

1.웹 프로그래밍의 시작

Index

1. Java 웹 개발 환경 만들기
2. 웹의 기본 동작 방식 이해하기
3. Web MVC 방식
4. HttpServlet
5. 모델과 컨트롤러

1.1 자바 웹 개발 환경 만들기

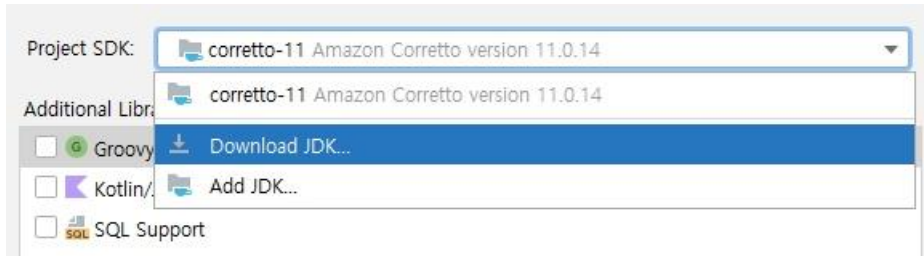
- IntelliJ(ultimate), Gradle을 이용한 프로젝트 생성
- Tomcat의 설치

웹 프로젝트의 기본구조

- 웹 프로젝트는 기본적으로 브라우저 - 서버 - 데이터베이스의 연동으로 구성된다.
- 브라우저 - 요청(request)과 응답(response)
- 웹 서버 혹은 WAS - 요청을 처리해서 원하는 결과를 만들어 내는 역할
- 데이터베이스 - 영속적으로 데이터를 보관

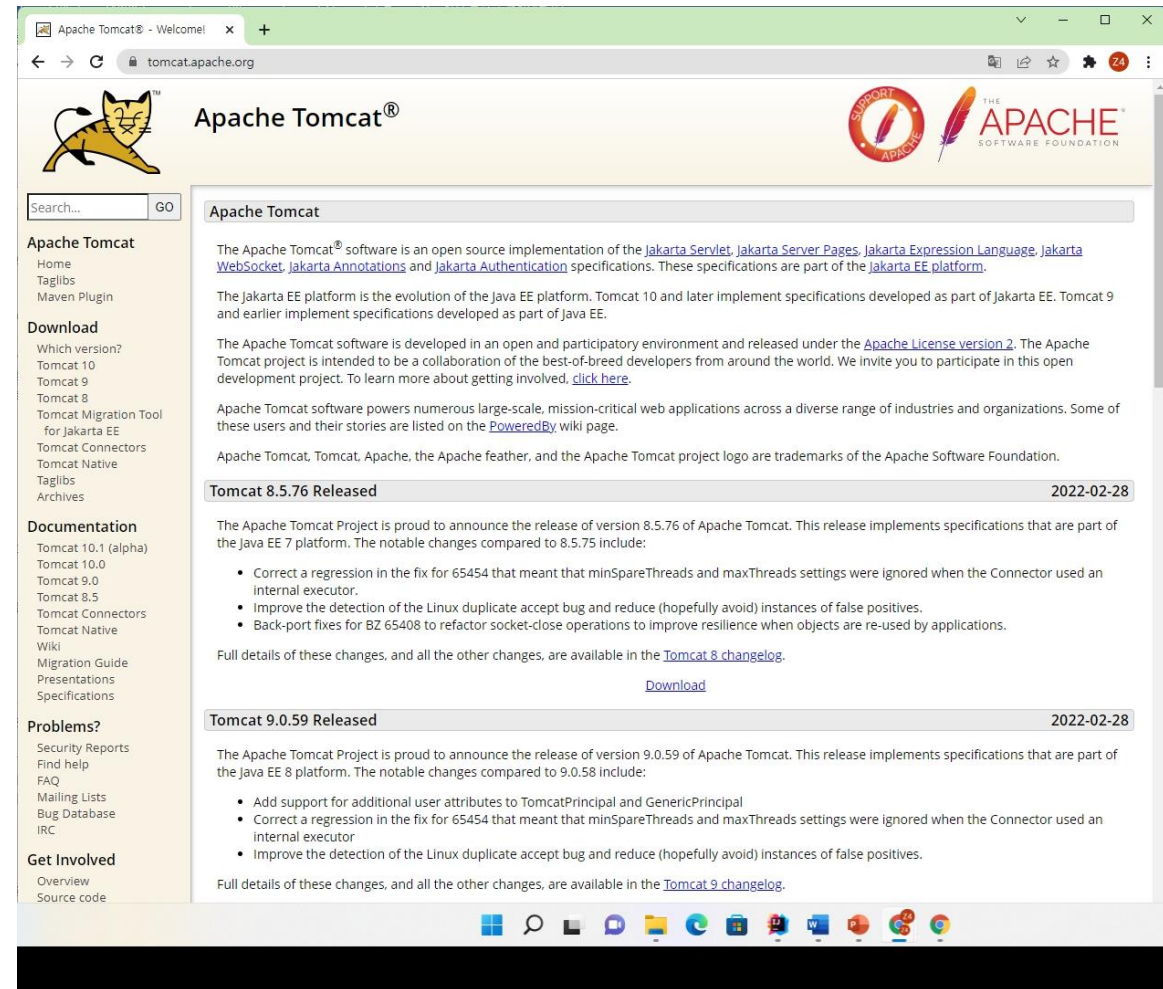
IntelliJ (ultimate)

- Community 에디션은 웹 프로젝트의 생성은 불가
- Ultimate 에디션은 30일 무료/ 유료 버전
- 프로젝트 생성시 JDK의 다운로드나 설정

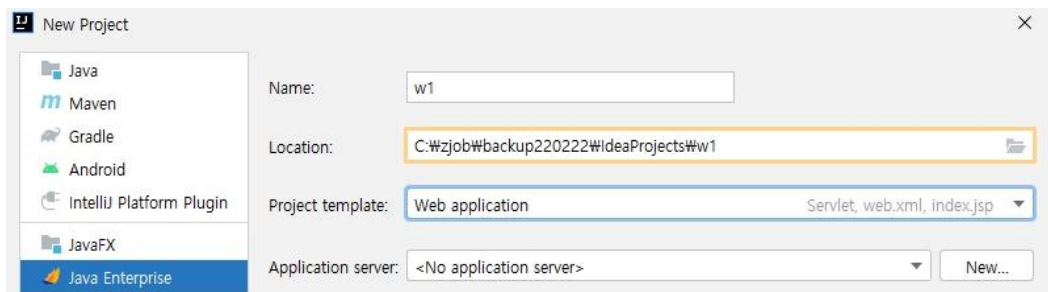


Tomcat의 다운로드와 설치

- tomcat.apache.org에서 다운로드
- 10버전은 API가 다르므로 주의
- 9버전 다운로드 및 압축해제

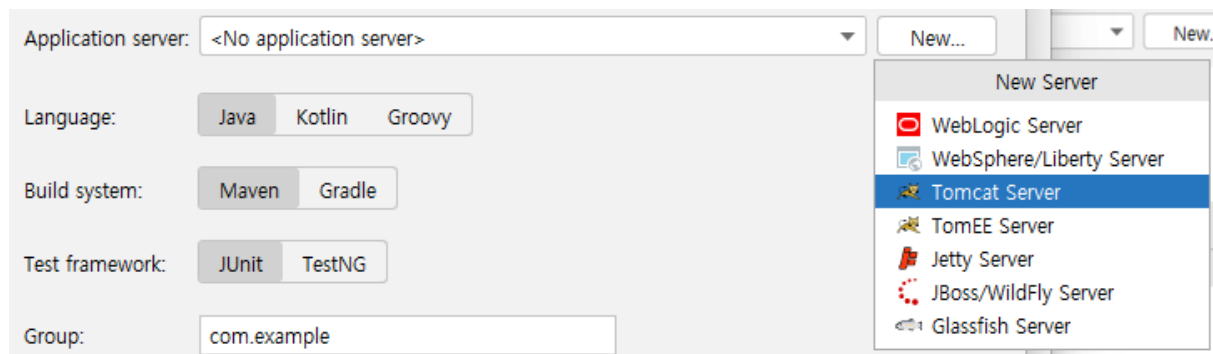


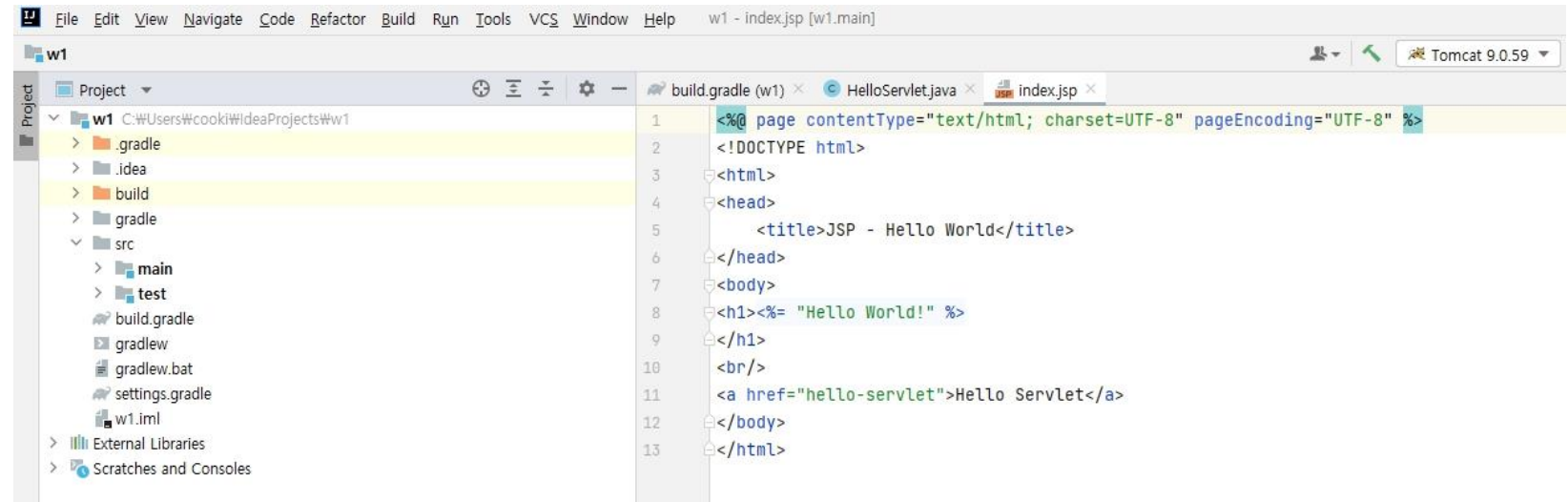
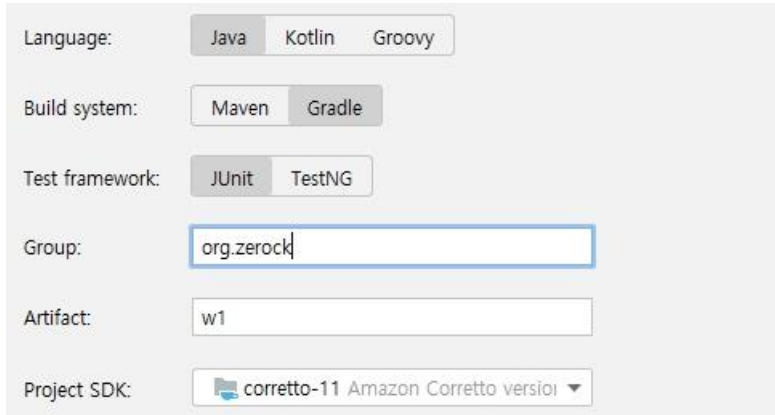
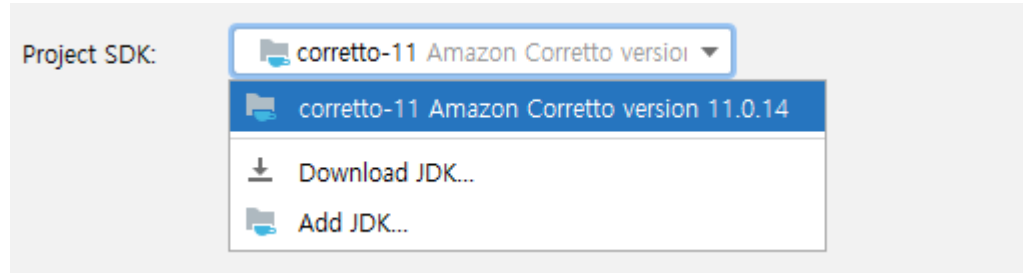
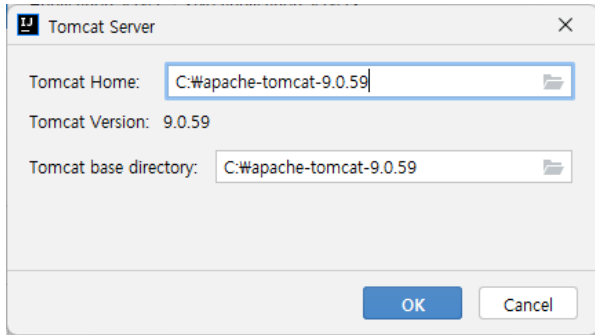
Java Enterprise 프로젝트 생성



IntelliJ 2022.2 이후 버전에서는 JakartaEE

JavaEE 8 버전으로 선택





- VM 옵션 수정

```

Output
startup.VersionLoggerListener.log 2025/10/25 14:08:00 -Dcatalina.home=C:\apache-tomcat-9
startup.VersionLoggerListener.log 2025/10/25 14:08:00 -Djava.io.tmpdir=C:\apache-tomcat-
core.AprLifecycleListener.lifecycleEvent 2025/10/25 14:08:00 [http-nio-8080]
stractProtocol.init 2025/10/25 14:08:00 [http-nio-8080]
startup.Catalina.load [380] 2025/10/25 14:08:00 [Catalina]
core.StandardService.startInternal 2025/10/25 14:08:00 [Catalina]
core.StandardEngine.startInternal 2025/10/25 14:08:00 [Apache Tomcat/9.0.59]
stractProtocol.start 2025/10/25 14:08:00 [http-nio-8080]
startup.Catalina.start 2025/10/25 14:08:00 [107]

```

```
rg.apache.catalina.startup.VersionLoggerListener.log 명령 행 아규먼트: -Dcatalina.home=C:\apache
rg.apache.catalina.startup.VersionLoggerListener.log 명령 행 아규먼트: -Djava.io.tmpdir=C:\apache
rg.apache.catalina.core.AprLifecycleListener.lifecycleEvent 프로덕션 환경들에서 최적의 성능을 제공하
rg.apache.coyote.AbstractProtocol.init 프로토콜 핸들러 ["http-nio-8080"]을(를) 초기화합니다.
rg.apache.catalina.startup.Catalina.load [339] 밀리초 내에 서버가 초기화되었습니다.
rg.apache.catalina.core.StandardService.startInternal 서비스 [Catalina]을(를) 시작합니다.
rg.apache.catalina.core.StandardEngine.startInternal 서버 엔진을 시작합니다: [Apache Tomcat/9.0.5
rg.apache.coyote.AbstractProtocol.start 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작합니다.
rg.apache.catalina.startup.Catalina.start 서버가 [115] 밀리초 내에 시작되었습니다.
```

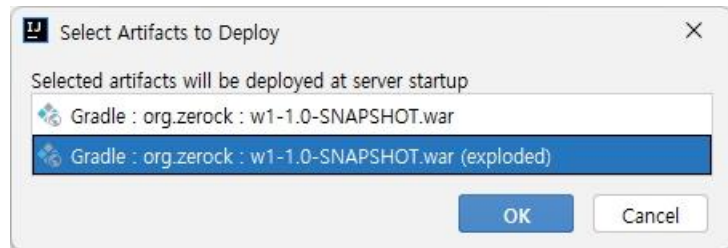
프로젝트의 경로 설정

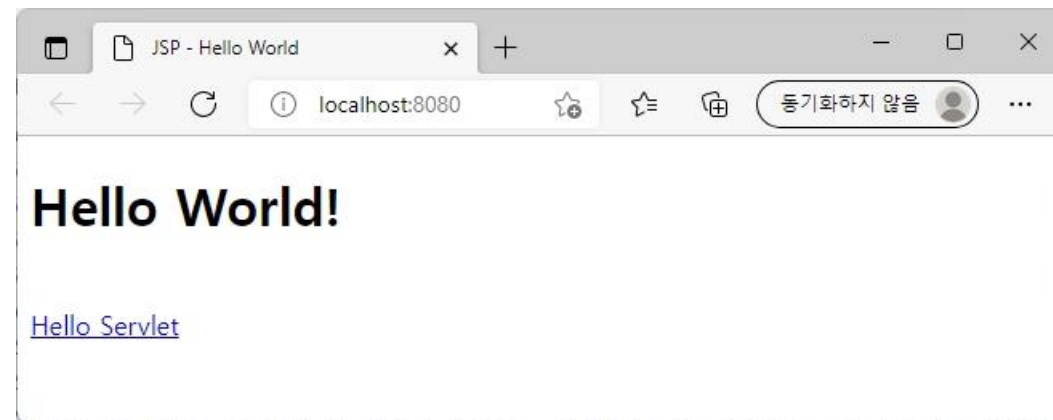
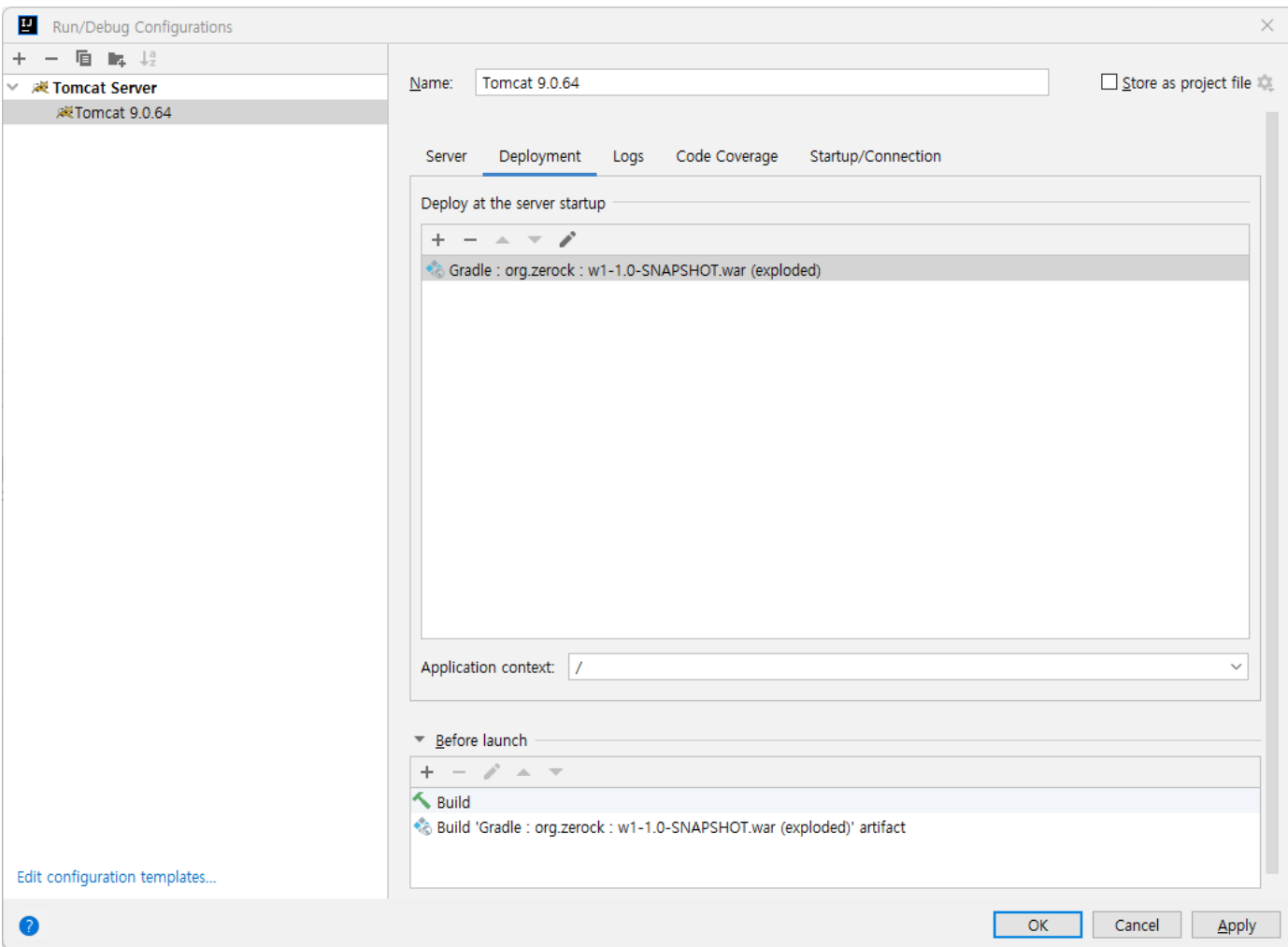
- 프로젝트의 실행 경로를 알아보기 쉽게 변경
- '/' 경로를 이용하도록 수정



Hello World!

[Hello Servlet](#)

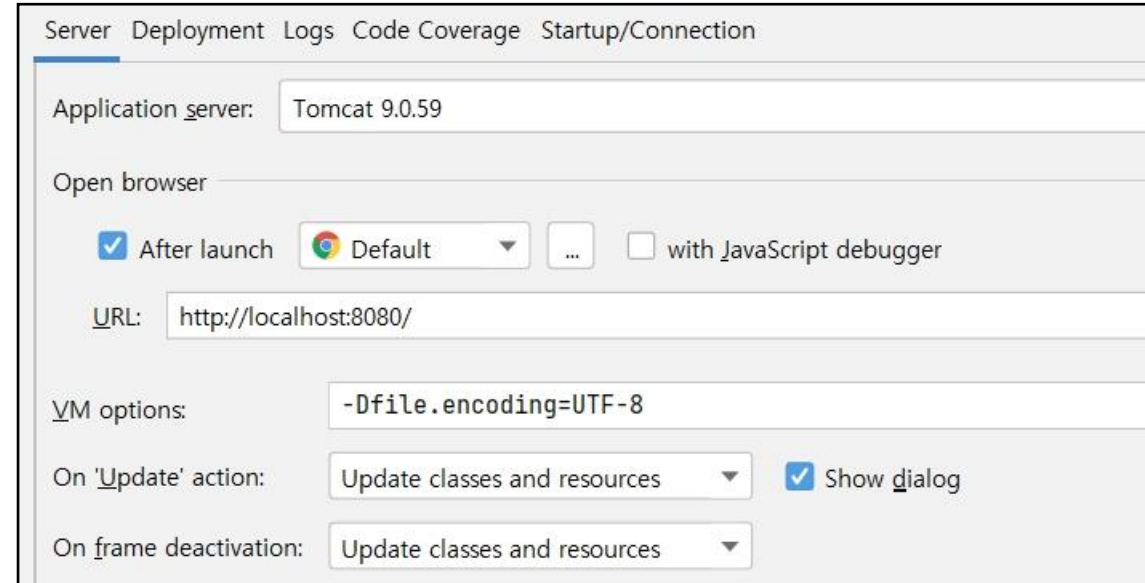




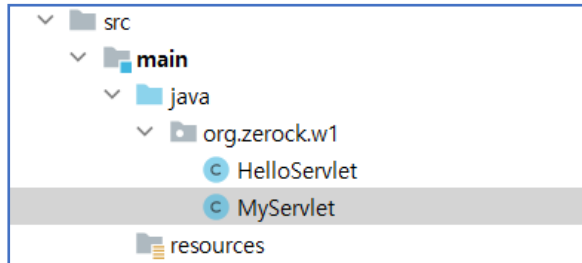
Tomcat의 자동 재시작 설정

코드의 변경 사항을 Tomcat에 자동으로 반영하도록 설정

- JSP의 경우 변경 내용 자동 반영
- 서블릿의 경우 redeploy



서블릿 코드 작성하기



```
package org.zerock.w1;

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

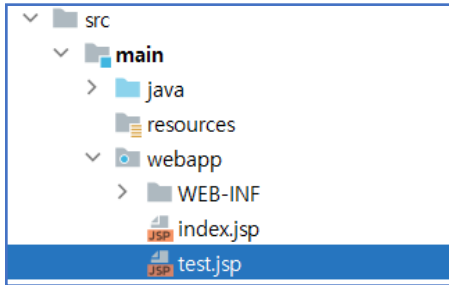
@WebServlet(name = "myServlet", urlPatterns = "/my")
public class MyServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

        resp.setContentType("text/html");

        PrintWriter out = resp.getWriter();
        out.println("<html><body>");
        out.println("<h1>MyServlet</h1>");
        out.println("</body></html>");
    }
}
```

JSP 코드 작성하기



```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
  <title>Title</title>
</head>
<body>
  <h1>Test JSP PAGE</h1>
</body>
</html>
```


1.2 웹의 기본 동작 방식 이해하기

- 브라우저와 서버와의 통신 과정
- HTTP(Hyper Text Transfer Protocol)의 의미
- GET/POST 방식의 데이터 실습

Request(요청)/Response(응답)

- 브라우저는 자신이 원하는 정보를 전달(요청)하기 위해서 주로 두 가지 방식을 이용

- GET 방식: 주소창에 직접 원하는 데이터를 적거나 링크를 클릭해서 호출
 - 원하는 웹의 주소를 호출할 때 필요한 데이터를 '?'와 '&,'를 이용해서 같이 전송하는 방식
 - 주소와 필요한 데이터를 한번에 같이 보내기 때문에 단순 링크로 처리되므로 다른 사람들에게 메신저나 SNS 등을 통해서 쉽게 공유가 가능
 - GET 방식은 특정한 정보를 조회하는 용도로 사용됩니다.
- POST 방식: 입력 화면에서 필요한 내용을 작성한 후에 '전송' 버튼 등을 클릭해서 호출
 - 주소와 데이터를 따로 보내는 방식임
 - 보통 회원 가입이나 로그인 등이 이에 해당
 - POST 방식은 웹 화면을 통해서 실제 처리가 필요한 작업을 하기 위해서 사용

HTTP 라는 약속

- 프로토콜(protocol) – 데이터 교환 약속
- 웹은 HTTP(Hyper Text Transfer Protocol)

프로토콜

호스트(도메인)

`https://www.google.com/`

The screenshot shows the Network tab of a web browser's developer tools. The top panel displays a timeline of network requests. The bottom panel shows the details of a selected request, including headers, response, and cookies.

Request List:

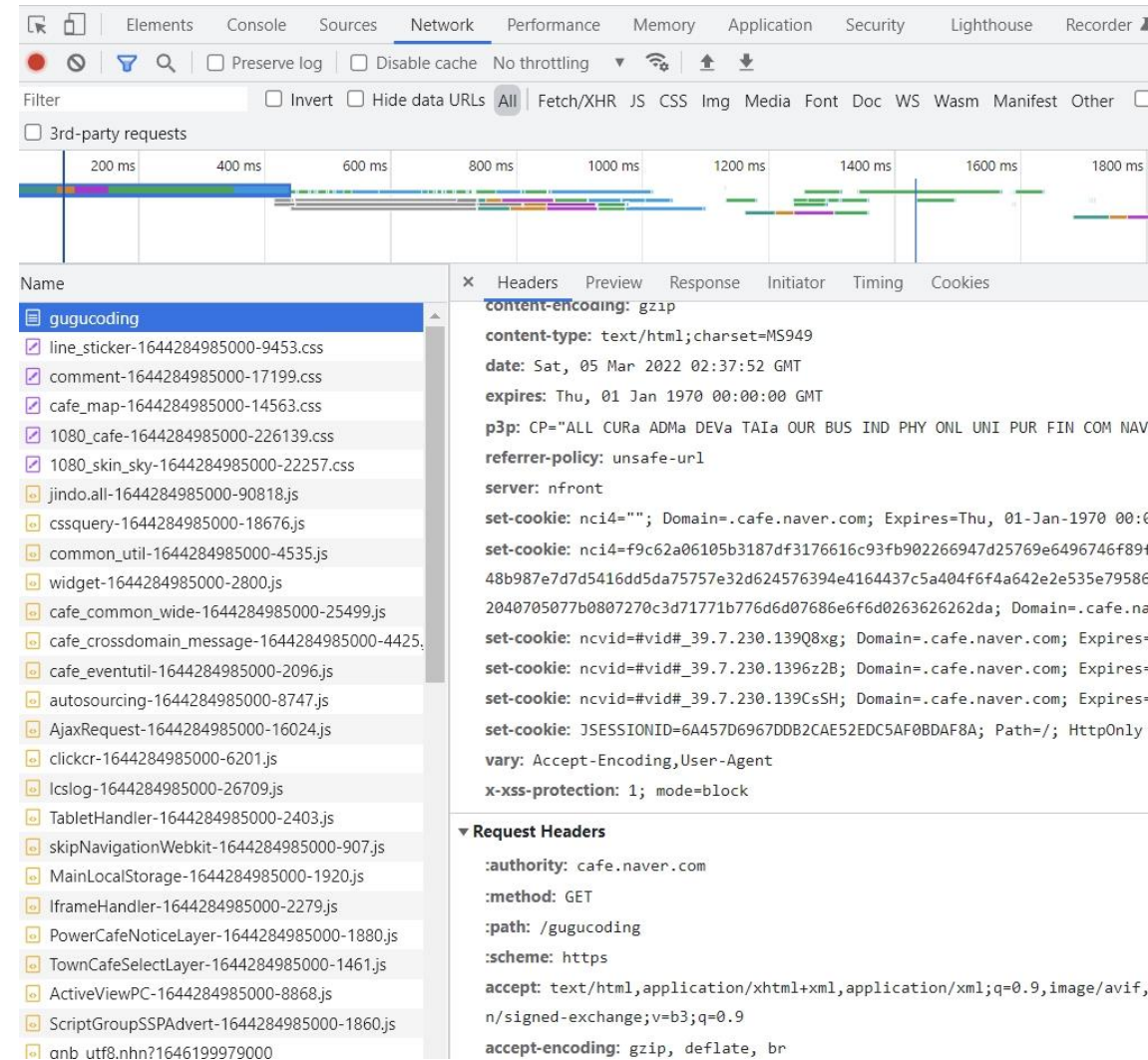
- gugucoding
- line_sticker-1644284985000-9453.css
- comment-1644284985000-17199.css
- cafe_map-1644284985000-14563.css
- 1080_cafe-1644284985000-226139.css
- 1080_skin_sky-1644284985000-22257.css
- jindo.all-1644284985000-90818.js
- cssquery-1644284985000-18676.js
- common_util-1644284985000-4535.js
- widget-1644284985000-2800.js
- cafe_common_wide-1644284985000-25499.js
- cafe_crossdomain_message-1644284985000-4425.js
- cafe_eventutil-1644284985000-2096.js
- autosourcing-1644284985000-8747.js
- AjaxRequest-1644284985000-16024.js
- clickcr-1644284985000-6201.js
- lcslog-1644284985000-26709.js
- TabletHandler-1644284985000-2403.js
- skipNavigationWebkit-1644284985000-907.js
- MainLocalStorage-1644284985000-1920.js
- IframeHandler-1644284985000-2279.js
- PowerCafeNoticeLayer-1644284985000-1880.js
- TownCafeSelectLayer-1644284985000-1461.js
- ActiveViewPC-1644284985000-8868.js
- ScriptGroupSSPAdvert-1644284985000-1860.js
- onb_utf8.nhn?1646199979000

Selected Request Details:

- Name:** gugucoding
- Headers:**
 - content-encoding: gzip
 - content-type: text/html; charset=MS949
 - date: Sat, 05 Mar 2022 02:37:52 GMT
 - expires: Thu, 01 Jan 1970 00:00:00 GMT
 - p3p: CP="ALL CURa ADMa DEVa TAIa OUR BUS IND PHY ONL PUR FIN COM NAV
 - referrer-policy: unsafe-url
 - server: nfront
 - set-cookie: nci4=""; Domain=.cafe.naver.com; Expires=Thu, 01-Jan-1970 00:00:00 GMT
 - set-cookie: nci4=f9c62a06105b3187df3176616c93fb902266947d25769e6496746f89H48b987e7d7d5416dd5da75757e32d624576394e4164437c5a404f6f4a642e535e795862040705077b0807270c3d7171b776d6d07686e6f6d0263626262da; Domain=.cafe.na
 - set-cookie: ncvid=#vid#_39.7.230.139Q8xg; Domain=.cafe.naver.com; Expires=
 - set-cookie: ncvid=#vid#_39.7.230.1396z2B; Domain=.cafe.naver.com; Expires=
 - set-cookie: ncvid=#vid#_39.7.230.139CsSH; Domain=.cafe.naver.com; Expires=
 - set-cookie: JSESSIONID=6A457D6967DDB2CAE52EDC5AF0BDAF8A; Path=/; HttpOnly
 - vary: Accept-Encoding, User-Agent
 - x-xss-protection: 1; mode=block
- Request Headers:**
 - authority: cafe.naver.com
 - method: GET
 - path: /gugucoding
 - scheme: https
 - accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,n/signed-exchange;v=b3;q=0.9
 - accept-encoding: gzip, deflate, br

Http 메시지의 구성

- 헤더(header)와 몸체(body)로 구성
- request header/request body
- reponse header/response body



The screenshot shows the Chrome DevTools Network tab. The top panel displays a timeline of network requests. The bottom panel shows the details of a selected request, including headers, preview, response, initiator, timing, and cookies.

Request List:

- gugucoding
- line_sticker-1644284985000-9453.css
- comment-1644284985000-17199.css
- cafe_map-1644284985000-14563.css
- 1080_cafe-1644284985000-226139.css
- 1080_skin_sky-1644284985000-22257.css
- jindo.all-1644284985000-90818.js
- cssquery-1644284985000-18676.js
- common_util-1644284985000-4535.js
- widget-1644284985000-2800.js
- cafe_common_wide-1644284985000-25499.js
- cafe_crossdomain_message-1644284985000-4425.js
- cafe_eventutil-1644284985000-2096.js
- autosourcing-1644284985000-8747.js
- AjaxRequest-1644284985000-16024.js
- clickcr-1644284985000-6201.js
- lcslog-1644284985000-26709.js
- TabletHandler-1644284985000-2403.js
- skipNavigationWebkit-1644284985000-907.js
- MainLocalStorage-1644284985000-1920.js
- IframeHandler-1644284985000-2279.js
- PowerCafeNoticeLayer-1644284985000-1880.js
- TownCafeSelectLayer-1644284985000-1461.js
- ActiveViewPC-1644284985000-8868.js
- ScriptGroupSSPAadvert-1644284985000-1860.js
- anb_utf8.nhn?1646199979000

Request Details (gugucoding):

- Content-Type:** text/html; charset=MS949
- Date:** Sat, 05 Mar 2022 02:37:52 GMT
- Expires:** Thu, 01 Jan 1970 00:00:00 GMT
- Cache-Control:** CP="ALL CURa ADMa DEVa TAIa OUR BUS IND PHY ONL UNI PUR FIN COM NAV
- Referrer-Policy:** unsafe-url
- Server:** nfront
- Set-Cookie:** nci4="" ; Domain=.cafe.naver.com; Expires=Thu, 01-Jan-1970 00:00:00 GMT
- Set-Cookie:** nci4=f9c62a06105b3187df3176616c93fb902266947d25769e6496746f89448b987e7d7d5416dd5da75757e32d624576394e4164437c5a404f6f4a642e2e535e795862040705077b0807270c3d71771b776d6d07686e6f6d0263626262da; Domain=.cafe.naver.com; Expires=Thu, 01-Jan-1970 00:00:00 GMT
- Set-Cookie:** ncvid=#vid#_39.7.230.139Q8xg; Domain=.cafe.naver.com; Expires=Thu, 01-Jan-1970 00:00:00 GMT
- Set-Cookie:** ncvid=#vid#_39.7.230.1396z2B; Domain=.cafe.naver.com; Expires=Thu, 01-Jan-1970 00:00:00 GMT
- Set-Cookie:** ncvid=#vid#_39.7.230.139CsSH; Domain=.cafe.naver.com; Expires=Thu, 01-Jan-1970 00:00:00 GMT
- Set-Cookie:** JSESSIONID=6A457D6967DDB2CAE52EDC5AF0BDAF8A; Path=/; HttpOnly
- Vary:** Accept-Encoding, User-Agent
- X-XSS-Protection:** 1; mode=block

Request Headers:

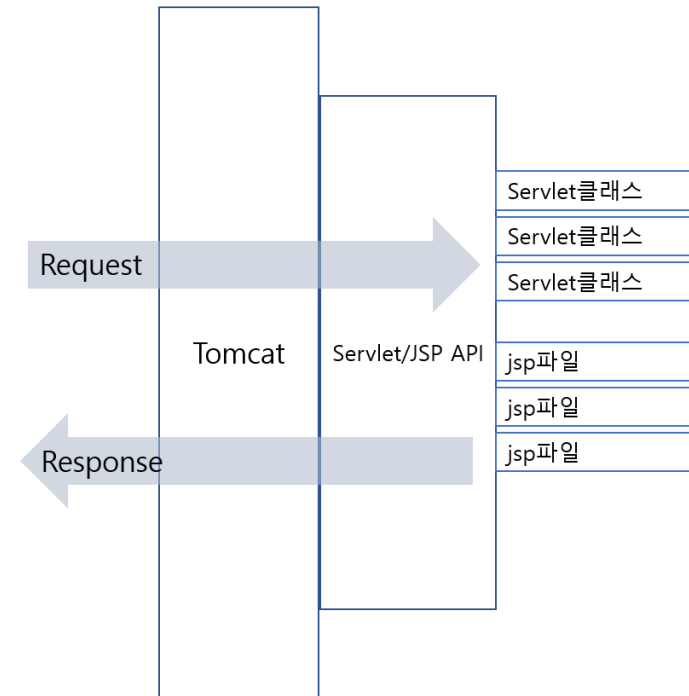
- Authority:** cafe.naver.com
- Method:** GET
- Path:** /gugucoding
- Scheme:** https
- Accept:** text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
- Accept-Encoding:** gzip, deflate, br

무상태(stateless)

- 웹은 기본적으로 지난번 상태를 기억하지 않는다.
- 모든 요청은 항상 새로운 요청일뿐
- 이를 처리하기 위한 쿠키나 세션/ Storage 방식등이 사용됨

서블릿(Servlet) 기술

- 개발자가 서버에서 처리되어야 하는 기능의 일부(조각)만 작성할 수 있도록 만들어진 JavaEE 기술 스펙의 일부



기존 JAVA 프로그래밍과 달라지는 점

- 객체를 생성하거나 호출하는 주체가 사용자가 아닌 서블릿 컨테이너 (Tomcat)가 처리
 - main()을 이용해서 프로그램을 실행하지 않는다.
- 서블릿 클래스에서 생성하는 객체의 관리 자체가 서블릿 컨테이너에 의해서 관리
 - 객체의 라이프사이클이 아닌 서블릿의 라이프사이클에 대한 이해가 필요
- 서블릿/JSP의 코드 개발은 기본적인 자바 API와 더불어 서블릿 API도 같이 사용
 - 서블릿 API를 활용해야 하므로 서블릿 관련 라이브러리의 필요

JSP 기술



- JSP는 'Java Server Pages'의 약자로 서블릿 기술과 동일하게 동적으로 서버에서 데이터를 구성하는 기술
- Servlet이나 JSP모두 동적으로 데이터를 생성해서 전송하는 것은 동일하지만 구현 방식은 다르다.
- Servlet = 'Java코드 + HTML'
- JSP = 'HTML + Java코드 '

Servlet	JSP
<pre>PrintWriter out = resp.getWriter(); out.println("<html><body>"); out.println("<h1>MyServlet</h1>"); out.println("</body> </html>");</pre>	<pre><h1><%= "Hello World!" %> </h1>
 Hello Servlet</pre>

실제로 JSP는 별도의 Java 파일로 만들어져서 실행되는 방식으로 동작

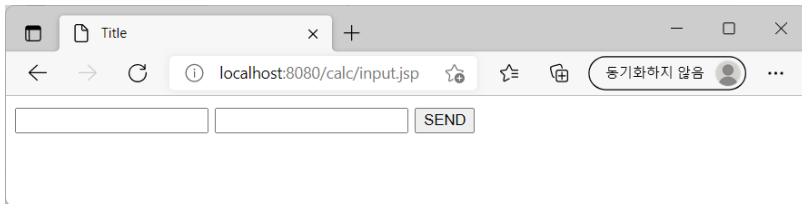
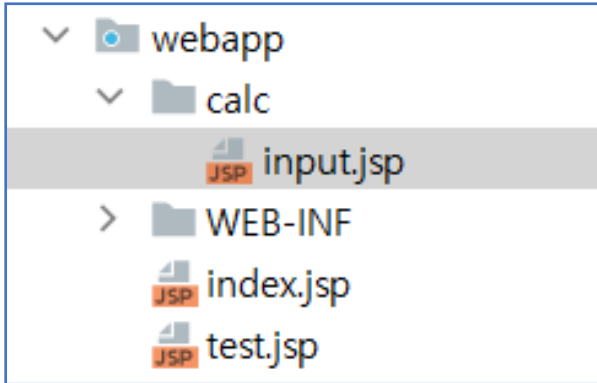
```
out.write("\n");
out.write("<!DOCTYPE html>\n");
out.write("<html>\n");
out.write("<head>\n");
out.write("    <title>JSP - Hello World</title>\n");
out.write("</head>\n");
out.write("<body>\n");
out.write("\n");
out.write("<h1>");
out.print( "Hello World!" );
out.write("</h1>\n");
out.write("\n");
out.write("<br/>\n");
out.write("<a href=\"hello-servlet\">Hello Servlet</a>\n");
out.write("</body>\n");
out.write("</html>");
```

사용자 > cooki > AppData > Local > JetBrains > IntelliJIdea2021.3 > tomcat > a3d3e22d-94e1-478a-a299-cfc0785da824 > work > Catalina > localhost > ROOT > org >

이름	수정한 날짜	유형	크기
 test.jsp.class	2022-03-05 오전 11:14	CLASS 파일	6KB
 test.jsp	2022-03-05 오전 11:14	Java 원본 파일	6KB

JSP를 이용해서 GET/POST 방식 처리하기

- GET방식은 눈에 보이는 용도
 - 링크를 통해 다른 사용자들에게 알려줄 수 있는 용도
 - 주로 입력과 조회 처리



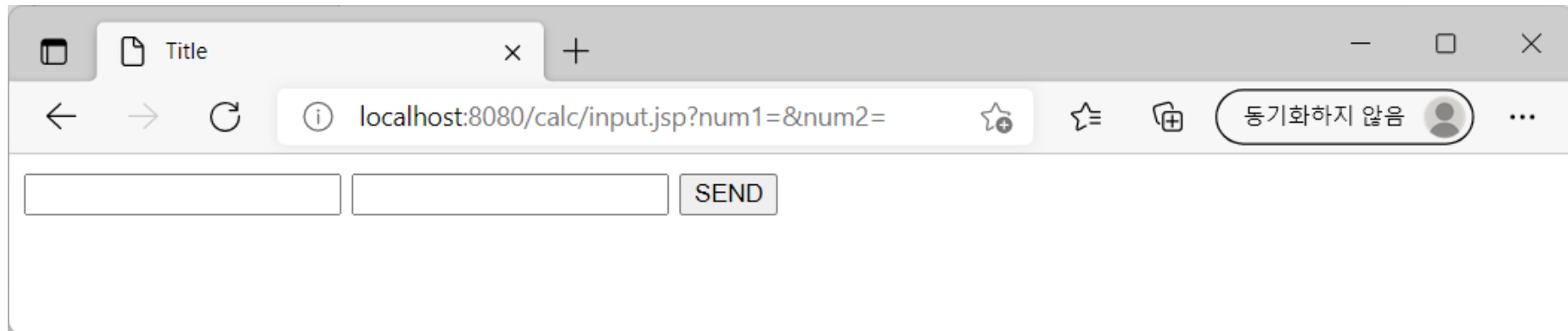
```
<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<html>
<head>
  <title>Title</title>
</head>
<body>

<form>
  <input type="number" name="num1">
  <input type="number" name="num2">
  <button type="submit">SEND</button>
</form>

</body>
</html>
```

<form> 태그와 POST 방식

- <form>태그는 '입력양식' 자체를 의미
- 한번에 여러 개의 데이터를 묶어서 전송(submit)
- <form>태그에 action 속성은 데이터를 어디로 전달할 것인지 목적지를 지정
- action 속성값이 지정되지 않으면 현재 URL 그대로 전송
- method 속성값은 데이터를 GET/POST방식으로 전송할 것인지 결정- 기본값은 GET

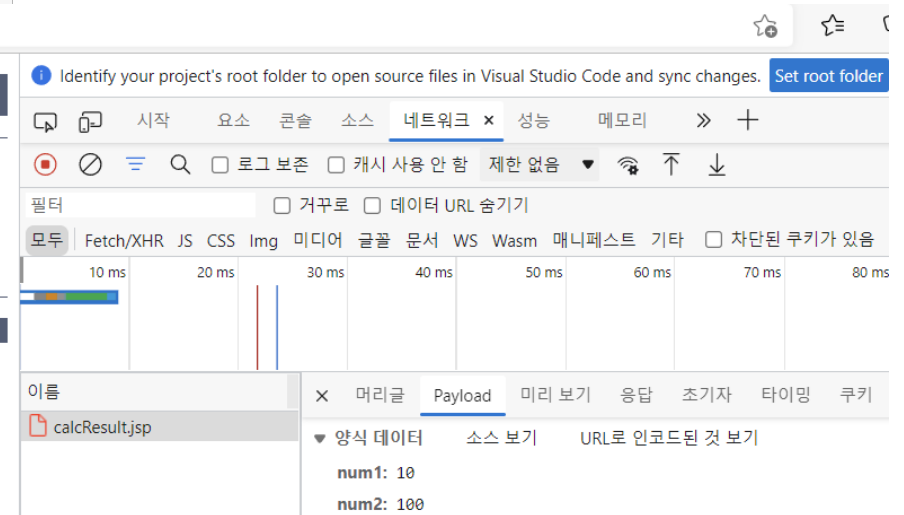
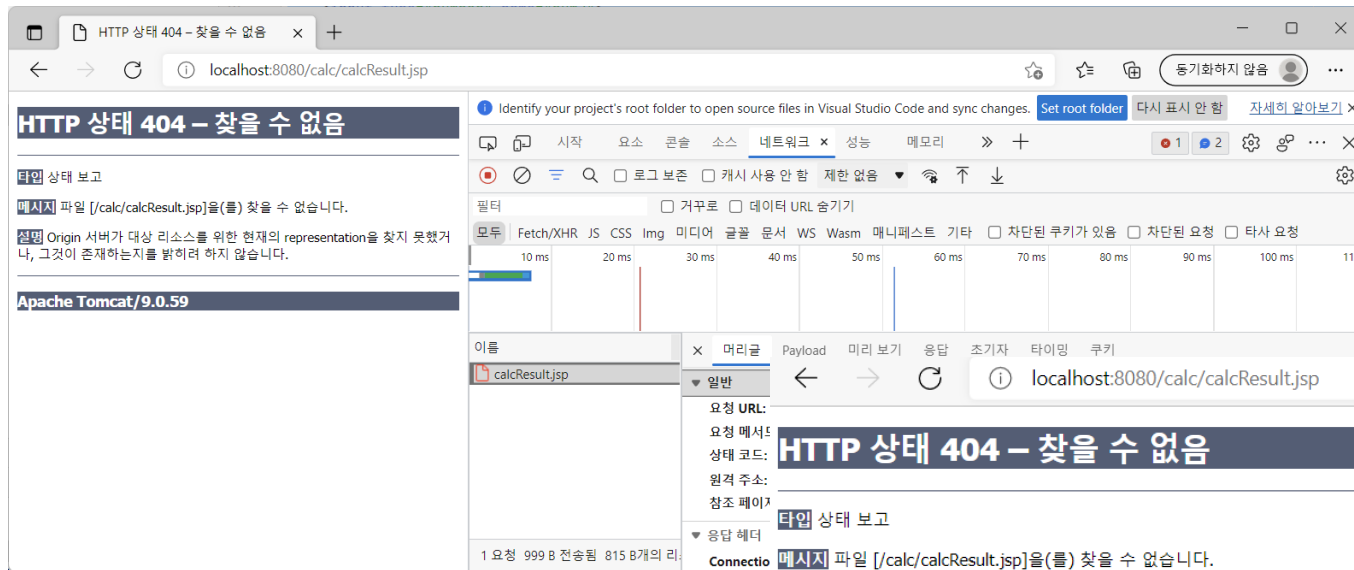


쿼리스트링(query string)

- 특정한 주소(URL)에 포함된 추가적인 데이터
- GET방식의 경우 '?'뒤에 '키(key)=값(value)'의 형태로 구성
 - 동일한 키(key)가 여러번 나올 수 있다.
 - 문자열로만 전송 가능
- POST 방식의 경우 동일한 문자열이지만 전송되는 위치가 다르다.
- 서버사이드 프로그래밍에서는 쿼리 스트링에 있는 키(key)들을 파라미터라고 한다.

개발자 도구와 쿼리스트링

- 브라우저내 개발자 도구를 이용하면 브라우저에서 보내는 데이터들을 확인할 수 있다.
- Payload 항목을 통해서 확인 가능



GET/POST 방식의 비교

	GET	POST
주용도	조회	등록/수정/삭제와 같은 처리
구성	URL뒤의 '?' 와 쿼리 스트링	URL 전달 후 HTTP 몸체(Body)로 쿼리 스트링
효과	손쉽게 남들이 사용할 수 있는 링크를 제공할 수 있음	단순 조회가 아니라 원하는 작업을 처리할 수 있게 됨
한계	브라우저에 따라 길이의 제한이 있다	GET방식에 비해서 많은 양의 데이터를 전송할 수 있음
	URL뒤의 쿼리 스트링으로 모든 정보가 전달되는 단점	주소창만으로는 테스트가 어려움
	쿼리 스트링의 길이에 대한 제한이 존재(일반적으로 2kb 혹은 브라우저마다 차이가 있음)	

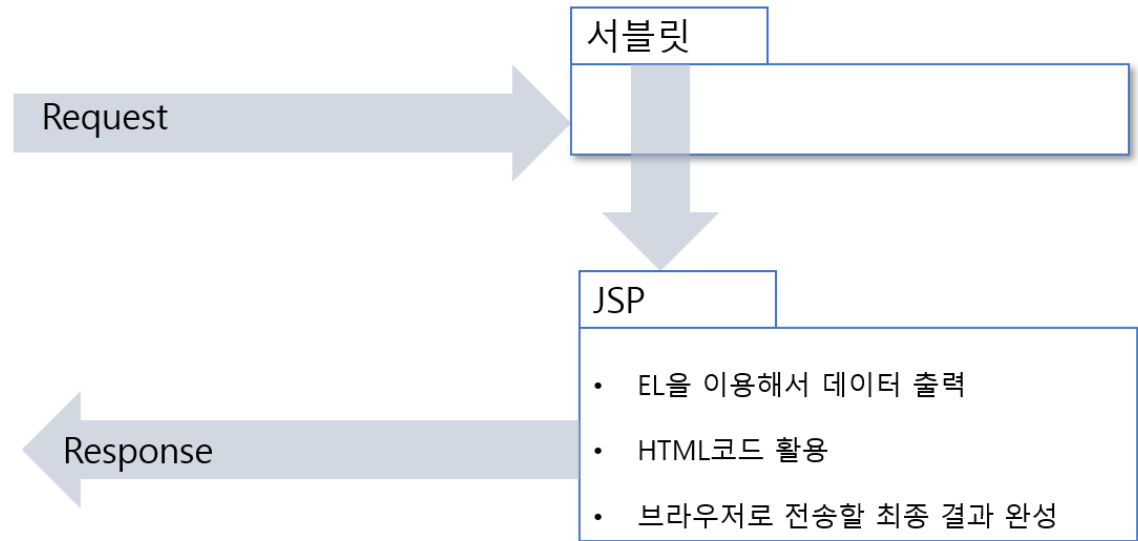
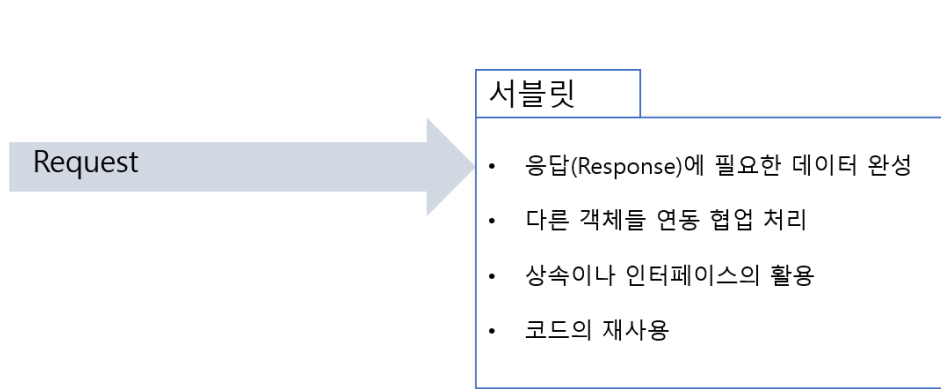
JSP의 올바른 사용법

- JSP는 기본적으로 GET/POST의 구분이 없다
 - POST 방식으로 호출하는 경우에 대해서 추가적인 코드 필요
- JSP파일의 이름이 변경되면 관련된 모든 링크를 수정해야 하는 불편함
- JSP를 제한적인 용도로 사용하기 시작
 - 쿼리스트링에 대한 처리는 Servlet에서
 - JSP는 입력화면과 조회(결과)화면으로만
 - 링크는 Servlet을 통해서 처리하고 JSP에서는 화면만 처리

WebMVC

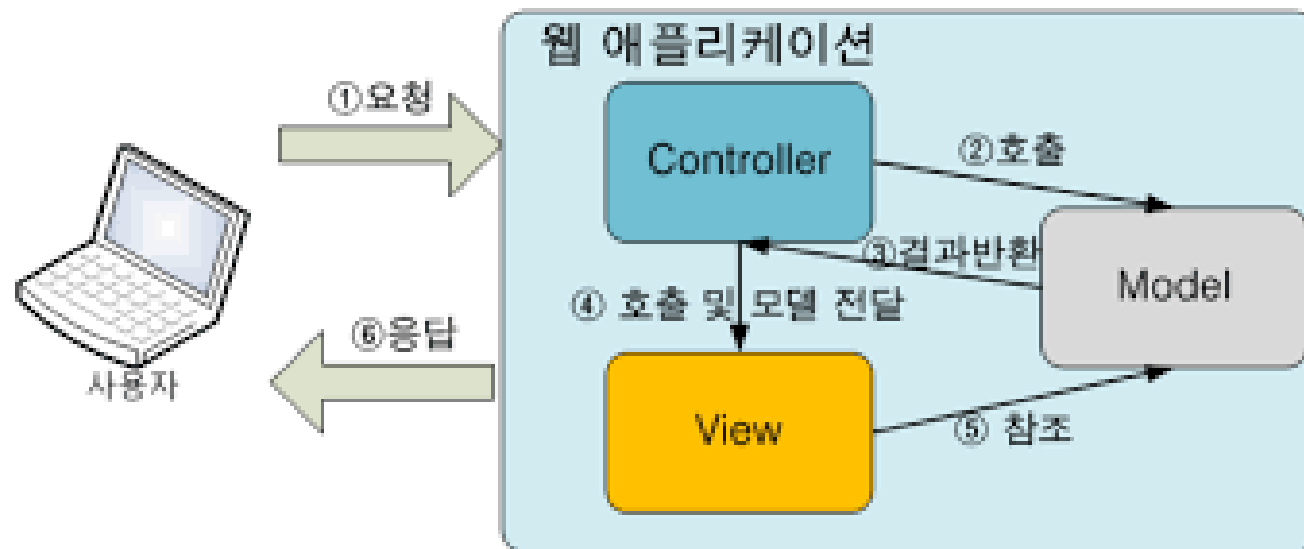
MVC 방식과 Servlet/JSP

- Servlet의 장점: Java코드를 자유롭게 사용할 수 있고, 상속이나 조합등 모든 처리가 가능
- JSP의 장점: 복잡한 화면 구성을 위한 처리에 유용



Web MVC

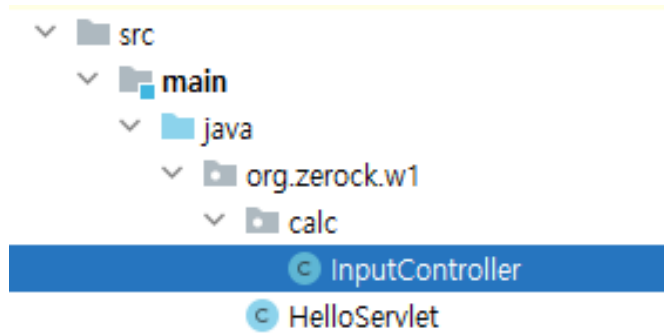
- 기존 어플리케이션에서 사용하던 MVC(model-view-controller) 패턴을 웹으로 이식
- Request를 처리하는 Controller는 서버릿으로
- 화면은 JSP로



Web MVC의 원칙

- 브라우저의 모든 호출은 반드시 컨트롤러를 향한다.
 - 서블릿은 내부적으로 화면을 결정하므로 나중에 URL이 변경될 일이 없다는 장점
- JSP는 컨트롤러를 통해서만 호출되며 컨트롤러에서 만들어진 데이터(Model)를 보여주는 용도로만 사용

계산기 실습 - 입력화면



```
package org.zerock.w1.calc;

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

@WebServlet(name = "inputController" , urlPatterns = "/calc/input")
public class InputController extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {

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

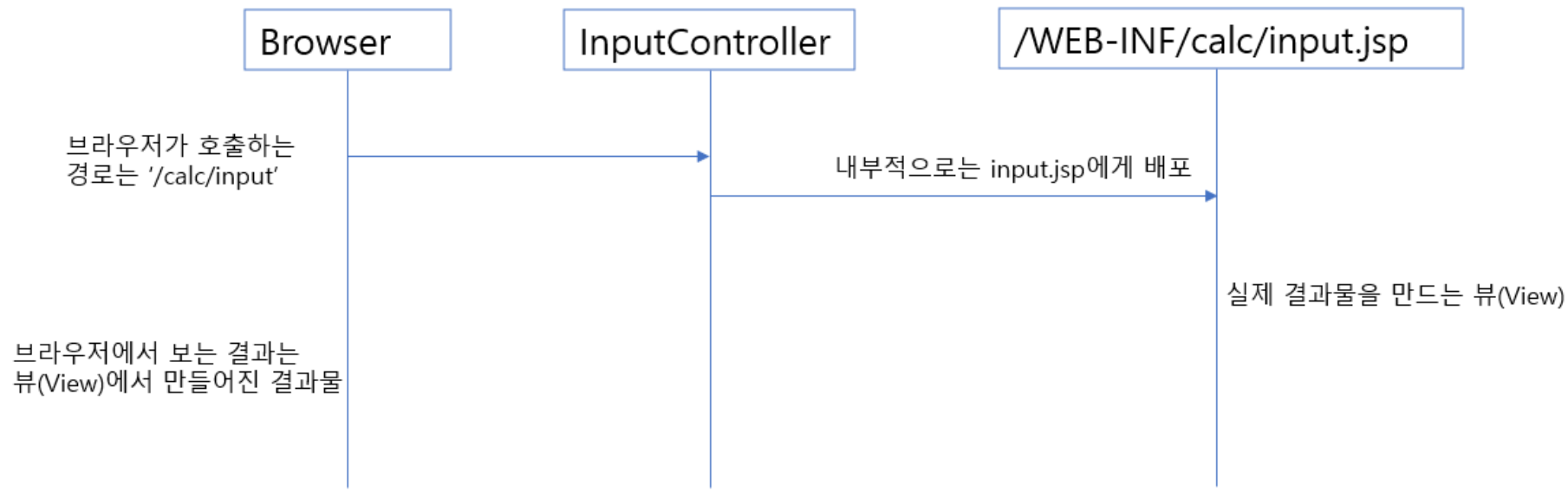
        RequestDispatcher dispatcher = req.getRequestDispatcher("/WEB-INF/calc/input.jsp");

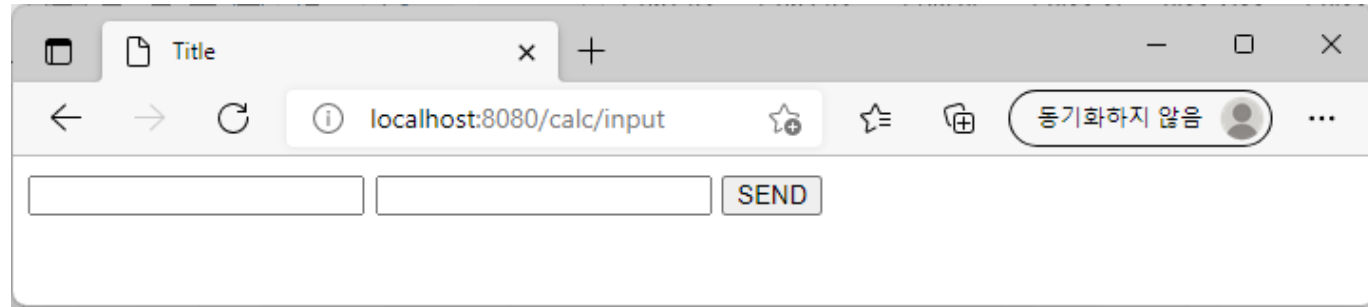
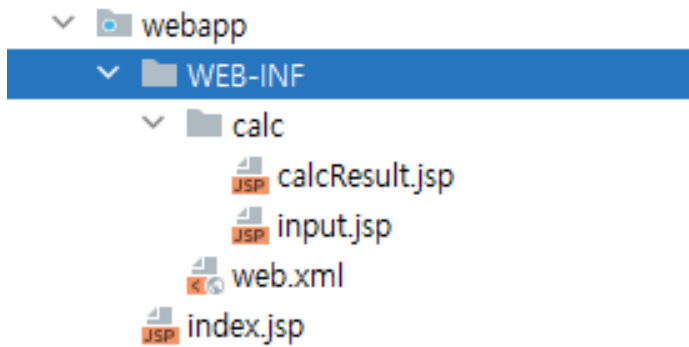
        dispatcher.forward(req, resp);

    }
}
```

RequestDispatcher

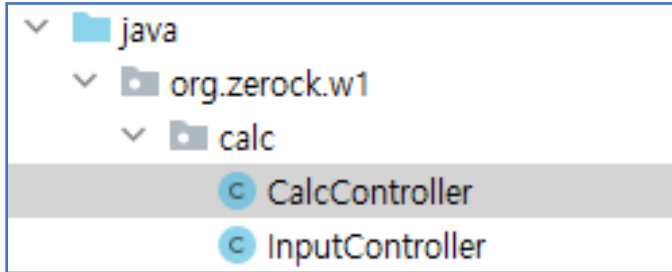
- 컨트롤러(Servlet)에서 요청(Request)에 대한 처리를 배포(dispatch) 하기 위해서 사용
- 컨트롤러에서는 JSP로 forward()를 이용





- *브라우저는 화면의 결과가 Servlet에서 나온 것인지 JSP에서 나온 것인지 알 수 없다는 점 – 서버의 내부에서 Servlet ->JSP를 거쳐서 나온 결과라는 점
- JSP파일은 WEB-INF 폴더 내에 있으므로 브라우저는 반드시 컨트롤러를 통해서만 JSP 결과를 볼 수 있다는 점
- 컨트롤러 역할을 하는 서블릿에서는 GET/POST에 따라서 doGet()/doPost()를 나누어서 처리할 수 있다는 점

POST방식을 통한 처리



```
package org.zerock.w1.calc;

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

@WebServlet(name = "calcController" , urlPatterns = "/calc/makeResult")
public class CalcController extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
    IOException {

        String num1 = req.getParameter("num1");
        String num2 = req.getParameter("num2");

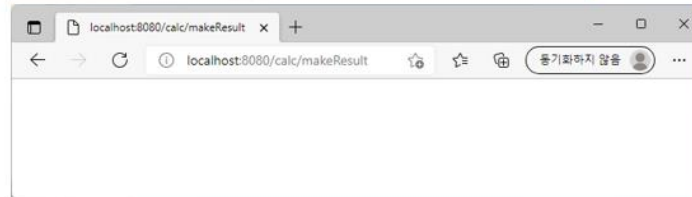
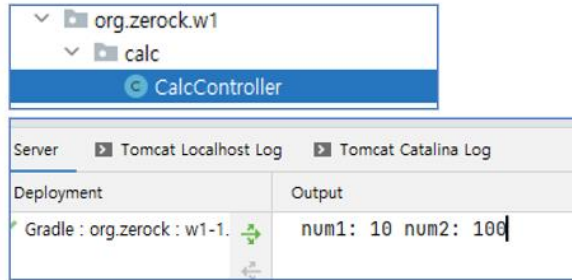
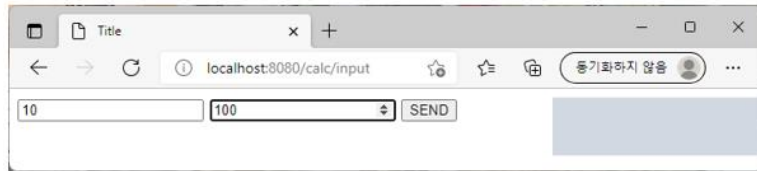
        System.out.printf(" num1: %s", num1);
        System.out.printf(" num2: %s", num2);

    }
}
```

- urlPatterns 속성값이 '/calc/makeResult'로 지정되어 있음 - 브라우저에서 <form> 태그의 submit경로를 수정할 필요가 있습니다.
- doPost()를 오버라이드 - 브라우저에서 POST방식으로 호출하는 경우에만 호출이 가능하게 됩니다.
- req.getParameter()라는 메서드를 이용해서 쿼리 스트링으로 전달되는 num1, num2 파라미터를 처리하고 있으며 이 때 숫자가 아닌 문자열(String)로 처리하고 있습니다. JSP에서는 \${param.num1}과 같이 단순하게 사용하지만 서블릿에서는 HttpServletRequest라는 API를 이용해야만 합니다.

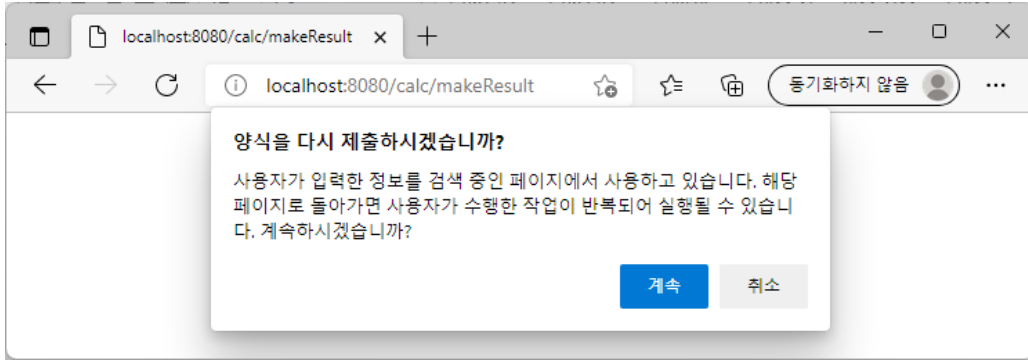
input.jsp의 수정

```
<form action="/calc/makeResult" method="post">
  <input type="number" name="num1">
  <input type="number" name="num2">
  <button type="submit">SEND</button>
</form>
```



sendRedirect()

- POST방식 처리후 브라우저의 새로고침 문제
- POST방식의 처리후에 브라우저가 다른 경로로 이동하도록 지정
- 사용자로 하여금 처리후 reset



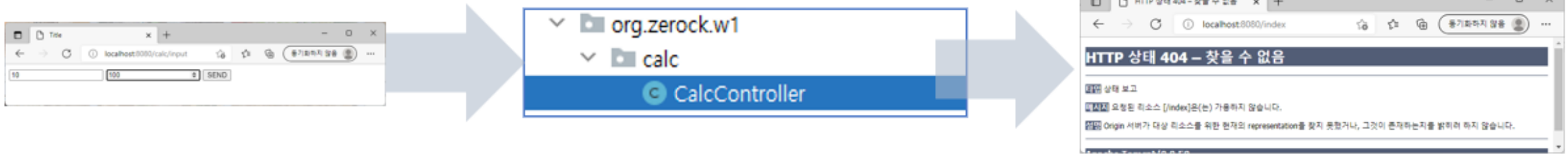
```
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {

    String num1 = req.getParameter("num1");
    String num2 = req.getParameter("num2");

    System.out.printf(" num1: %s", num1);
    System.out.printf(" num2: %s", num2);

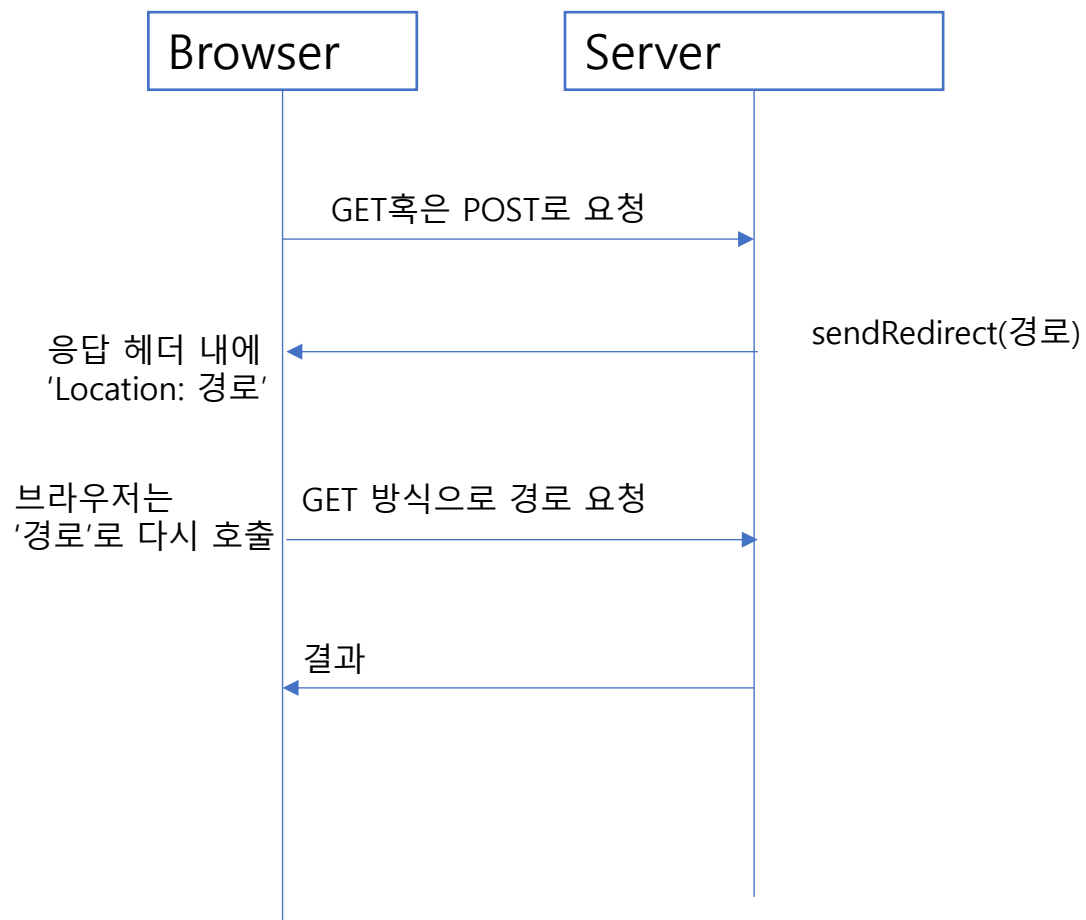
    resp.sendRedirect("/index");

}
```



PRG 패턴(POST-Redirect-GET)

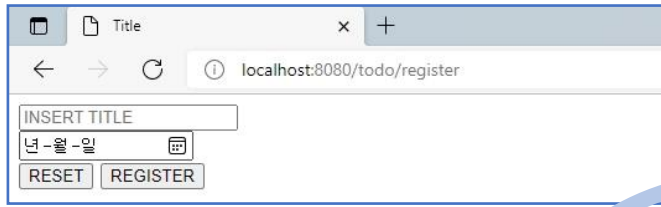
- 사용자의 POST방식으로 컨트롤러에 원하는 작업을 처리하기를 요청
- POST방식을 컨트롤러에서 처리하고 브라우저는 다른 페이지로 이동(GET)하라는 응답
- 브라우저의 GET방식이동



Todo 와이어프레임 작성

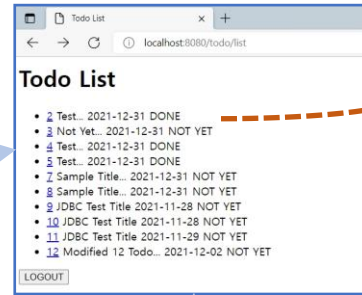
- PRG패턴을 이용해서 Todo를 대상으로 프로그램이 어떻게 동작해야 하는지 설계
- 와이어프레임 - 인터페이스를 시각적으로 작성

/todo/register (GET)



/todo/register (POST) /todo/list (Redirect)

TodoRegisterController
doPost()



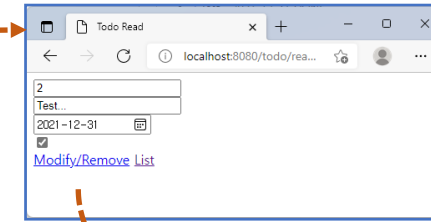
/todo/list (Redirect)

/todo/list (Redirect)

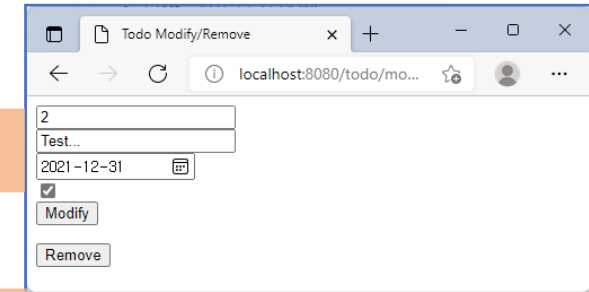
TodoModifyController
doPost()

TodoRemoveController
doPost()

/todo/read (GET)



/todo/modify (GET)



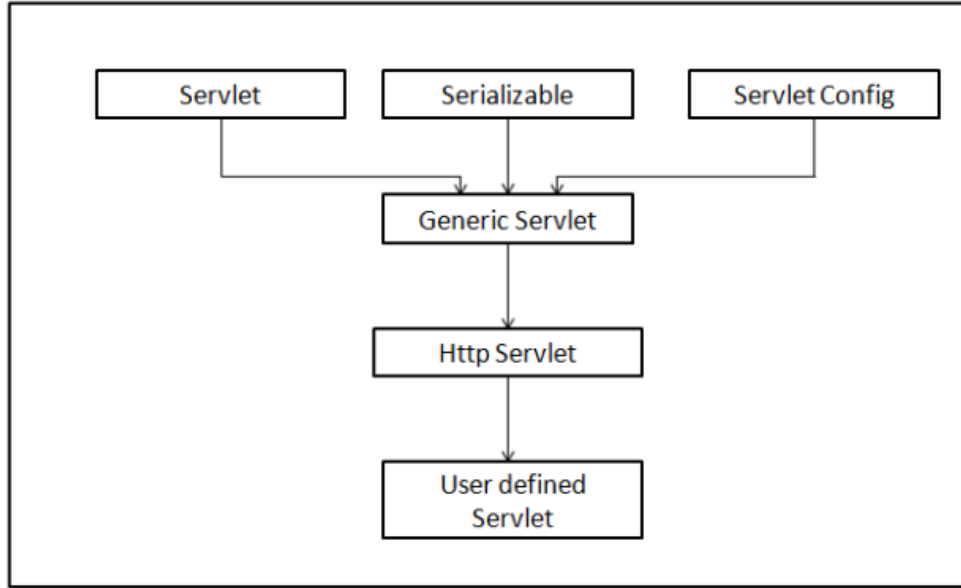
구현 목록의 정리

기능	동작 방식	컨트롤러(org.zerock.w1.todo)	JSP
목록	GET	TodoListController	WEB-INF/todo/list.jsp
등록(입력)	GET	TodoRegisterController	WEB-INF/todo/register.jsp
등록(처리)	POST	TodoRegisterController	Redirect
조회	GET	TodoReadController	WEB-INF/todo/read.jsp
수정(입력)	GET	TodoModifyController	WEB-INF/todo/modify.jsp
수정(처리)	POST	TodoModifyController	Redirect
삭제(처리)	POST	TodoRemoveController	Redirect

HttpServlet

- HttpServlet은 GET/POST 등에 맞게 doGet(), doPost()등을 제공하므로, 개발자들은 본인에게 필요한 메서드를 오버라이드 하는것 만으로 GET/POST 방식 처리를 나누어서 처리할 수 있습니다.
- HttpServlet을 상속받은 클래스의 객체는 Tomcat과 같은 WAS의 내부에서 자동으로 객체를 생성되고, 관리되기 때문에 개발자가 객체의 관리에 신경쓸 필요가 없습니다.
- HttpServlet의 동작시에는 동시에 여러 쓰레드에 의해서 실행될 수 있도록 처리되기 때문에 개발자는 동시에 많은 사용자들을 어떻게 처리해야 하는지에 대한 고민을 줄일 수 있습니다.

Servlet 상속 구조

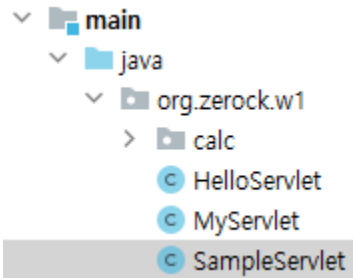


- 프로토콜에 관계없이 처리할 수 있는 GenericServlet
- HTTP에 특화된 HttpServlet
- 웹 개발시에는 주로 HttpServlet을 상속해서 사용

HttpServlet의 라이프사이클

• HttpServlet의 처리 과정

- 1) 브라우저가 Tomcat에 특정한 서블릿이 처리해야 하는 경로를 호출합니다.
- 2) Tomcat은 해당 경로에 맞는 서블릿 클래스를 로딩하고 객체를 생성합니다. 이 과정에서 `init()`라는 이름의 메서드를 실행해서 서블릿 객체가 동작하기 전에 수행해야 하는 일들을 처리할 수 있습니다(최초 필요한 시점에 서블릿 클래스를 로딩하는 대신에 Tomcat실행시에 로딩하도록 하는 `load-on-startup`이라는 옵션도 있습니다.).
- 3) 생성된 서블릿 객체는 브라우저의 요청(Request)에 대한 정보를 분석해서 GET/POST등의 정보와 함께 같이 전달되는 파라미터(쿼리스트링의 내용)들을 `HttpServletRequest`라는 타입의 파라미터로 전달받습니다. 이 과정에서 응답을 처리하는데 필요한 기능들은 `HttpServletResponse`라는 타입의 객체로 전달받습니다.
- 4) 서블릿의 내부에서는 GET/POST에 맞게 `doGet()`/`doPost()`등의 메서드를 실행합니다. 이 후 동일한 주소의 호출이 있는 경우 Servlet은 동일한 객체 하나만을 이용해서 이를 처리합니다.
- 5) Tomcat이 종료되는 경우에는 Servlet의 `destroy()`라는 메서드를 실행됩니다.



```
package org.zerock.w1;

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

@WebServlet(name = "sampleServlet", urlPatterns = "/sample")
public class SampleServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        System.out.println("doGet...." + this);
    }

    @Override
    public void destroy() {
        System.out.println("destory.....");
    }

    @Override
    public void init(ServletConfig config) throws ServletException {
        System.out.println("init(ServletConfig).....");
    }
}
```

HttpServletRequest

- HttpServletRequest는 HTTP 메시지의 형태로 들어오는 요청(Request)에 대한 정보를 파악하기 위해서 제공

기능	메서드	설명
HTTP헤더 관련	getHeaderNames() getHeader(이름)	HTTP헤더 내용들을 찾아내는 기능
사용자 관련	getRemoteAddress()	접속한 사용자의 IP주소
요청 관련	getMethod() getRequestURL() getRequestURI() getServletPath()	GET/POST 정보, 사용자가 호출에 사용한 URL정보 등
쿼리 스트링 관련	getParameter() getParameterValues() getParameterNames()	쿼리 스트링 등으로 전달되는 데이터를 추출하는 용도
쿠키 관련	getCookies()	브라우저가 전송한 쿠키 정보
전달 관련	getRequestDispatcher()	
데이터 저장	setAttribute()	전달하기 전에 필요한 데이터를 저장하는 경우에 사용

HttpServletResponse

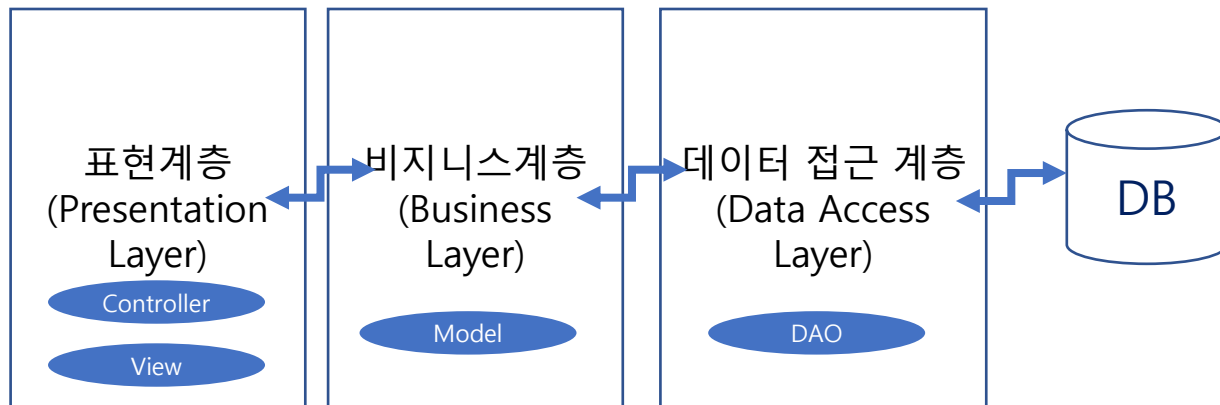
- 브라우저로 전송하기 위해서 데이터를 만들어내는데 필요한 기능들을 제공

기능	메서드	설명
MIME타입	setContentType()	응답 데이터의 종류를 지정(이미지/html/xml 등)
헤더 관련	setHeader()	특정 이름의 HTTP 헤더 지정
상태 관련	setStatus()	404, 200, 500등 응답 상태 코드 지정
출력 관련	getWriter()	PrintWriter를 이용해서 응답 메시지 작성
쿠키 관련	addCookie()	응답시에 특정 쿠키 추가
전달 관련	sendRedirect()	브라우저에 이동을 지시

Model과 컨트롤러

모델과 서비스 계층

- 컨트롤러에서는 화면에 필요한 데이터를 화면쪽으로 전달해주는데 이런 데이터들을 담당하는 객체를 모델(Model)이라고 함
- JSP로 전달된 모델은 EL등을 이용해서 처리
- JSP에서는 EL과 JSTL을 이용



3-tier 구조

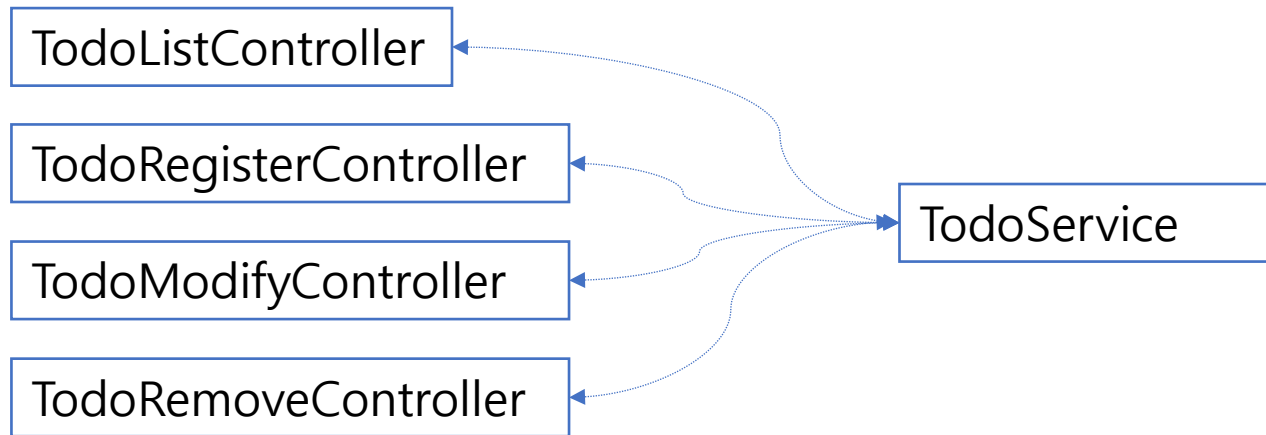
- 표현 계층: 실제 화면 처리를 담당하는 객체들로 이루어진 계층으로 앱의 화면이나 GUI화면의 화면들, 웹 MVC등이 이에 해당
- 서비스(비즈니스) 계층: 고객의 요구사항을 반영하고 각 기능들에 대한 API를 제공하는 계층. 모든 기능에 대한 입력과 출력을 정의한 객체들로 구성
- 영속 계층: 오직 데이터의 관리만을 목적으로 하며 데이터베이스나 네트워크, 파일 처리등의 연결을 담당하는 객체들로 구성

계층의 데이터 전달을 위한 DTO

- Data Transfer Object의 약어
- 여러 개의 데이터를 묶어서 하나의 단위 객체로 구성하기 위해서 주로 사용
- 서비스나 컨트롤러의 경우 오랜 시간 유지되는데 비해 DTO는 아주 짧은 생명주기
- DTO의 구성 규칙 – Java Beans 규칙 준수
 - 최소한의 규칙
 - 생성자가 없거나 반드시 파라미터가 없는 생성자 함수를 가지는 형태
 - 속성(멤버 변수)은 private으로 작성
 - getter/setter를 제공할 것

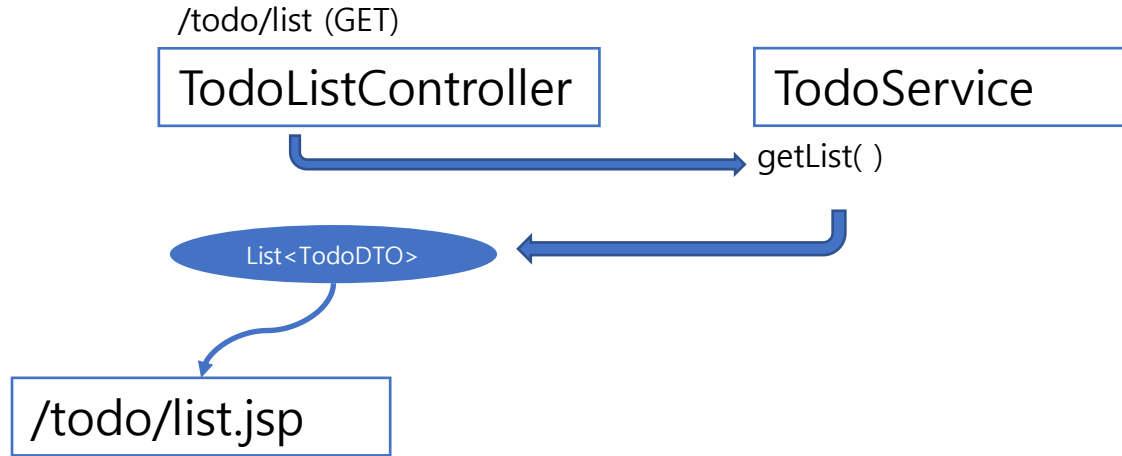
서비스 객체

- 실제 필요한 기능의 구현은 서비스 객체를 구성해서 처리
- 서비스 객체는 POJO(Plain Old Java Object)로 구성
 - 특별한 API에 종속적이지 않도록 구성
- Web MVC 구조에서는 여러 개의 컨트롤러들이 하나의 서비스 객체를 사용하는 방식



컨트롤러에서 모델 처리하기

- 목록 화면의 흐름



JSP와 EL

- 초기의 JSP에서는 <% %>등을 이용해서 화면에서 Java 코드를 이용하는 방식으로 출력
- JSP2.0부터 EL(Expression Language)의 등장
- 간단한 표현식으로 데이터 출력(내부적으로 getter등을 활용)
- EL은 출력만을 담당하므로 반복문과 같은 구문(statement)처리를 위해 JSTL사용
- JSTL(Java Standard Tag Library)

> test

build.gradle

gradlew

gradlew.bat

```
dependencies {  
    compileOnly('javax.servlet:javax.servlet-api:4.0.1')  
  
    testImplementation("org.junit.jupiter:junit-jupiter-api:${junitVersion}")  
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junitVersion}")  
  
    implementation group: 'jstl', name: 'jstl', version: '1.2'  
}
```

JSP에서 JSTL사용하기

- 페이지 상단에 JSTL 설정

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

- <c:forEach>

- var – EL에서 사용될 변수이름
- items – List, Set, Map, Enumeration, Iterator 등의 컬렉션
- begin – 값을 읽어올 인덱스 시작값
- end – 값을 읽어올 인덱스 끝 값

```
<ul>  
  <c:forEach var="dto" items="${list}">  
    <li>${dto}</li>  
  </c:forEach>  
</ul>
```

- `<c:if>` `<c:choose>`
 - if ~ else를 처리하기 위해서 사용

```
<c:if test="${list.size() % 2 == 0}">
    짝수
</c:if>
<c:if test="${list.size() % 2 != 0}">
    홀수
</c:if>
```

```
<c:choose>
    <c:when test="${list.size() % 2 == 0}">
        짝수
    </c:when>
    <c:otherwise>
        홀수
    </c:otherwise>
</c:choose>
```

- `<c:set>`
 - 추가적인 변수를 선언하기 위해서 사용

```
<c:set var="target" value="5"></c:set>

<ul>
    <c:forEach var="num" begin="1" end="10">
        <c:if test="${num == target}">
            num is target
        </c:if>
    </c:forEach>
</ul>
```

