스프링 MVC의 요청-응답 흐름 이해

미션	[백엔드] 스프링 MVC
상태	완료
설명	스프링 MVC에서 클라이언트의 요청이 컨트롤러를 거쳐 뷰로 전달되기까지의 과정을 단계별로 정리합니다. 각 단계에서 사용되는 주요 컴포넌트와 그 역할(DispatcherServlet, Controller, ViewResolver 등)을 요약하여 설명하는 문서를 작성해봅니다. 결과물로 요청-응답 흐름을 나타내는 다이어그램과 설명 문서를 제출합니다.
유형	미션

서블릿과 JSP의 한계

서블릿으로 개발

뷰 화면을 위한 HTML을 만드는 작업이 자바 코드에 섞여 있어서 코드가 지저분하고 복잡하다. JSP를 사용한 덕분에 뷰를 생성하는 HTML 작업을 깔끔하게 가져가고, 동적으로 변경이 필요한 부분만 자바 코드를 적용할 수 있다.

JSP 문제

JSP 코드의 절반은 비즈니스 로직으로 구성되어 있고, 나머지 절만만 결과를 HTML로 보여주기 위한 뷰 영역이다. 자바 코드, 데이터 조회 등 다양한 코드가 JSP에 노출되어 있다. JSP가 너무 많은 역할을 하기 때문이다. 이렇게 된다면 유지보수 측면에서도 어려울 것이고 JSP 코드가 길어질 것이다.

MVC 패턴의 등장

비즈니스 로직은 서블릿처럼 다른 곳에서 처리하고, JSP는 목적에 맞게 HTML로 화면을 구성하는 일에 집중하도록 한다.

MVC 패턴

하나의 서블릿이나 JSP만으로 비즈니스 로직과 뷰를 모두 처리하게 되면 너무 많은 역할을 하게 된다. 결과적으로 유지보수가 어려워진다. **MVC 패턴은 하나의 서블릿이나 JSP가 처리하던 것들을 Controller, View, Model로 나눈 것을 말한다.**

모델

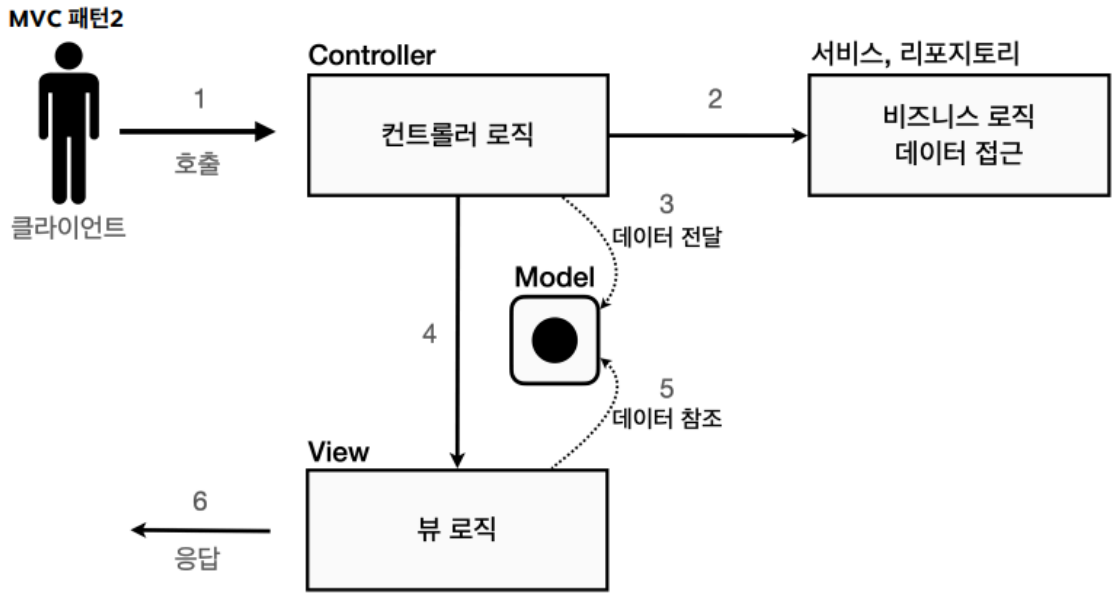
- 뷰에 출력할 데이터를 담아둔다. 뷰가 필요한 데이터를 모두 모델에 담아서 전달해주는 덕분에, 뷰는 화면 렌더링에만 집중할 수 있다.

뷰

- 모델에 담겨있는 데이터를 사용해서 화면을 그리는 일을 한다.

컨트롤러

- HTTP 요청을 받아 파라미터 검증, 비즈니스 로직 실행
- 뷰에 전달할 결과 데이터를 조회해서 모델에 담는다.



> 컨트롤러에 비즈니스 로직을 둘 수 있지만, 컨트롤러의 부담이 커지게 되어, 비즈니스 로직은 서비스라는 계층을 별도로 두어 처리한다. 그리고 컨트롤러는 비즈니스 로직이 있는 서비스를 호출하는 역할을 담당한다.

MVC 패턴 적용

> 서블릿을 컨트롤러로, JSP를 뷰로 사용해서 MVC 패턴을 적용

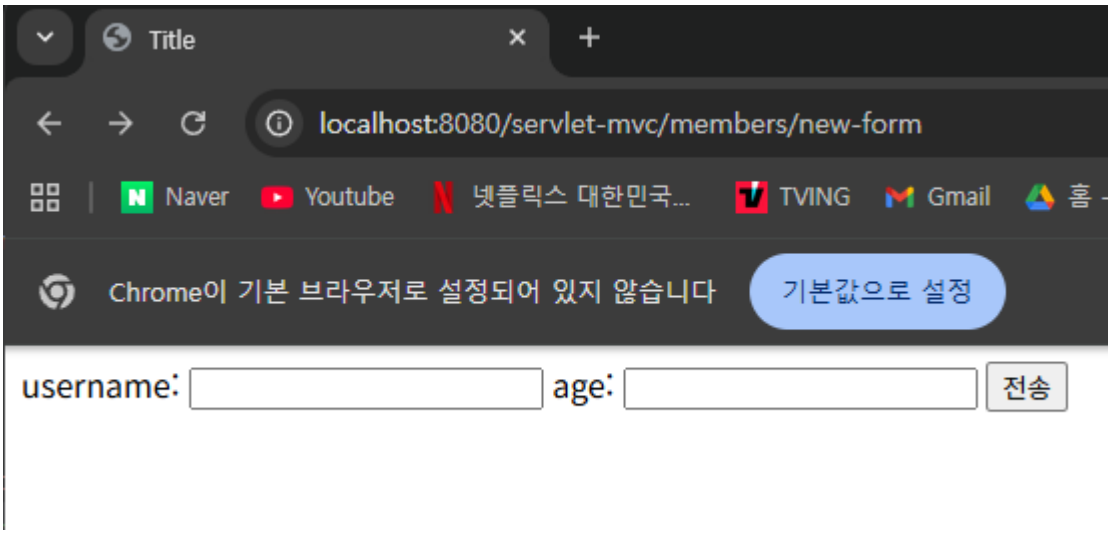
회원 등록 폼

Controller

```
package com.example.servlet.web.servletmvc;

import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.HttpServletBean;

import java.io.IOException;

@WebServlet(name = "mvcMemberFormServlet", urlPatterns = "/servlet-m
vc/members/new-form")
public class MvcMemberFromServlet extends HttpServletBean {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String viewPath = "/WEB-INF/views/new-form.jsp";
        RequestDispatcher dispatcher = request.getRequestDispatcher(viewP
ath);
        dispatcher.forward(request, response);
    }
}
```

View

```
<%--
  Created by IntelliJ IDEA.
  User: user
  Date: 25. 12. 17.
  Time: 오후 10:08
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<!-- 상대경로 사용, [현재 URL이 속한 계층 경로 + /save] -->
<form action="save" method="post">
    username: <input type="text" name="username" />
    age: <input type="text" name="age" />
    <button type="submit">전송</button>
</form>
</body>
</html>
```



회원 저장

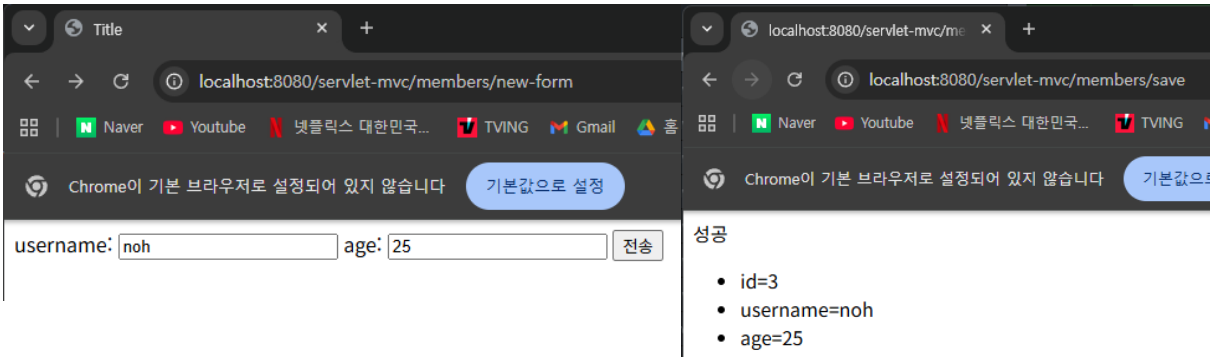
Controller

```
package com.example.servlet.web.servletmvc;

import com.example.servlet.domain.member.Member;
import com.example.servlet.domain.member.MemberRepository;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

@WebServlet(name = "mvcMemberSaveServlet", urlPatterns = "/servlet-m
vc/members/save")
public class MvcMemberSaveServlet extends HttpServlet {

    private MemberRepository memberRepository = MemberRepository.getI
nstance();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String username = request.getParameter("username");
        int age = Integer.parseInt(request.getParameter("age"));

        Member member = new Member(username, age);
        System.out.println("member : " + member);
        memberRepository.save(member);

        request.setAttribute("member", member);

        String viewPath = "/WEB-INF/views/save-result.jsp";
        RequestDispatcher dispatcher = request.getRequestDispatcher(viewP
ath);
        dispatcher.forward(request, response);
```

```
    }
}
```

View

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
성공
<ul>
    <li>id=${member.id}</li>
    <li>username=${member.username}</li>
    <li>age=${member.age}</li>
</ul>
<a href="/index.html">메인</a>
</body>
</html>
```



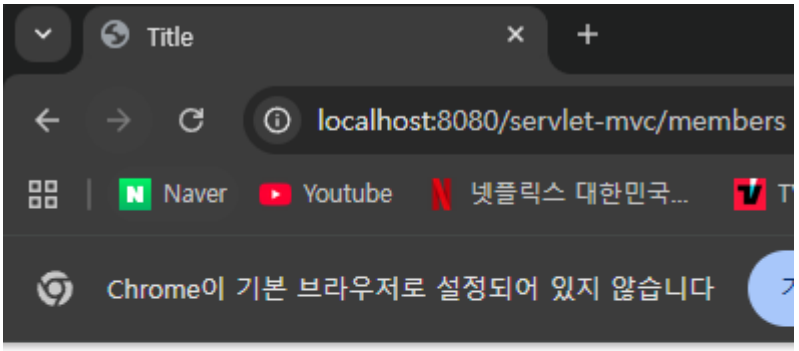
회원 조회

Controller

```
package com.example.servlet.web.servletmvc;

import com.example.servlet.domain.member.Member;
import com.example.servlet.domain.member.MemberRepository;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.util.List;

@WebServlet(name = "mvcMemberListServlet", urlPatterns = "/servlet-mv
c/members")
public class MvcMemberListServlet extends HttpServlet {

    private MemberRepository memberRepository = MemberRepository.getI
nstance();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        System.out.println("MvcMemberListServlet.service");

        List<Member> members = memberRepository.findAll();

        request.setAttribute("members", members);

        String viewPath = "/WEB-INF/views/members.jsp";
        RequestDispatcher dispatcher = request.getRequestDispatcher(viewP
ath);
        dispatcher.forward(request, response);
    }
}
```

View

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<a href="/index.html">메인</a>
<table>
    <thead>
    <th>id</th>
    <th>username</th>
    <th>age</th>
    </thead>
    <tbody>
    <c:forEach var="item" items="${members}">
        <tr>
            <td>${item.id}</td>
            <td>${item.username}</td>
            <td>${item.age}</td>
        </tr>
    </c:forEach>
    </tbody>
</table>
</body>
</html>
```



메인

id	username	age
1	noh	25
2	noh	123
3	asd	333

MVC 패턴의 한계

MVC 패턴 덕분에 컨트롤러와 뷰의 역할을 명확히 구분할 수 있었다. 특히 뷰는 화면을 그리는 역할에 충실한 덕분에, 코드가 깔끔하고 직관적이다.

하지만 컨트롤러는 중복된 코드가 많아 보이고, 필요없는 코드도 있다는 것을 알 수 있다.

MVC 컨트롤러의 단점

포워드 중복

> view로 이동하는 코드가 항상 중복 호출이 된다.

```
viewPath = "~~~/~~~/~~~.jsp"
RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
dispatcher.forward(request, response);
```

ViewPath

> 경로에 중복되는 부분 발생

- prefix : `/WEB-INF/views`
- sufix : `.jsp`

사용하지 않는 코드

> 사용할 때도 있고, 사용하지 않을 때도 있다.

```
HttpServletRequest request, HttpServletResponse response
```

공통처리가 어렵다.

> 기능이 복잡해질수록 컨트롤러에서는 공통으로 처리해야 할 부분이 점점 증가할 것이다.

정리

> 컨트롤러 호출 전에 공통 기능을 먼저 처리해야하는 기능이 필요하다.

프론트 컨트롤러(Front Controller) 패턴을 도입하여 다음 문제를 해결할 수 있다. 또한 스프링 MVC의 핵심이 바로 프론트 컨트롤러이다.