Servlet & JSP

웹 프로그래밍 개요

인터넷(네트워크 통신)의 이해

- World Wide Web
  - 인터넷과 동일시하는 경향이 있으나, 인터넷 기반 대표 서비스 중 하나

이름	프로토콜	포트	기능
Email	SMTP/POP3/IMAP	25/110/114	이메일 서비스
FTP	FTP	21	파일 전송 서비스
DNS	DNS	53	도메인 네임 서비스
NEWS	NNTP	119	인터넷 뉴스 서비스
WWW	HTTP/HTTPS	80/443	웹 서비스

  - 인터넷 (Internet)
    - TCP/IP 기반의 네트워크가 전세계적으로 확대되어 하나로 연결된 네트워크의 네트워크
    - Network of Networks (네트워크의 결합체)

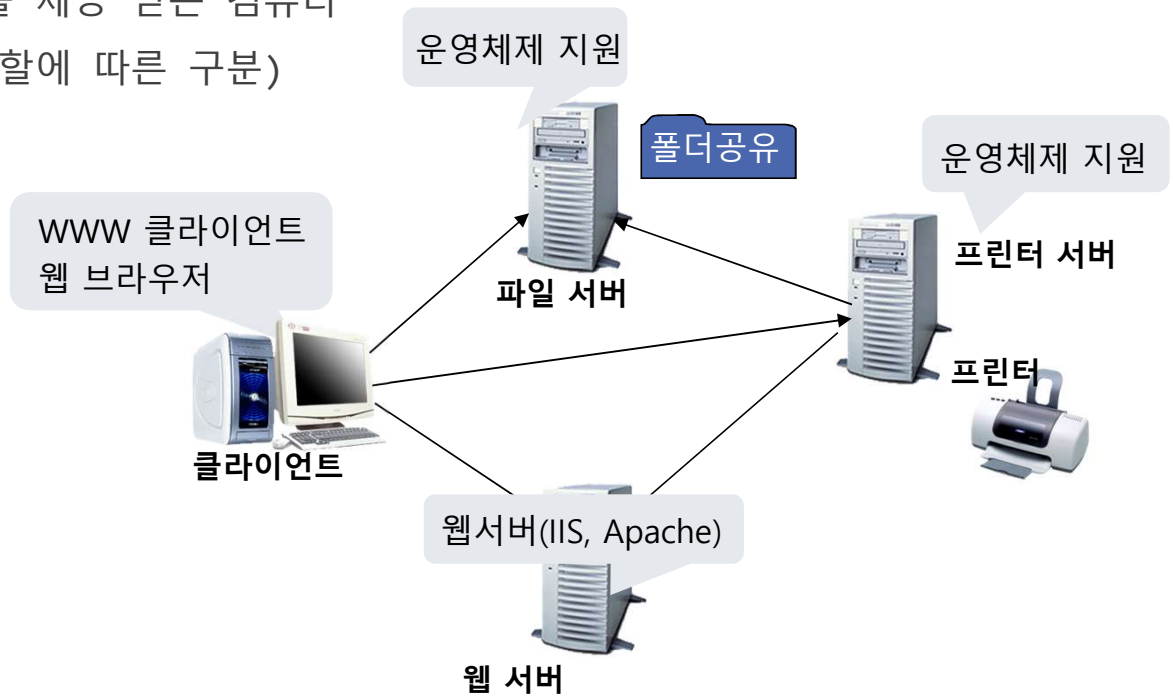
인터넷(네트워크 통신)의 이해

- TCP/IP
  - 하드웨어, 운영체계, 접속 매체와 관계 없이 동작할 수 있는 개방형 구조
  - OSI 7 계층에서 4계층으로 단순화

OSI 7 계층	TCP/IP 4 계층	
응용 계층	응용 계층	• 네트워크를 사용하는 응용프코그램 • WWW, FTP, 텔넷, SMTP 등
표현 계층		
세션 계층	전송 계층	• 도착지까지 데이터를 전송 • 각각의 시스템을 연결 • TCP 프로토콜을 이용하여 데이터를 전송
전송 계층		
네트워크 계층	인터넷 계층	• 데이터 정의 및 경로 지정 • 정확한 라우팅을 위해 IP 프로토콜 사용 • IP 주소가 위치하는 계층
데이터 링크 계층		
물리 계층	물리 계층	• 물리적 계층 (이더넷 카드 등 하드웨어)

인터넷(네트워크 통신)의 이해

- 서비스(SERVICE : 클라이언트 / 서버 통신)
  - 서버 : 네트워크에서 서비스를 제공하는 컴퓨터
  - 클라이언트 : 네트워크에서 서비스를 제공 받는 컴퓨터
  - 하드웨어적 구분은 사실상 없음 (역할에 따른 구분)



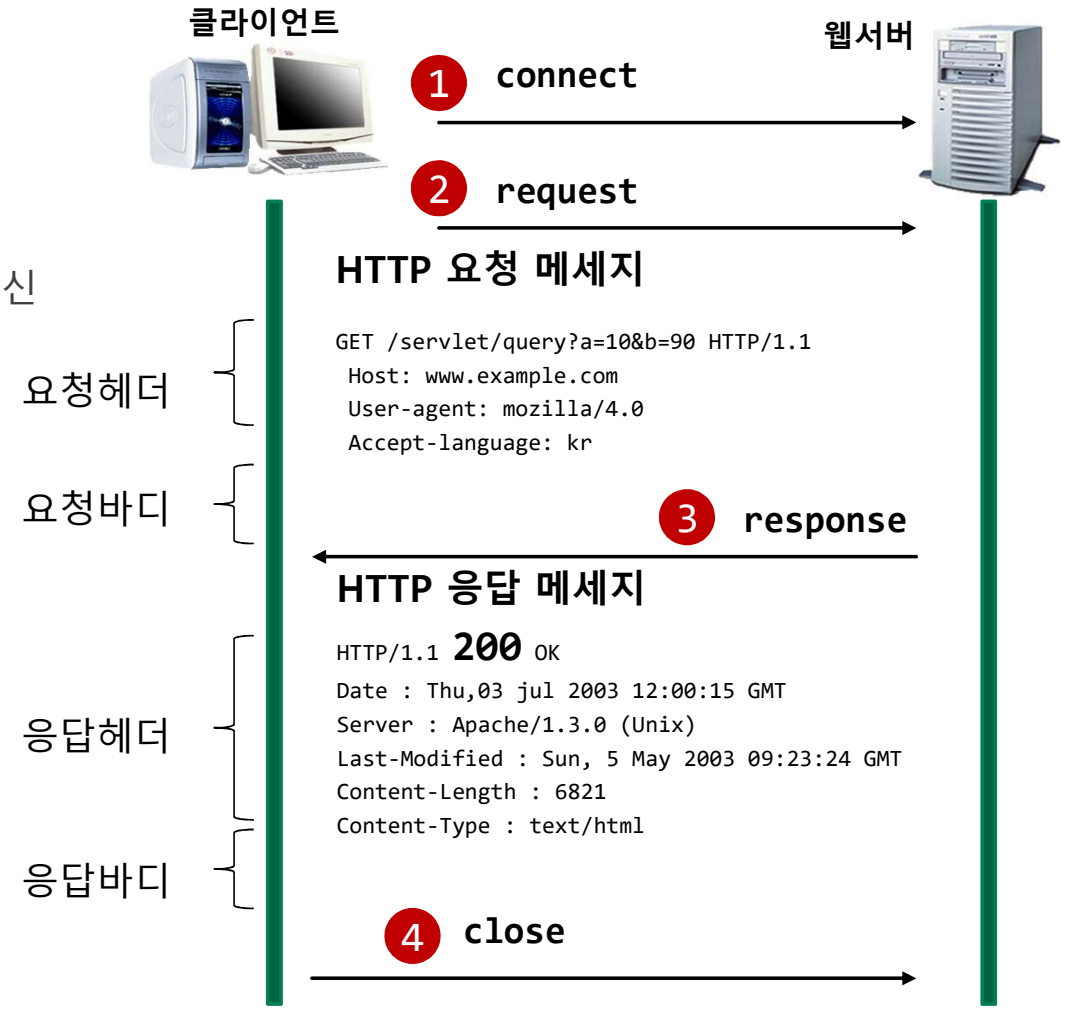
인터넷(네트워크 통신)의 이해

- 프로토콜(PROTOCOL: 규약)
  - 컴퓨터나 원거리 장비 사이에서 메시지를 주고 받는 양식과 규칙의 체계
  - 신호 체계, 인증, 오류 감지 및 수정 기능을 포함할 수 있음

- 포트(PORT)
  - 컴퓨터 상호 통신을 위해 프로토콜에서 이용하는 가상의 연결 종단점
  - 0 ~ 65,535 사이의 숫자

HTTP Protocol

- HTTP (Hyper Text Transfer Protocol)
  - WWW 서비스를 위한 TCP/IP 기반 응용 계층 프로토콜 중 하나
  - 웹 서버와 클라이언트는 HTTP를 이용하여 통신
  - 무 상태 연결(Stateless Connection)

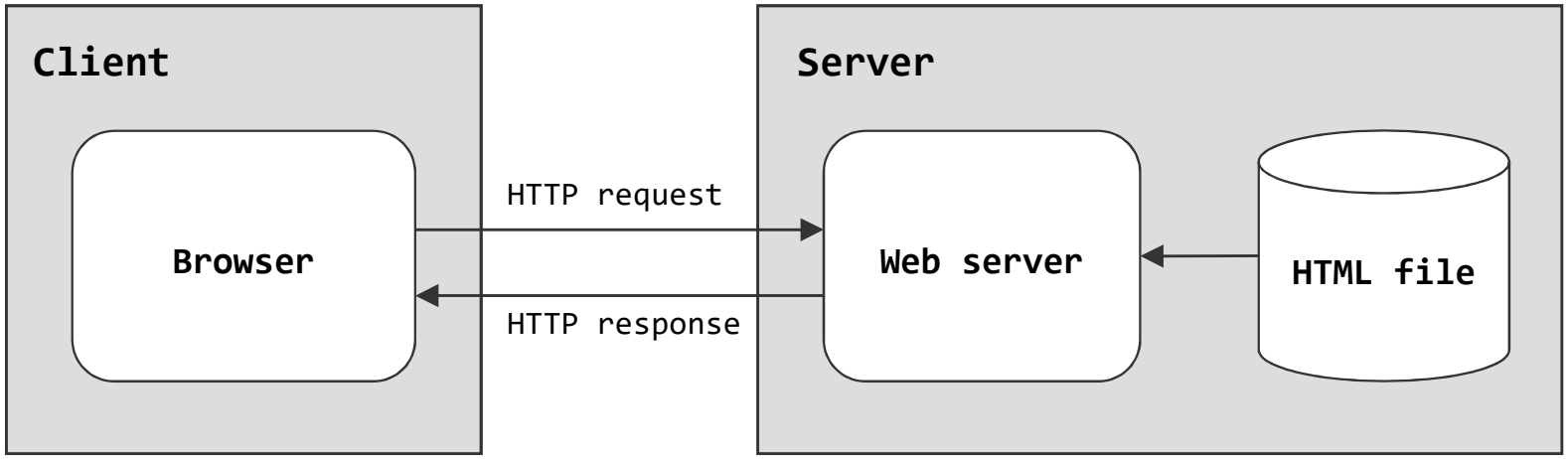


URL

- URL (Uniform Resource Locator)
  - 인터넷 상 자원의 위치
  - 특정 웹 서버의 특정 파일에 접근하기 위한 경로 혹은 주소

웹 프로그래밍

- 정적(Static) 웹 페이지 (~~웹 프로그래밍~~, 퍼블리싱)



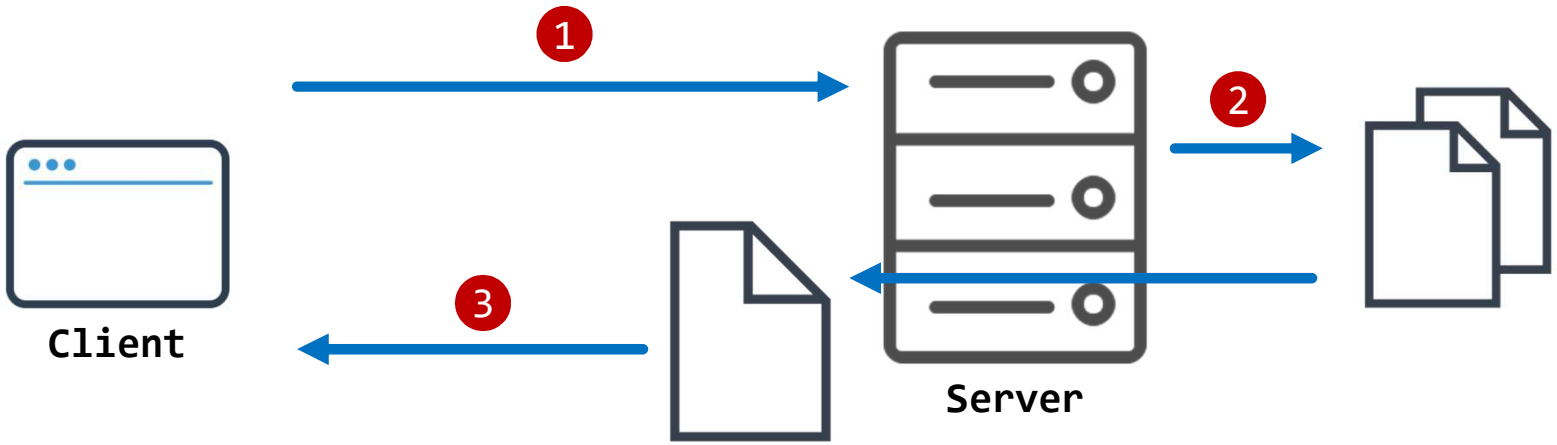
- 웹 페이지는 HTML이라는 표준 마크업 언어로 작성

웹 프로그래밍

- 정적(Static) 웹 페이지의 처리 방식
  - HTML(Hyper Text Martup Language)는 브라우저가 웹 페이지로 변환하는 언어
  - 정적 웹 페이지는 파일 형태로 저장되어 있으면서 사용자의 입력에 따라 변하지 않는 HTML 문서
  - HTTP(Hyper Text Transfer Protocol)는 웹 브라우저와 웹 서버가 통신하는 프로토콜이다
  - 웹 브라우저는 HTTP 요청(HTTP Request) 메시지를 서버에 전송하여 웹 서버의 페이지를 요청한다
  - 웹 서버는 HTTP 응답(HTTP Response) 메시지를 전달하여 HTTP 요청에 응답한다.
  - 정적인 웹 페이지에서는 HTTP 응답이 HTML 문서를 포함한다

웹 프로그래밍

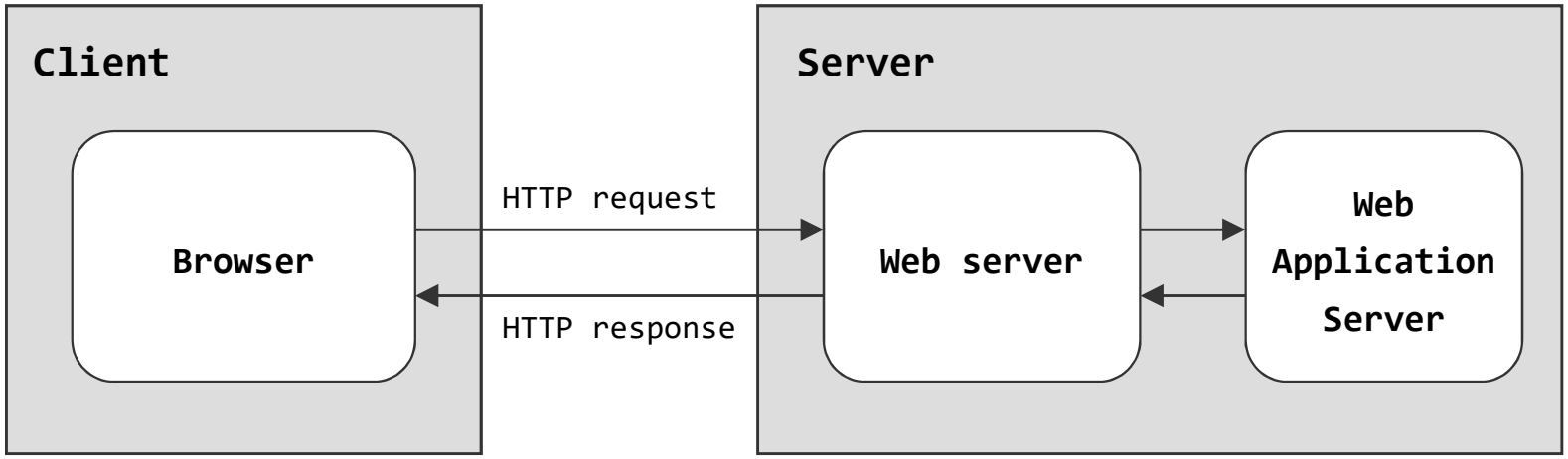
- 정적(Static) 웹 페이지의 처리 방식



1) 클라이언트가 웹 페이지를 서버에 요청
2) 웹 서버는 요청한 페이지를 내부에서 검색
3) 검색된 HTML 스트림을 브라우저에 반환

웹 프로그래밍

- 동적(Dynamic) 웹 페이지 (웹 프로그래밍)



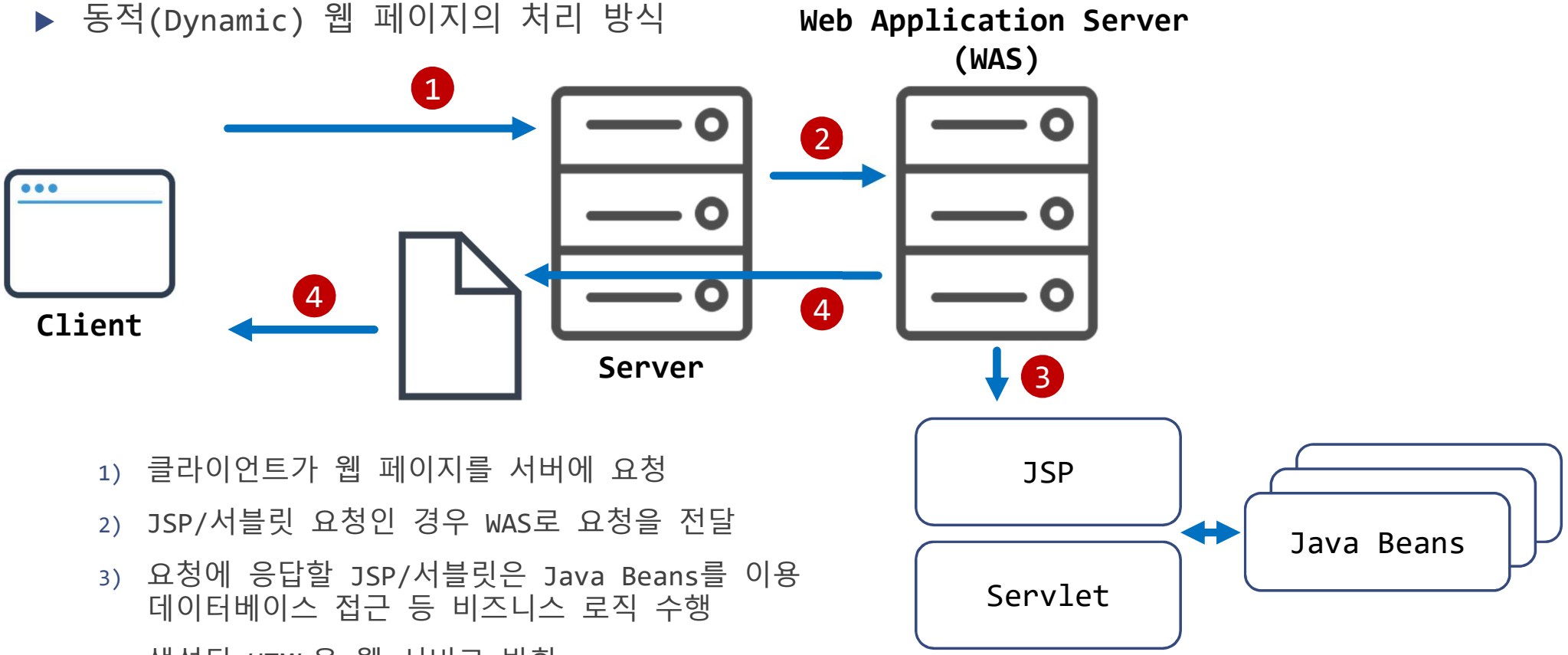
- Java에서 동적 웹 페이지를 만들어 내는 기술
  - Servlet
  - JSP (Java Server Pages)

웹 프로그래밍

- 동적(Dynamic) 웹 페이지의 처리 방식
  - 동적 웹 페이지는 웹 응용프로그램에 의해 생성되는 HTML 문서이다.
  - 동적 웹 페이지는 웹 브라우저가 웹 응용프로그램에 전달한 파라미터 값에 따라 웹 페이지가 변한다.
  - 웹 서버가 동적인 웹 페이지에 대한 요청을 받으면 서버는 웹 응용프로그램으로 요청을 넘긴다. 그러면 응용프로그램이 HTML 문서를 생성하여 웹 서버로 결과를 전달한다.
  - 웹 서버는 전달 받은 HTML 문서를 HTTP 응답(HTTP Response)로 감싼 후 브라우저로 결과를 전달한다.
  - 브라우저는 응답으로 전달 받은 HTML 문서가 정적인 HTML 파일인지 웹 응용프로그램에 의해 동적으로 생성된 문서인지 알지 못한다. 어느 쪽이든 브라우저는 전달 받은 HTML 문서를 해석하여 화면에 표시해 준다.

웹 프로그래밍

- 동적(Dynamic) 웹 페이지의 처리 방식



1) 클라이언트가 웹 페이지를 서버에 요청
2) JSP/서블릿 요청인 경우 WAS로 요청을 전달
3) 요청에 응답할 JSP/서블릿은 Java Beans를 이용 데이터베이스 접근 등 비즈니스 로직 수행
4) 생성된 HTML을 웹 서버로 반환
5) 웹 서버는 전달 받은 HTML을 브라우저로 전송

웹 응용프로그램

- 웹 응용프로그램(Web Application)에 필요한 구성 요소
  - Java 웹 응용프로그램은 JSP와 Servlet으로 구성
  - JSP/Servlet 엔진(혹은 컨테이너)은 서버에서 JSP와 Servlet을 구동할 수 있게 하는 소프트웨어
  - J2EE(Java 2 Platform, Enterprise Edition)는 웹 서버와 JSP/Servlet 엔진이 어떻게 상호 작용해야 하는지를 명세하고 있음
  - JSP/Servlet 엔진이 작동하기 위해서는 SDK에 접근해야 한다.
  - EJB(Enterprise Java Beans)를 사용하는 웹 응용프로그램은 EJB Container로 알려진 추가적인 서버 컴포넌트가 필요

웹 응용프로그램

- JSP (Java Server Pages)
  - JSP(Java Server Pages)는 HTML 코드 내에 Java 코드를 포함하는 형태로 구성
  - JSP 페이지가 처음 호출될 때, JSP 엔진은 JSP 코드를 Servlet으로 변환하고 컴파일한다. 그리고 Servlet 엔진이 서블릿을 구동

- 서블릿(Servlet)
  - 서블릿(Servlet)은 서버에서 동작하는 Java 클래스
  - 서블릿은 HttpServlet 클래스를 상속
  - HTML 코드를 브라우저로 반환하기 위해서, 서블릿은 out 객체의 println 메서드를 이용 -> 이는 HTML 코드를 작성하기 어렵게 만드는 요소가 된다
  - 서블릿과 JSP로부터 최상의 결과를 얻으려면, 웹 페이지를 개발할 때 이 두 가지 컴포넌트를 조화롭게 사용해야 한다.
    - 웹 페이지를 구성하는 화면(HTML) -> JSP로 작성
    - 로직 및 프로세스를 처리하는 부분 -> 서블릿이 처리

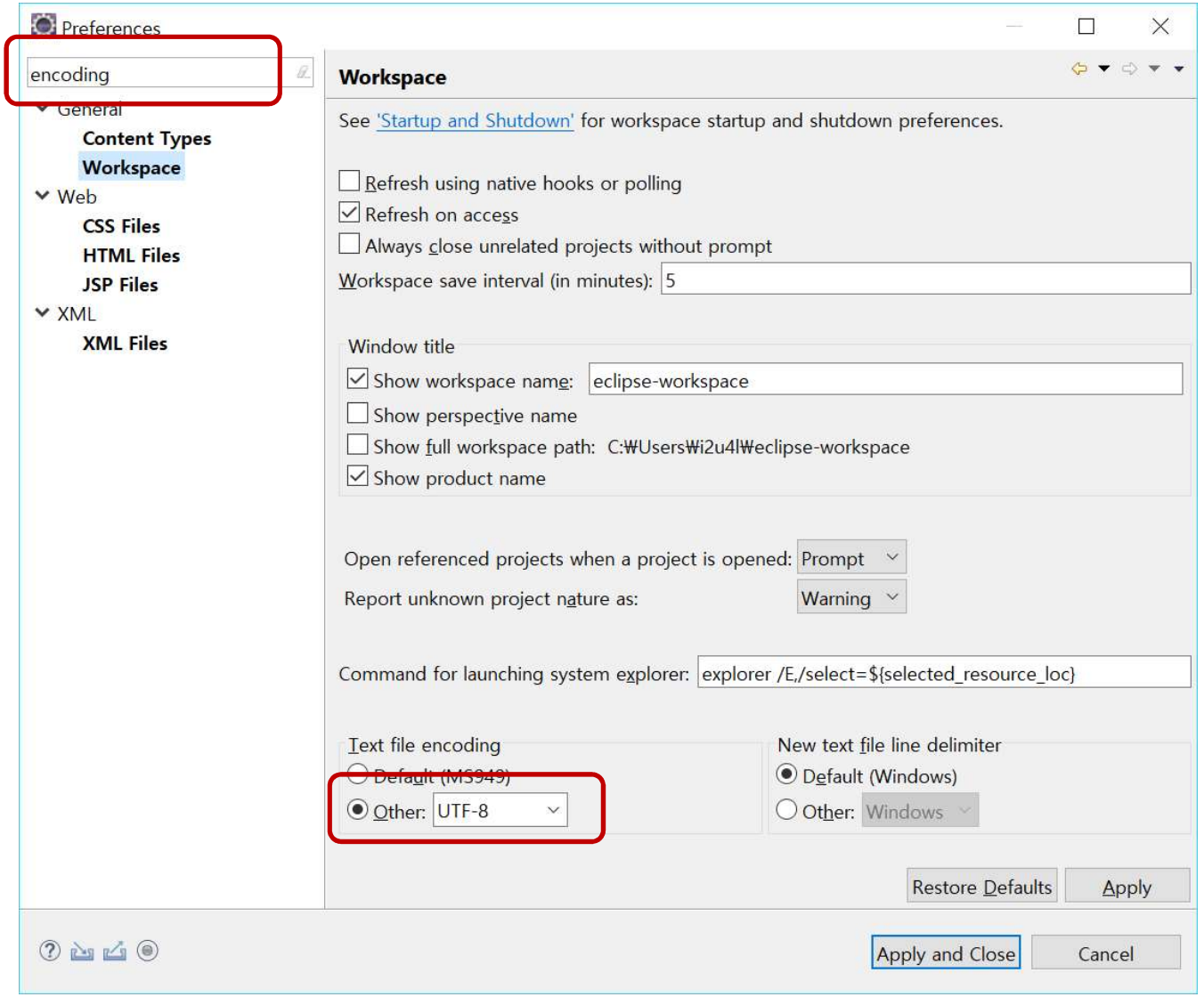
개발 환경 구축과 첫 번째 웹 프로젝트

개발 환경 설정

- Requirements
  - JDK 1.8 이상
    - http://java.oracle.com
  - Eclipse IDE
    - http://www.eclipse.org
    - 가급적 Java EE IDE for Web Developer 버전으로 설치
  - Apache Tomcat 8.0 이상
    - http://tomcat.apache.org/

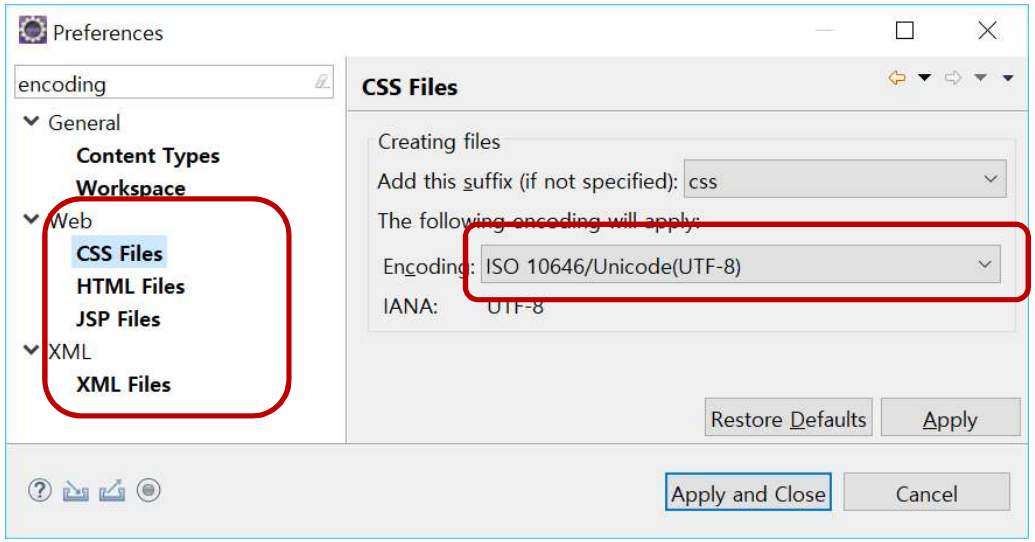
개발 환경 설정
: Workspace Encoding

- Workspace의 Encoding을 UTF-8로 설정
  - Window > Preferences > General > Workspaces
  - 좌상단 검색창에 encoding을 입력하면 쉽게 찾을 수 있음



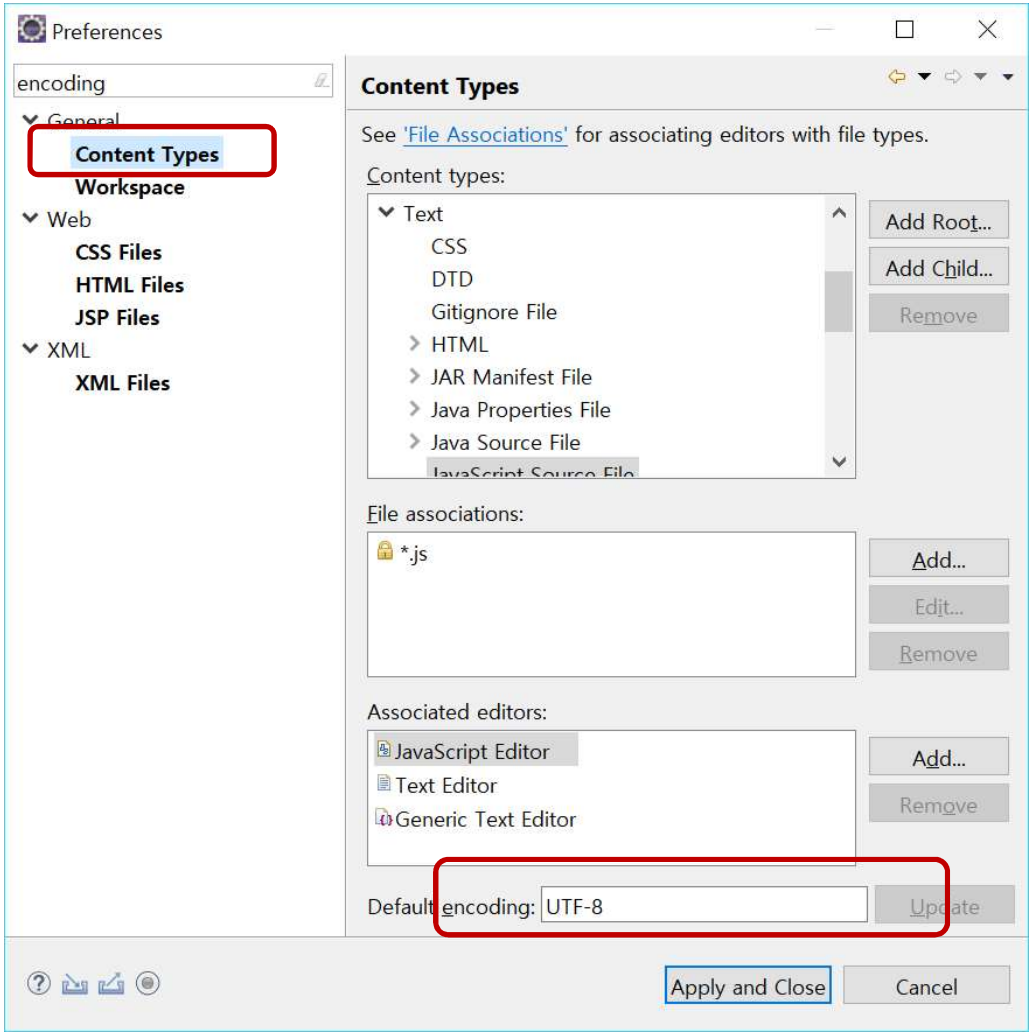
개발 환경 설정
: File Encoding

- 아래 형식의 파일 Encoding을 모두 UTF-8로 설정
  - Window > Preferences > Web, XML
    - CSS Files
    - HTML Files
    - JSP Files
    - XML Files
  - 좌상단 검색창에 encoding을 입력하면 쉽게 찾을 수 있음



개발 환경 설정
: Contents Type Encoding

- Contents Types 내 주요 파일 Encoding도 UTF-8인지 확인
  - 예) JavaScript Source File



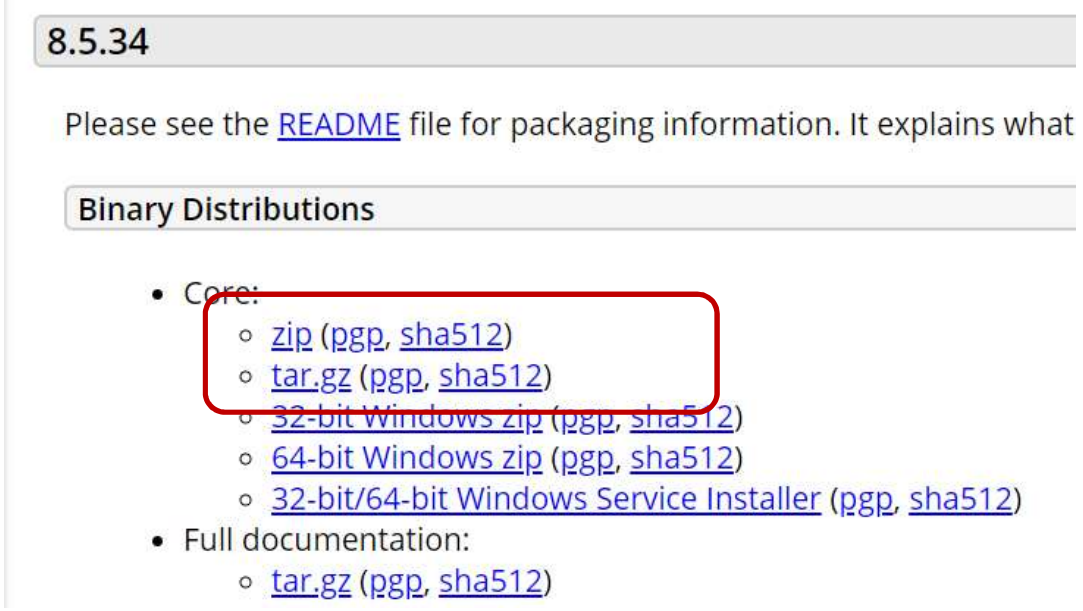
개발 환경 설정
: Apache Tomcat

- https://tomcat.apache.org/ 에서 적절한 버전 다운로드

- 적절한 위치에 압축 해제
  (개발 환경을 위해서는 인스톨러 버전보다 압축 버전을 선호)

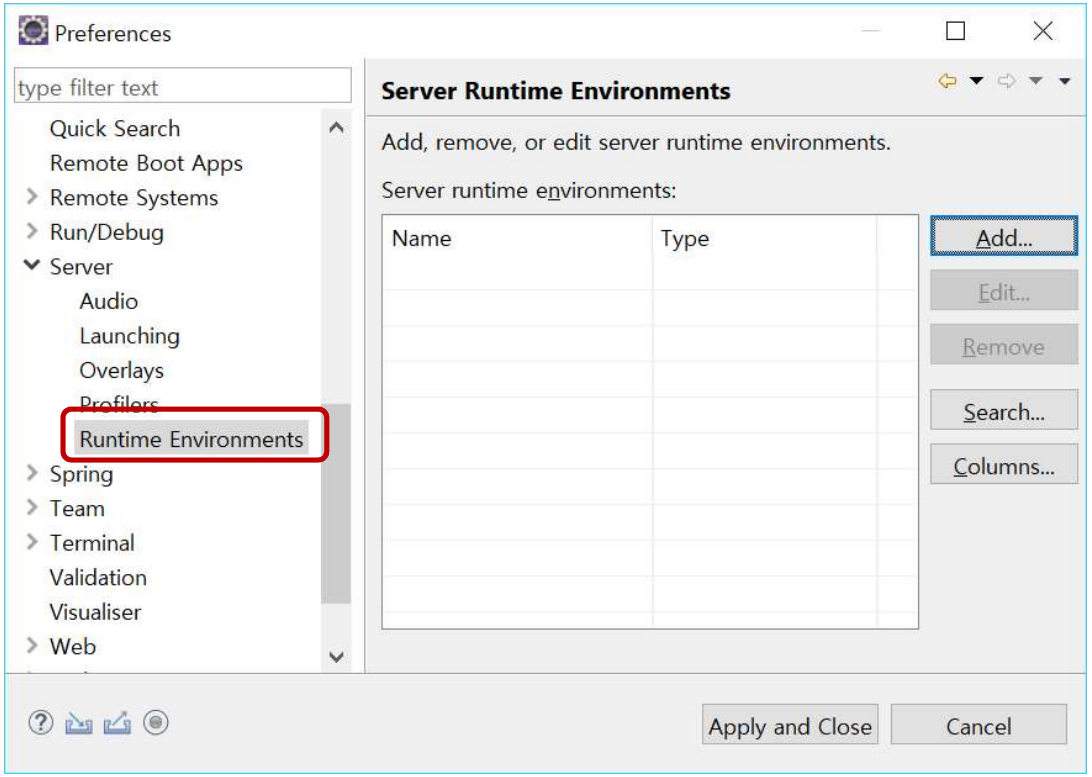
Apache Tomcat

- Apache 재단에서 관리되는 WAS
- 웹 서버 기능 포함



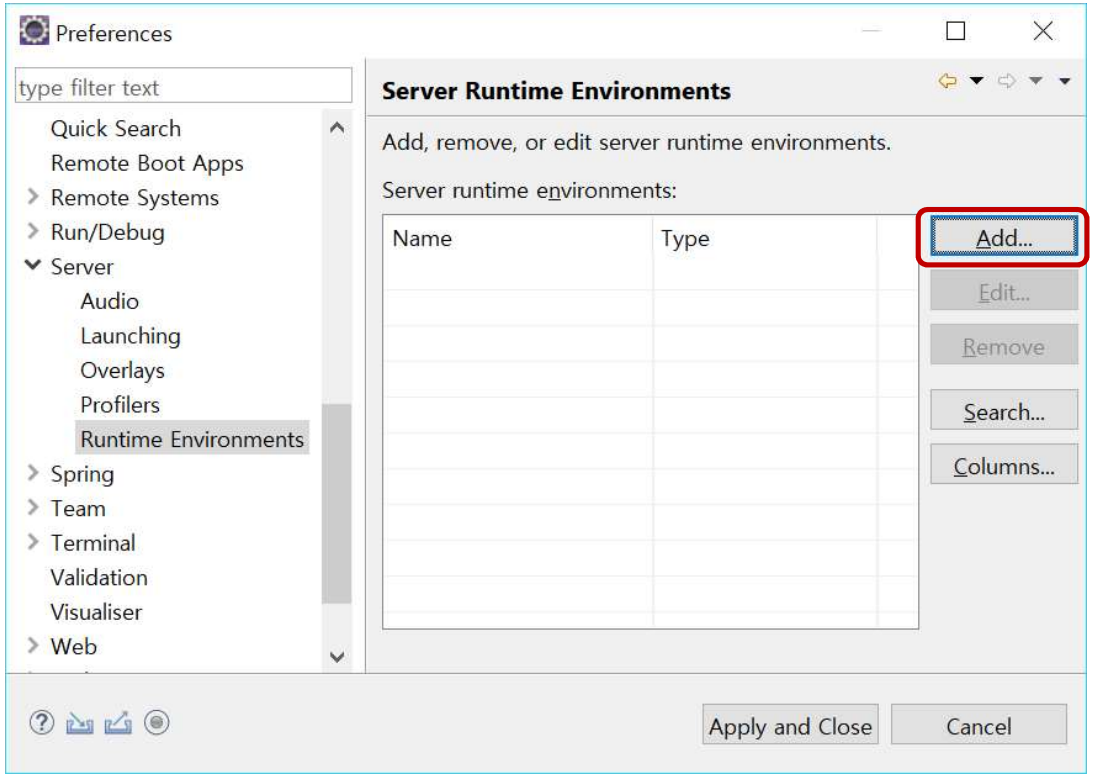
개발 환경 설정
: Apache Tomcat

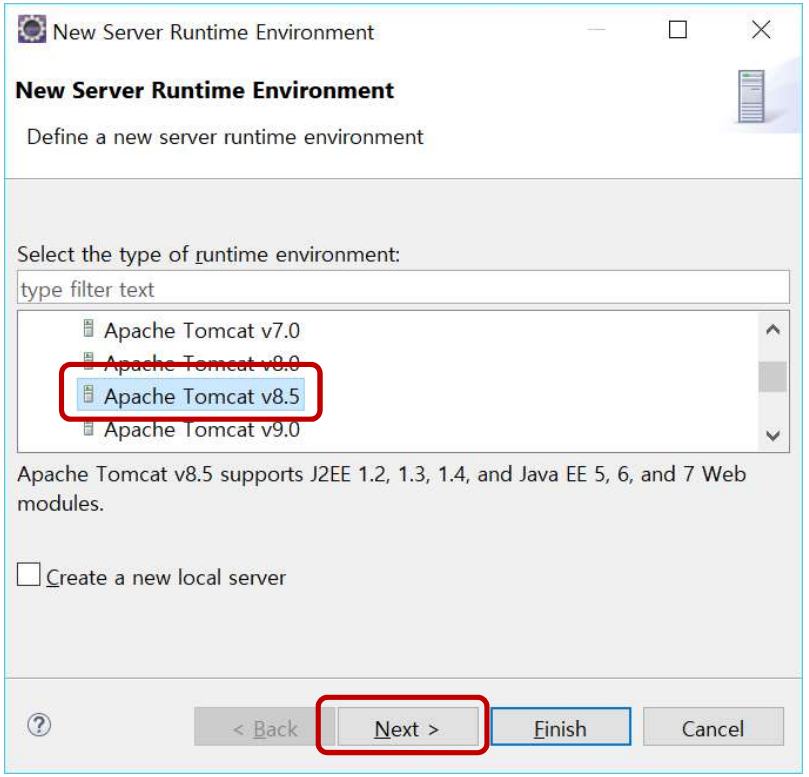
- Eclipse Workspace 내에 Server Runtime Environment 등록
  - Window > Preferences >
    Server > Runtime Environment



개발 환경 설정
: Apache Tomcat

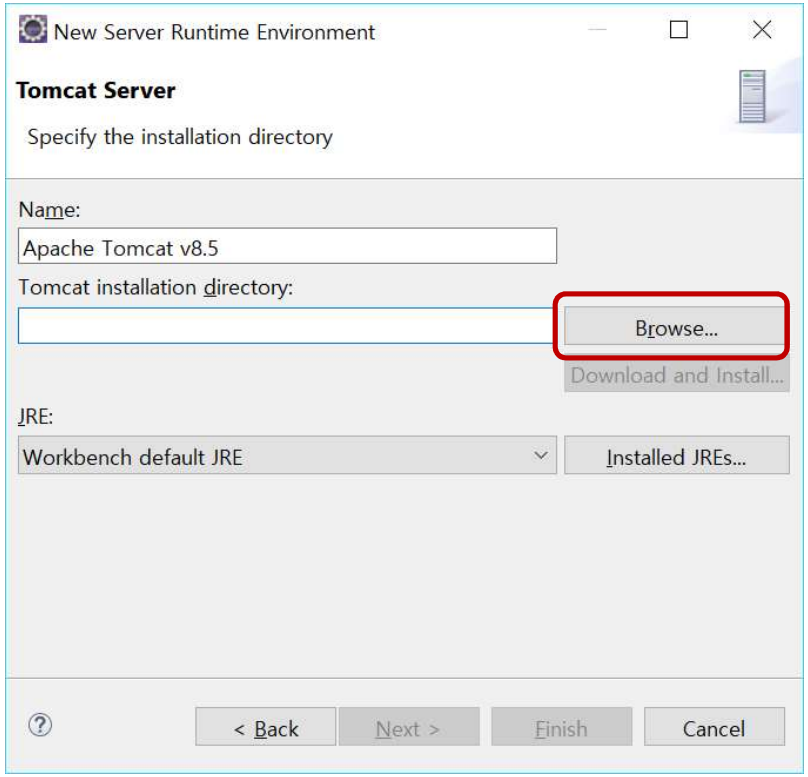
- Tomcat 서버 추가 및 삭제
  - 개발, 테스트 실행 환경에 새로운 톰캣 추가 및 삭제 가능

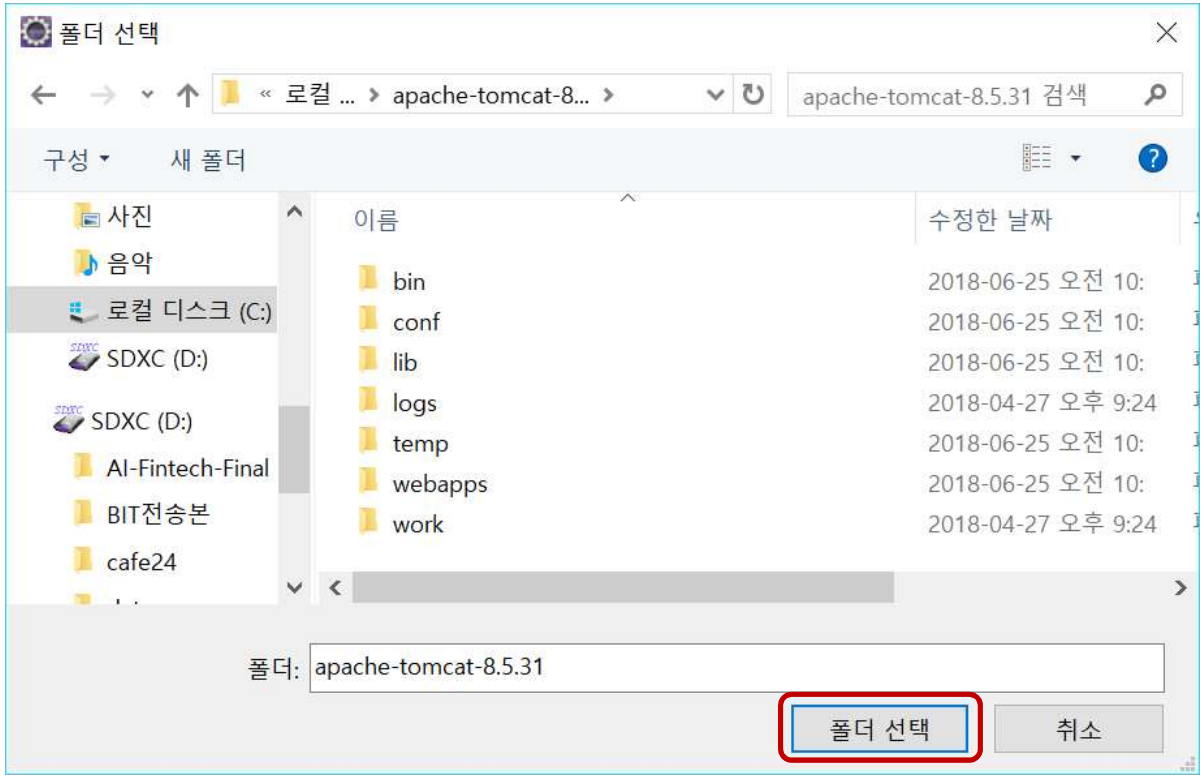




개발 환경 설정
: Apache Tomcat

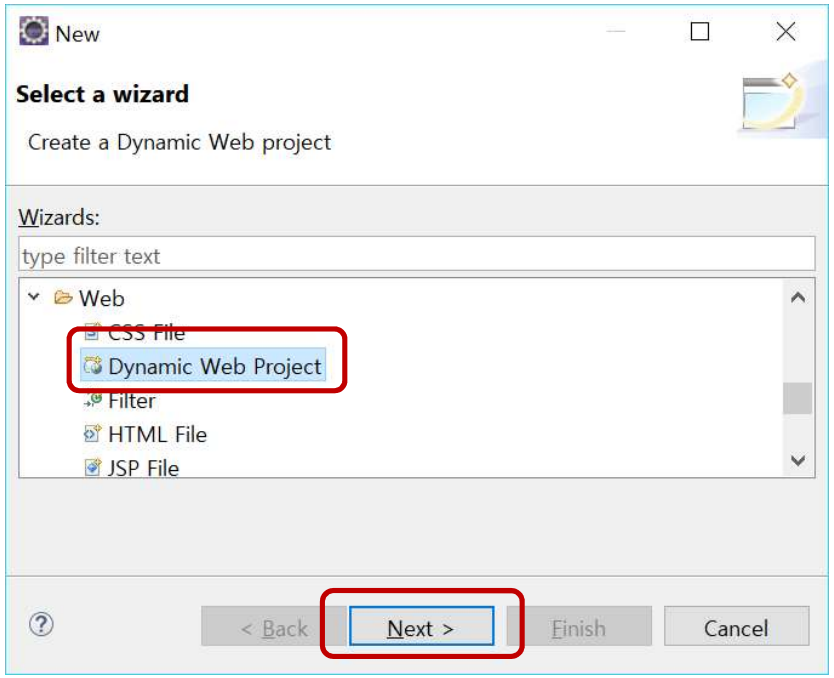
- Tomcat 서버 추가 및 삭제
  - Tomcat 실행 디렉터리 선택





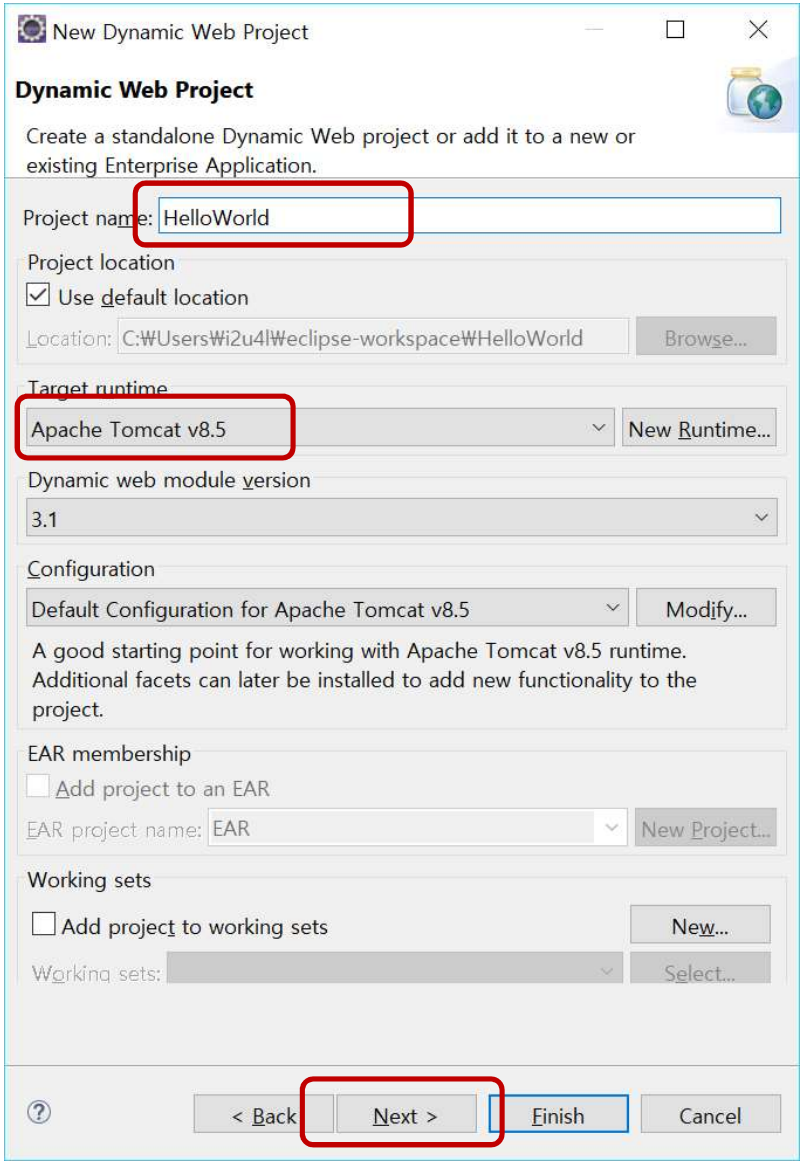
간단한 웹 프로젝트 구성
: HelloWorld

- Eclipse에서 웹 프로젝트를 만들기 위해서는
  File > New > Dynamic Web Project를 선택
  - 원하는 프로젝트 형식이 보이지 않을 때는
    File > New > Other... 에서 Web > Dynamic Web Project 선택



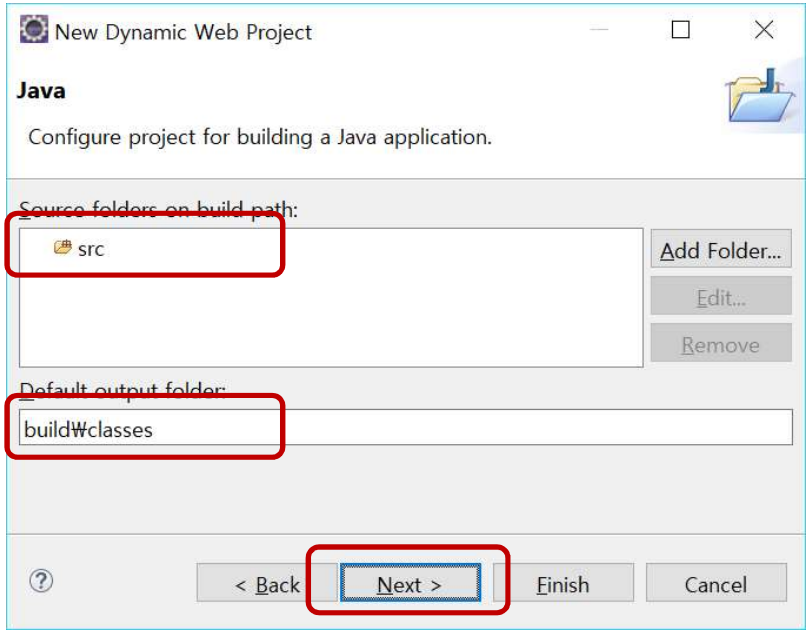
간단한 웹 프로젝트 구성
: HelloWorld

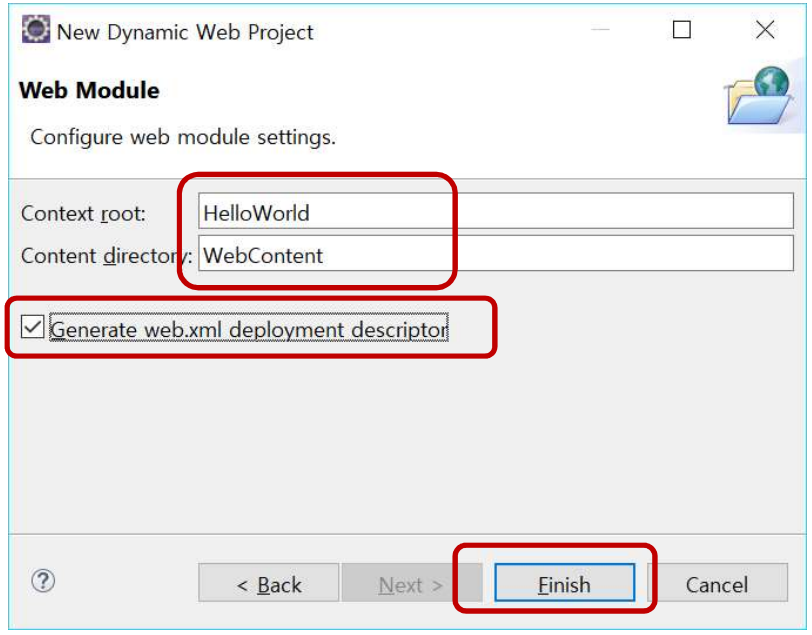
- 프로젝트 이름 및 Target Runtime 선택



간단한 웹 프로젝트 구성
: HelloWorld

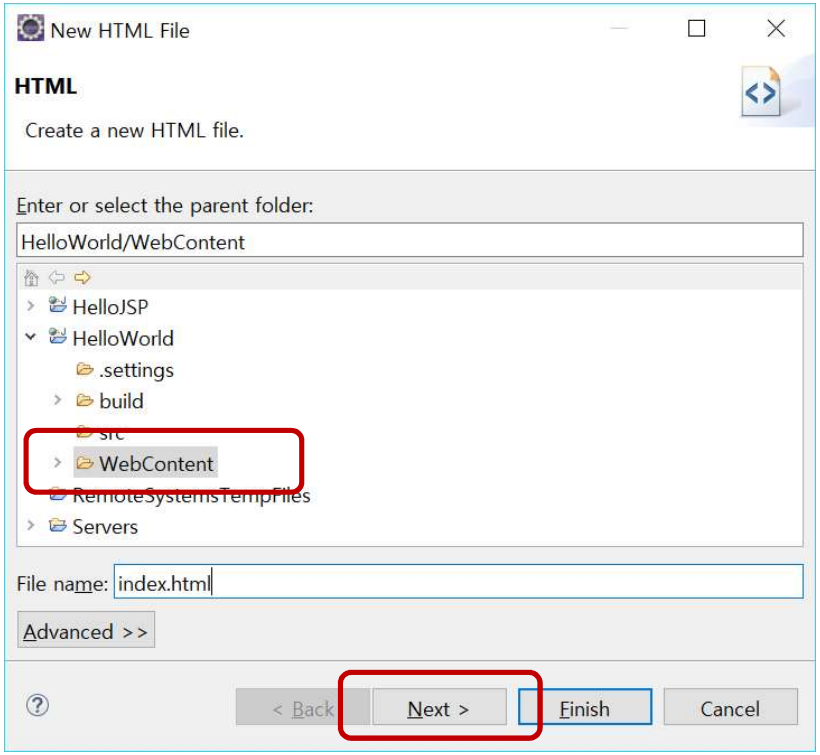
- Source 폴더 및 Output 폴더 위치 지정
- Context Root 및 web.xml 파일 생성

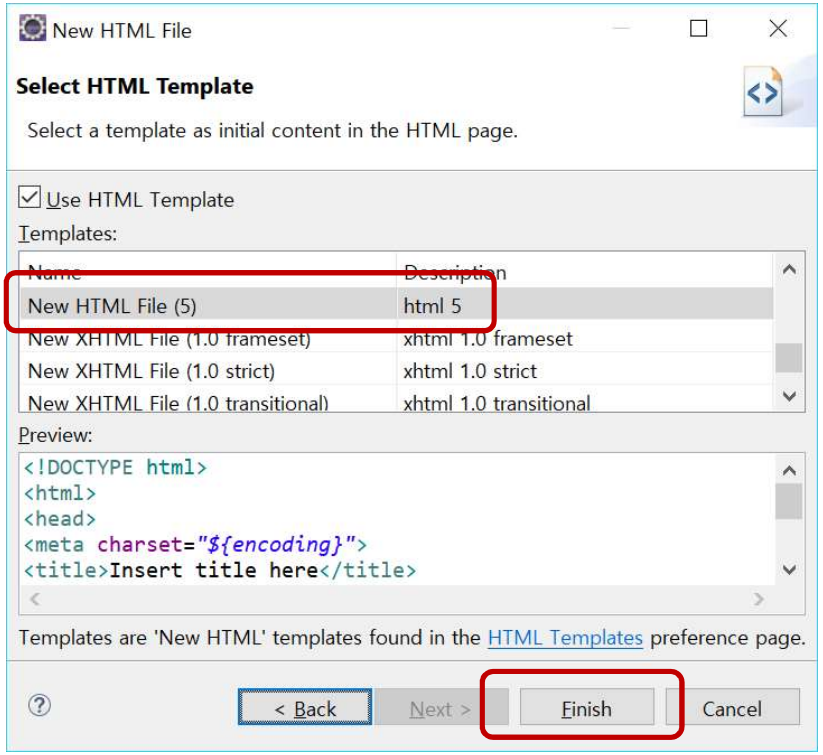




간단한 웹 프로젝트 구성
: HelloWorld

- 첫 번째 웹 페이지 만들기
  - 프로젝트 내 WebContent 디렉터리에서 우클릭 > New > HTML File
  - 파일명은 index.html(html 5 Template)로 설정

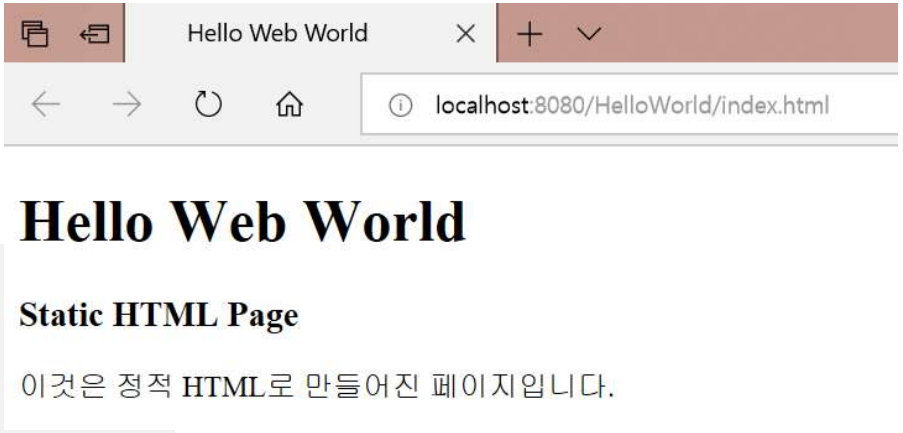




간단한 웹 프로젝트 구성
: HelloWorld

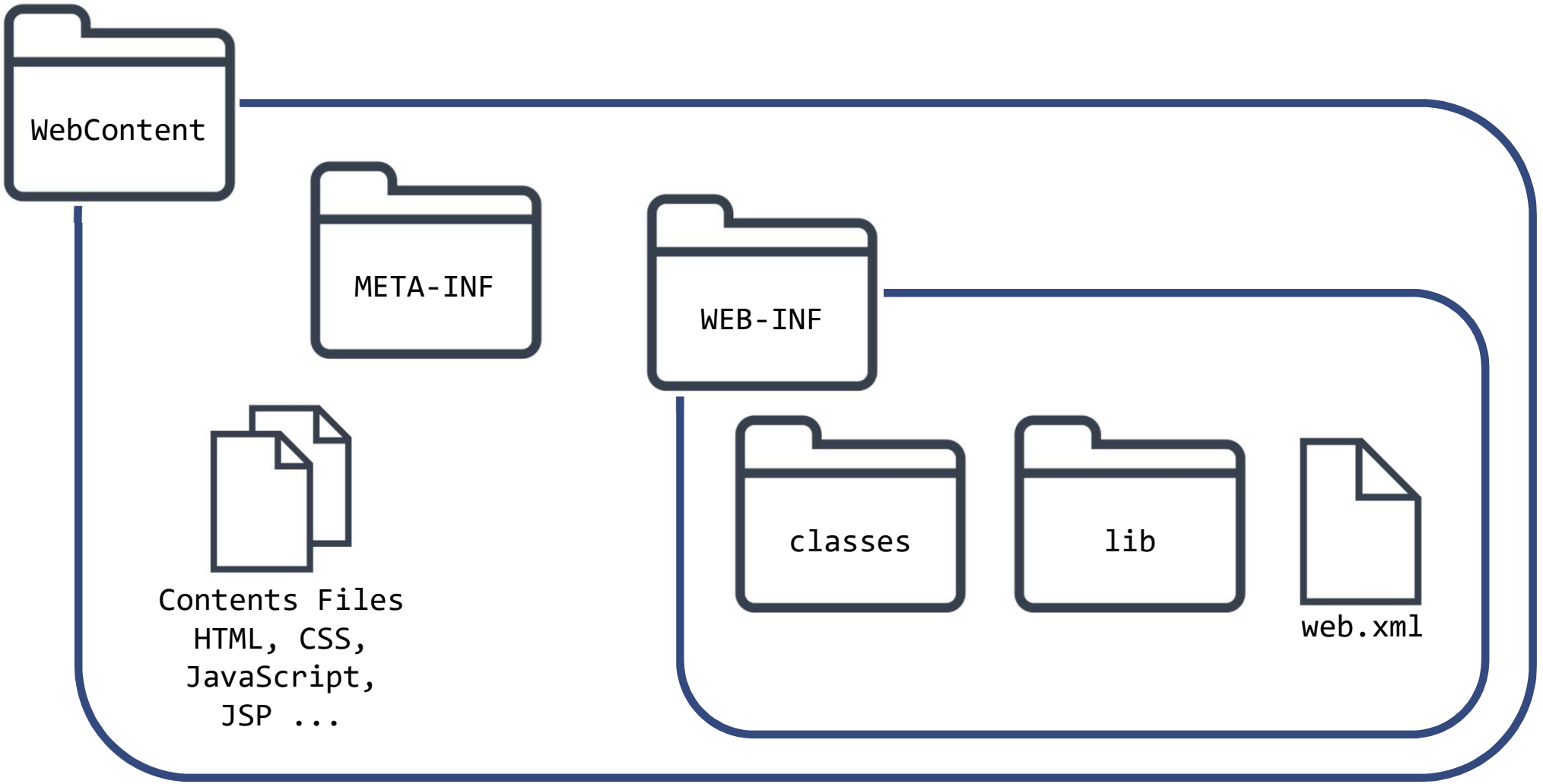
- 첫 번째 웹 페이지 만들기
  - 새로 만들어진 index.html 파일을 수정

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello Web World</title>
</head>
<body>
    <h1>Hello Web World</h1>
    <h3>Static HTML Page</h3>
    <p>이것은 정적 HTML로 만들어진 페이지입니다.</p>
</body>
</html>
```



  - Ctrl + F11로 Tomcat 구동
    - Window > Web Browser 메뉴에서 페이지를 표시할 브라우저 선택 가능

Java Web Project Structure

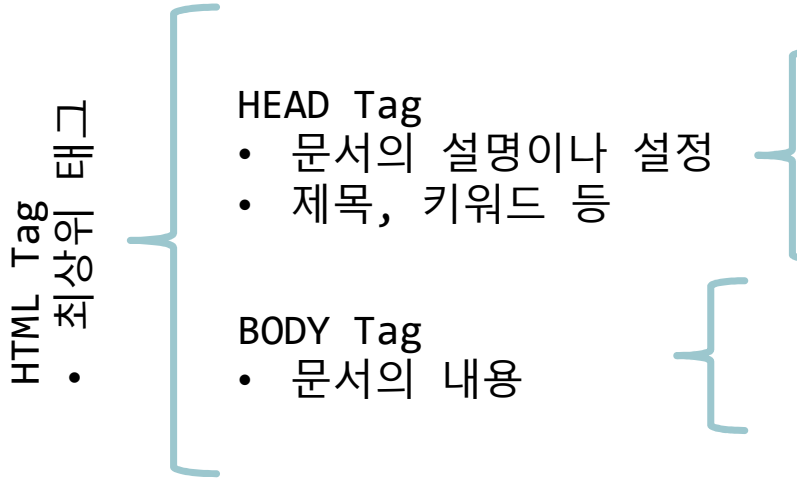


HTML 기초

HTML Basic

- Hyper Text Markup Language
  - 웹 문서의 구조와 내용을 담고 있는 구조화된 문서 파일(Text)

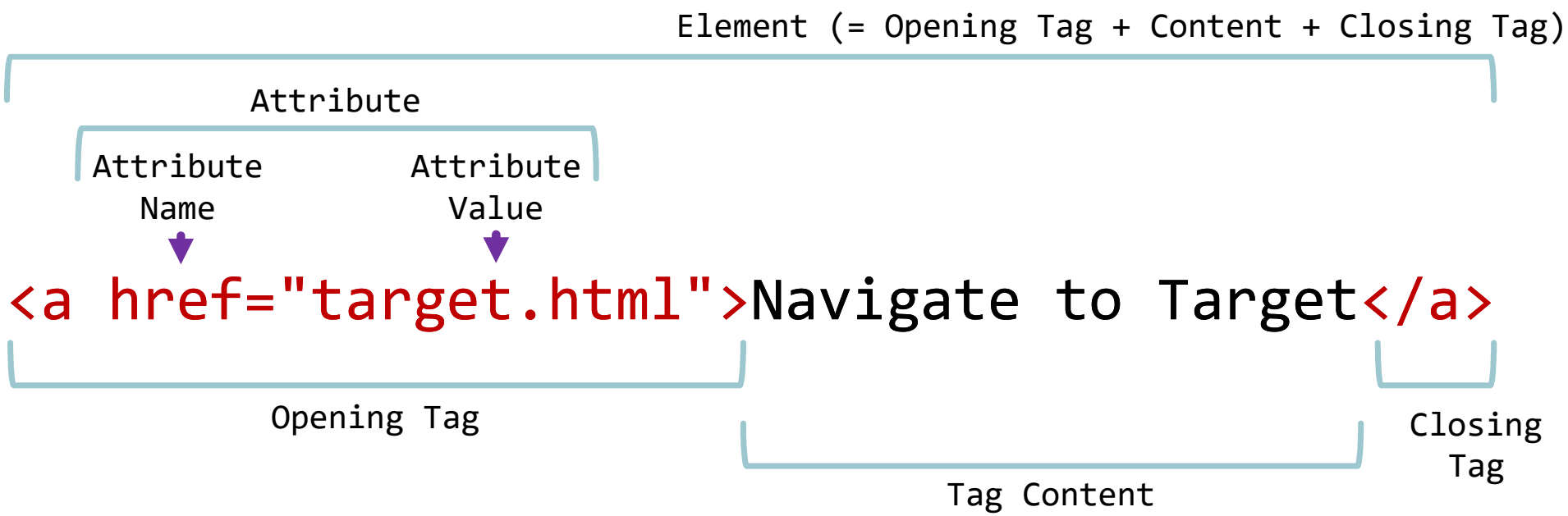
  - HTML의 기본 구조




```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Hello Web World</title>
    </head>
    <body>
        <h1>Hello Web World</h1>
    </body>
</html>
```

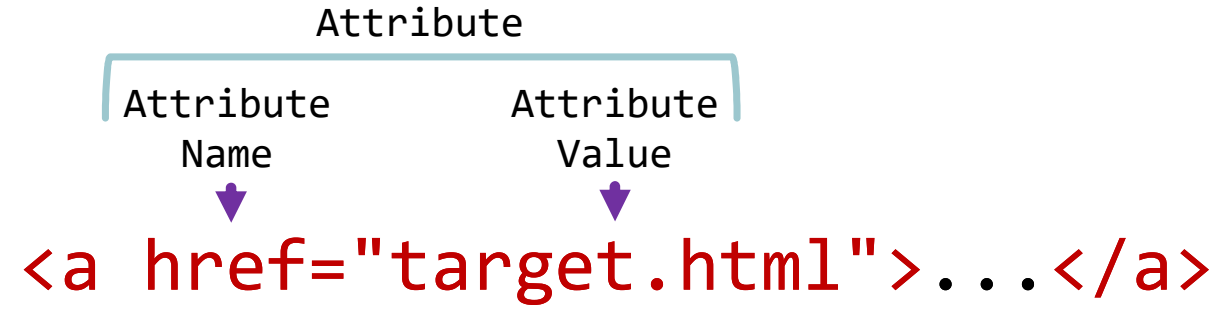
HTML Basic
: Tag

- 태그(Tag)
  - <> 사이에 오는 단어나 문자. ex) <head>, <p>, <h1>
  - 작성한 텍스트의 구조와 의미에 관해 브라우저에게 알려줌



HTML Basic
: Tag

- 속성(Attribute)
  - html 요소(Element)에 부가적인 정보를 전달하기 위해 제공하는 키와 값의 쌍
  - 속성은 요소를 구성하는 태그에 부여되며 요소의 특성과 역할에 따라 다양한 속성을 가질 수 있다



  - a 태그는 타 웹 문서로 연결되는 링크(link)를 정의하는 태그로 href 속성에 연결할 문서의 URL을 값으로 부여

HTML Basic
: Basic Tags

- 기본 마크업 태그 h1 ~ h6
  - 문서의 헤더 정보를 정의. 가장 중요한 헤더는 <h1>, 가장 낮은 중요도의 헤더 정보는 <h6>

- 기본 마크업 태그 p
  - 한 개의 문단을 의미

- 기본 마크업 태그 br
  - 줄 바꿈

- 기본 마크업 태그 hr
  - 주제를 분리, 가로 줄을 삽입

```
<h1>Main Title</h1>
<h2>Sub Title</h2>

<p>P 태그는 하나의 문단을 나타냅니다.
<br/>BR 태그는 줄 바꿈 태그입니다.</p>

<hr/>
```

- **br, hr 처럼 Content가 없는 태그는 Closing Tag 없이
, <hr/> 방식으로 표기**

HTML Basic
: Basic Tags

- 기본 마크업 태그 img
  - HTML 문서에 이미지를 삽입하고자 할 때 사용

속성명	설명
src	이미지의 경로 지정
alt	대체 텍스트
width	이미지의 너비
height	이미지의 높이

```
<img src="profile.png" alt="프로필 사진"
    width="320" height="240" />
```

- 기본 마크업 태그 a
  - 단일 페이지에서 벗어나 다른 페이지와 연결할 수 있게 해 주는 태그
  - 웹의 근간을 이루는 Hyper Text 연결을 위한 Anchor Tag
  - href 속성은 링크의 목적지를 명시

```
<a href="another-page.html">Hyper Text Link</a>
```

HTML Basic
: Table Tags

- table 태그
  - 표를 만들 때 사용하는 태그
  - 테이블 캡션 : caption
  - 테이블 헤더 : thead
  - 테이블 본문 : tbody
  - 테이블 푸터 : tfoot

성적표

이름	나이	점수
철수	23	80
영희	22	90
평균		85

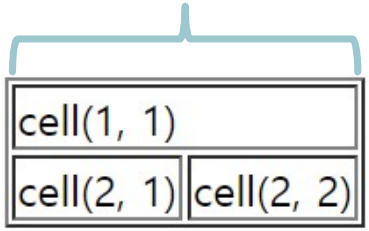
  - tr 태그 (Table Row)
    - 테이블의 한 열(row)을 표현
  - td 태그 (Table Data)
    - tr 태그 내에서 하나의 행(column)을 표현
  - th 태그 (Table Header)
    - td 대신 사용. 제목 셀로 사용

```
<table border="1">
    <caption>성적표</caption>
    <thead>                          ← thead
        <tr>
            <th>이름</th>
            <th>나이</th>
            <th>점수</th>
        </tr>
    </thead>
    <tbody>                          ← tbody
        <tr>
            <td>철수</td>
            <td>23</td>
            <td>80</td>
        </tr>
        ...
    </tbody>
    <tfoot>                          ← tfoot
        <tr>
            <td>평균</td>
            <td></td>
            <td>85</td>
        </tr>
    </tfoot>
</table>
```

HTML Basic
: Table Tags

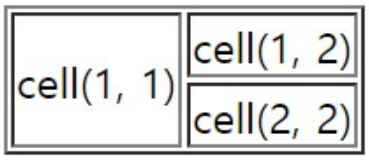
- 열(TR)과 행(TD)의 확장
  - rowspan : 열의 확장
  - colspan : 행의 확장

colspan="2"




```
<table border="1">
    <tr>
        <td colspan="2">cell(1, 1)</td>
    </tr>
    <tr>
        <td>cell(2, 1)</td>
        <td>cell(2, 2)</td>
    </tr>
</table>
```

rowspan="2"




```
<table border="1">
    <tr>
        <td rowspan="2">cell(1, 1)</td>
    <td>cell(1, 2)</td>
    </tr>
    <tr>
        <td>cell(2, 2)</td>
    </tr>
</table>
```

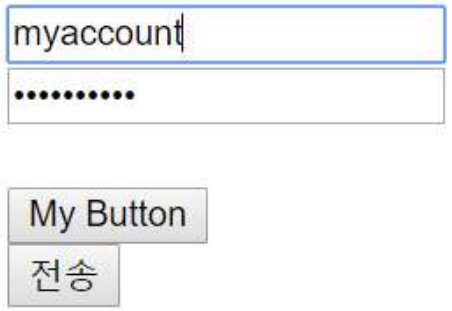
HTML Basic
: Form Tags

- input 태그
  - 전송할 데이터를 입력하기 위한 입력 영역을 만드는 태그. form 태그 안쪽에 기술됨
  - type 속성에 따라 다른 형태의 입력 영역이 표시

속성명	설명
type	입력 영역의 종류 (기본값: text)
name	전송할 데이터의 이름

type	설명
text	텍스트 입력 영역
password	암호 입력 영역(입력 값을 가림)
hidden	숨겨진 입력 영역
file	파일 정보 선택 영역
submit	전송 버튼
button	사용자 정의 버튼

```
<form method="POST" action="profile.jsp">
<input type="text" name="userid" /><br/>
<input type="password" name="password" /><br/>
<input type="hidden" name="secret_key" /><br/>
<input type="button" value="My Button" /><br/>
<input type="submit" value="전송" />
</form>
```



HTML Basic
: Form Tags

- input 태그 : checkbox와 radio

type	설명
checkbox	체크 박스 (중복 선택 가능)
radio	라디오 버튼 (단일 선택만 가능)

- checkbox (type="checkbox")

```
<input type="checkbox" name="pet" value="dog" checked>개
<input type="checkbox" name="pet" value="cat" checked>고양이
<input type="checkbox" name="pet" value="bird">새
```

☑ 개 ☑ 고양이 ☐ 새

- radio (type="radio")

```
<input type="checkbox" name="pet" value="dog" checked>개
<input type="checkbox" name="pet" value="cat" checked>고양이
<input type="checkbox" name="pet" value="bird">새
```

○ 사과 ○ 포토 ◉ 오렌지

HTML Basic
: Form Tags

- textarea 태그
  - 장문의 텍스트(여러 줄)를 전송하고자 할 때 사용

속성명	설명
rows	열 수
cols	행 수

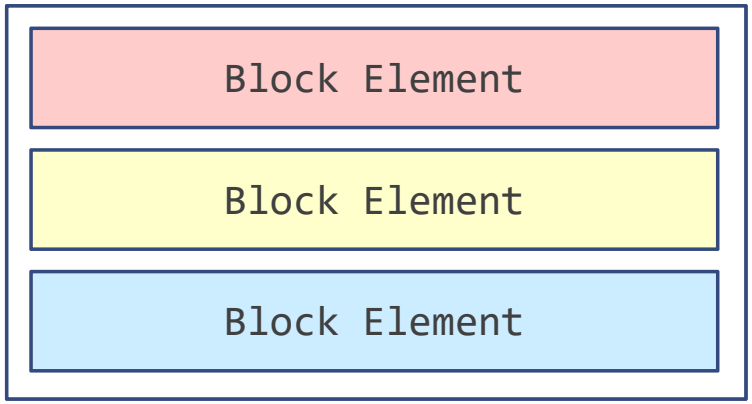
```
<textarea name="content" rows="10" cols="30"></textarea>
```

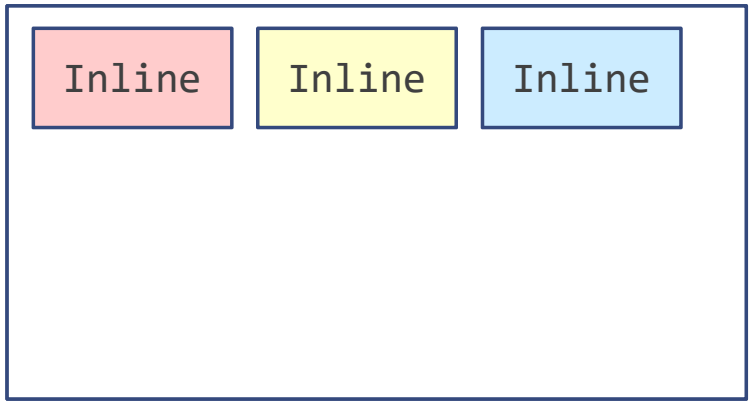
- select 태그
  - 여러 개의 선택 옵션(option) 가운데서 하나를 선택


```
<select name="telecom">
    <option value="S">SKT</option>
    <option value="K">KT</option>
    <option value="L">LGT</option>
    <option value="E">기타</option>
</select>
```

HTML Basic
: 공간 정의 태그

- HTML 요소가 공간을 점유하는 방식 : block vs inline





  - 모든 시각적인(공간을 점유하는) HTML 요소들은 block 혹은 inline 방식으로 영역을 차지한다.

- div와 span은 임의의 작은 구역을 설정할 때 사용하는 태그
  - div : Block 요소로 줄 바꿈이 일어난다.
  - span : inline 요소로 줄 바꿈이 일어나지 않는다. 문맥의 흐름이 유지된다.

Servlet & JSP 기초

첫 번째 JSP 구현

- HelloWorld 프로젝트 WebContent 폴더 아래 다음과 같이 hello.jsp 파일을 작성

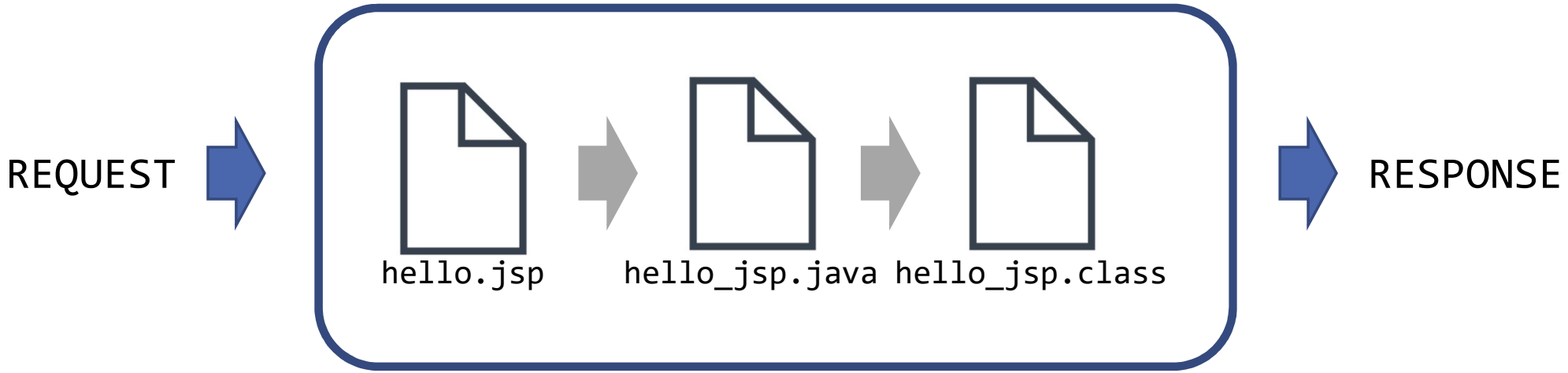
```jsp
<%@ page language="java"
    contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%
    String name = request.getParameter("name");
    if (name == null) {
        name = "Anonymous";
    }
    %>
    <h1>Hello, JSP</h1>
    <p>Welcome <%= name %></p>
</body>
</html>
```

첫 번째 JSP 구현

- 프로젝트를 실행하고 다음의 결과를 확인
  - http://localhost:8080/HelloWorld/hello.jsp
  - http://localhost:8080/HelloWorld/hello.jsp?name=World
- 브라우저의 소스 보기 기능을 이용하여 원본 소스 파일과 브라우저에서 출력하는 소스를 비교
  - JSP 소스에 <% ~ %>로 작성했던 코드는 실제 브라우저에서는 어떻게 표시되고 있는가?

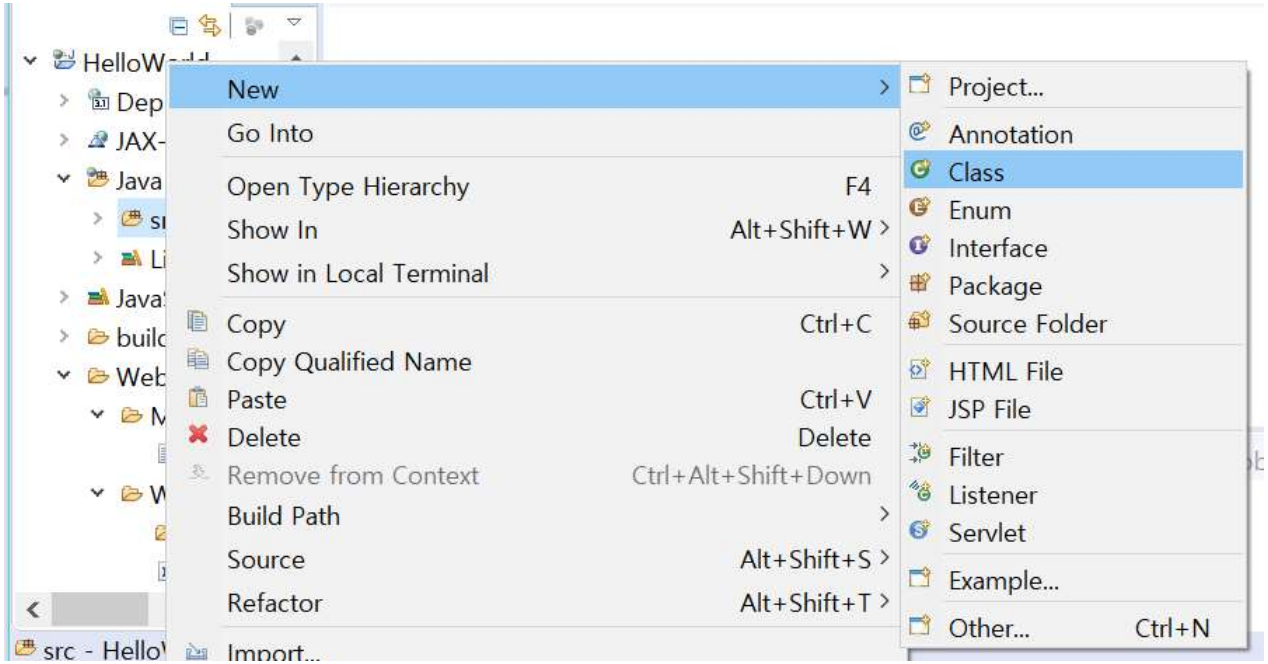
JSP는 어떻게 작동하는가?

- JSP (Java Server Pages)
  - JSP는 HTML 코드 내에 Java 코드를 포함하는 형태로 구성
  - JSP가 처음 호출되었을 때, JSP 엔진은 JSP 코드를 서블릿으로 변환하고 컴파일
  - 서블릿 엔진이 컴파일된 서블릿 클래스를 호출하여 요청을 처리

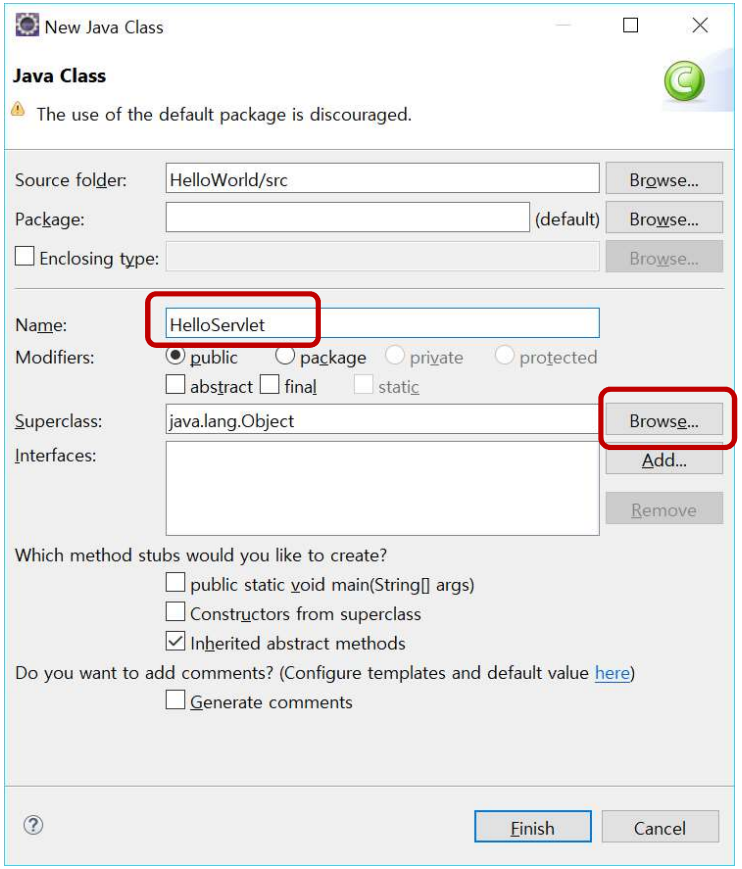


첫 번째 Servlet 구현
: HelloServlet

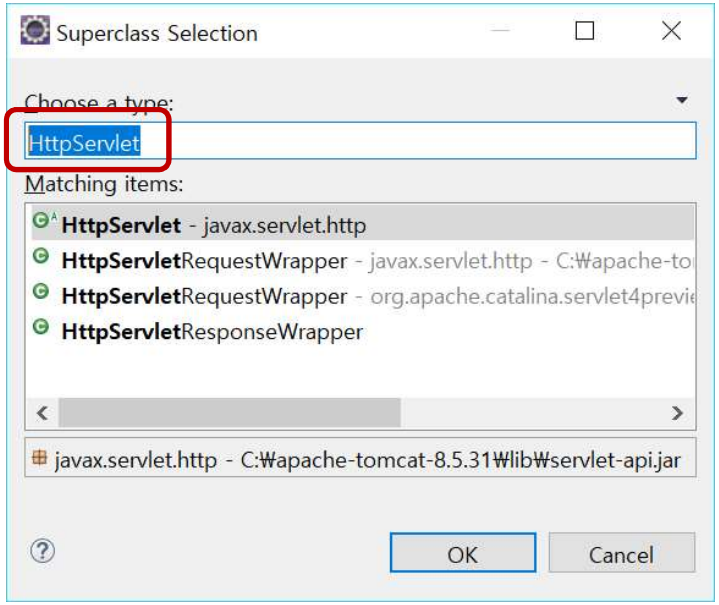
- URL /hs 요청에 응답하는 서블릿을 구현해 봅시다
- HelloServlet 구현
  - Java Resources > src 에서 New > Class 선택



첫 번째 Servlet 구현
: HelloServlet



- HelloServlet 구현
  - HttpServlet을 Superclass로 하는 HelloServlet 클래스 생성



첫 번째 Servlet 구현
: HelloServlet

- doGet 메서드를 Override 하고 다음과 같이 코드를 구성

```
/* import 문 생략 */

public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        resp.setContentType("text/html; charset=UTF-8");
        String name = req.getParameter("name");
        if (name == null) {
            name = "Anonymous";
        }
        PrintWriter out = resp.getWriter();
        out.println("<h1>Hello Servlet</h1>");
        out.println("<p>Welcome, " + name + "</p>");
    }
}
```

첫 번째 Servlet 구현
: HelloServlet

- Servlet 매핑
  - WEB-INF > web.xml에 요청 URL과 Servlet 클래스를 매핑

```
<servlet>
  <servlet-name>MyFirstServlet</servlet-name>
  <servlet-class>HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>MyFirstServlet</servlet-name>
  <url-pattern>/hs</url-pattern>
</servlet-mapping>
```

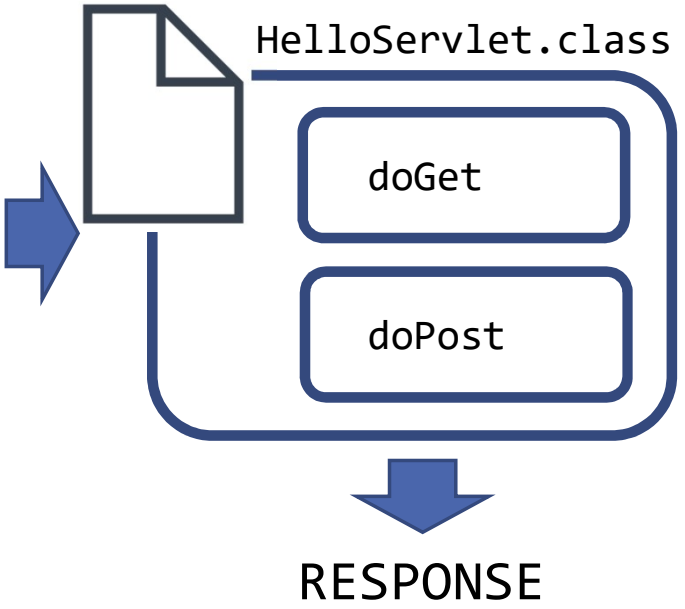
- 프로젝트를 실행하고 다음 URL을 확인
  - http://localhost:8080/HelloWorld/hs
  - http://localhost:8080/HelloWorld/hs?name=World

Servlet은 어떻게 작동하는가?

- URL 패턴에 적합한 Servlet Name 확인
- 해당 Servlet Name의 class 확인
- Container는 해당 클래스로 HttpServletRequest와 HttpServletResponse 객체를 전달
- Servlet 클래스는 HTTP 요청에 적합한 메서드(doGet, doPost, service) 내에서 전달 받은 HttpServletRequest와 HttpServletResponse를 이용, 요청을 처리

REQUEST

```
<servlet>
  <servlet-name>MyFirstServlet</servlet-name>
  <servlet-class>HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MyFirstServlet</servlet-name>
  <url-pattern>/hs</url-pattern>
</servlet-mapping>
```



Request와 Response 확인

- 브라우저의 개발자 도구를 이용, Request와 Response 확인
- HTTP 상태 코드(status)

코드	요약	상세 설명
2xx	성공	요청을 성공적으로 처리함
3xx	리다이렉션	요청을 마치기 위해 추가 동작이 필요함
4xx	요청 오류	클라이언트 요청에 오류가 있음
5xx	서버 오류	서버가 요청을 정상적으로 처리하지 못함

- 주요 상태 코드
  - 200 : 성공
  - 404 : Not Found. 요청한 리소스를 찾을 수 없음
  - 500 : Internal Server Error. 서버 오류로 요청을 처리할 수 없음

HTTP Request Method와 Servlet Method

- HTTP 요청은 Get, Post, Put, Delete 등 다양하지만,
  주로 Get 메서드와 Post 메서드를 이용하여 서버로 요청을 전송
- GET vs POST

	GET	POST
요청 방법	주로 링크(link)를 이용	폼(form)을 이용
데이터 전송	URL 파라미터 이용	form tag 이용
데이터 제한	4KB 이하 / 단순 텍스트	제한 없음
중복 요청	허용	허용하지 않음
Cache/북마크	허용	허용하지 않음
Servlet Method	doGet	doPost
	service	

- service 메서드는 Get, Post를 이용한 모든 요청을 처리할 수 있으나
  내부적으로 HttpServletRequest의 getMethod 메서드를 이용하여 요청별 분기를 직접
  구현해 주어야 함

Servlet 호출

- Servlet 호출을 위한 URL 형식

```
http://{host}:{port}/{ContextPath}/{ServletName}
```

- Servlet 호출 URL에 데이터 전송을 위한 파라미터 추가(GET 방식)

```
{ServletName}?{key1}={val1}&{key2}={val2}
```

  - Parameter 전송을 위해서는 ? 기호를 시작으로, key = value 형식의 파라미터 쌍을 & 기호로 연결하여 URL 뒤에 붙여 보낸다

- Form을 이용한 서블릿 호출

```
<form action="{ServletName}" method="GET">
```

```
<form action="{ServletName}" method="POST">
```

  - method를 생략하면 GET 방식
  - Form을 GET 방식으로 전송하면 파라미터 값들이 URL과 연결되어 전송되고, 브라우저 주소 창에 노출

Servlet 호출

- [실습 프로젝트]
  - HelloWorld 웹 응용프로그램의 index.html 페이지 하단에 다음의 작업을 진행, HelloServlet을 호출해 봅니다

  - 작업 1:
    - GET 호출
    - a 태그를 이용, name 파라미터에 Servlet이라는 값을 담아 전송

  - 작업 2:
    - POST 호출
    - form을 작성하고 아래와 같은 텍스트 입력 태그를 작성
      - last_name : 성(Family Name)
      - first_name : 이름
    - 전송 버튼을 누르면 폼을 이용하여 전송
    - HelloServlet에 doPost 메서드를 오버라이드하여 폼으로부터 전송된 파라미터를 출력

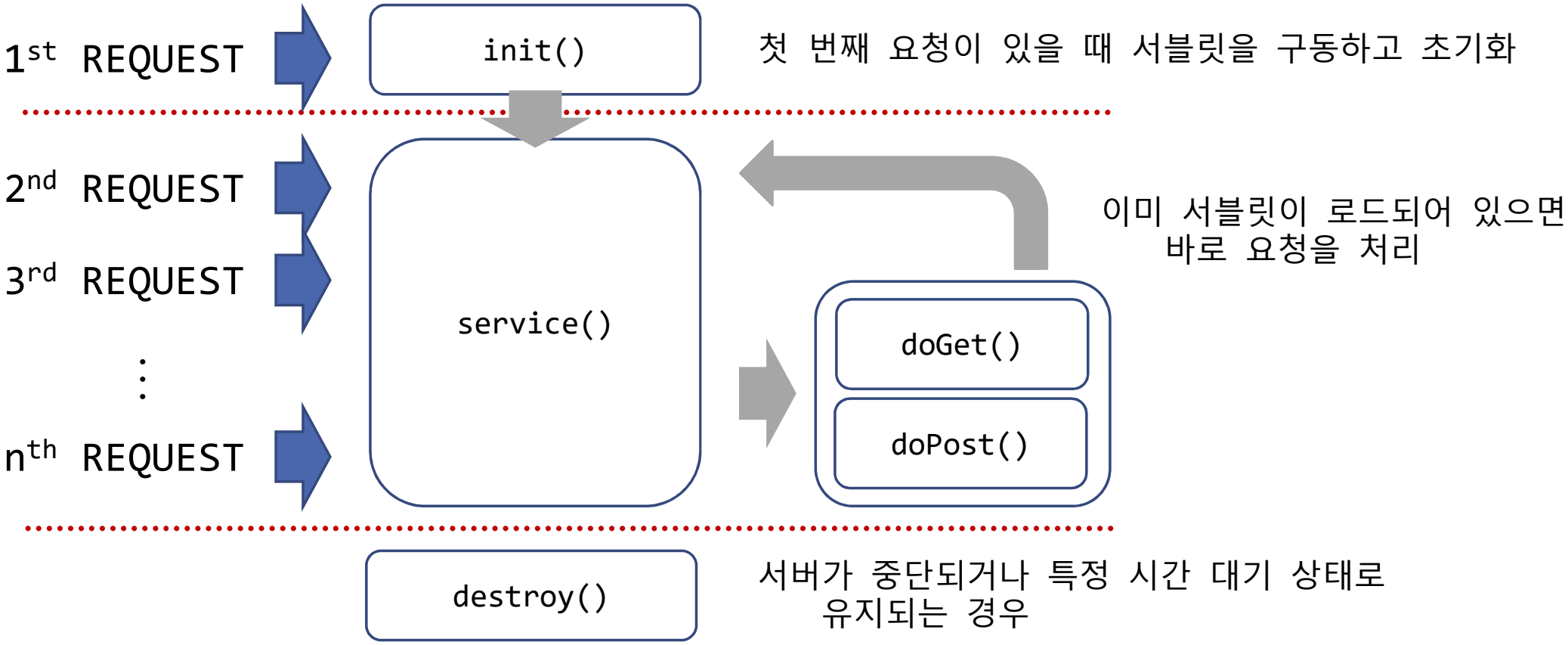
Servlet 생명 주기(LifeCycle)

- Servlet의 주요 기본 메서드

```
public class LifeCycleServlet extends HttpServlet {
    @Override
    public void init() throws ServletException { }
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException { }
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException { }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException { }
    @Override
    public void destroy() { }
}
```

Servlet 생명 주기(LifeCycle)

- Servlet의 생명 주기와 요청 처리

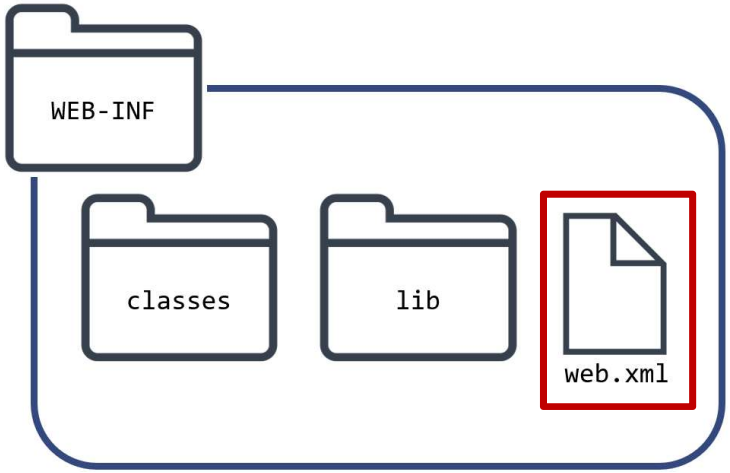


Servlet 생명 주기(LifeCycle)

- [실습 프로젝트]
  - LifeCycleServlet을 작성하여 Servlet 생명 주기를 확인해 봅니다.
  - 주요 메서드를 Override 하고 콘솔에 메시지를 출력하여 어떤 순서로 메서드가 실행되는지 확인합니다.
  - 처음 호출했을 때와 여러 번 호출했을 때, 실행되는 메서드와 그렇지 않은 메서드를 확인해 봅니다.

web.xml의 이해

- web.xml
  - WEB-INF 디렉터리 아래에 위치
  - 웹 응용프로그램의 Deployment Descriptor(환경파일: 배포 서술자)
  - XML 형식이므로 대소문자를 확실히 구분해야 함
  - 요소들의 순서가 틀리면 web.xml을 읽어 들일 때 서블릿 컨테이너가 오류를 표시
  - web.xml 파일의 설정들은 웹 응용프로그램이 시작될 때 메모리에 로딩됨
  - web.xml의 내용을 수정하면 서블릿 컨테이너를 재구동 해야 변경 내용이 반영
- 주요 작성 내용
  - 초기화 파라미터 (컨텍스트, 서블릿)
  - 서블릿/JSP에 대한 정의와 매핑
  - Welcome 파일 목록과 Error Page 처리
  - 세션, 리스너, 필터 설정
  - 보안



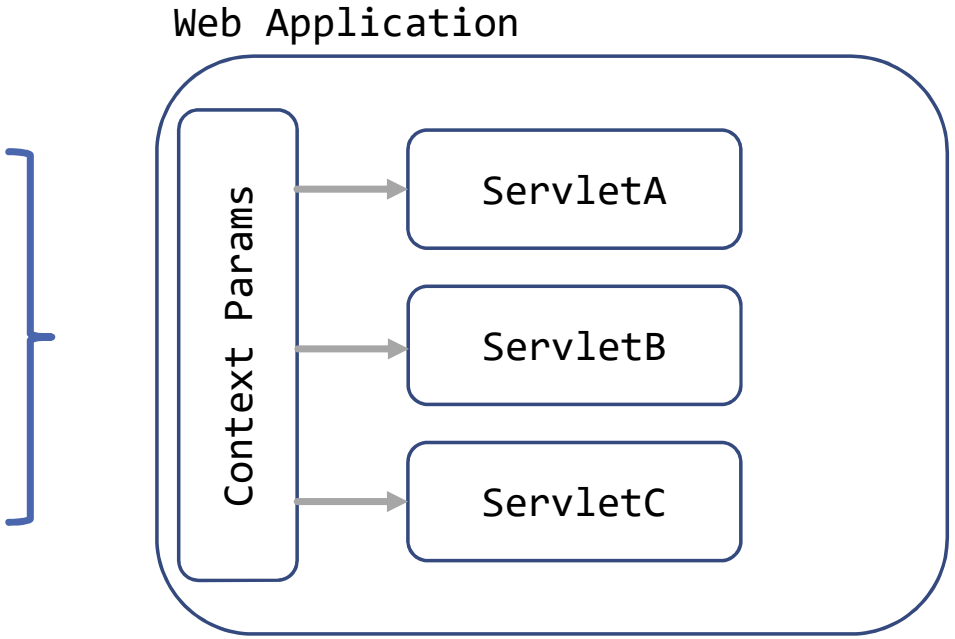
web.xml의 이해
: 초기화 파라미터

- 초기화 파라미터 : 서블릿 컨테이너가 구동될 때 읽어 들이는 파라미터
- 종류
  - 서블릿 초기화 파라미터 (Servlet Initialize Parameters)
    - 개별 Servlet 내에서만 읽고 활용할 수 있음
    - getServletConfig() 메서드를 이용, ServletConfig를 반환 받아 사용
  - 컨텍스트 초기화 파라미터 (Context Initialize Parameters)
    - 웹 응용프로그램 내에 존재하는 모든 서블릿과 JSP에서 읽고 활용할 수 있음
    - getServletContext() 메서드를 이용, ServletContext를 반환 받아 사용

web.xml의 이해
: 컨텍스트 초기화 파라미터

▶ 정의 (in web.xml)

```
<web-app>
  ...
  <context-param>
    <param-name>dbName</param-name>
    <param-value>myportal</param-value>
  </context-param>
  <context-param>
    <param-name>dbUser</param-name>
    <param-value>webdb</param-value>
  </context-param>
  ...
</web-app>
```



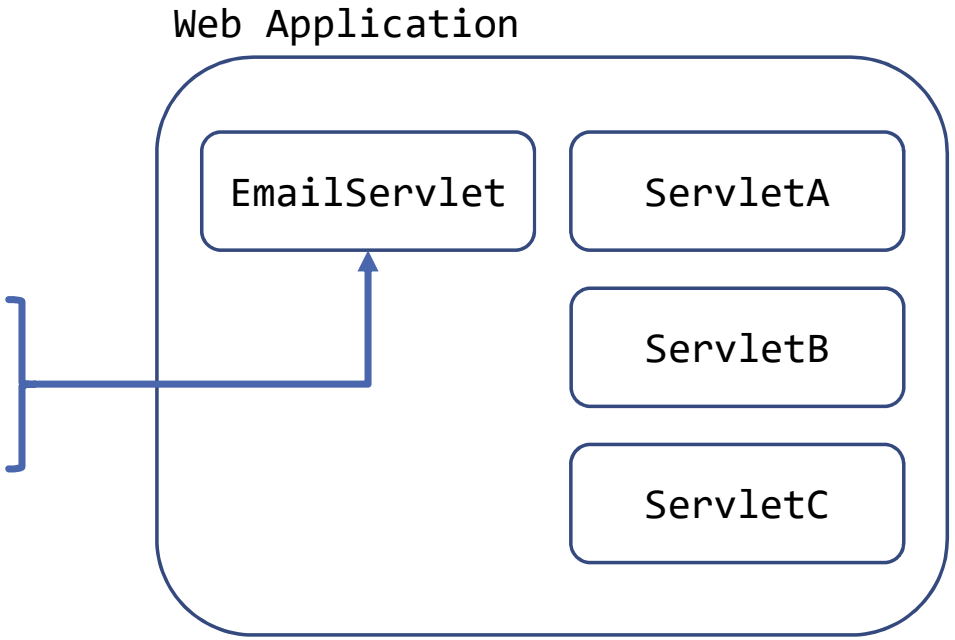
▶ 사용 (in Servlet Class)

```
ServletContext servletContext = getServletContext();
String dbName = servletContext.getInitParameter("dbName");
String dbUser = servletContext.getInitParameter("dbUser");
```

web.xml의 이해
: 서블릿 초기화 파라미터

▶ 정의 (in web.xml)

```
<servlet>
  <servlet-name>EmailList</servlet-name>
  <servlet-class>
    com.example.EmailListServlet
  </servlet-class>
  <init-param>
    <param-name>filename</param-name>
    <param-value>email_list.txt</param-value>
  </init-param>
</servlet>
```



▶ 사용 (in Servlet Class)

```
ServletConfig config = getServletConfig();
String filename = config.getInitParameter("filename");
```

web.xml의 이해
: 오류 페이지

- <error-page> 요소를 이용하면 다음과 같은 상황에서 특정 페이지가 출력되도록 제어할 수 있음
  - 에러 (Uncaught Exceptions) : <exception-type>으로 정의
  - 특정 HTTP 상태 코드가 발생했을 때 : <error-code>로 정의
- <error-page> 요소는 <servlet>과 <servlet-mapping> 요소 다음에 위치해야 함
- 오류 페이지의 위치는 <location> 요소를 이용하여 지정한다

web.xml의 이해
: 오류 페이지

- 특정 Exception 타입에 대한 오류 페이지 처리

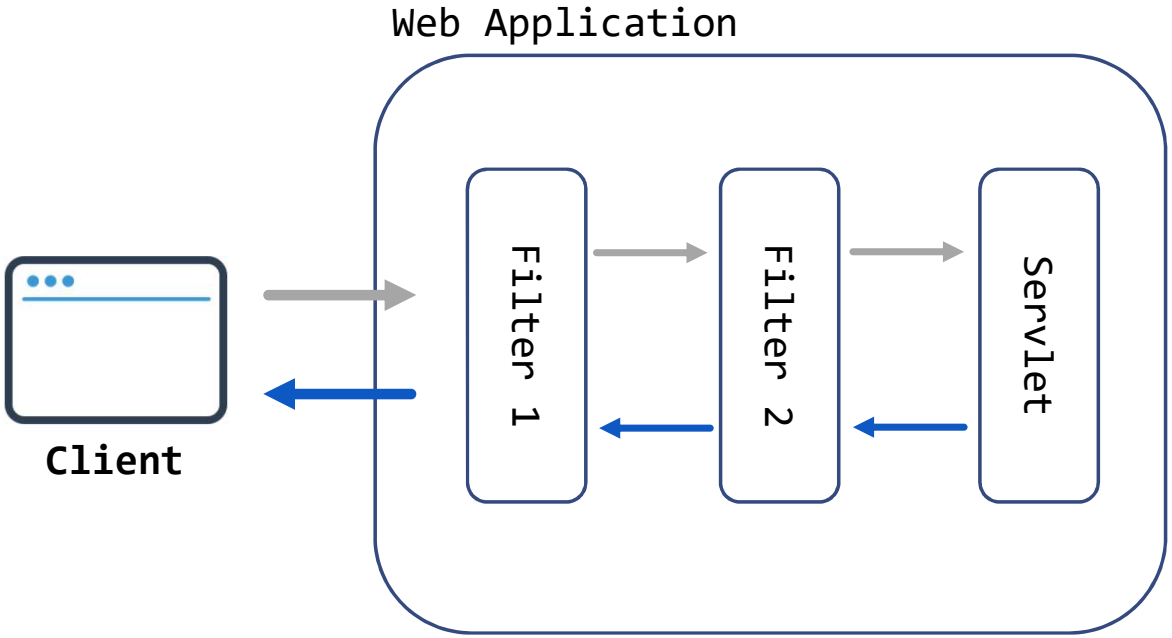
```
<error-page>
  <exception-type>java.lang.Throwable</exception-type>
  <location>/view/error/error.htm</location>
</error-page>
```

- 특정 HTTP 코드에 대한 오류 페이지 처리

```
<error-page>
  <error-code>404</error-code>
  <location>/view/error/error.404.jsp</location>
</error-page>
```

Filter

- 클라이언트가 서블릿에 요청하거나 응답할 때, 미리 요청이나 응답에 관련된 여러 작업을 수행하는 기능
- 여러 서블릿에서 반복적으로 처리해야 하는 작업들을 필터에서 처리
  - 예) 유니코드 인코딩



Filter
: 필터의 생성

- 필터는 javax.servlet 패키지의 Filter 인터페이스를 구현하여 만듦
- 필터 역시 생명 주기를 가지고 있다
  - init : 필터 생성시 컨테이너에 의해 호출되어 초기화 작업을 수행
  - doFilter : 요청/응답시 컨테이너에 의해 호출되어 기능을 수행
  - destroy : 필터 소멸시 컨테이너에 의해 호출되어 종료 작업을 수행

```
public class EncodingFilter implements Filter {
    @Override
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
        throws IOException, ServletException {
        req.setCharacterEncoding("UTF-8");
        //...
    }
}
```

Filter
: 필터의 설정

- 필터를 사용하기 위해서는 WEB-INF > web.xml에 설정
- 필터는 특정 URL Pattern에 대응하는 서블릿 요청/응답 처리의 앞뒤에서 추가된 기능을 수행하므로 설정 위치는 servlet 설정 이전에 마무리 되어야 한다.

- url-pattern에 /* 를 설정하면 동일 컨텍스트 내 모든 서블릿 URL 요청에 같은 필터를 적용할 수 있다

```
<!-- 필터 설정은 서블릿 세팅 이전에 -->
<filter>
  <filter-name>EncodingFilter</filter-name>
  <filter-class>com.bit.filter.EncodingFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>EncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Filter
: 필터의 설정

- [필터 적용 실습]
  - 현재 작성된 HelloWorld 응용프로그램의 doGet, doPost에서 Encoding 관련 설정 기능을 별도의 Filter 클래스로 작성합니다. 필터 클래스의 이름은 EncodingFilter로 합니다.
  - 작성된 필터를 모든 서블릿 요청에 대응할 수 있도록 web.xml에 설정해 봅니다.

JSP 프로그래밍 연습

JSP와 Web Application Model 1

JSP 태그
: <% ~ %>

- JSP 코드는 HTML 페이지 내에 부분적으로 Java 코드를 포함한다
- 하나 이상의 Java 문장을 포함하는 스크립틀릿(Scriptlet)을 작성하기 위해 <% 와 %> 태그를 사용한다
- 문자열로 변환되는 표현식(Expression)을 출력하기 위해 <%= 과 %> 태그를 사용한다
- JSP 페이지 내에서는 암묵적 요청 객체(implicit request object)를 사용할 수 있고, 해당 요청 객체는 request라 명명된다.
- JSP 페이지로 전달된 request 객체의 getParameter 메서드를 이용, 전달되는 파라미터의 값을 얻을 수 있다

JSP 태그

- JSP 태그 종류와 용도

태그	명칭	용도
<% %>	JSP 스크립틀릿	Java 구문을 JSP 페이지에서 사용
<%= %>	JSP 표현식	Java 표현식을 문자열로 출력
<%@ %>	JSP 지시자	JSP 페이지 전체에 적용되는 조건을 설정
<%-- --%>	JSP 주석	JSP 페이지에 주석을 삽입
<%! %>	JSP 선언문	인스턴스 변수 및 메서드를 선언

- JSP 지시자를 이용한 Java 클래스 import

```
<%@ page import="com.example.emaillist.vo.EmailVo" %>
<%@ page import="com.example.emaillist.dao.EmaillistDao" %>
<%@ page import="java.util.List" %>
```

JSP 태그

- 스크립틀릿(Scriptlet)과 표현식(Expression)
  - 스크립틀릿 문법
    : Java 문법이므로 각 문장을 세미콜론으로 마무리

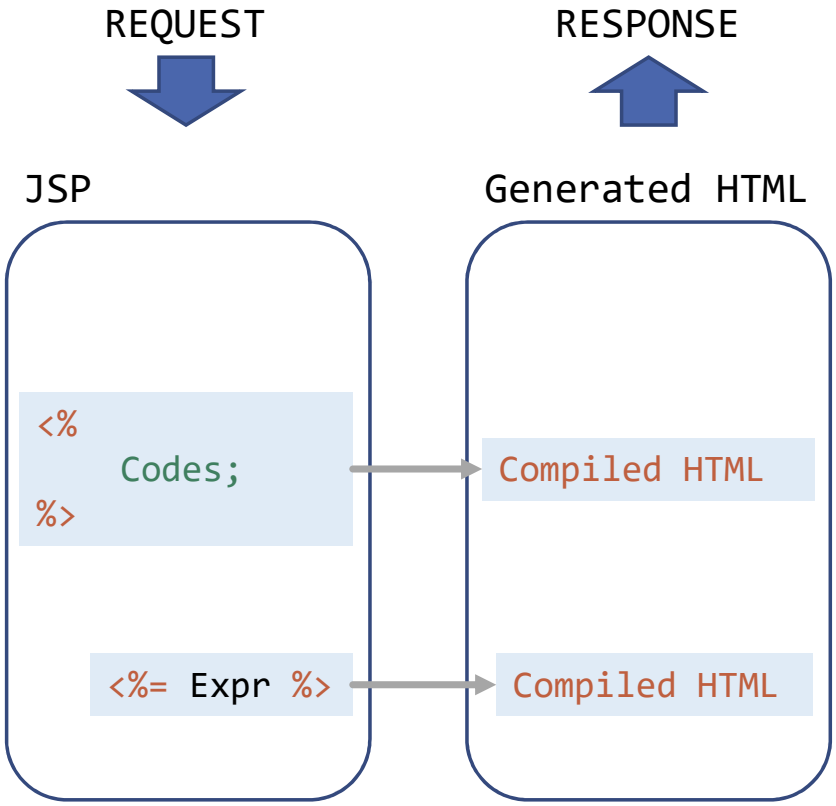
```
<% Java Statements %>
```

  - 표현식 문법
    : 내부적으로는 객체의 toString 메서드로 변환
    : 세미콜론을 붙이지 않음

```
<%= 문자열로 변경될 수 있는 Java 표현식 %>
```

  - request 객체로부터 파라미터 값을 받는 방법

```
request.getParameter( parameterName );
```



JSP 태그

- 스크립틀릿(Scriptlet)과 표현식(Expression) 사용 예
  - 파라미터 값을 화면에 출력하는 스크립틀릿 + 표현식의 예

```
<% String firstName = request.getParameter("firstName"); %>
The first name is <%= firstName %>.
```

  - 파라미터 값을 화면에 출력하는 표현식의 예

```
The first name is <%= request.getParameter("firstName") %>.
```

  - HTML 블록을 5회 출력하는 스크립틀릿 + 표현식의 예

```
<%
    for (int i = 1; i <= 5; i++) { %>
<h1>This line is shown <%= i %> of 5 times.</h1>
    <%
    }
%>
```

JSP 태그

- JSP 주석
  - JSP 주석과 스크립틀릿 내의 Java 주석은 컴파일되지 않으며 실행도 되지 않는다
  - JSP 주석의 예

```
<%--
    Today's data is <%= new java.util.Date() %>
 --%>
```

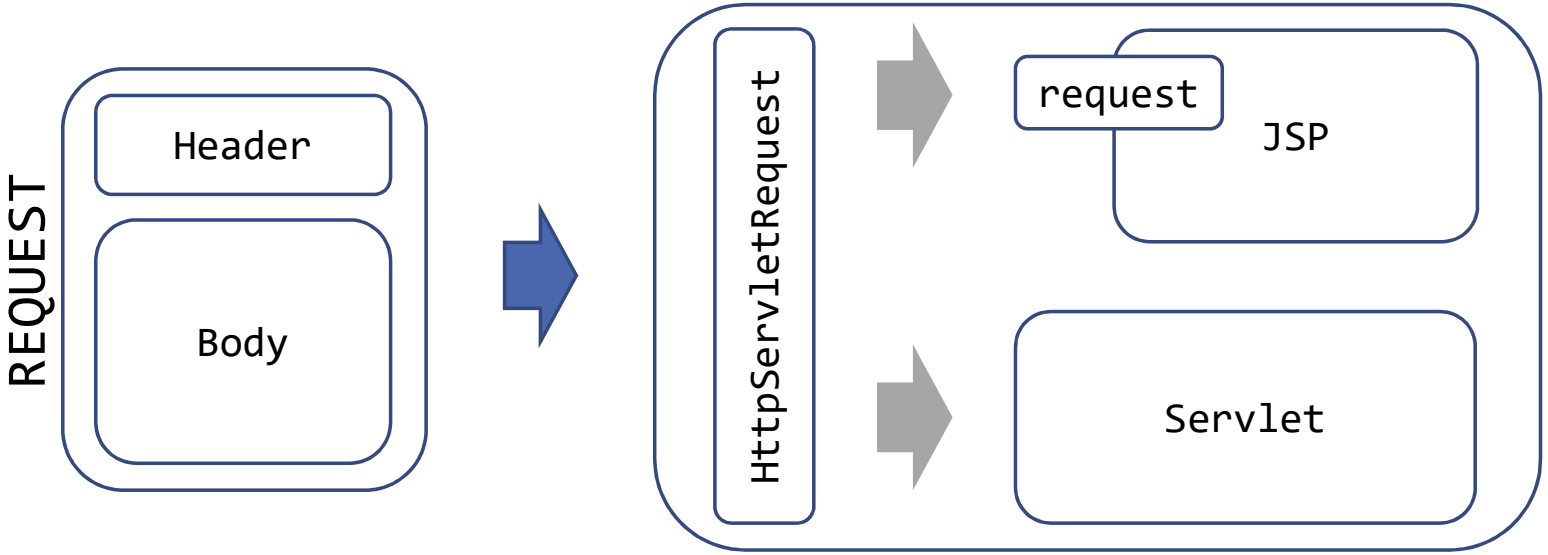
  - HTML 주석

```
<!--
    Today's data is <%= new java.util.Date() %>
 -->
```

    - HTML 주석 내에 스크립틀릿이나 표현식이 포함되어 있을 경우, 해당 내용은 컴파일 되고 실행도 된다.

request 객체의 이해

- 브라우저로부터 전송된 요청 정보는 HttpServletRequest에 적재되어 Servlet의 service, doGet, doPost 등의 메서드에 전달된다.
- JSP에는 암묵적으로 request 라 명명되어 전달된다.
  특별한 작업을 하지 않아도 JSP 스크립틀릿 내에서는 request 객체를 이용,
  브라우저로부터의 요청 정보를 활용할 수 있다.



request 객체의 이해

- 매개 변수(Parameter) 관련 메서드

메서드	리턴타입	설명
getParameter (String param)	String	주어진 이름의 파라미터가 갖는 값을 리턴 지정된 파라미터가 없는 경우는 null
getParameterValues (String param)	String[]	주어진 이름의 파라미터가 갖는 모든 값을 문자열 배열로 리턴. 파라미터가 다중 선택이 가능한 리스트 또는 체크박스 값이라면 여러 개의 값이 하나의 이름으로 전달될 수 있다.
getParameterNames()	Enumeration	request 객체에 포함되어 있는 모든 파라미터의 이름을 열거형 객체로 리턴. request에 파라미터가 하나도 없는 경우에도 비어 있는 Enumeration 객체를 리턴

request 객체의 이해

- 요청 헤더(Header) 관련 메서드

메서드	리턴타입	설명
getHeader (String headerName)	String	요청 Header에서 headerName에 할당된 값을 리턴. 지정한 이름이 없을 시에는 null
getHeaderNames()	Enumeration	요청 Header에 포함된 모든 헤더의 이름을 Enumeration 객체로 리턴

- 쿠키(Cookie) 및 세션(Session) 관련 메서드

메서드	리턴타입	설명
getSession()	HttpSession	요청한 클라이언트에 지정된 HttpSession의 객체를 리턴. 이전에 생성된 객체가 없을 때는 새로운 객체를 생성
getCookies()	Cookie[]	요청 Header에 포함된 쿠키를 Cookie 객체의 배열로 리턴

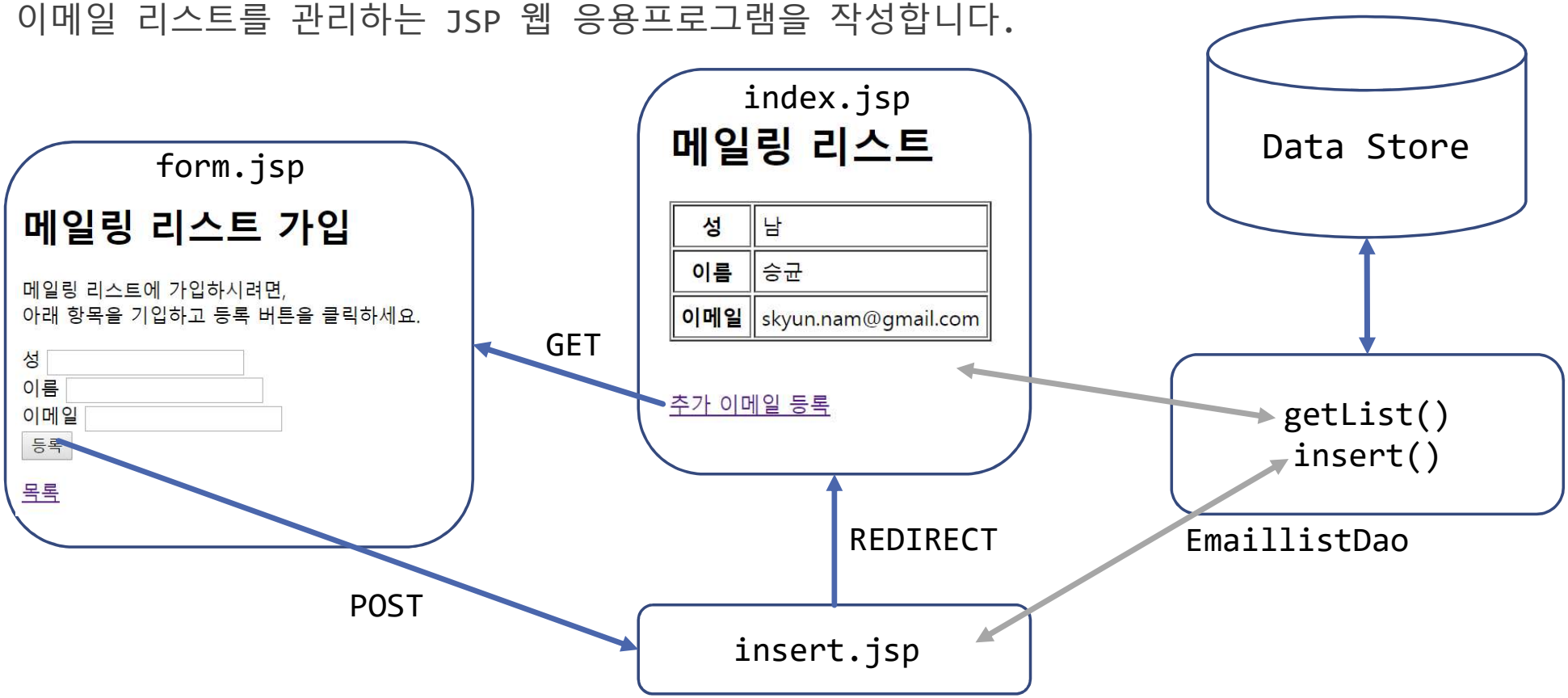
request 객체의 이해

- URL, URI, 요청 Method 관련 메서드

메서드	리턴타입	설명
getServerName()	String	서버의 도메인 이름을 리턴
getServerPort()	int	서버의 포트 번호를 리턴
getRequestURL()	StringBuffer	요청 URL을 리턴
getRequestURI()	String	요청 URI를 리턴
getQueryString()	String	요청에 사용된 쿼리 문장 전체를 리턴
getRemoteHost()	String	클라이언트의 호스트 이름을 리턴
getRemoteAddr()	String	클라이언트의 IP 주소를 리턴
getProtocol()	String	요청에서 사용된 프로토콜 이름(예: http)을 리턴
getMethod()	String	요청에서 사용된 요청 방식(GET, POST 등)을 리턴
getContextPath()	String	해당 서블릿/JSP의 컨텍스트 경로를 리턴

JSP 프로그래밍 연습
: emaillist

- 이메일 리스트를 관리하는 JSP 웹 응용프로그램을 작성합니다.



JSP 프로그래밍 연습
: emaillist

- Database 구성 및 Dao의 설계

emaillist (MySQL의 예)

컬럼명	데이터 타입	제약	설명
no	int	PK, AUTO_INCREMENT	식별번호
last_name	varchar(20)		성
first_name	varchar(20)		이름
email	varchar(128)		이메일 주소
created_at	datetime	NOT NULL	등록 일시

JSP 프로그래밍 연습
: emaillist

- Database 구성 및 Dao의 설계

emaillist (Oracle의 예)

컬럼명	데이터 타입	제약	설명
no	number	PK, AUTO_INCREMENT	식별번호
last_name	varchar2(20)		성
first_name	varchar2(20)		이름
email	varchar2(128)		이메일 주소
created_at	date	NOT NULL	등록 일시

- Primary Key에 사용하기 위한 시퀀스 SEQ_EMAILLIST_PK를 작성해 봅시다.

JSP 프로그래밍 연습
: emaillist

- Database 구성 및 Dao의 설계
  - 테이블을 생성하고 데이터베이스 결과를 바인딩하기 위한 VO를 작성

```
public class EmailVo {
    //Fields
    private Long no;
    private String firstName;
    private String lastName;
    private String email;
}
```

    - GETTER와 SETTER를 작성하고, toString 메서드를 Override 하여 VO를 완성해 봅시다

JSP 프로그래밍 연습
: emaillist

- Database 구성 및 Dao의 설계
  - Dao 구현을 위한 interface를 우선 작성

```
public interface EmaillistDao {
    public List<EmailVo> getList();
    public int insert(EmailVo vo);
}
```

  - EmaillistDao interface를 구현한 DaoImpl 클래스를 작성해 봅시다

```
public class EmaillistDaoMySQLImpl implements EmaillistDao {
    private Connection getConnection() throws SQLException { ... }
    @Override
    public List<EmailVo> getList() { ... }
    @Override
    public int insert(EmailVo vo) { ... }
}
```

JSP 프로그래밍 연습
: emaillist

- index.jsp 구성
  - EmaillistDao에서 getList를 호출하여 EmailVo의 리스트를 반환

```
<%
    EmaillistDao dao = new EmaillistDaoMySQLImpl();
    List<EmailVo> list = dao.getList();
%>
```

  - 반환 받은 EmailVo의 리스트를 화면에 출력해 봅시다

```
<!-- 리스트 -->
<%
for (EmailVo vo : list) {
%>
    <!-- vo 객체의 getter를 이용, 리스트를 표시 --><%
}
%>
```

index.jsp

메일링 리스트

성	남
이름	승균
이메일	skyun.nam@gmail.com

추가 이메일 등록

JSP 프로그래밍 연습
: emaillist

- form.jsp 구성
  - 다음과 같이 POST 전송 HTML Form을 갖는 JSP 페이지를 작성

```
<form action="<%= request.getContextPath() %>/insert.jsp"
    method="POST">
    <label for="ln">성</label>
    <input type="text" name="ln" value=""><br />
    <label for="fn">이름</label>
    <input type="text" name="fn" value=""><br/>

    <label for="email">이메일</label>
    <input type="text" name="email" value=""><br />

    <input type="submit" value="등록">
</form>
```

JSP 프로그래밍 연습
: emaillist

- insert.jsp 구성
  - request 객체로부터 파라미터를 받아 오기

```
<%
    request.setCharacterEncoding("UTF-8");

    String firstName = request.getParameter("fn");
    String lastName = request.getParameter("ln");
    String email = request.getParameter("email");
%>
```

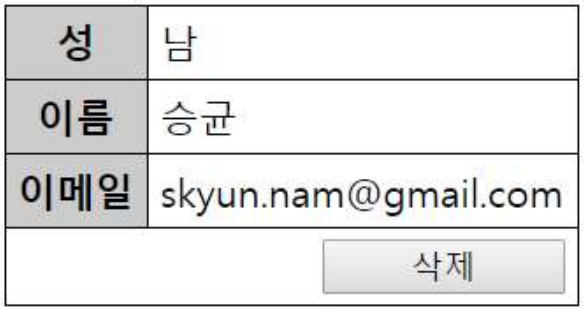
  - EmailVo 객체를 생성하여 Setter를 이용 파라미터를 적재한 후 EmaillistDao의 insert 메서드를 이용하여 데이터베이스에 저장
  - 처리가 완료되면 목록으로 요청을 리다이렉트(Redirect)

```
<%
    response.sendRedirect(request.getContextPath());
%>
```

insert.jsp

JSP 프로그래밍 연습
: emaillist

- 추가 구현 과제
  - 이메일 정보 삭제 기능을 작성해 봅니다.
    - index(리스트) 페이지의 하단에 삭제 버튼을 추가
    - javascript의 confirm 함수를 이용, 삭제 동의를 받은 후
    - 동의하지 않으면: 삭제 작업 진행하지 않음
    - 동의하면: delete.jsp로 해당 이메일 정보의 no 필드값을 전송, 데이터베이스로부터 레코드를 삭제
  - 정보 수정 HTML 폼과 해당 폼의 정보를 이용, 이메일 정보를 데이터베이스로부터 갱신하는 기능도 만들어 봅시다.

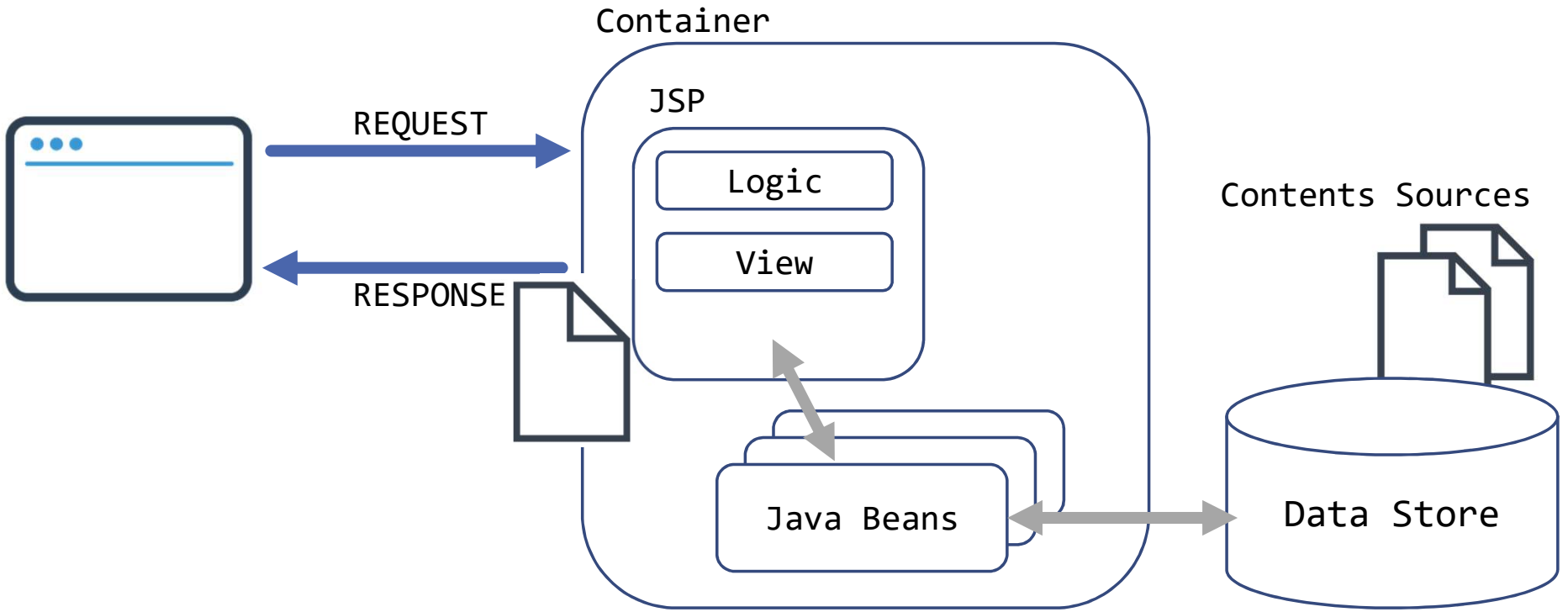


Servlet 프로그래밍 연습

Servlet과 Web Application Model 2

지금까지의 이야기

- 우리가 만든 웹 응용프로그램의 구동 방식



지금까지의 이야기
: Web Application Architecture Model 1

- JSP 내에서 요청 처리, 데이터/서비스 처리 로직, 화면 처리를 위한 뷰 처리 모두 실행하는 구조 : Model 1

- Model 1 Architecture
  - 비즈니스 클래스는 자바 빈즈(Java Beans)로 작성
    - 비즈니스 클래스는 애플리케이션의 비즈니스 프로세스와 데이터를 표현
  - 데이터베이스나 파일 시스템을 데이터 저장소(Data Store)로 활용
    - 영구적인 데이터 저장소 (Persistent Data Storage)
  - VO 클래스와 같은 데이터 클래스는 데이터 저장소의 데이터를 조화하거나 전달, 저장하는데 사용
  - 제한된 요구 사항의 웹 응용프로그램을 개발할 때 주로 사용

Model 1 방식 장단점

- Model 1 방식 요약
  - HTTP Request가 들어오면 JSP에서 요청을 받음
  - JSP 자신 혹은 다른 클래스(ex: DAO)를 이용하여 작업을 처리
  - 처리한 결과를 클라이언트에 응답(출력)

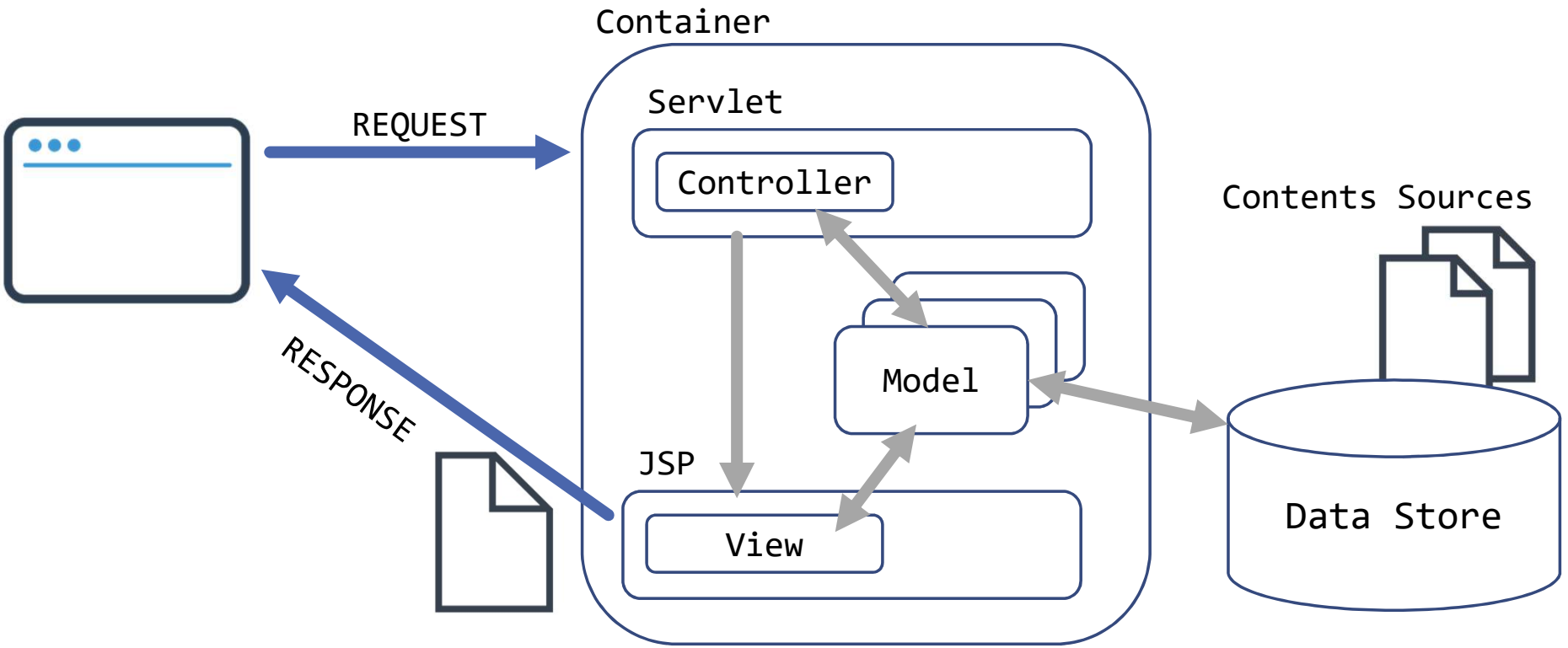
장점	단점
• 구조가 단순하며 익히기 쉬움 • 숙련된 개발자가 아니어도 구현이 용이 • 중소형 프로젝트에 적합	• 출력을 위한 뷰 코드와 로직 처리를 위한 Java 코드가 함께 있어 코드가 복잡해짐 • JSP 코드에 백 엔드와 프론트 엔드가 혼재되어 있어 분업이 용이하지 않음 • 웹 응용프로그램이 복잡해질수록 유지보수가 어려워짐

MVC
: Web Application Architecture Model 2

- 복잡한 요구사항을 처리하는 웹 응용프로그램에서는 Model 2 아키텍처를 사용
  - MVC (Model – View – Controller) 패턴
- 역할별로 다음과 같이 레이어를 분리하여 작성한다
  - Controller : HTTP 요청의 처리, 응용프로그램의 흐름을 제어 – Servlet이 담당
  - Model : 데이터 생성, 조회, 저장 등의 처리 담당. 로직을 가지고 있음
    - Java Beans가 담당
  - View : 화면에 내용을 보여주는 역할 – JSP가 담당

MVC
: Web Application Architecture Model 2

- Servlet을 이용한 Controller 분리
- JSP로부터 로직을 분리, View를 전담



Model 2 방식 장단점

- Model 2 방식 요약
  - HTTP Request가 들어오면 처리를 위한 흐름 제어는 Controller인 Servlet이 담당
  - 요청 처리에 필요한 로직은 Model 클래스가 담당
  - 처리한 결과는 View인 JSP를 통해 클라이언트에 응답(출력)

장점	단점
• 출력을 위한 뷰 코드와 로직 처리를 위한 Java 코드가 분리 • 백 엔드 코드는 서블릿이, 프론트 엔드 코드는 JSP가 전담, 분업과 유지보수가 용이	• Model 1에 비해 구조가 복잡하여 습득 난이도가 높으며 작업량이 많아짐 • Java에 대한 조금 더 깊은 이해가 필요 • 개발 초기 구조 설계를 위한 시간이 필요

  - MVC 패턴을 사용할 때는 가능한 각 레이어를 독립적으로 구성해야 한다
    : 하나의 레이어에 수정이 발생해도 타 레이어에 미치는 영향 최소화

애너테이션(@)을 이용한 서블릿 매핑

- web.xml을 이용한 서블릿 설정 방식은 구성이 복잡해진다는 단점
- Tomcat 7 버전 이후에는 @WebServlet 애너테이션을 이용한 서블릿 매핑 방식을 지원
  - 애너테이션 : 소스코드에 직접 기능을 설정하는 방식
    - 서블릿 클래스에 직접 URL 매핑을 설정

```
@WebServlet("/myurl")
public class MyServlet extends HttpServlet { }
```

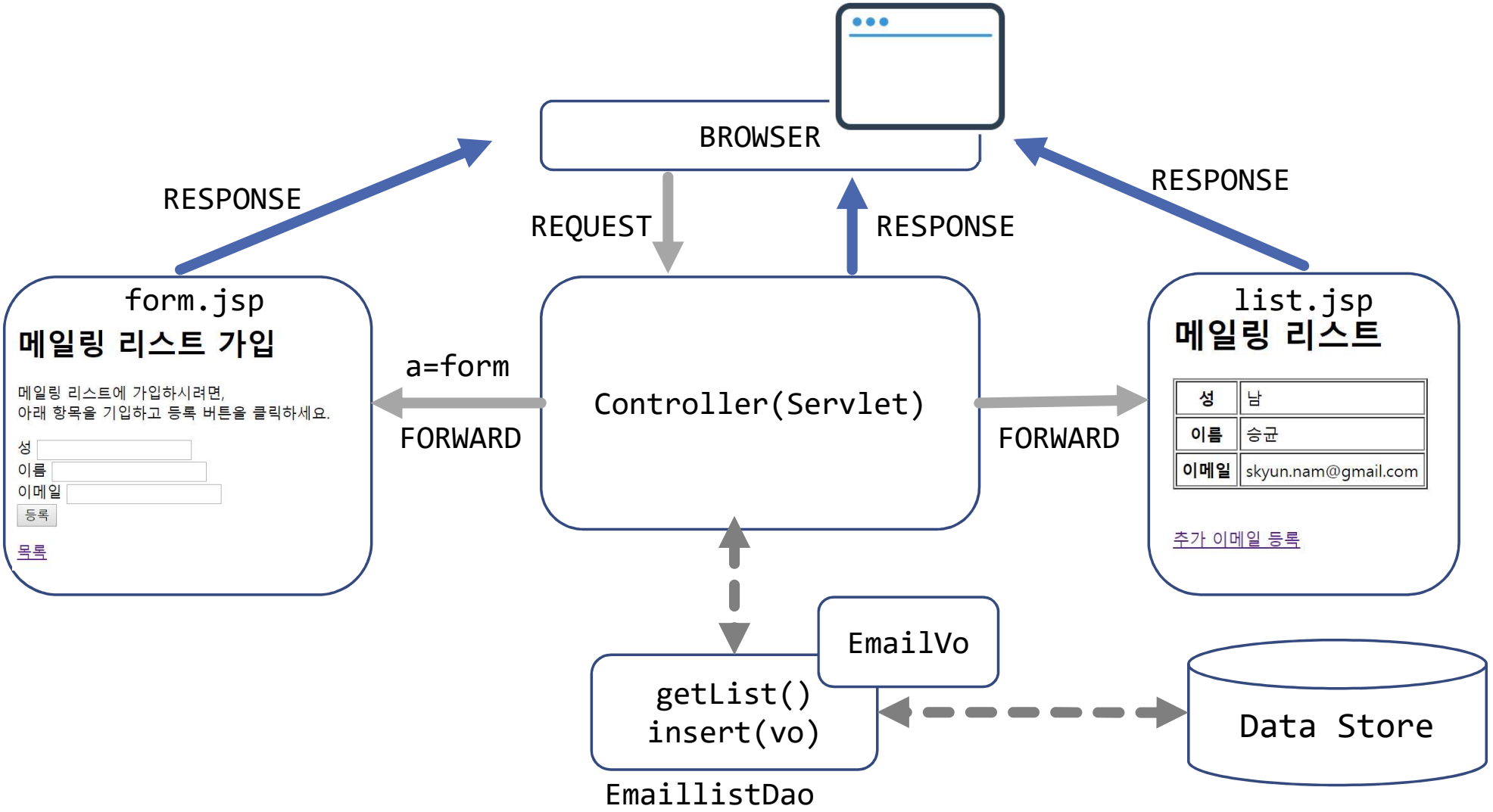
    - 서블릿 클래스에 서블릿 이름과 URL 매핑을 설정

```
@WebServlet(name="MyServlet", urlPatterns="/myurl")
public class MyServlet extends HttpServlet { }
```

    - 서블릿 클래스에 서블릿 이름, URL 매핑, 초기화 파라미터를 설정

```
@WebServlet(name="MyServlet", urlPatterns="/myurl",
initParams= { @WebInitParam(name="paramName", value="paramValue") })
public class MyServlet extends HttpServlet { }
```

Servlet 프로그래밍 연습



Servlet 프로그래밍 연습

- Model 1 방식으로 작성된 emaillist 웹 응용프로그램을 MVC 방식의 Model 2 방식으로 재작성해 봅니다.

- HTTP 요청 처리를 담당할 Servlet (Controller)를 구현

```
@WebServlet(name="EmailList", urlPatterns="/el")
public class EmaillistServlet extends HttpServlet { }
```

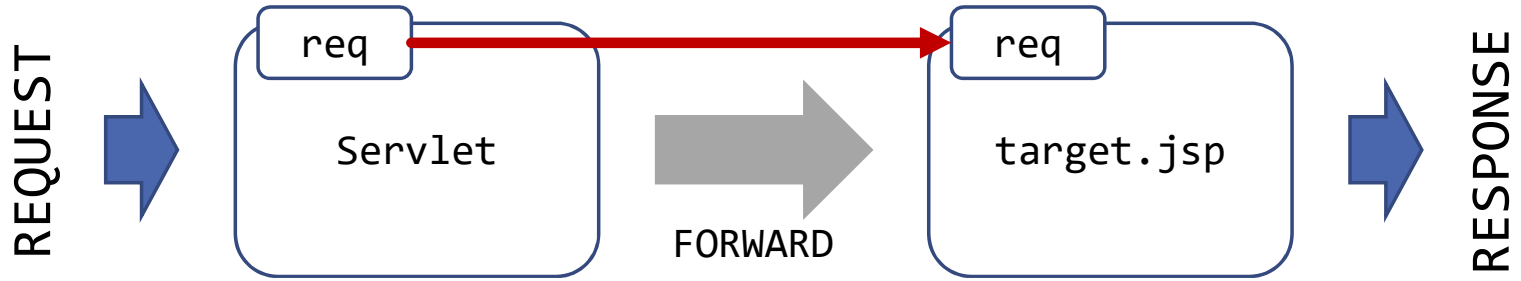
- doGet 메서드를 Override 하여 form 페이지가 필요한 상황일 때 form.jsp로 포워드

```
String actionName = req.getParameter("a");

if ("form".equals(actionName)) {
    RequestDispatcher rd = getServletContext()
                            .getRequestDispatcher("/WEB-INF/views/form.jsp");
    rd.forward(req, resp);
}
```

Servlet 프로그래밍 연습
: 요청 Dispatch (Forward)

- 요청 Dispatch (Forward)
  - 서버 내부에서 요청 처리를 다른 객체에 전달하여 처리하도록 함
  - Forward 작업을 수행하기 위해서는 Context로부터 RequestDispatcher를 얻어온다

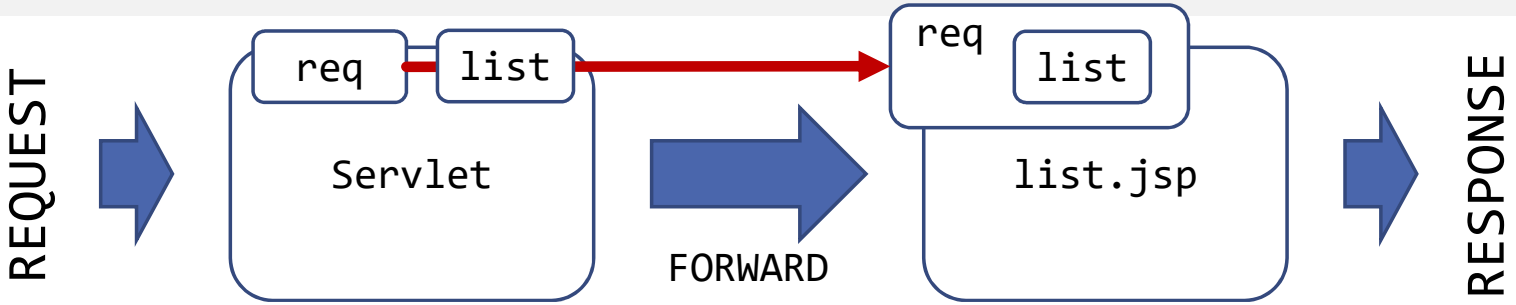
```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    ...
    RequestDispatcher rd = getServletContext().getRequestDispatcher("target.jsp");
    rd.forward(req, resp);
}
```



Servlet 프로그래밍 연습

- list 기능 구현
  - doGet 메서드에서 기본 처리 로직으로 구현, 요청 객체에 처리된 데이터 객체를 추가하여 전송

```
/*Default 요청 처리 (list) */
EmaillistDao dao = new EmaillistDaoMySQLImpl();
List<EmailVo> list = dao.getList();

req.setAttribute("list", list);

RequestDispatcher rd = getServletContext()
                        .getRequestDispatcher("/WEB-INF/views/list.jsp");
rd.forward(req, resp);
```



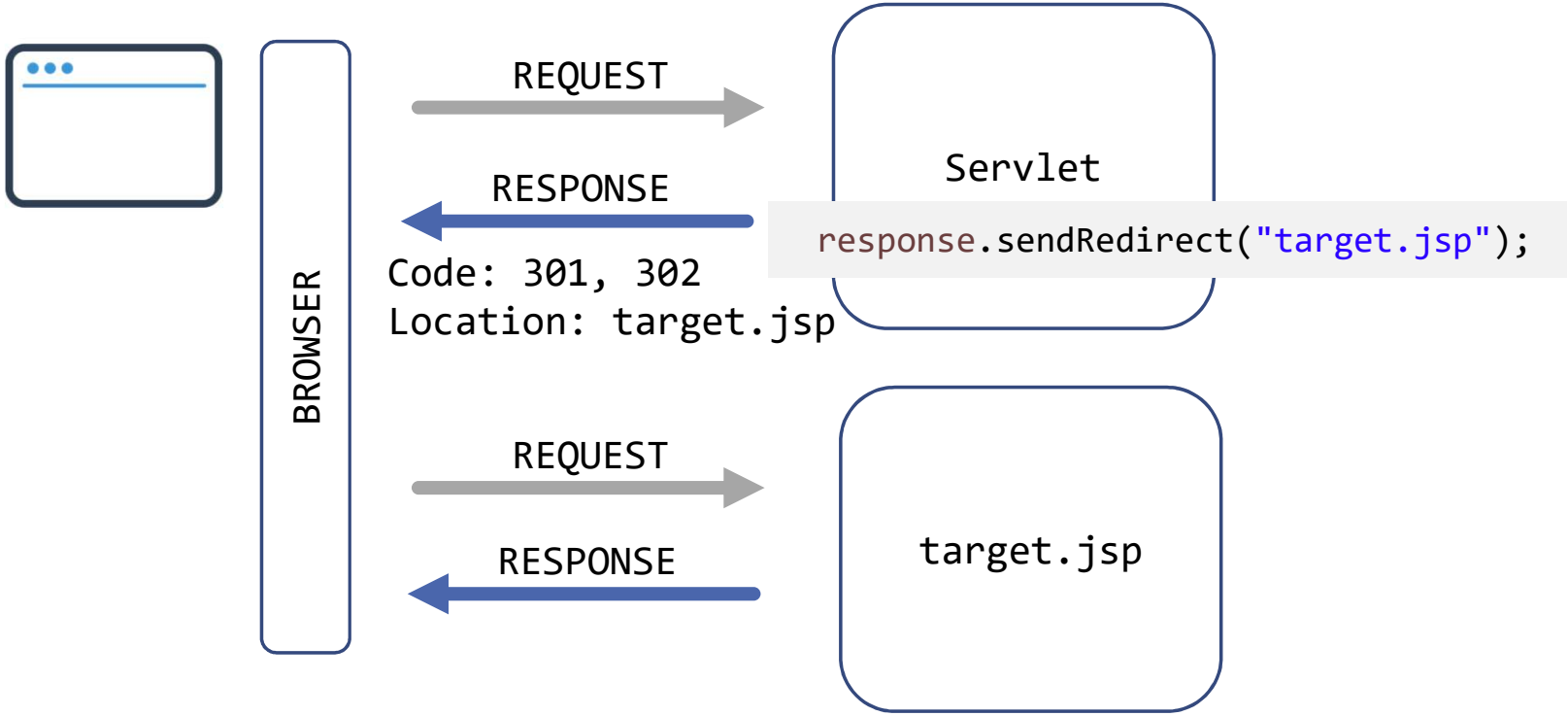
Servlet 프로그래밍 연습

- form insert 기능
  - doPost 메서드를 오버라이드하여 POST form으로부터 전송된 데이터를 저장
  - 저장이 완료되면 리스트 페이지로 리다이렉트(Redirect)

```
//Method = POST, a = insert일 때의 처리
String firstName = req.getParameter("fn");
...

EmailVo vo = new EmailVo();
...
vo.setEmail(email);

new EmaillistDaoMySQLImpl().insert(vo);

resp.sendRedirect(req.getContextPath() + "/el");
```

Servlet 프로그래밍 연습
: Redirect

- 웹 서버가 브라우저에게 지정한 페이지로 이동하도록 지시



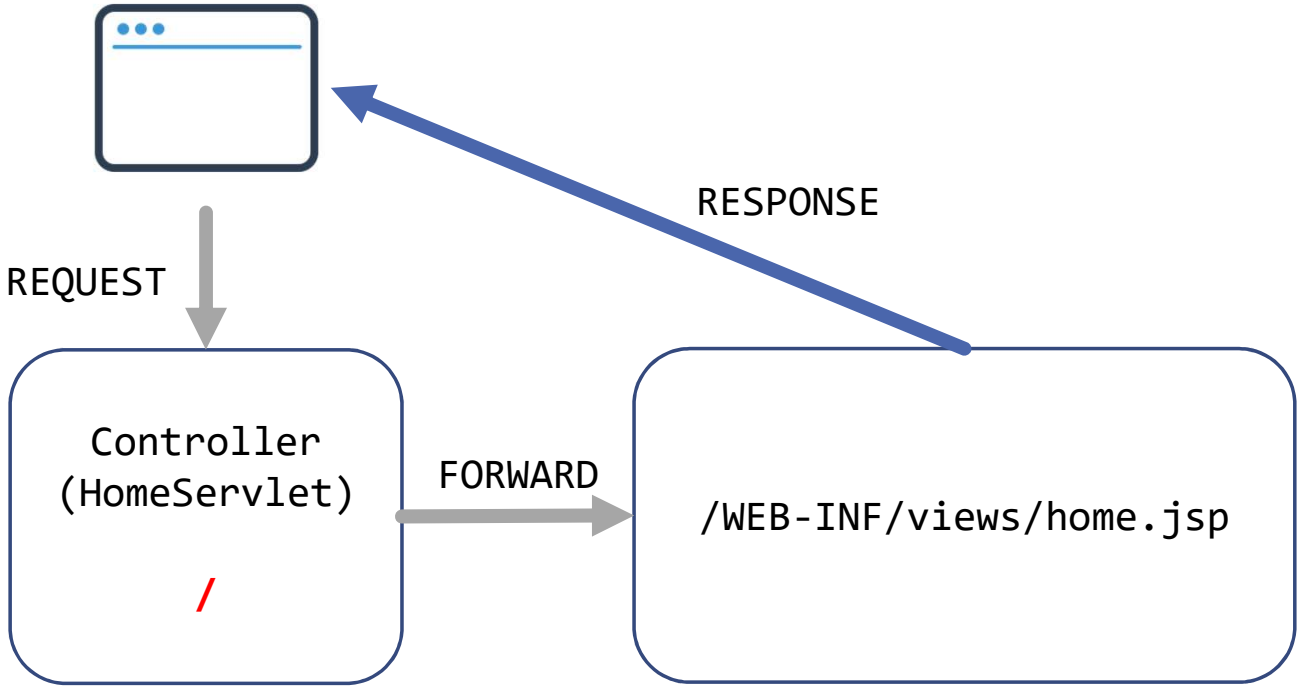
Servlet 프로그래밍 연습
: Redirect vs Forward

▶ 공통점과 차이점

	FORWARD	REDIRECT
공통점	다른 페이지를 호출하는 방법으로 사용	
차이점	• 브라우저로부터 전송 받은 요청을 컨테이너 내부의 다른 객체로 전송 • 새 페이지는 이전 페이지로부터 요청 객체를 전달 받아 처리하여 응답한다	• 브라우저에게 새로운 페이지로 다시 요청하라는 응답을 전송 • 브라우저는 이 응답을 받으면 즉시 새로운 페이지로 새 요청을 전송한다
요청	동일 요청의 처리	별개의 요청

Servlet 프로그래밍 연습
: 과제 프로젝트 - 메인 페이지

- MyHome 웹 응용프로그램을 작성하고 HomeServlet이 메인 페이지를 처리하도록 구현해 봅시다



Servlet 프로그래밍 연습
: 과제 프로젝트 - 회원 가입

- USERS 테이블 설계(MySQL)

컬럼명	데이터 타입	제약	설명
no	int	PK, AUTO_INCREMENT	식별번호
name	varchar(20)	NOT NULL	사용자 이름
password	varchar(20)	NOT NULL	비밀번호
email	varchar(128)	UNIQUE, NOT NULL	이메일 주소
gender	enum('M', 'F')	NOT NULL	성별
created_at	datetime	NOT NULL	등록 일시

- UserVo와 UserDao를 구현해 USERS 테이블을 위한 CRUD를 작성해 봅시다

Servlet 프로그래밍 연습
: 과제 프로젝트 - 회원 가입

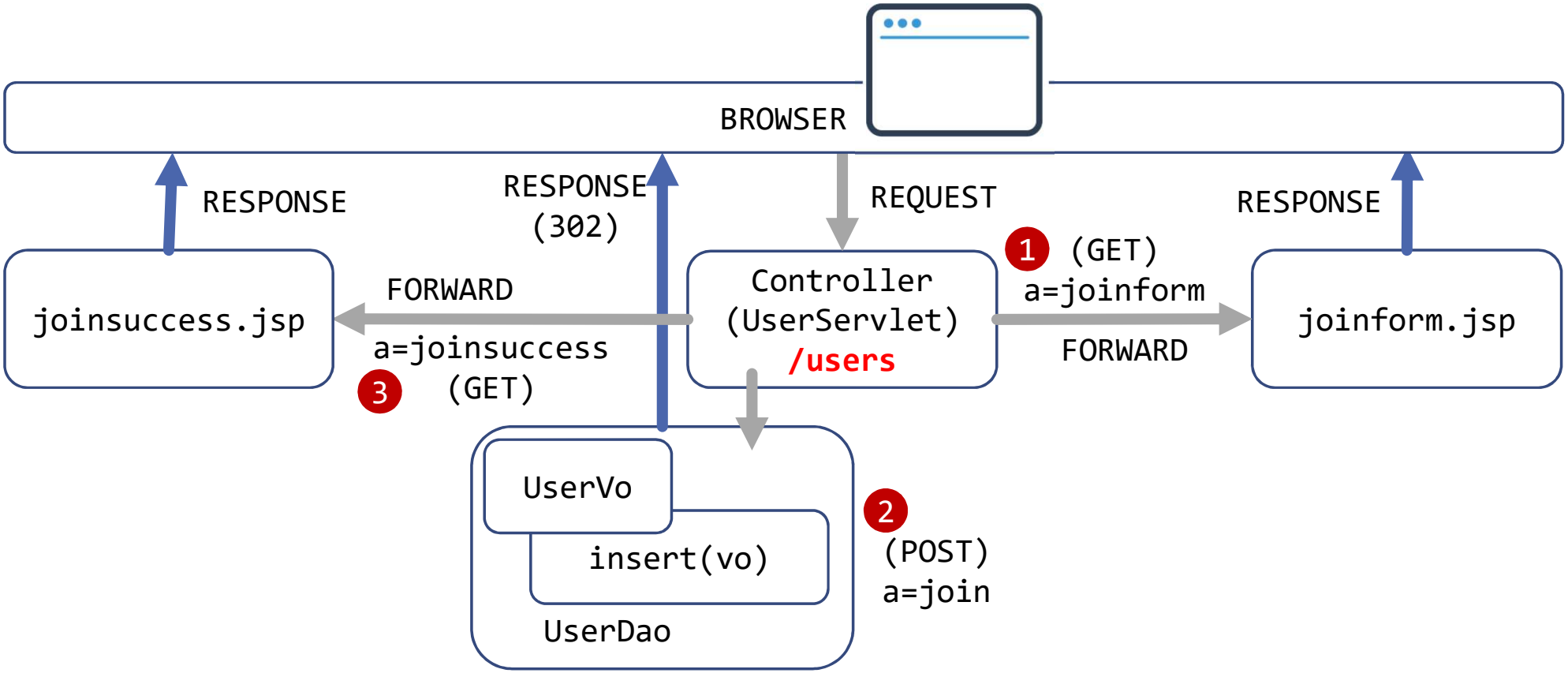
- USERS 테이블 설계(Oracle)

컬럼명	데이터 타입	제약	설명
no	number	PK, AUTO_INCREMENT	식별번호
name	varchar2(20)	NOT NULL	사용자 이름
password	varchar2(20)	NOT NULL	비밀번호
email	varchar2(128)	UNIQUE, NOT NULL	이메일 주소
gender	char(1)	NOT NULL, CHECK(gender IN ('M', 'F'))	성별
created_at	datetime	NOT NULL	등록 일시

- Primary Key에 사용하기 위한 시퀀스 SEQ_USERS_PK를 작성해 봅시다
- UserVo와 UserDao를 구현해 USERS 테이블을 위한 CRUD를 작성해 봅시다

Servlet 프로그래밍 연습
: 과제 프로젝트 - 회원 가입

- UserServlet과 JSP View 작성



다른 페이지의 파일을 포함하기

: <jsp:include ...> vs <%@ include ...%>

- 웹 페이지를 여러 조각으로 나누어 개발하고자 할 때 <jsp:include ...> 혹은 <%@ include ...> 태그를 이용

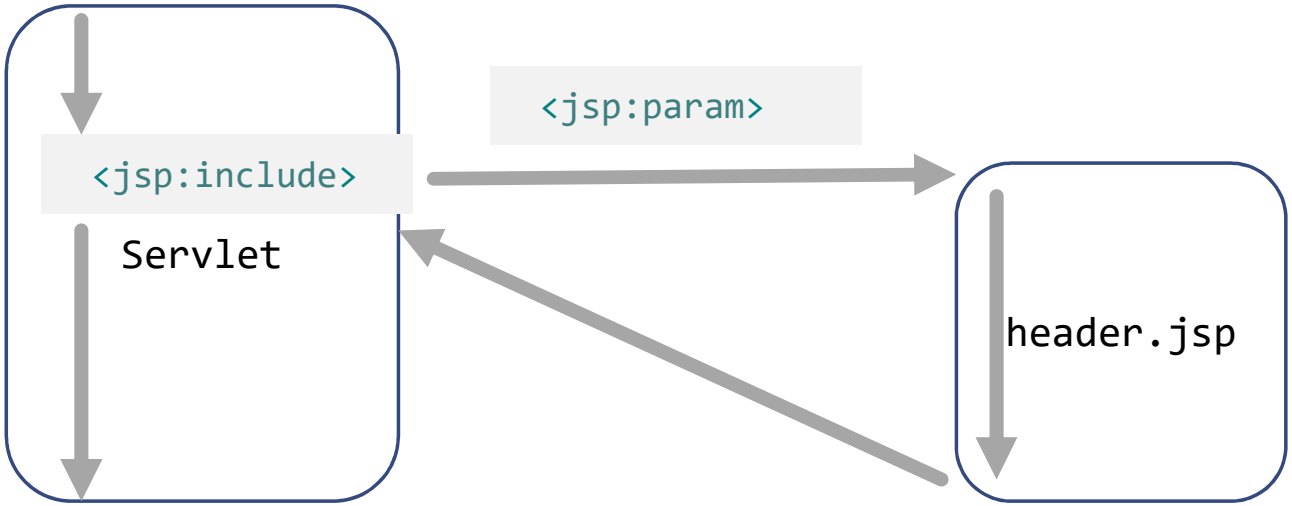
항목	<jsp:include>	<%@ include %>
처리 시간	요청 시간에 처리	JSP 파일을 자바 소스로 변환할 때 처리
흐름	별도의 파일로 요청 처리 흐름을 이동	현재 파일에 삽입
데이터 전달	Request 기본 객체나 <jsp:param>을 이용	페이지 내의 변수를 선언한 후, 변수에 값 저장
용도	화면의 레이아웃 일부를 모듈화할 때 주로 사용	다수의 JSP 페이지에서 공통으로 사용하는 변수를 지정하는 코드 혹은 저작권 등의 문장을 포함

다른 페이지의 파일을 포함하기
: <jsp:include ...>

- <jsp:include ...> 사용 예

```
<jsp:include page="/WEB-INF/views/includes/header.jsp">
    <jsp:param name="paramName1" value="paramValue1" />
    <jsp:param name="paramName2" value="paramValue2" />
</jsp:include>
```

- JSP, Servlet, HTML 등 include 할 수 있다
- 제어가 이동될 때, 현재까지의 출력을 브라우저로 전송하고 싶다면 flush 속성을 true로 지정



Servlet 프로그래밍 연습
: 과제 프로젝트

- MyHome 웹 응용프로그램의 JSP 페이지 중, 공통으로 사용될 영역을 별도의 JSP 로 추출하여 작성해 봅니다
  - header.jsp
  - footer.jsp

- 기존 JSP 페이지의 공통 사용 영역을 제거하고, <jsp:include > 액션 태그를 이용, header와 footer를 동적으로 로딩하여 페이지에 포함해 봅니다

```
<jsp:include page="/WEB-INF/views/includes/header.jsp" />
...
<jsp:include page="/WEB-INF/views/includes/footer.jsp" />
```

Cookie

동적 페이지를 위한 브라우저 상태 유지

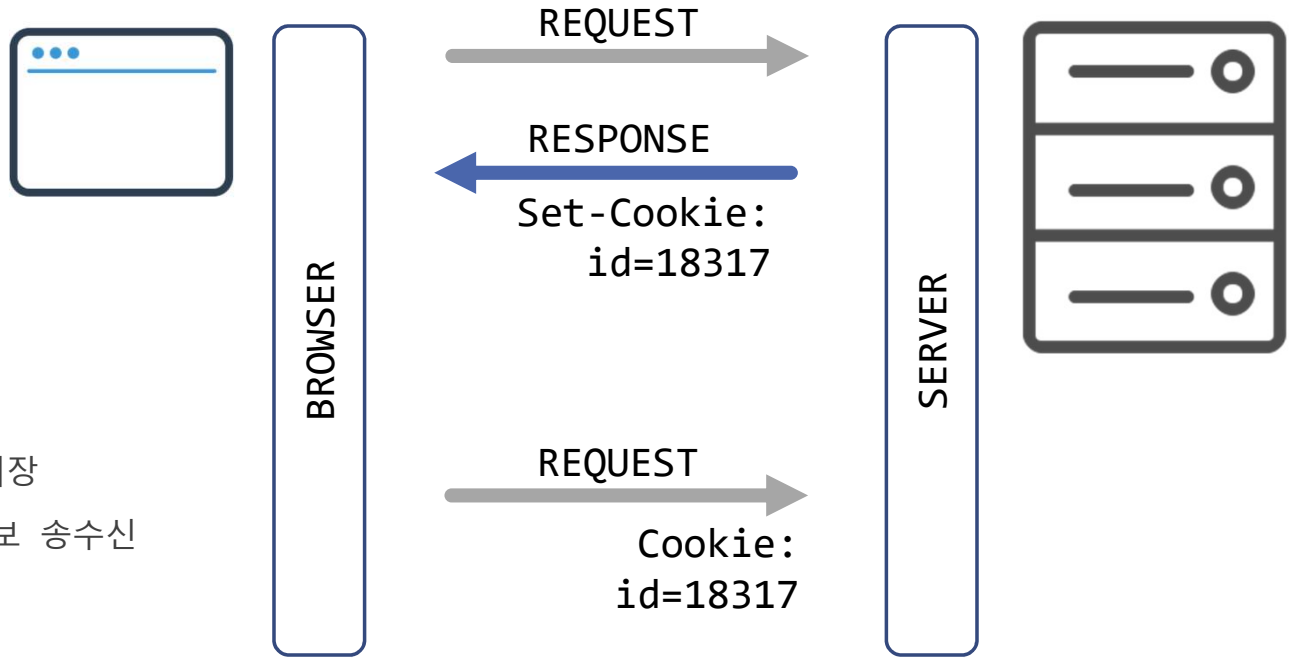
- 기본적으로 HTTP 프로토콜은 상태를 저장하지 않음(Stateless)
  - 모든 요청-응답 사이클은 서로 관계가 없음
  - HTTP 클라이언트와 무관하게 HTTP 연결의 데이터는 서로 상관이 없음
  - 동적 맞춤형 자료를 제공하기 위해 접속 브라우저를 식별해야 할 필요가 발생

- 접속 브라우저 식별(상태 유지) 및 임시 데이터 저장을 위한 다양한 노력들
  - 클라이언트 IP
  - HTTP Header를 이용
  - 인증 식별 정보를 가진 값을 파라미터로 주고 받음 등

- 서버가 브라우저를 식별하고 간단한 데이터를 저장해 둘 수는 없을까...?

Cookie

- Cookie : 서버가 사용자의 컴퓨터에 설치하는 작은 기록 정보 파일
  - HTTP 쿠키, 브라우저 쿠키, 웹 쿠키라고도 일컬음
  - 쿠키에 담긴 정보는 인터넷 사용자가 웹 사이트를 방문할 때, 서버가 조회하고 변경할 수 있음
  - 쿠키 안에 개인 데이터가 담겼을 때는 악용의 여지가 있지만 그 자체는 악성 파일이 아님
  - 쿠키로 할 수 있는 일들
    - 웹 사이트 기본 설정
    - 쇼핑 카트 등의 임시 데이터 저장
    - 페이지 개인화를 위한 식별 정보 송수신



Cookie
: 구성 요소

- 쿠키는 HTTP 헤더 정보에 포함되어 전달

구성요소	설명
이름	각 쿠키의 값을 식별하기 위한 키(key)
값	지정된 이름으로 쿠키에 저장된 값(value)
유효시간	쿠키의 유지 시간(초)
도메인	쿠키를 전송할 도메인
경로	쿠키를 전송할 경로

- Java에서 Cookie를 다루기 위한 API는 java.servlet.http 패키지에 포함

- 제약 조건
  - 클라이언트에 저장할 수 있는 총 쿠키 개수와 하나의 도메인 당 쿠키 수가 제한
  - 하나의 쿠키 값은 4KB까지 저장 가능

Cookie
: 생성 및 변경

- 쿠키를 생성 및 변경하고자 할 때는 Cookie 클래스의 생성자를 이용

```
Cookie jcookie = new Cookie("example", value);
```

- 생성된 쿠키를 브라우저에 기록하기 위해서는 응답 헤더에 쿠키를 추가하여 브라우저로 전송

```
resp.addCookie(jcookie);
```

Cookie
: 조회 및 삭제

- 요청 헤더에 getCookies() 메서드를 호출하면 브라우저에 설정된 쿠키의 배열을 반환한다.

```
Cookie[] cookies = req.getCookies();
if (cookies != null) {
    for (Cookie cookie:cookies) {
        String name = cookie.getName();
        String value = cookie.getValue();
            ...
    }
}
```

- 쿠키 삭제를 위한 API는 없으며 쿠키의 유효시간을 0으로 설정, 무효화한다

```
if (cookie.getName().equals("cname")) {
    Cookie deletedCookie = new Cookie("cname", "");
    deletedCookie.setMaxAge(0);
    resp.addCookie(deletedCookie);
}
```

Session

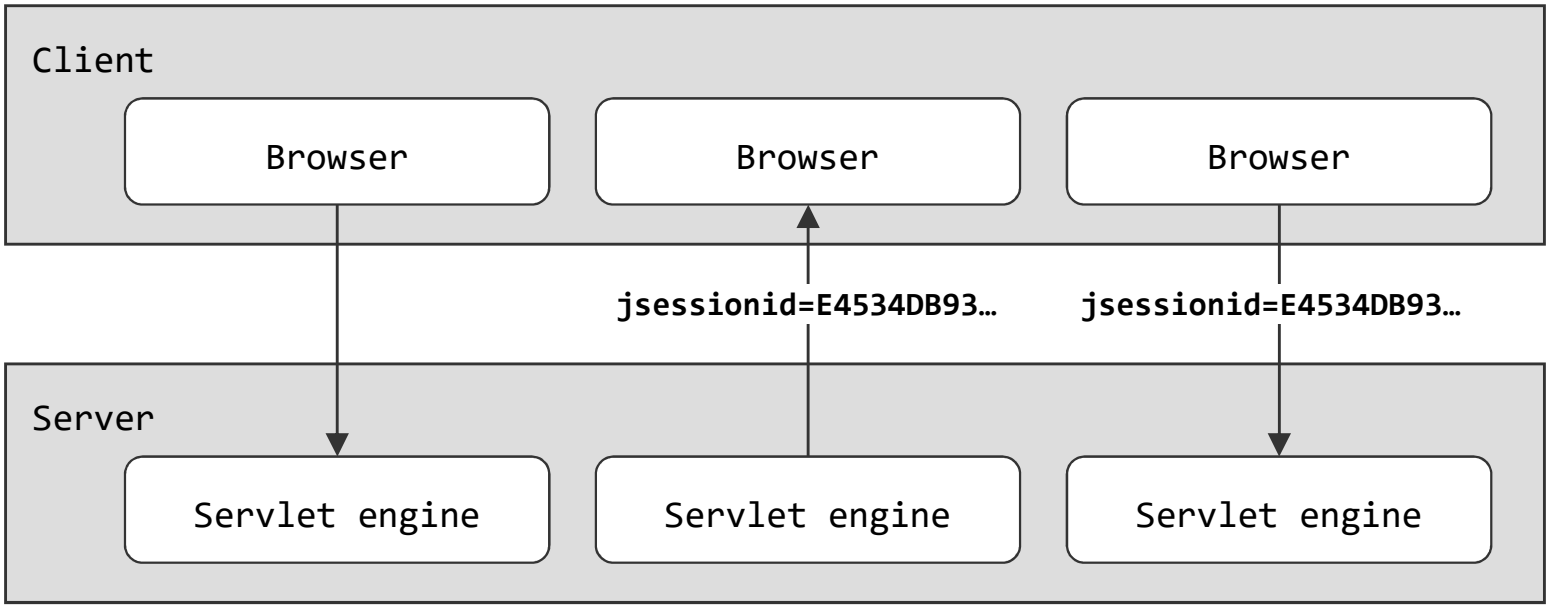
Session

- 상태 없는 HTTP 프로토콜 연결에 상태를 유지해주는 기술
- 쿠키와 비슷하지만, 세션은 클라이언트(브라우저)에 세션 ID만 부여해주고 실제 데이터는 서버에 저장

- 세션의 장점
  - 각 클라이언트에 고유한 ID를 부여(정확한 구분 가능)
  - 세션 ID로 클라이언트를 구분, 클라이언트의 요구에 맞는 맞춤형 서비스 제공 가능
  - 실제 데이터들은 서버에 저장하기 때문에 쿠키보다 보안 면에서 우수
- 세션의 단점
  - 세션 데이터는 서버에 저장되므로 서버상에 처리를 요구하는 부하와 저장 공간이 필요

Session

- 서블릿 엔진에서 세션을 유지하는 방식



최초 HTTP 요청 :
브라우저에서 JSP 또는 서블릿을 요청한다. 서블릿 엔진은 세션 객체를 생성하고 ID를 할당한다.

최초 HTTP 응답 :
서버는 요청된 페이지와 세션에 할당된 ID를 함께 보낸다.

이후의 HTTP 요청 :
브라우저에서 웹 페이지를 요청한다. 서블릿 엔진은 세션ID를 이용해 브라우저에 할당된 세션 객체를 확인한다.

Session

- 세션 객체(HttpSession)에 접근
  - 세션 객체 얻기 (request 객체의 메서드)

```
public HttpSession getSession()
```

    - request.getSession(true) : request에 대한 새로운 세션을 생성한 후 리턴
    - request.getSession(false) : 현재 세션이 존재하면 기존 세션 리턴, 없으면 null 리턴
    - request.getSession() : 현재 세션이 존재하면 기존 세션 리턴, 없으면 새로 생성한 세션 리턴
  - 세션 객체의 기본 메서드

메서드	설명
public void setAttribute(String name, Object value)	value 객체를 name 키로 저장
public Object getAttribute(String name)	name 키로 저장된 객체를 얻어 오기
public void removeAttribute(String name)	name 키로 지정된 객체를 삭제
public void invalidate()	세션을 무효화(없애기)

Session

- 실습: 세션 변수 저장 (session_write.jsp)

```
...
<%@ page session="true" %>
<%
    session.setAttribute("sess1", "세션 String 테스트");
    session.setAttribute("sess2", Integer.valueOf(10));
    session.setAttribute("sess3", Double.valueOf(3.14));
%>
<!DOCTYPE html>
<html>
...
<h2>세션 변수 저장</h2>
<p><a href="session_read.jsp">세션 변수 확인</a></p>
...
```

  - JSP 페이지 내에서는 특별히 지정하지 않아도 HttpSession 객체를 session 이라는 이름으로 사용 가능

Session

- 실습: 세션 변수 읽기 (session_read.jsp)

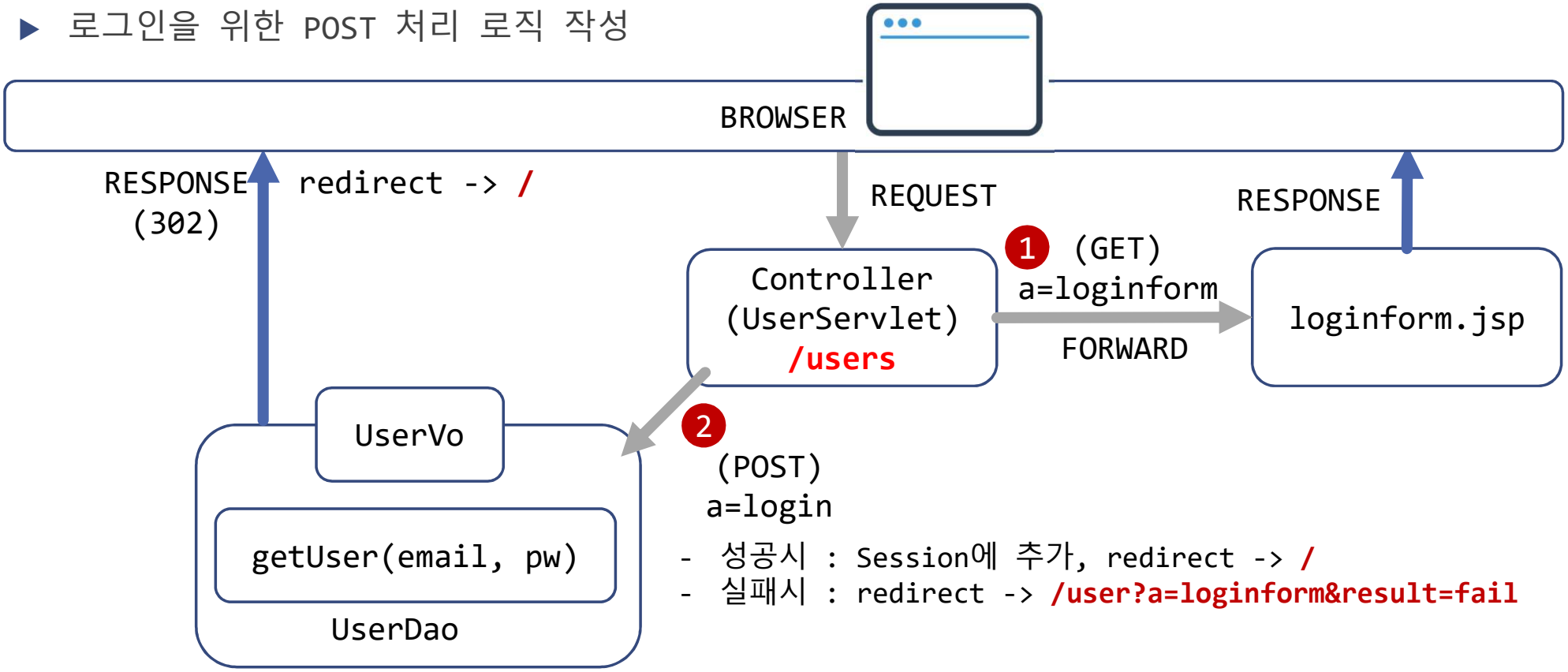
```
<%
String str = (String)session.getAttribute("sess1");
Integer i = (Integer)session.getAttribute("sess2");
Double d = (Double)session.getAttribute("sess3");
%>
...
<h4>sess1 : <%= str %></h4>
<h4>sess2 : <%= i %></h4>
<h4>sess3 : <%= d %></h4>
```

- 세션 무효화 (session_remove.jsp")

```
<%
session.invalidate();
%>
```

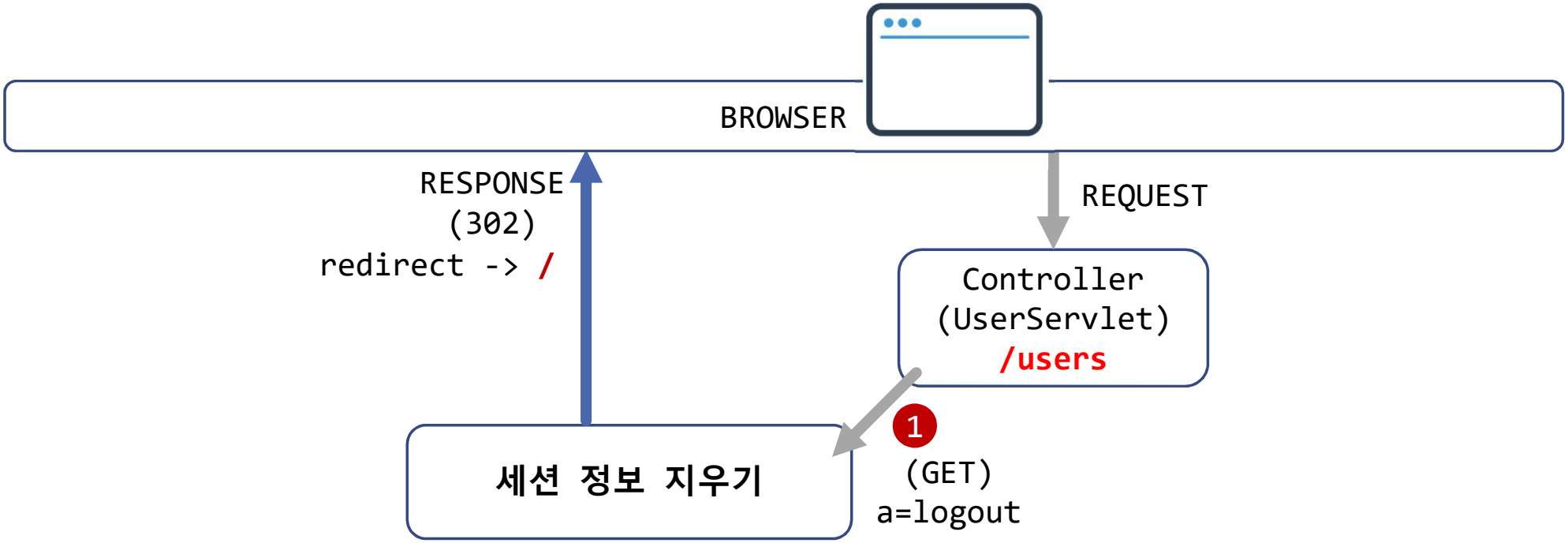
Servlet 프로그래밍 연습
: 과제 프로젝트 - Session을 이용한 로그인

- 로그인을 위한 form JSP 작성
- 로그인을 위한 POST 처리 로직 작성



Servlet 프로그래밍 연습
: 과제 프로젝트 - Session을 이용한 로그아웃

- 로그아웃을 위한 GET 로직 작성




```
HttpSession session = req.getSession(false);
session.removeAttribute(SESSION_NAME);
session.invalidate();
```

- 세션 지우기
- 세션 무효화(invalidate)