

스프링 부트 웹 프로젝트

chapter04

서블릿과 스프링MVC

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

Contents

part.1

HTTP 프로토콜

part.2

웹 기술 발전 과정

part.3

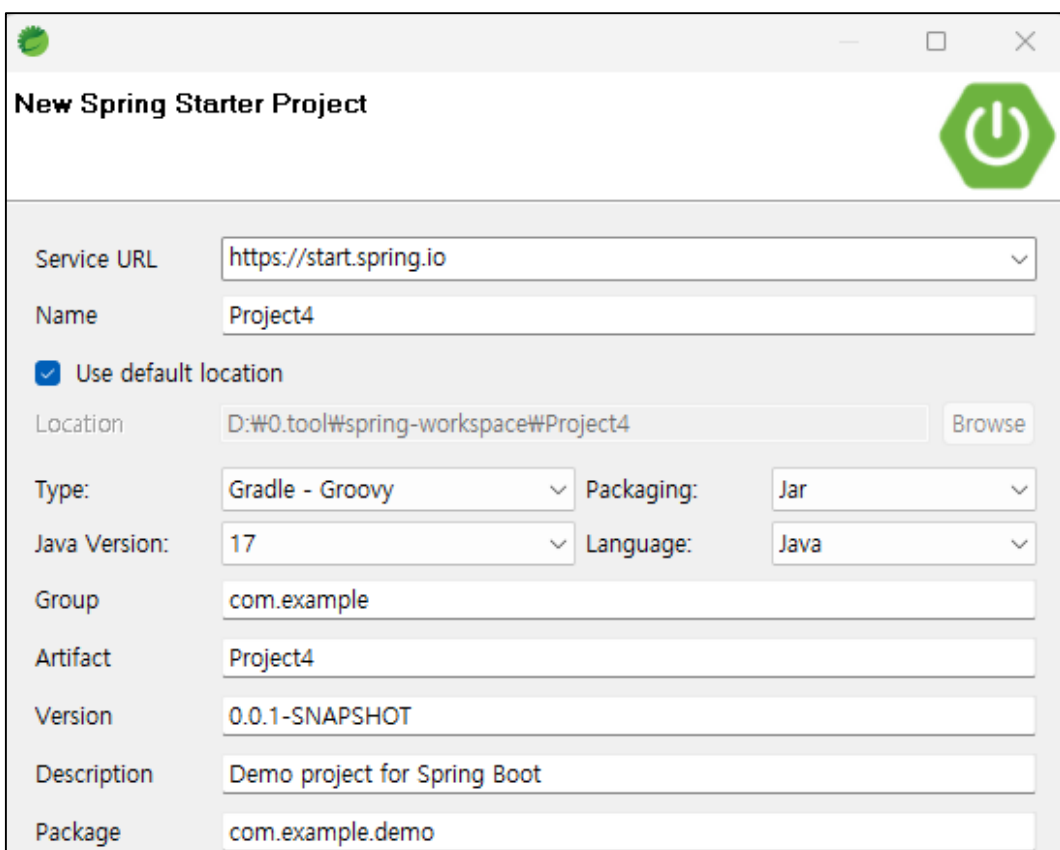
서블릿/JSP

part.4

모델2와 스프링MVC

실습을 위해 프로젝트를 생성한다.

라이브러리는 'Lombok, Spring Web'를 선택한다.

A screenshot of the 'New Spring Starter Project' dialog box. It contains various input fields for project configuration. The 'Service URL' is set to 'https://start.spring.io'. The 'Name' is 'Project4'. The 'Location' is 'D:\#0.tool\#spring-workspace\#Project4'. The 'Type' is 'Gradle - Groovy', 'Packaging' is 'Jar', 'Java Version' is '17', and 'Language' is 'Java'. The 'Group' is 'com.example', 'Artifact' is 'Project4', 'Version' is '0.0.1-SNAPSHOT', 'Description' is 'Demo project for Spring Boot', and 'Package' is 'com.example.demo'. There is a 'Browse' button next to the location field.

New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

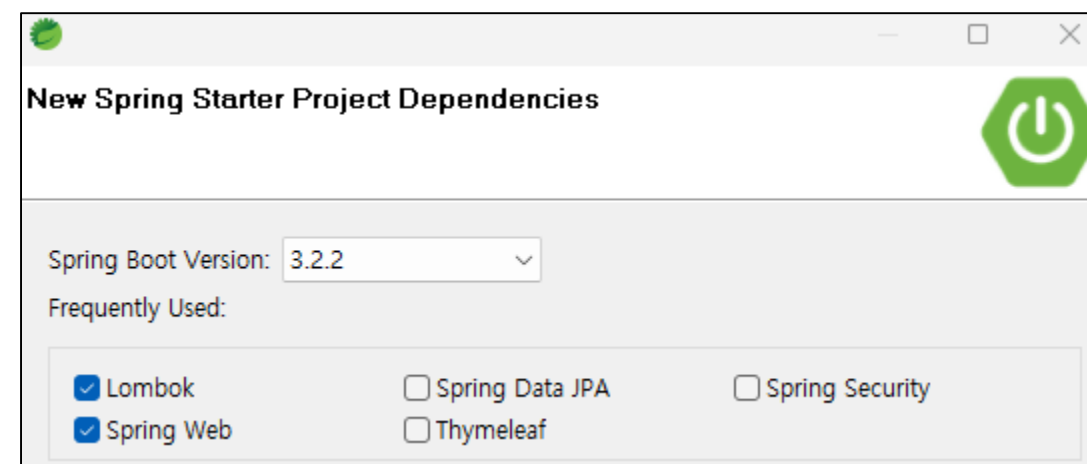
Group:

Artifact:

Version:

Description:

Package:

A screenshot of the 'New Spring Starter Project Dependencies' dialog box. It shows the 'Spring Boot Version' as '3.2.2'. Under 'Frequently Used', 'Lombok' and 'Spring Web' are checked, while 'Spring Data JPA', 'Thymeleaf', and 'Spring Security' are unchecked.

New Spring Starter Project Dependencies

Spring Boot Version:

Frequently Used:

<input checked="" type="checkbox"/> Lombok	<input type="checkbox"/> Spring Data JPA	<input type="checkbox"/> Spring Security
<input checked="" type="checkbox"/> Spring Web	<input type="checkbox"/> Thymeleaf	

HTTP 프로토콜은 인터넷에서 서버와 클라이언트 간에 데이터를 주고 받기 위한 통신 규약이다.
클라이언트가 서버에게 콘텐츠를 요청하면 HTTP 요청을 생성하고, 서버는 그 요청에 대한 응답을 반환한다.
이를 통해 웹 페이지, 파일 등 다양한 데이터를 주고 받을 수 있다.



HTTP 메시지는 HTTP 메시지는 브라우저와 서버 간에 데이터를 주고 받기 위한 형식이다.
HTTP 메시지는 몇 줄의 텍스트로 이루어져 있으며, 요청(request) 메시지와 응답(response) 메시지로 나뉜다.

Request 메시지

Header : 요청 메소드(GET, POST 등), 요청 URL, 파라미터, 데이터 형식(Content-Type)

Body : 전송할 데이터(폼 데이터, JSON, XML)

Response 메시지

Header : HTTP 상태 코드(예: 200, 404), 응답 데이터의 형식(Content-Type)

Body : 서버가 반환할 응답 데이터 (예: HTML, JSON)

(a) Request message		(b) Response message	
GET /test/hi-there.txt HTTP/1.0	Start line	HTTP/1.0 200 OK	
Accept: text/* Accept-Language: en,fr	Headers	Content-type: text/plain Content-length: 19	
	Body	Hi! I'm a message!	

URL이란?

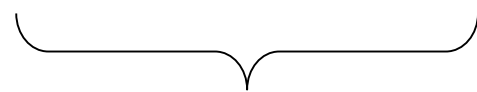
사용자는 'URL' 주소를 통해 서버에 요청을 보낼 수 있다.

URL은 기본 주소, 포트 번호, 상세 주소, 파라미터로 구성된다.

상세 주소는 요청의 내용에 따라 달라진다.

파라미터는 추가적인 정보를 서버에 전달하는 역할을 한다.

www.test.com:8080/get/board?no=5



기본주소



포트



상세주소



파라미터

자바 웹 기술은 시간에 따라 다음과 같이 발전해 왔다.

서블릿 (Servlet): HTTP 요청과 응답을 처리하는 기본적인 웹 기술.

JSP (JavaServer Pages): HTML과 자바 코드를 결합하여 동적인 웹 페이지를 생성하는 기술.

MVC 패턴 (Model-View-Controller): 비즈니스 로직, 프레젠테이션 로직, 사용자 입력 처리를 분리한 아키텍처 패턴.

프론트 컨트롤러 패턴: 모든 요청을 하나의 중앙 컨트롤러에서 처리하는 패턴.

스프링 MVC: 프론트 컨트롤러 패턴을 기반으로 한 웹 프레임워크.

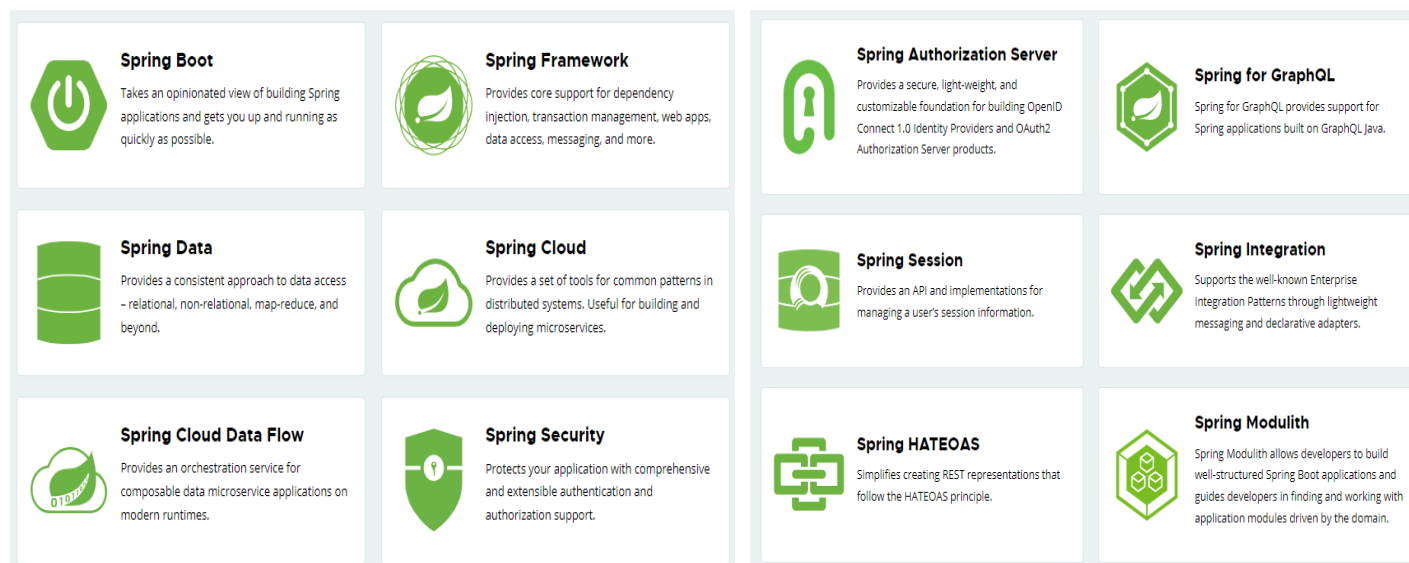
서블릿 → JSP → MVC패턴 → 프론트 컨트롤러 패턴 → 스프링MVC

웹 프로그램을 구현하는 방법은 다양하다.

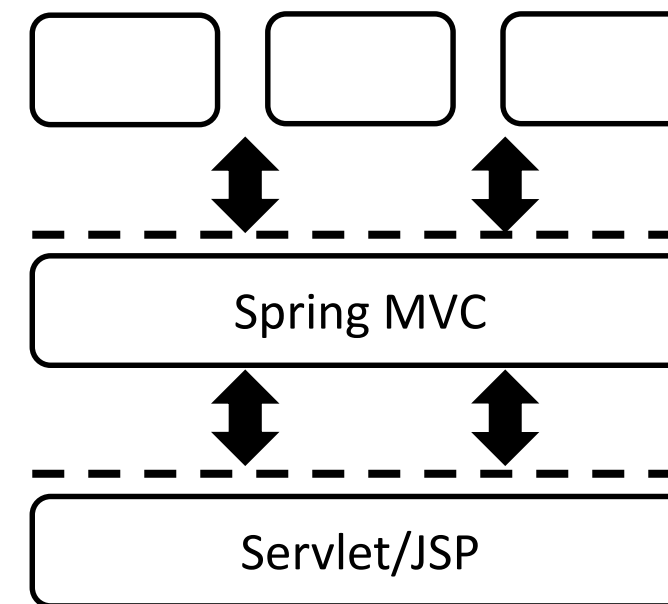
전통적인 서블릿/JSP 방식부터 스프링 MVC와 같은 최신 프레임워크까지 선택하여 사용할 수 있다.

최근에는 대부분 스프링의 서브 프로젝트인 스프링 MVC를 주로 사용한다.

하지만 스프링 MVC는 내부적으로는 서블릿을 기반으로 동작하므로, 스프링 MVC를 제대로 이해하려면 서블릿을 먼저 이해해야 한다.



스프링의 프레임워크들



스프링 MVC 구조

웹 기술 발전 과정 순수 자바 클래스로 회원 리파지토리 만들기

Member
MemberRepository

스프링 MVC 프레임워크 없이, 순수하게 자바클래스를 사용하여 사용자 요청을 처리할 준비를 한다.

데이터 저장을 위한 모델 객체와 리파지토리를 순수한 자바 클래스로 작성한다.

가상 리파지토리는 테이블 대신 Map을 사용하여 회원 정보를 저장하고, 조회, 수정, 삭제 기능을 처리한다.

Member (모델 역할)

```
public class Member {  
  
    int no;  
  
    String userId; // 회원아이디  
  
    String password; // 비밀번호  
  
}
```

MemberRepository (저장소 역할)

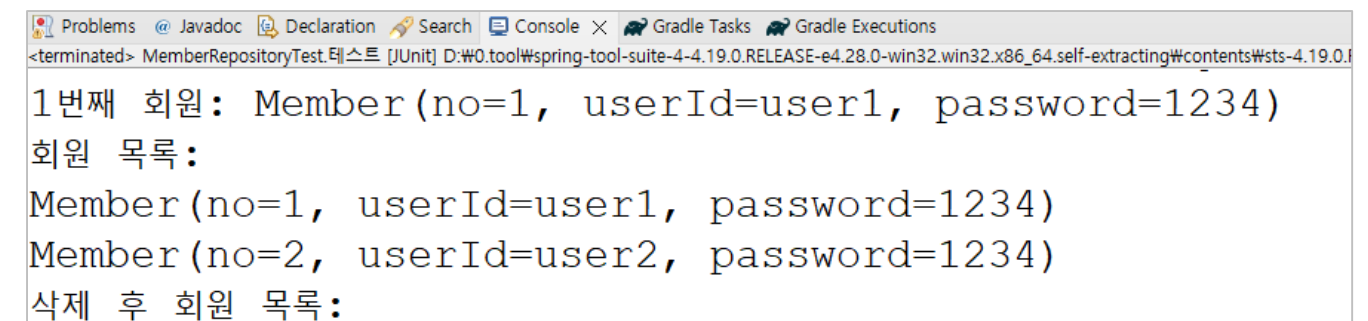
```
public class MemberRepository {  
  
    static Map<Integer, Member> store = new HashMap<>();  
    static int sequence = 0;  
  
    public Member save(Member member) {  
        member.setNo(++sequence);  
        store.put(member.getNo(), member);  
        return member;  
    }  
  
    public Member findById(Long id) {  
        return store.get(id);  
    }  
  
    public List<Member> findAll() {  
        return new ArrayList<>(store.values());  
    }  
  
    public void clearStore() {  
        store.clear();  
    }  
}
```

Q. MemberRepository의 기능을 테스트하기 위해 "MemberRepositoryTest" 클래스를 작성하세요.
MemberRepository를 사용하여 회원 데이터를 추가, 조회, 삭제하는 기능을 테스트하세요.
테스트는 메모리 저장소(Map)에 저장되므로, 모든 테스트를 하나의 메소드에서 순차적으로 진행해 주세요.

테스트 클래스

```
public class MemberRepositoryTest {  
  
    // 스프링을 사용하지 않았기 때문에 직접 리파지토리 객체 생성해야함  
    MemberRepository repository = new MemberRepository();  
  
    @Test  
    public void 테스트 () {  
  
        // 1.회원 등록 (2명)  
  
        // 2.회원 목록 조회  
  
        // 3.1번째 회원 조회  
  
        // 4.모든 회원 삭제
```

결과 화면



```
<terminated> MemberRepositoryTest.테스트 [JUnit] D:\#0.tool\spring-tool-suite-4-4.19.0.RELEASE-e4.28.0-win32.win32.x86_64.self-extracting\contents\sts-4.19.0.1  
1번째 회원: Member(no=1, userId=user1, password=1234)  
회원 목록:  
Member(no=1, userId=user1, password=1234)  
Member(no=2, userId=user2, password=1234)  
삭제 후 회원 목록:
```

서블릿/JSP

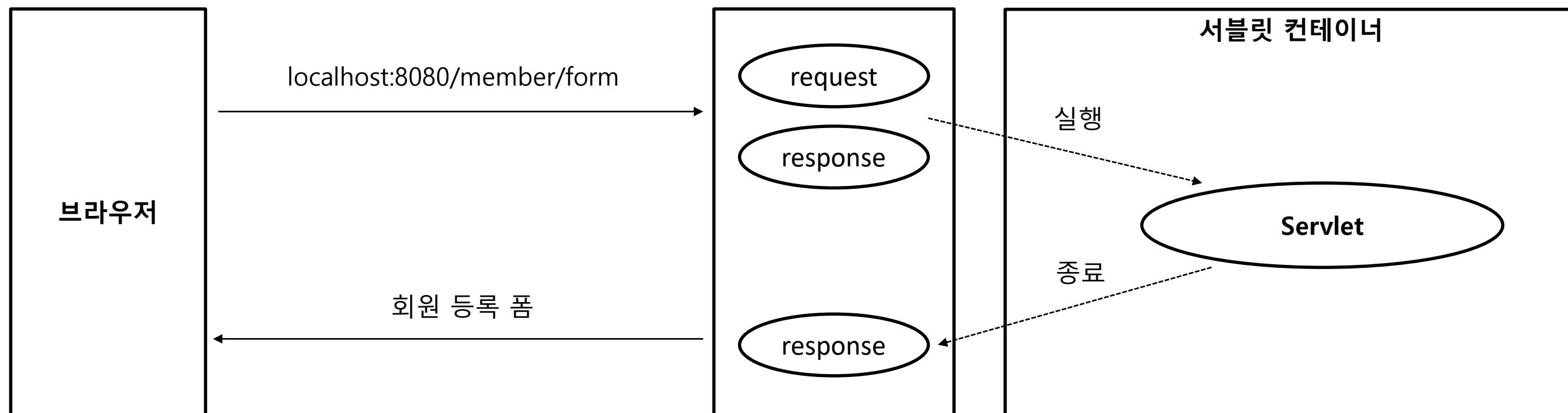
서블릿이란?

서블릿은 사용자 요청을 처리하고, 응답을 생성하는 클래스이다.

사용자 요청이 들어오면 서블릿이 자동으로 request와 response 객체를 생성한다.

request 객체를 통해 사용자의 요청 정보를 분석하고, response 객체를 통해 응답을 생성한다.

이 과정에서 HTTP 메시지가 자동으로 생성되어 사용자에게 전송된다.



객체	설명
@WebServlet	서블릿의 이름, URL 등 매핑정보를 설정한다.
HttpServletRequest 클래스	사용자가 보낸 요청 메시지를 나타내는 객체이다. 예) 파라미터
HttpServletResponse 클래스	사용자에게 보낼 응답 메시지를 나타내는 객체이다. 예) 콘텐츠 타입, 상태코드, 결과데이터

화면을 통해 회원 정보를 입력 받아 등록하는 기능을 구현하기 위해, 서블릿을 사용하여 회원 관리 프로그램을 작성한다.
이를 위해 회원정보를 입력 받는 서블릿과 저장하는 서블릿을 구현한다.

서블릿 구현 단계

1. HTTP 요청을 처리하기 위해 HttpServlet 클래스를 상속받는다.
2. request 객체를 사용하여 사용자의 요청 정보를 분석한다. (폼데이터 또는 URL 파라미터)
3. response 객체를 사용하여 응답을 생성하고 사용자에게 보낸다.

```
@WebServlet(name = "FormServlet", urlPatterns = "/servlet/form")
public class FormServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");

        PrintWriter w = response.getWriter();

        w.write("<!DOCTYPE html>\n" +
            "<html>\n" +
            "<head>\n" +
            "  <meta charset=\"UTF-8\">\n" +
            "  <title>Title</title>\n" +
            "</head>\n" +
            "<body>\n" +
```

회원 폼을 반환하는 서블릿

```
@WebServlet(name = "SaveServlet", urlPatterns = "/servlet/save")
public class SaveServlet extends HttpServlet {

    MemberRepository repository = new MemberRepository();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

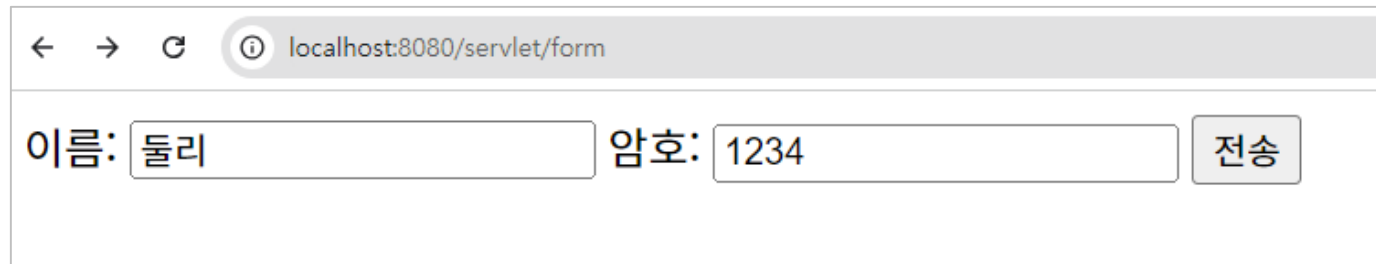
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        Member member = new Member(0, username, password);
        repository.save(member);
        request.setAttribute("member", member);

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

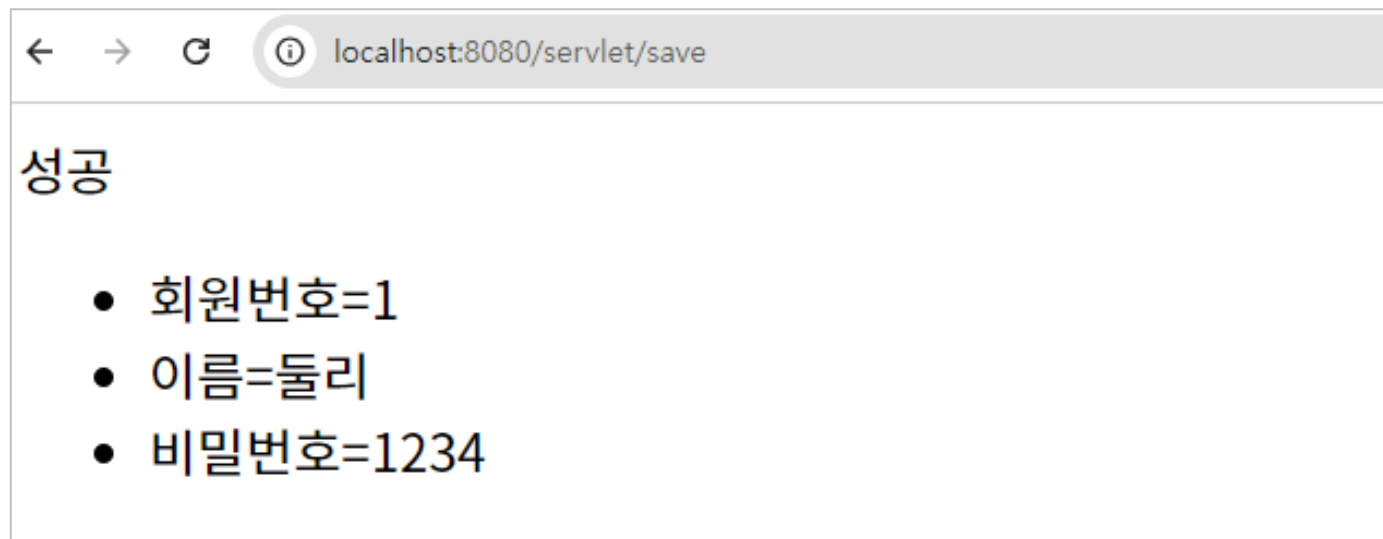
회원 정보를 저장하는 서블릿

1. 브라우저에서 `http://localhost:8080/servlet/form` 주소를 호출하여 회원 등록 폼을 실행한다.
2. 폼에서 이름과 암호를 입력한 후, 전송 버튼을 클릭한다.
3. 다음 페이지에서 새로운 회원 정보가 성공적으로 등록되었는지 확인한다.



← → ↻ ⓘ localhost:8080/servlet/form

이름: 암호:



← → ↻ ⓘ localhost:8080/servlet/save

성공

- 회원번호=1
- 이름=둘리
- 비밀번호=1234

서블릿/JSP

서블릿의 한계와 템플릿 엔진의 필요성

이전 예제에서 보았듯이, 자바 코드로 HTML 작성하는 것은 매우 복잡한 작업이다.

줄바꿈과 따옴표는 자바 코드에서 특수 기호로 처리해야 하기 때문에 코드가 더 복잡해진다.

이 문제를 해결하기 위해 나온 것이 **템플릿 엔진**이다.

템플릿 엔진을 사용하면 HTML 문서에서 필요한 부분에만 자바 코드를 삽입할 수 있어 코드가 훨씬 간단해진다.

대표적인 템플릿 엔진으로 JSP, Thymeleaf, Freemarker 등이 있다.

자바 코드 내에서 HTML 작성

```
response.setContentType("text/html");
response.setCharacterEncoding("utf-8");
PrintWriter w = response.getWriter();

w.write("<!DOCTYPE html>\n" +
"<html>\n" +
"<head>\n" +
" <meta charset=\"UTF-8\">\n" +
" <title>Title</title>\n" +
"</head>\n" +
"<body>\n" +
```

HTML

템플릿 엔진을 사용하여 HTML 내에서 자바 코드 작성

```
<body>
  <p>
    <!-- 자바 객체 출력 -->
    <%= newMember.getNo() %> 번째
  </p>
</body>
```

```
<!-- 자바 객체 출력 -->
<%= newMember.getNo() %> 번째
```

자바코드

템플릿 엔진 중에서 가장 먼저 나온 기술은 1999년에 발표된 **JSP**이다.

JSP는 HTML 코드에 자바 코드를 넣어서 동적인 웹 페이지를 생성하는 기술이다.

JSP를 실행하면 서블릿으로 변환되며, 이를 통해 사용자 요청을 처리하고 동적인 페이지를 사용자에게 전송할 수 있다.

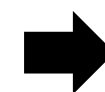
JSP 구조

HTML

Java

JSP 코드와 결과 화면

```
<html>
<body>
<h1><c:out value="${변수}" /></h1>
</body>
</html>
```



동적인페이지입니다

JSP 파일에서는 자바 코드를 삽입하여 동적으로 데이터를 처리하고 출력할 수 있다.

JSP에서 사용하는 문법은 다음과 같다.

`<% %>` : 이 태그 안에 자바 코드를 작성할 수 있으며, 로직을 처리할 때 사용한다.

`<%= %>` : 이 태그는 변수 값이나 메소드 결과를 바로 출력할 때 사용한다.

JSP 예제

```
<%
    MemberRepository repository = new MemberRepository();

    String username = request.getParameter("username");
    String password = request.getParameter("password");

    Member member = Member.builder()
        .userId(username).password(password)
        .build();

    Member newMember = repository.save(member);
%>
```

```
<p>
    <%= newMember.getNo() %> 번째 <%= newMember.getUserId() %> 회원을 추가했습니다!
</p>
```

JSP 라이브러리 추가

1. jsp 파일을 사용하기 위해 jsp 라이브러리를 추가한다.
2. gradle 설정파일을 수정한 후에 'gradle refresh' 를 실행 한다.

build.gradle

```
implementation 'org.apache.tomcat.embed:tomcat-embed-jasper'  
implementation 'jakarta.servlet:jakarta.servlet-api'  
implementation 'jakarta.servlet.jsp.jstl:jakarta.servlet.jsp.jstl-api'  
implementation 'org.glassfish.web:jakarta.servlet.jsp.jstl'
```

이 내용을 복사해서
넣어주세요!

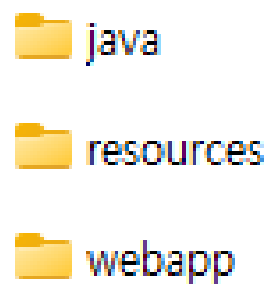
JSP 폴더 추가

프로젝트 디렉토리에서 src/main 폴더 아래에 webapp 폴더를 생성한다.
webapp 폴더 안에 view 폴더를 생성하고, 그 안에 v1 폴더를 순차적으로 생성한다.

뷰 기본 경로 설정

application.properties 파일에 JSP 파일의 경로와 확장자를 설정한다.
아래와 설정하면 JSP 경로가 기본값으로 설정이 된다.
기본 경로는 /src/main/webapp/view/ 이다.

JSP 폴더 위치



application.properties 파일

```
spring.mvc.view.prefix=/view/  
spring.mvc.view.suffix=.jsp
```

실습 파일의 내용을
복사해서 넣어주세요!

이번에는 이전에 작성했던 프로그램을 JSP로 다시 구현해본다.

회원 등록 폼 JSP는 대부분 HTML로 작성하고, 첫번째 줄에는 JSP 파일임을 표시한다.

JSP는 실행 시 서블릿으로 변환되며, 이전에 만들었던 서블릿 코드와 유사하게 변환된다.

JSP에서 자바 클래스를 사용할 때는 import를 해주어야 한다.

회원 저장 JSP는 HTML 중심으로 작성하고 중간에 자바 코드를 삽입한다.

회원 저장을 처리하는 부분은 `<% %>` 태그를, 저장 결과를 출력하는 부분은 `<%= %>` 태그를 사용한다.

회원 등록 폼 JSP

```
<%@ page language="java" contentType="text/html; cha
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <!-- 상대경로 -->
    <form action="/view/v1/save.jsp" method="post">
        이름: <input type="text" name="username" />
        암호: <input type="text" name="password" />
        <button type="submit">전송</button>
    </form>
```

회원 저장 JSP

```
<%@ page import="com.example.demo.domain.MemberRepository" %>
<%@ page import="com.example.demo.domain.Member" %>
<%@ page language="java" contentType="text/html; charset=UTF-8" %>

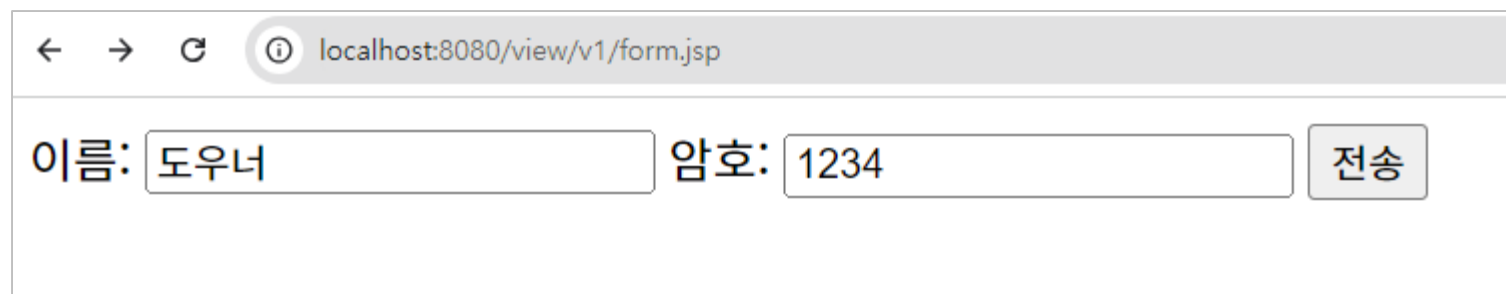
<!-- 자바코드를 사용 -->
<%
    MemberRepository repository = new MemberRepository();

    String username = request.getParameter("username");
    String password = request.getParameter("password");

    Member member = Member.builder()
        .userId(username).password(password)
        .build();

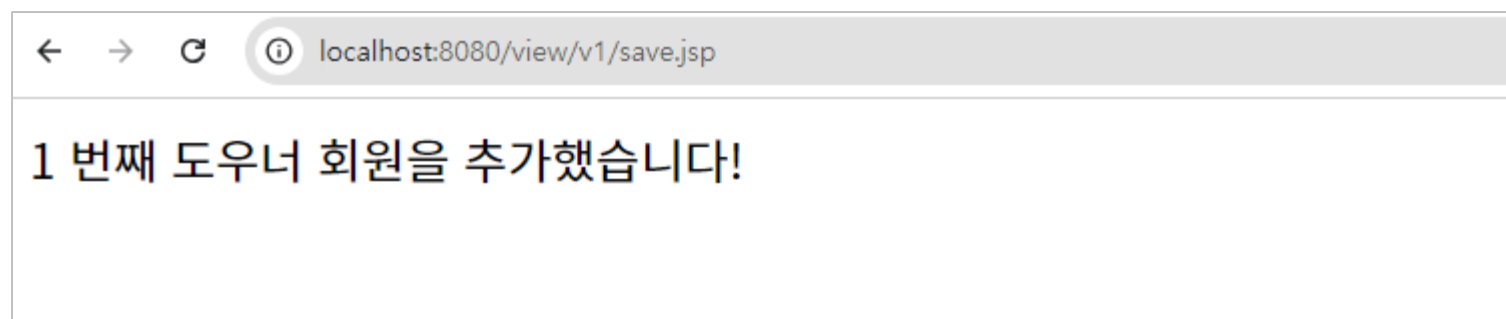
    Member newMember = repository.save(member);
%>
```

1. 브라우저에서 `http://localhost:8080/localhost:8080/view/v1/form.jsp` 주소를 호출하여 회원 등록 폼을 실행한다.
2. 폼에서 이름과 암호를 입력한 후, 전송 버튼을 클릭한다.
3. 다음 페이지에서 새로운 회원 정보가 성공적으로 등록되었는지 확인한다.



← → ↻ ⓘ localhost:8080/view/v1/form.jsp

이름: 암호:



← → ↻ ⓘ localhost:8080/view/v1/save.jsp

1 번째 도우너 회원을 추가했습니다!

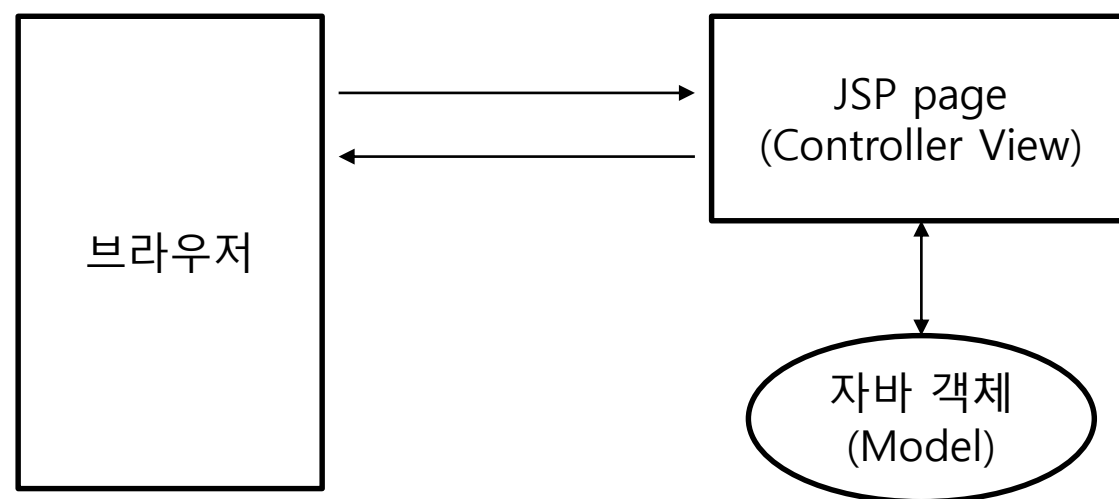
지금까지 작성한 JSP 프로그램은 **MVC 모델1** 구조이다.

모델1 구조에서는 JSP가 사용자 요청을 모두 처리하며, 컨트롤러와 뷰의 역할을 모두 수행한다.

MVC 모델1의 단점

- JSP 파일에 HTML코드와 자바 코드가 섞여서 복잡해진다.
- 모든 로직이 한 파일에 코드가 모여 있어서 유지보수가 어렵다.

MVC 모델1 구조



JSP 처리 과정

```
<%
    MemberRepository repository = new MemberRepository();

    String username = request.getParameter("username");
    String password = request.getParameter("password");

    Member member = Member.builder()
        .userId(username).password(password)
        .build();

    Member newMember = repository.save(member);
%>

<html>
<body>
<p>
    <%= newMember.getNo() %> 번째 <%= newMember.getUserId() %> 회원을 추가했습니다!
  </p>
</body>
</html>
```

비즈니스 로직에서
오류가 발생하면..

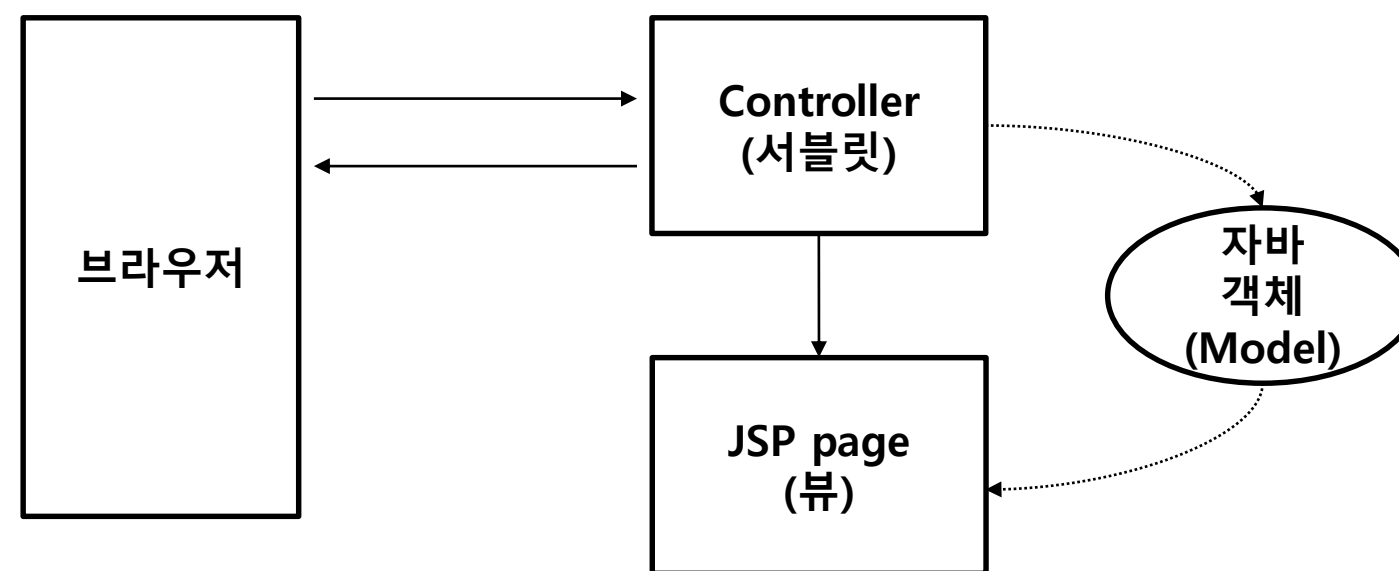
뷰 렌더링도
실패할 수 있다!

스프링 MVC

- 스프링 mvc는 '모델2'라는 방식으로 처리되는 구조이다.
- 모델 2방식은 비즈니스 로직과 화면을 분리하여 개발하는 방식이다.

스프링의 동작 순서

1. 서블릿이 사용자의 요청을 처리하고 jsp에 데이터를 전달한다.
2. 사용자에게 결과 화면(jsp)를 보낸다.



Model(모델)

- 데이터 관리를 담당한다.
- 여기에는 Entity, Dto, Repository, Service 클래스가 포함된다.

View(뷰)

- 사용자에게 보여주는 화면을 만든다.
- html, jsp 또는 타임리프 같은 템플릿 엔진을 사용한다.

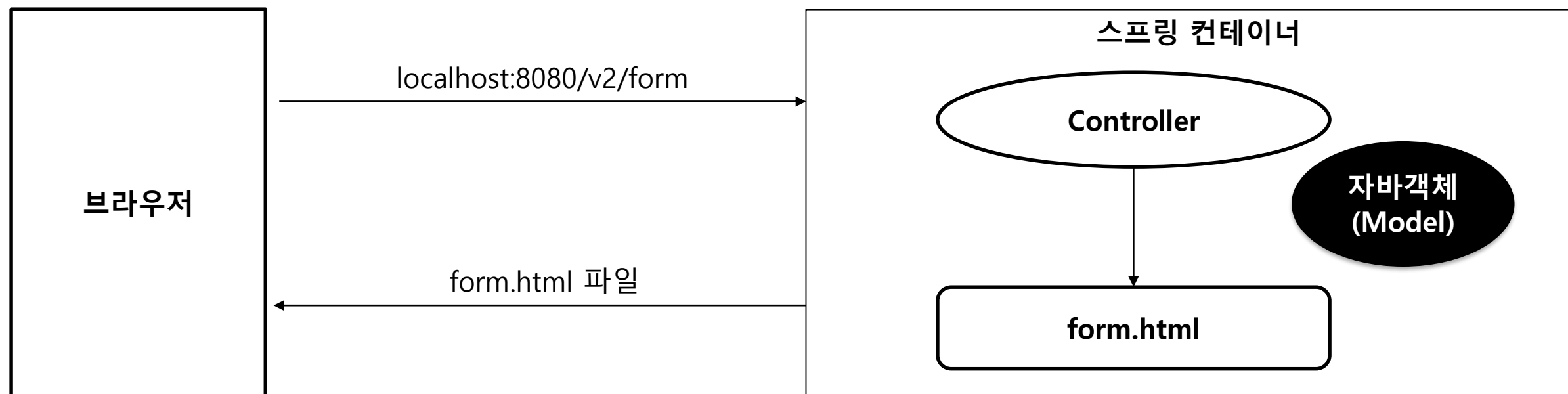
Controller(컨트롤러)

- 모델과 뷰를 연결하고, 사용자 요청을 처리한다.
- 사용자의 요청을 받아 작업을 수행하고, 결과를 화면에 표시한다.
- 내부적으로 서블릿 기술을 사용한다.

MVC 동작 과정

사용자가 서버에 데이터를 요청하면 다음과 같이 동작 과정이 이루어진다.

1. 사용자가 브라우저의 주소창에 URL을 입력한다.
2. 서버에 사용자 요청이 도착한다.
3. 서버는 요청을 컨트롤러에 전달한다.
4. 컨트롤러는 로직을 처리하고, 결과데이터를 뷰에 전달한다.
5. 적절한 뷰를 찾아서 사용자한테 보내준다.



컨트롤러 만들기

- 컨트롤러에 기본 URL주소를 지정한다.
- 사용자 요청을 처리할 메소드를 추가한다.
- 메소드에서 사용자의 요청을 해석하고 응답데이터를 전달한다.
- 반환할 JSP 경로를 지정한다.

컨트롤러 예시

```
@Controller
@RequestMapping("/v2")
public class MemberController {

    @GetMapping("/form")
    public String method1() {
        return "/v2/form";
    }
}
```

어노테이션	설명
@Controller	<ul style="list-style-type: none">• 사용자 요청이 들어오면 처리하는 클래스이다.• 스프링 컨테이너에 빈으로 등록된다.
@RequestMapping	<ul style="list-style-type: none">• 사용자 요청을 처리한다.• 요청URL에 /sample을 포함되면 해당 클래스로 연결된다
@GetMapping	<ul style="list-style-type: none">• HTTP GET 요청을 처리한다.• 요청URL에 /ex1을 포함되면 해당 메소드로 연결된다.

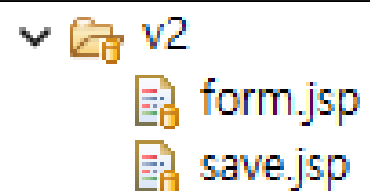
뷰 만들기

- webapp 폴더 밑에 jsp파일을 생성한다.

컨트롤러

```
public String method1() {  
    return "/v2/form";  
}
```

JSP



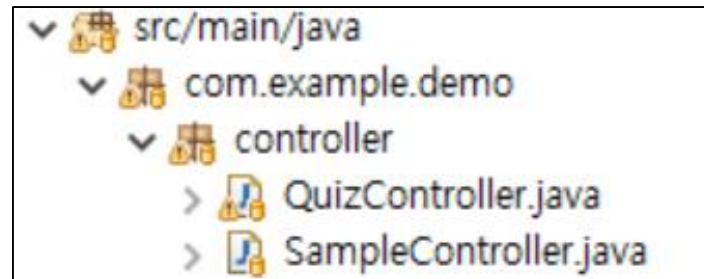
File explorer showing the v2 folder containing form.jsp and save.jsp files.

localhost:8080/v2/form 주소에 접속하면, 서버가 사용자에게 form.jsp 파일을 보내준다.

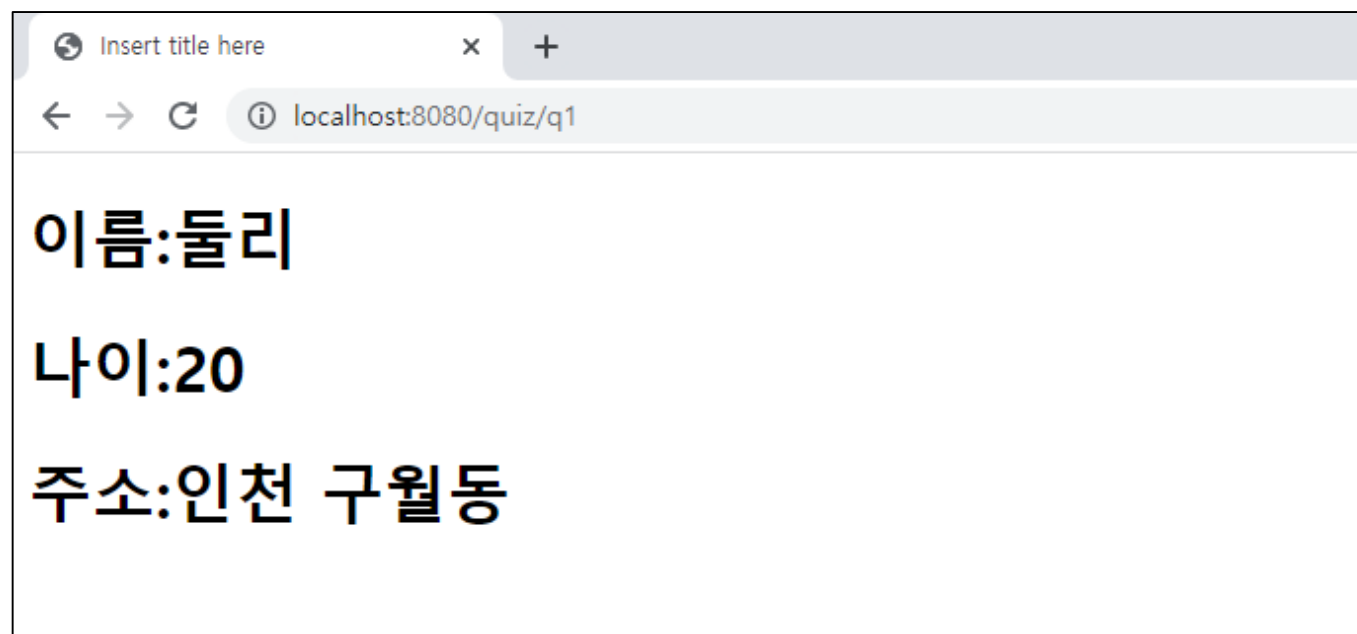
화면

이름: 암호:

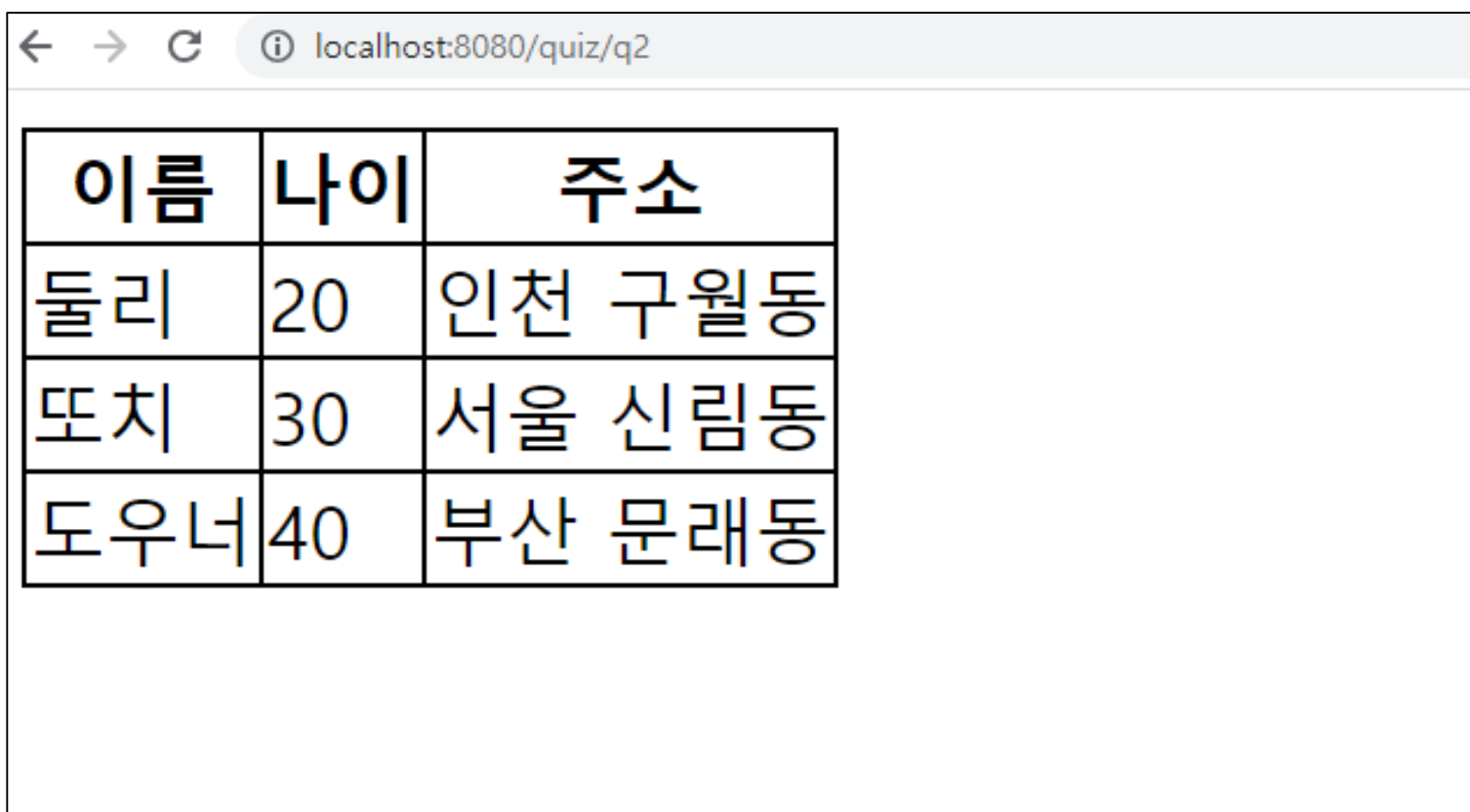
먼저 controller 패키지에 QuizController 클래스를 추가하세요.



Q1. localhost:8080/quiz/q1 주소에 접속했을 때 아래와 같이 화면을 표시하도록 코드를 구현하세요.



Q2. localhost:8080/quiz/q2 주소에 접속했을 때 아래와 같이 화면을 표시하도록 코드를 구현하세요.



이름	나이	주소
둘리	20	인천 구월동
또치	30	서울 신림동
도우너	40	부산 문래동