은 자바 프로그래밍 언어의 try문과 비슷한 기능을 한다.
- <c:catch> 커스텀 액션의 시작 태그와 끝 태그 사이에서 에러가 발생하면 실행의 흐름이 곧바로 <c:catch> 액션 다음에 있는 코드로 넘어간다.



- <c:catch> 커스텀 액션은 자바의 try 블록에 해당하는 일만 하기 때문에 catch 블록에 해당하는 일은 별도로 코딩해야 한다.

- <c:catch> 커스텀 액션의 사용 방법

- var 애트리뷰트에 지정된 변수(익셉션 객체가 저장되는 변수)는 <c:catch> 액션의 범위 밖에서도 EL 식을 통해 사용할 수 있으므로, 이를 이용해서 에러 처리를 하면 된다.

익셉션이 발생했는지 체크하는 조건식

<c:if test= “\${e != null} ” >

에러 메시지: \${e.message}

</c:if>

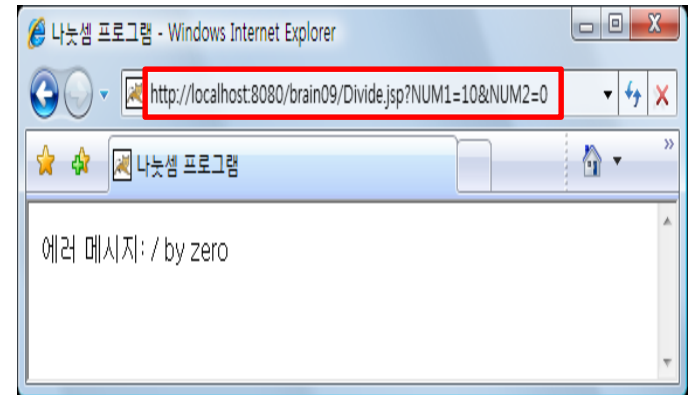
← 에러 메시지를 출력하는 코드

- \${e.message}라는 EL 식은 익셉션 객체 e에 대해 getMessage 메서드를 호출하는 일을 한다.

- <c:catch> 커스텀 액션의 사용 방법

jstl_Divide.jsp

```
<% @page contentType= "text/html; charset=euc-kr" %>
<%@taglib prefix= "c" uri= "http://java.sun.com/jsp/jstl/core" %>
<%
    String str1 = request.getParameter( "NUM1 " );
    String str2 = request.getParameter( "NUM2 " );
    int num1 = Integer.parseInt(str1);
    int num2 = Integer.parseInt(str2);
%>
<HTML>
    <HEAD><TITLE>나눗셈 프로그램</TITLE></HEAD>
    <BODY>
        <c:catch var= "e">
            <% int result = num1 / num2; %>
            나눗셈의 결과는? <%= result %>
        </c:catch>
        <c:if test= "${e != null}">
            에러 메시지: ${e.message}
        </c:if>
    </BODY>
</HTML>
```



[그림 9-16] 예제 9-8의 실행 결과

■ 수치를 포맷하는 <fmt:formatNumber> 커스텀 액션

- 출력할 수치 값은 <fmt:formatNumber>의 value 애트리뷰트에 지정하면 된다.

```
<fmt:formatNumber value= "10000 " />
```

출력할 수치 데이터

- 세 자리마다 쉼표를 찍은 포맷으로 출력하려면 groupingUsed라는 애트리뷰트를 추가하고, 그 값으로 true를 지정하면 된다.

```
<fmt:formatNumber value= "1234500" groupingUsed= "true" />
```

주어진 값을 '1,234,500' 포맷으로 출력하도록 지시합니다

- pattern 애트리뷰트를 사용하면 소수점 아래의 숫자를 원하는 만큼 끊거나 늘려서 표시할 수 있다

```
<fmt:formatNumber value= "3.14158 " pattern= "#.### " />
```

주어진 값을 소수점 아래 2자리까지 끊어서 출력하도록 지시합니다.

- pattern 애트리뷰트의 값에서 0이라고 쓴 위치는 표시할 유효숫자가 없으면 0으로 채워진다.

```
<fmt:formatNumber value= "10.5 " pattern= "#.00 " />
```

주어진 값을 소수점 아래 2자리까지 끊어서 출력하도록 지시합니다
반올림된 값

jstl_NumberFormat.jsp

```
<% @page contentType= "text/html; charset=euc-kr" %>
<% @taglib prefix= "fmt " uri= "http://java.sun.com/jsp/jstl/fmt" %>
<HTML>
  <HEAD><TITLE>숫자 포맷</TITLE></HEAD>
  <BODY>
    첫번째 수: <fmt:formatNumber value= "1234500" groupingUsed= "true" /> <BR>
    두번째 수: <fmt:formatNumber value= "3.14558" pattern= "#.###" /> <BR>
    세번째 수: <fmt:formatNumber value= "3.14558" pattern= "#.00" />
  </BODY>
</HTML>
```

- type 애트리뷰트에 percent라는 값을 지정하면 주어진 수치를 퍼센트 단위로 표시할 수 있다.

```
<fmt:formatNumber value= "0.5" type= "percent" />
```

주어진 수치를 퍼센트 단위로 포맷하여
출력하도록 지시합니다

- type 애트리뷰트에 currency라는 값을 지정하면 주어진 수치가 금액에 적합한 포맷으로 만들어져서 출력된다.

```
<fmt:formatNumber value= "2500000" type= "currency" />
```

주어진 수치를 금액으로 표시하여
출력하도록 지시합니다

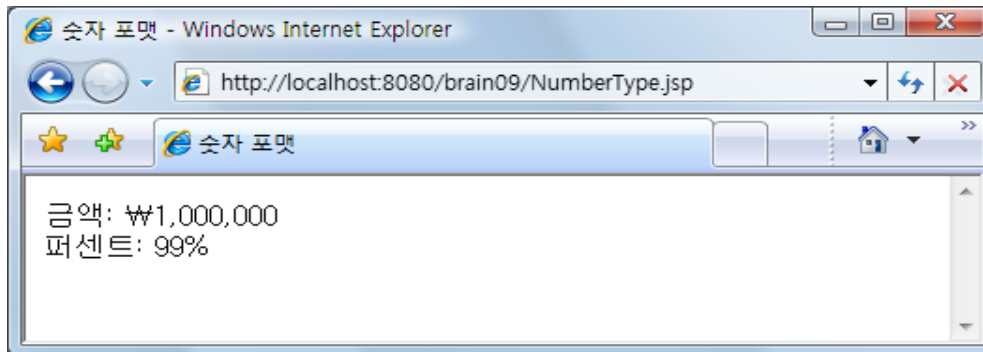
- 화폐 단위를 표시하기 위해서는 currencySymbol이라는 애트리뷰트를 이용하면 된다.

```
<fmt:formatNumber value= "2500000 " type= "currency " currencySymbol= "₩" />
```

금액 앞에 붙는 화폐 단위 표시

jstl_NumberType.jsp

```
<% @page contentType= "text/html; charset=euc-kr"%>
<% @taglib prefix= "fmt" uri= "http://java.sun.com/jsp/jstl/fmt" %>
<HTML>
  <HEAD><TITLE>숫자 포맷</TITLE></HEAD>
  <BODY>
    금액: <fmt:formatNumber value= "1000000 " type= "currency" currencySymbol= "₩" /> <BR>
    퍼센트: <fmt:formatNumber value= "0.99" type= "percent" />
  </BODY>
</HTML>
```



[그림 9-25] 예제 9-17의 실행 결과

■ <fmt:parseNumber>

- 문자열을 숫자(Number 타입)로 변환해 주는 기능을 제공하는 태그

jstl_format.jsp

```
<%@page contentType= "text/html; charset=euc-kr"%>
<%@ taglib prefix= "c" uri= "http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix= "fmt" uri= "http://java.sun.com/jsp/jstl/fmt" %>
<HTML>
  <HEAD><TITLE>숫자 포맷</TITLE></HEAD>
  <body>
    <c:set var= "num" value= "10.5555" />
    10.5555<br>

    반올림 : <fmt:formatNumber value= "${num}" pattern= "#.00" /><br>
    소숫점없앰 : <fmt:parseNumber var= "integerVal" value= "${num}" integerOnly= "true"/>
    ${integerVal}
  </body>
</HTML>
```

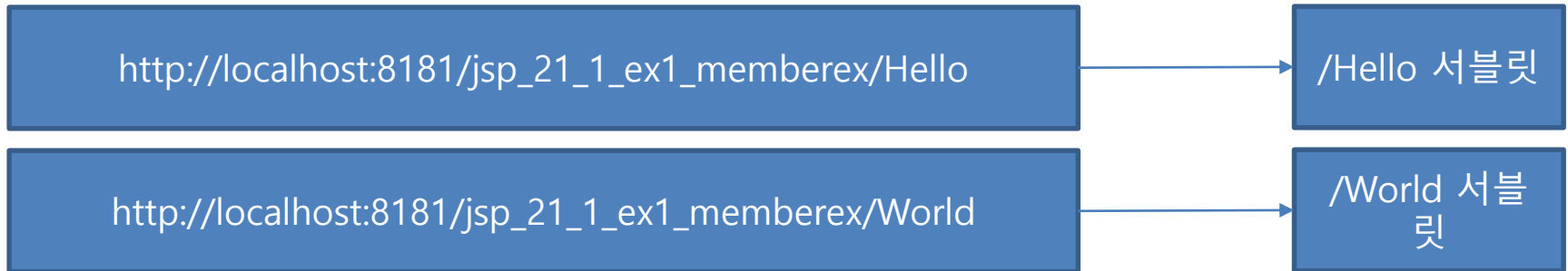
10.5555

반올림 : 10.56

소숫점없앰 : 10

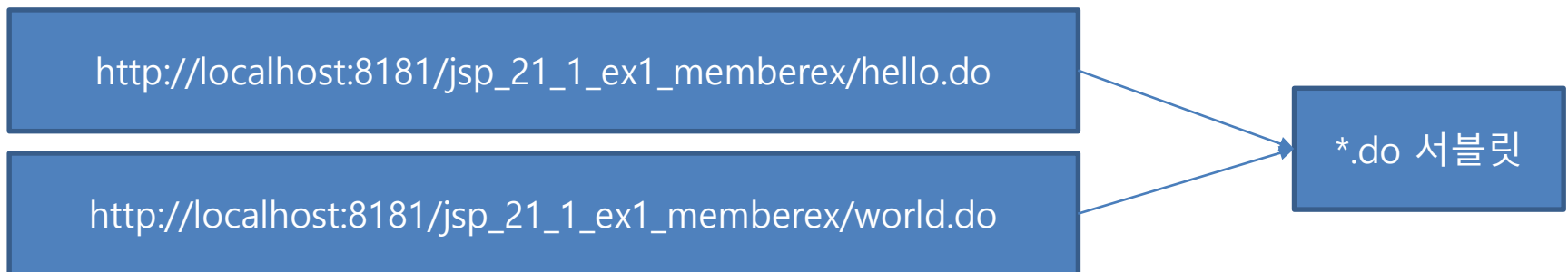
- **디렉토리 패턴**

- 디렉터리 형태로 서버의 해당 컴포넌트를 찾아서 실행하는 구조

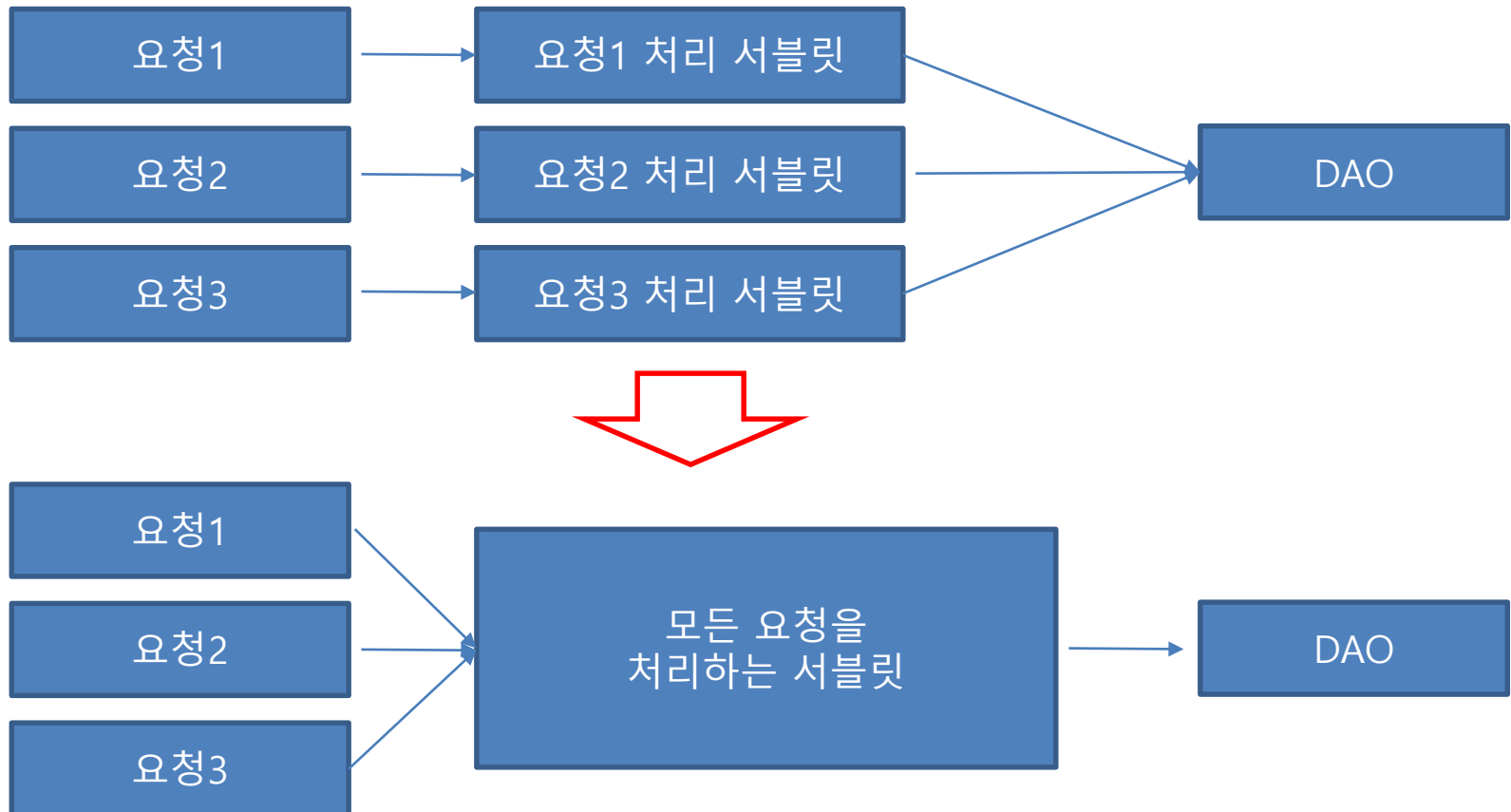


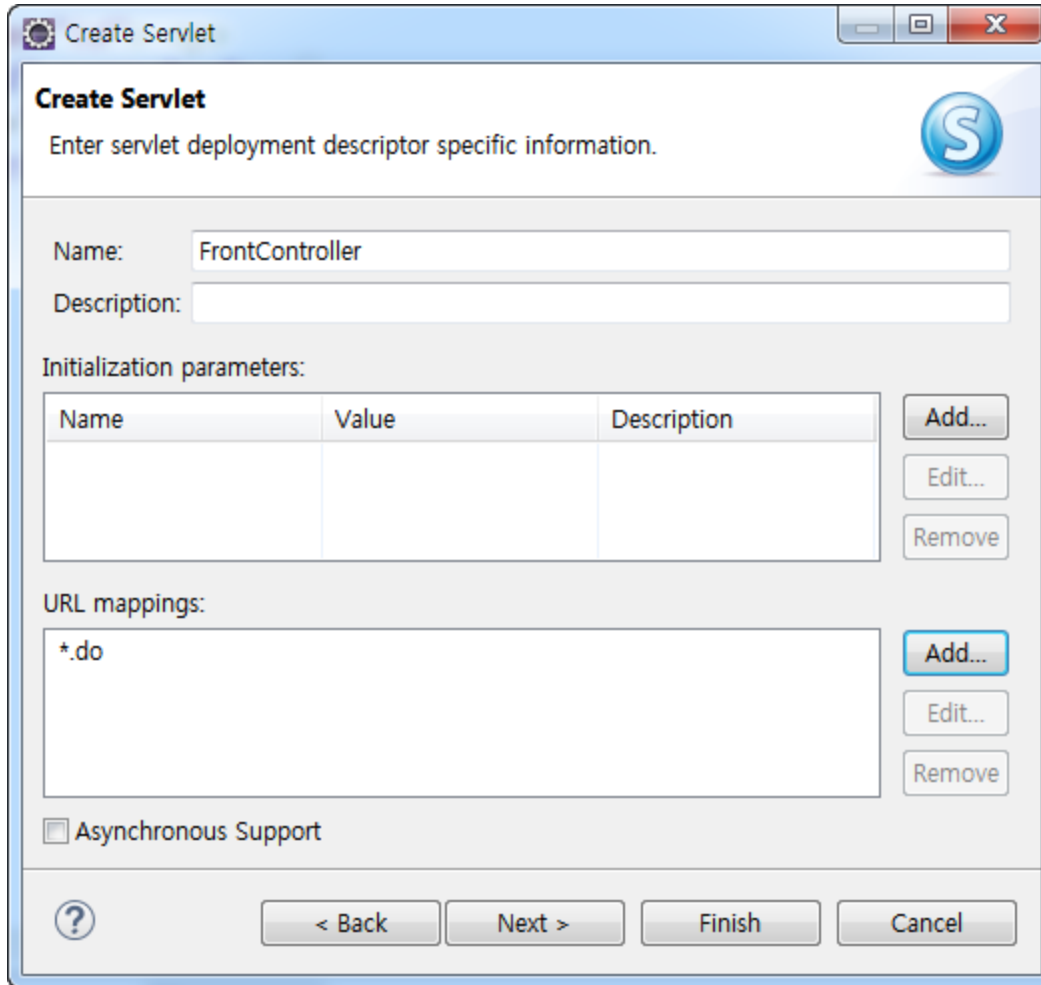
- **확장자패턴**

- 확장자 형태로 서버의 해당 컴포넌트를 찾아서 실행하는 구조



- 클라이언트의 다양한 요청을 한곳으로 집중시켜, 개발 및 유지보수에 효율성을 극대화 시킴
- Tracking이나 Security를 적용할 때 편하게 구현이 가능하고, URL 구성이 간편해짐





Create Servlet

Enter servlet deployment descriptor specific information.

Name:

Description:

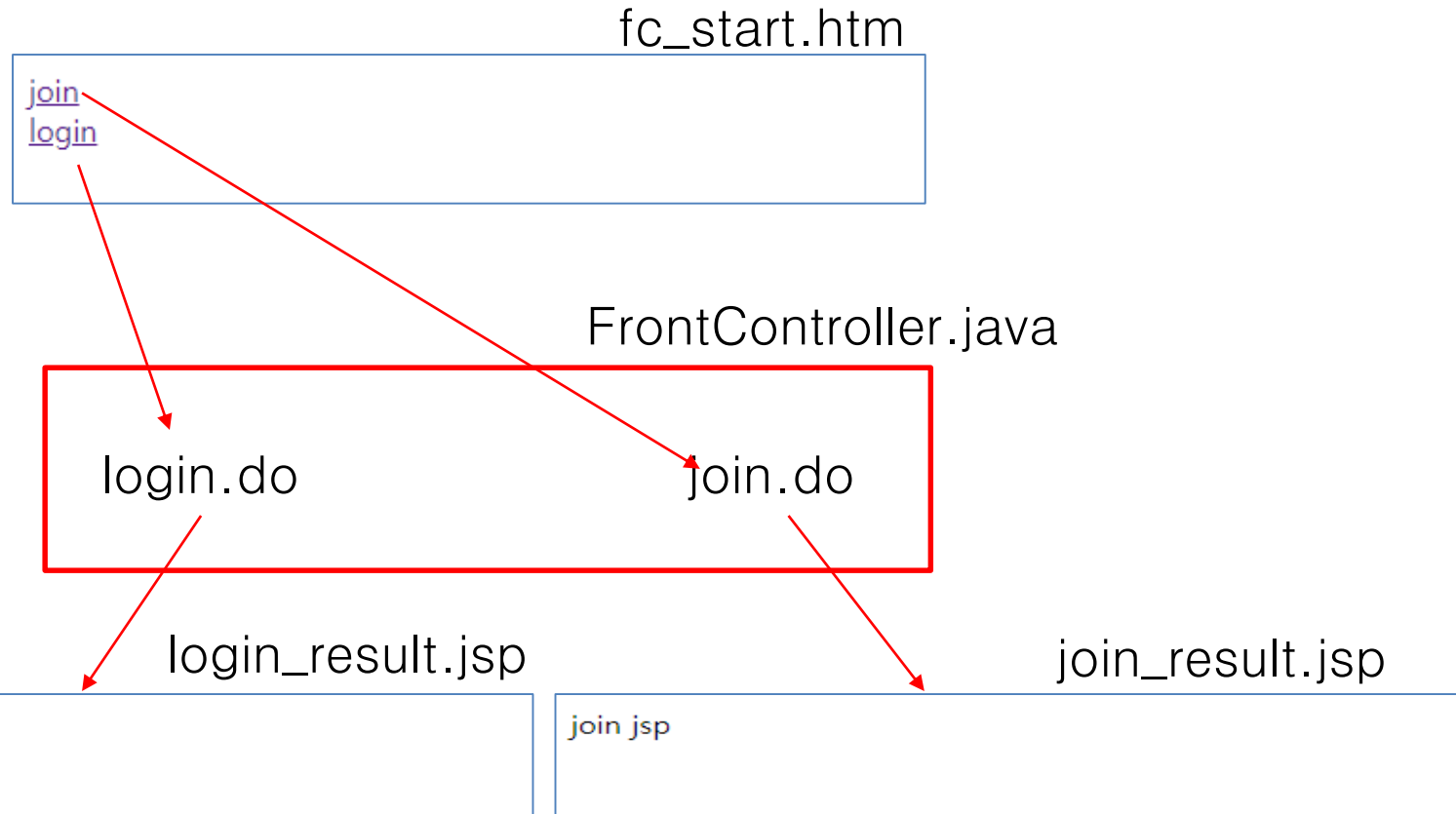
Initialization parameters:

Name	Value	Description

URL mappings:

☐ Asynchronous Support

FrontController 패턴



fc_start.html

```
<body>
  <a href="join.do">join</a><br>
  <a href="login.do">login</a><br>
</body>
```

FrontController.java

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

FrontController.java

```
private void actionPerformed(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    System.out.println("actionDo");

    String uri = request.getRequestURI();
    System.out.println("uri : " + uri);
    String conPath = request.getContextPath();
    System.out.println("conPath : " + conPath);
    String command = uri.substring(conPath.length());
    System.out.println("command : " + command);

    if(command.equals("/join.do")){
        System.out.println("join");
        System.out.println("-----");
        response.sendRedirect("join_result.jsp");
    }else if(command.equals("/login.do")){
        System.out.println("login");
        System.out.println("-----");
        response.sendRedirect("login_result.jsp");
    }
}
```

```
uri - /servlet_lecture_command/insert.bo
conPath - /servlet_lecture_command
com - /insert.bo
```

login_result.html

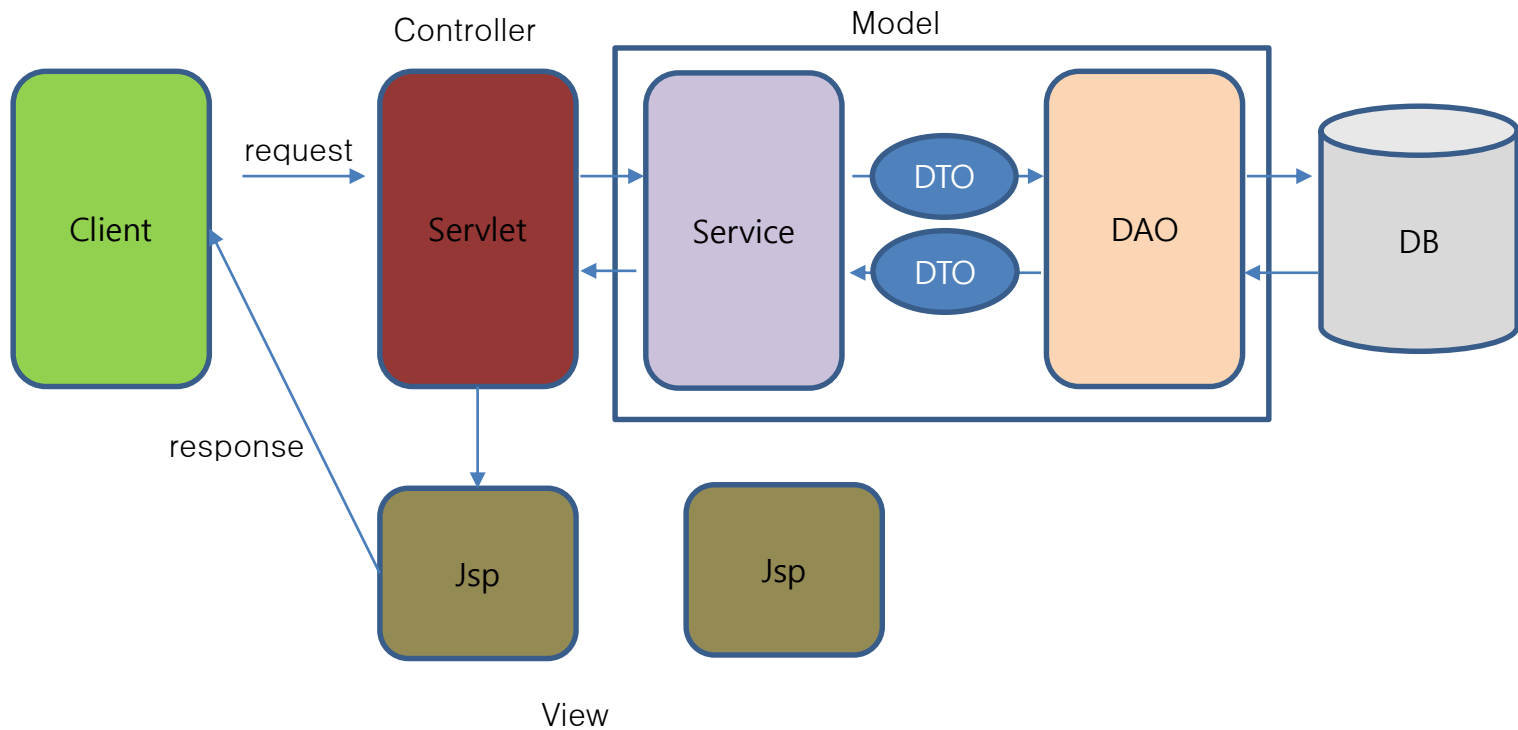
```
<body>  
  login.jsp  
</body>
```

join_result.java

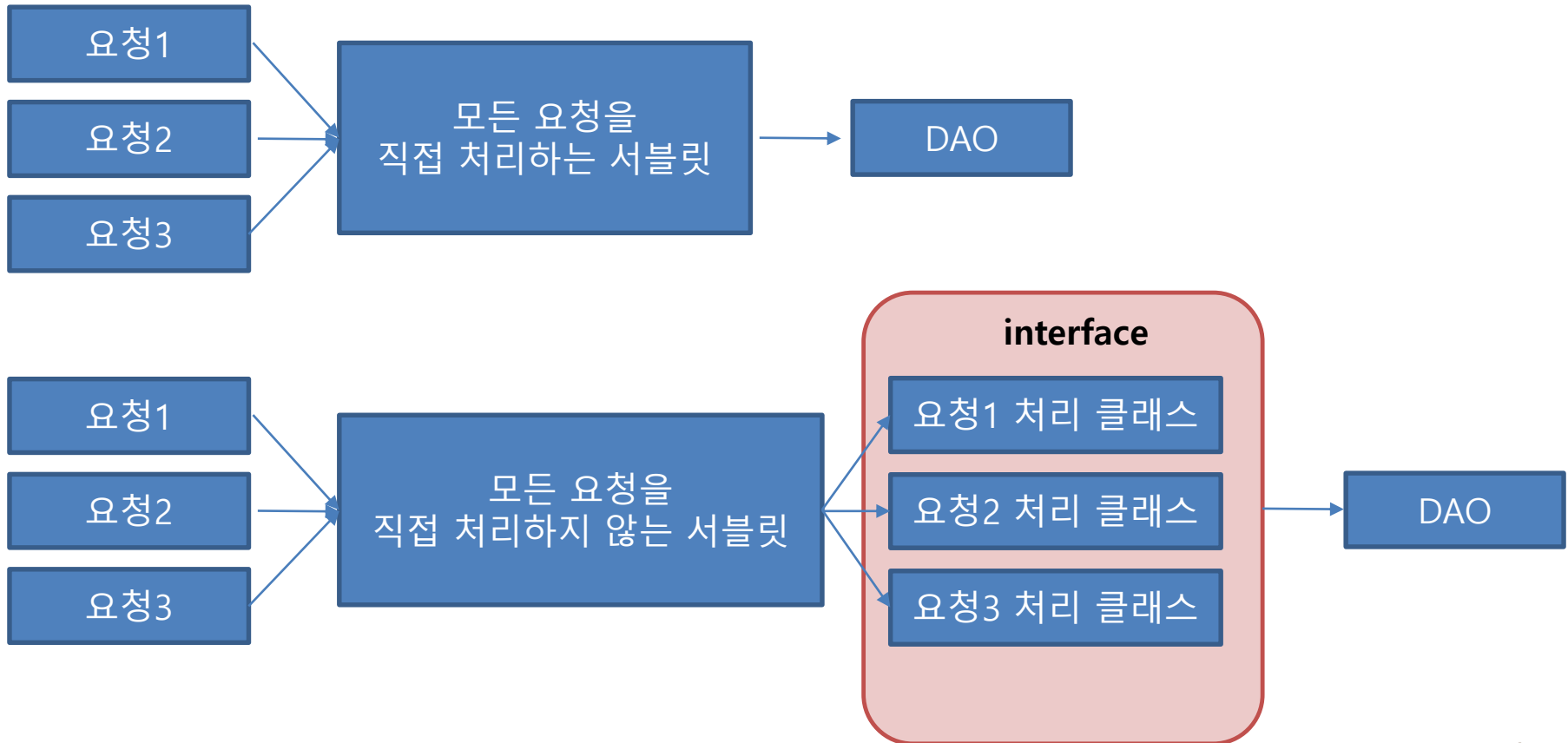
```
<body>  
  join.jsp  
</body>
```

- 확장자 패턴으로 서블릿 생성 : book
 - insert.do : controller 거쳐 insert_result.jsp
 - search.do : controller 거쳐 search.jsp

Command 패턴



- 클라이언트로부터 받은 요청들에 대해서, 서블릿이 작업을 직접 처리 하지 않고, 해당 클래스가 처리하도록 합니다
- 객체의 행위(메서드)를 클래스로 만들어 캡슐화 하는 패턴
 - 재사용성이 높은 클래스를 설계하는 패턴



Command 패턴

login_form.htm

아이디

비밀번호

[join](#)



login success



join_form.htm

join

id :

pw :

name :



join success

login_form.html

```
<form method="get" action="login.do" >
  <ul>
    <li>아이디 </li>
    <li><input type="text" name="id" value="apple"></input></li>
    <li>비밀번호</li>
    <li><input type="text" name="pw" value="orange"></input></li>
    <li><input type="submit" value="로그인"></input></li>
    <li><a href=join_form.html>join</a></li>
  </ul>
</form>
</body>
```

join_form.html

```
<body>
  <h2> join </h2>
  <form action="join.do">
    id : <input type="text" name="id"> <p>
    pw : <input type="password" name="pw"><p>
    name : <input type="text" name="name"> <p>
    <input type="submit" value="send">
  </form>
</body>
```


FrontController.java

```
private void actionDo(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    System.out.println("actionDo");
    request.setCharacterEncoding("utf-8");

    String uri = request.getRequestURI();
    String conPath = request.getContextPath();
    String com = uri.substring(conPath.length());
    String unit[ ] = com.split("WW/");
    System.out.println("uri - "+uri);
    System.out.println("conPath - "+conPath);
    System.out.println("com - "+com);

    MemberDto mdto = new MemberDto();
    Service command = null;
    String id = null;
    String pw = null;
    String name = null;
    int result=0;
    String result_page=null;
    HttpSession session = request.getSession();
```

FrontController.java

```
if(com.equals("/join.do")) {  
    id = request.getParameter("id");  
    pw = request.getParameter("pw");  
    name=request.getParameter("name");  
    mdto.setId(id);  
    mdto.setPw(pw);  
    mdto.setName(name);  
    command=new InsertCommand();  
    result=command.execute(mdto);  
    result_page="insertResult.jsp";  
    if(result==1)  
    {  
        session.setAttribute("insertResult", "join success");  
    }  
    else {  
        session.setAttribute("insertResult", "join fail");  
    }  
}
```

- Dto

LoginCommand.java

```
public class MemberDto {  
    public String id;  
    public String pw;  
    public String name;  
    public String getId() {  
        return id;  
    }  
    public void setId(String id) {  
        this.id = id;  
    }  
    public String getPw() {  
        return pw;  
    }  
    public void setPw(String pw) {  
        this.pw = pw;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

- FrontController

FrontController.java

```
else if(com.equals("/login.do")) {
    id = request.getParameter("id");
    pw = request.getParameter("pw");
    mdto.setId(id);
    mdto.setPw(pw);
    command=new LoginCommand();
    result=command.execute(mdto);
    System.out.println("com - "+com);

    result_page="loginResult.jsp";
    if(result==1)
    {
        session.setAttribute("loginResult", "login success");
    }
    else {
        session.setAttribute("loginResult", "login fail");
    }
} else {
    System.out.println("controller error");
}
response.sendRedirect(result_page);
}
```

- Service

service.java

```
public interface Service {  
    public int execute(MemberDto mdto) throws ServletException, IOException;  
}
```

InsertCommand.java

```
public class InsertCommand implements Service {  
  
    @Override  
    public int execute(MemberDto mdto) throws ServletException, IOException{  
  
        MemberDao mdao = new MemberDaoImpl();  
        int retVal=mdao.insertMember(mdto);  
        return retVal;  
    }  
}
```

LoginCommand.java

```
public class LoginCommand implements Service {  
  
    @Override  
    public int execute(MemberDto mdto) throws ServletException, IOException{  
  
        String userId=mdto.getId();  
        String userPw=mdto.getPw();  
        MemberDao mdao = new MemberDaoImpl();  
        String dbPw=mdao.loginMember(userId);  
        if(userPw.equals(dbPw)){  
            return 1;  
        }  
        else {  
            return -1;  
        }  
    }  
}
```

- Dao

MemberDao.java

```
import java.sql.*;

public interface MemberDao {

    public int insertMember(MemberDto mdto);
    public String loginMember(String id);
    public Connection getConnection() ;
    public void closeConnection(ResultSet set, PreparedStatement pstmt, Connection connection);

}
```

MemberDaoImpl.java

```
import java.sql.*;

public class MemberDaoImpl implements MemberDao{

    public int insertMember(MemberDto mdto)
    {
        int ret=0;
        String sql = "insert into members (id,pw,name) values (?, ?, ?)";
        Connection conn=null;
        PreparedStatement pstmt = null;

        try {
            conn=getConnection();
            System.out.println("connection ok");
            pstmt = conn.prepareStatement(sql);
            pstmt.setString(1,mdto.getId());
            pstmt.setString(2,mdto.getPw());
            pstmt.setString(3,mdto.getName());
            pstmt.executeUpdate();
            System.out.println("insert ok");
            ret=1;
        }catch (SQLException e){
            ret=-1;
            System.out.println("access error.");
            System.out.println(e.getMessage());
            e.printStackTrace();
        }finally
        {
            closeConnection(null,pstmt,conn);
        }
    }
}
```


MemberDaoImpl.java

```
public String loginMember(String id)
{
    String pw=null;
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rs=null;
    String pwDb=null;
    try{
        String sql="select pw from members where id = ?";
        conn=getConnection();
        System.out.println("connection ok");
        pstmt = conn.prepareStatement(sql);
        pstmt.setString(1, id);
        rs= pstmt.executeQuery();

        while (rs.next())
        {
            pwDb = rs.getString("pw");
            System.out.println(" pw : "+pwDb);
        }
        pw=pwDb;

    }catch (SQLException e){
        System.out.println("SQLException error.");
        e.printStackTrace();
    }finally{
        closeConnection(rs,pstmt,conn);
    }
    return pw;
}
```

MemberDaoImpl.java

```
public Connection getConnection() {  
  
    Connection conn=null;  
    String DBName = "jsp_servlet_db";  
    String dbURL = "jdbc:mysql://localhost:3306/" + DBName;  
    String sslStr="?useSSL=false";  
  
    try {  
  
        Class.forName("com.mysql.jdbc.Driver");  
        System.out.println("JDBC driver load success");  
  
        conn = DriverManager.getConnection(dbURL+sslStr  
            , "root", "1111133333");  
        System.out.println("DB connection success");  
    } catch (ClassNotFoundException e) {  
        System.out.println("JDBC driver load fail !!");  
    } catch (SQLException e) {  
        System.out.println("DB connection fail !!");  
    }  
  
    return conn;  
}
```

MemberDaoImpl.java

```
public void closeConnection(ResultSet set, PreparedStatement pstmt, Connection connection) {  
    if(set!=null)  
    {  
        try {  
            set.close();  
        } catch (Exception e2) {  
            e2.printStackTrace();  
        }  
    }  
    if(pstmt!=null)  
    {  
        try {  
            pstmt.close();  
        } catch (Exception e2) {  
            e2.printStackTrace();  
        }  
    }  
    if(connection!=null)  
    {  
        try {  
            connection.close();  
        } catch (Exception e2) {  
            e2.printStackTrace();  
        }  
    }  
}
```

- 확장자 패턴으로 서블릿 생성 : book
 - insert.do : controller 거쳐 insert_result.jsp
 - search.do : controller 거쳐 search.jsp
 - service, dao 추가

- **요청폼**

- **Multipart는 HTTP를 통해 File을 SERVER로 전송하기 위해 사용되는 Content-type**

모든 문자를 인코딩하지 않음을 명시함.

```
<h1>File Upload</h1>
<form method="post" action="UploadServlet" enctype="multipart/form-data">
    Select file to upload: <input type="file" name="file" size="60" /><br />
    <br /> <input type="submit" value="Upload" />
</form>
```

- **@MultipartConfig Annotation**

```
@MultipartConfig(  
    fileSizeThreshold = <size in bytes>,  
    maxFileSize = <size in bytes>,  
    maxRequestSize = <size in bytes>,  
)
```

- **fileSizeThreshold:**
 - 이 크기가 넘으면 디스크의 임시디렉토리에 저장
 - 기본값은 0
- **maxFileSize**
 - 파일의 최대크기
 - 기본값은 -1L -> 제한이 없음
- **maxRequestSize**
 - 여러파일 및 전송값 등을 합한 최대 크기
 - 기본값은 -1L -> 제한이 없음

- 1. 파일 저장 경로 설정
- 2. `HttpServletRequest` 로 부터 Part정보를 얻음

Method Summary

Methods

Modifier and Type	Method and Description
void	delete() Deletes the underlying storage for a part, including deleting any associated temporary disk file.
java.lang.String	getContentType() Obtain the content type passed by the browser or null if not defined.
java.lang.String	getHeader(java.lang.String name) Obtains the value of the specified part header as a String.
java.util.Collection<java.lang.String>	getHeaderNames() Returns a Collection of all the header names provided for this part.
java.util.Collection<java.lang.String>	getHeaders(java.lang.String name) Obtain all the values of the specified part header.
java.io.InputStream	getInputStream() Obtain an <code>InputStream</code> that can be used to retrieve the contents of the file.
java.lang.String	getName() Obtain the name of the field in the multipart form corresponding to this part.
long	getSize() Obtain the size of this part.
void	write(java.lang.String fileName) A convenience method to write an uploaded part to disk.

- **3. 2로부터 얻은 Part 로 부터 얻은 file name을 이용 파일 객체 생성**
 - **파일이름 얻기**
 - 파일 이름은 content-disposition 헤더 값을 파싱(parsing)해야 함
 - filename=""에 해당하는 문자열

Content-Disposition: form-data; name="(서버에서 받을 이름)"; filename="(보낼 파일 이름)"

- **2로 부터 얻은 Part 객체를 이용 파일 쓰기**
- **주의점**
 - 1. 저장되는 파일 이름은 같을 경우도 있으므로 년부터밀리세컨드까지 값을 얻어서 파일이름으로 이용
 - 2. 실제 파일이름과 2번의 파일이름, 크기, 경로를 db에 저장해야 함

```
private String genSaveFileName(String extName) {  
    String fileName = "";  
    Calendar calendar = Calendar.getInstance();  
    fileName += calendar.get(Calendar.YEAR);  
    fileName += calendar.get(Calendar.MONTH);  
    fileName += calendar.get(Calendar.DATE);  
    fileName += calendar.get(Calendar.HOUR);  
    fileName += calendar.get(Calendar.MINUTE);  
    fileName += calendar.get(Calendar.SECOND);  
    fileName += calendar.get(Calendar.MILLISECOND);  
    fileName += extName;  
    System.out.println("genSaveFileName() : "+fileName);  
    return fileName;  
}
```


- 1. File 객체 생성
- 2. FileInputStream 객체 생성
- 3. ServletOutputStream 객체 생성
- 4. http response header 설정
 - Content-Type
 - Content-Disposition
- 2번의 객체로 파일 읽기, 3번의 객체로 response에 파일 데이터 쓰기
- 6. 2번 객체 close

- 저장위치
 - C:\Tomcat 8.5\wtpwebapps\file

- **http response header 설정**

- **Content-Type**

- 다운로드 되는 파일의 타입 설정
 - 기타파일 : **application/octet-stream**

타입	일반적인 서브타입 예시
text	text/plain, text/html, text/css, text/javascript
image	image/gif, image/png, image/jpeg, image/bmp, image/webp
audio	audio/midi, audio/mpeg, audio/webm, audio/ogg, audio/wav
video	video/webm, video/ogg
application	application/octet-stream, application/vnd.ms-powerpoint, application/xhtml+xml, application/xml, application/pdf

- **Content-Disposition** : 파일 다운로드를 처리하는 HTTP 헤더 중 하나다. 웹 서버 응답에 이 헤더를 포함하면 그 내용(보통 사진 등의 파일 데이터)을 웹 브라우저로 바로 보거나 내려받도록 설정할 수 있다.
 - inline
 - attachment