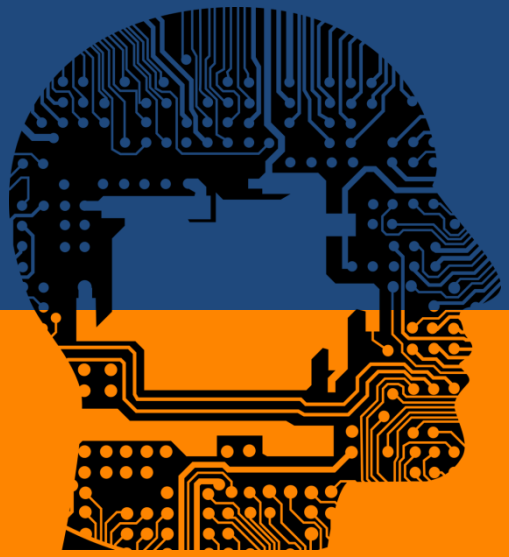
한국IT진흥부설
정보보호교육학원 아이섹

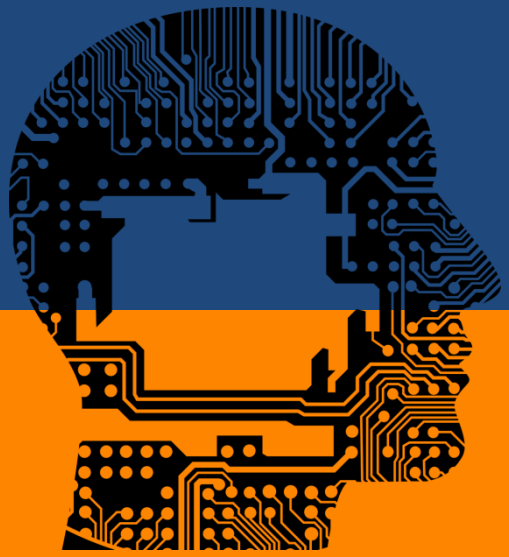


JAVA 웹 개발자 양성과정
Spring: Spring MVC

3강 - JSP, MVC패턴

By SoonGu Hong

한국IT진흥부설
정보보호교육학원 아이섹



JAVA 웹 개발자 양성과정

Spring: Spring MVC

1. JSP

템플릿 엔진으로

지금까지 서블릿과 자바 코드만으로 HTML을 만들어보았다. 서블릿 덕분에 동적으로 원하는 HTML을 마음껏 만들 수 있다. 정적인 HTML 문서라면 화면이 계속 달라지는 회원의 저장 결과라던가, 회원 목록 같은 동적인 HTML을 만드는 일은 불가능 할 것이다.

그런데, 코드에서 보듯이 이것은 매우 복잡하고 비효율 적이다. 자바 코드로 HTML을 만들어 내는 것 보다 차라리 HTML 문서에 동적으로 변경해야 하는 부분만 자바 코드를 넣을 수 있다면 더 편리할 것이다.

이것이 바로 템플릿 엔진이 나온 이유이다. 템플릿 엔진을 사용하면 HTML 문서에서 필요한 곳만 코드를 적용해서 동적으로 변경할 수 있다.

템플릿 엔진에는 JSP, Thymeleaf, Freemarker, Velocity등이 있다.

나음 시간에는 JSP로 동일한 작업을 진행해보사.

참고

JSP는 성능과 기능면에서 다른 템플릿 엔진과의 경쟁에서 밀리면서, 점점 사장되어 가는 추세이다. 템플릿 엔진들은 각각 장단점이 있는데, 강의에서는 JSP는 앞부분에서 잠깐 다루고, 스프링과 잘 통합되는 Thymeleaf를 사용한다.

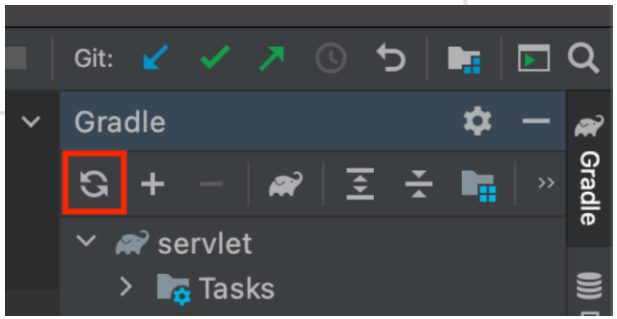
JSP 라이브러리 추가

JSP를 사용하려면 먼저 다음 라이브러리를 추가해야 한다.

`build.gradle` 에 추가

```
//JSP 추가 시작
implementation 'org.apache.tomcat.embed:tomcat-embed-jasper'
implementation 'javax.servlet:jstl'
//JSP 추가 끝
```

라이브러리를 추가하면 다음 버튼을 클릭해서 Gradle을 refresh 해주자.



회원 등록 폼 JSP

main/webapp/jsp/members/new-form.jsp

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

<form action="/jsp/members/save.jsp" method="post">
    username: <input type="text" name="username" />
    age:      <input type="text" name="age" />
    <button type="submit">전송</button>
</form>

</body>
</html>
```

- `<%@ page contentType="text/html;charset=UTF-8" language="java" %>`
  - 첫 줄은 JSP문서라는 뜻이다. JSP 문서는 이렇게 시작해야 한다.

회원 등록 폼 JSP를 보면 첫 줄을 제외하고는 완전히 HTML와 똑같다. JSP는 서버 내부에서 서블릿으로 변환되는데, 우리가 만들었던 MemberFormServlet과 거의 비슷한 모습으로 변환된다.

실행

- http://localhost:8080/jsp/members/new-form.jsp
  - 실행시 `.jsp` 까지 함께 적어주어야 한다.

JSP는 자바 코드를 그대로 다 사용할 수 있다.

- `<%@ page import="hello.servlet.domain.member.MemberRepository" %>`
  - 자바의 import 문과 같다.
- `<% ~~ %>`
  - 이 부분에는 자바 코드를 입력할 수 있다.
- `<%= ~~ %>`
  - 이 부분에는 자바 코드를 출력할 수 있다.

서블릿과 JSP의 한계

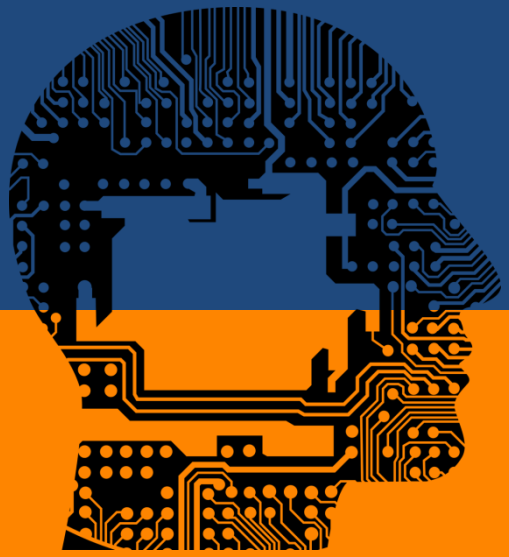
서블릿으로 개발할 때는 뷰(View)화면을 위한 HTML을 만드는 작업이 자바 코드에 섞여서 지저분하고 복잡했다.
JSP를 사용한 덕분에 뷰를 생성하는 HTML 작업을 깔끔하게 가져가고, 중간중간 동적으로 변경이 필요한 부분에만 자바 코드를 적용했다. 그런데 이렇게 해도 해결되지 않는 몇가지 고민이 남는다.

회원 저장 JSP를 보자. 코드의 상위 절반은 회원을 저장하기 위한 비즈니스 로직이고, 나머지 하위 절반만 결과를 HTML로 보여주기 위한 뷰 영역이다. 회원 목록의 경우에도 마찬가지다.
코드를 잘 보면, JAVA 코드, 데이터를 조회하는 리포지토리 등등 다양한 코드가 모두 JSP에 노출되어 있다.
JSP가 너무 많은 역할을 한다. 이렇게 작은 프로젝트도 벌써 머리가 아파오는데, 수백 수천줄이 넘어가는 JSP를 떠올려보면 정말 지옥과 같을 것이다. (유지보수 지옥 썰)

MVC 패턴의 등장

비즈니스 로직은 서블릿 처럼 다른곳에서 처리하고, JSP는 목적에 맞게 HTML로 화면(View)을 그리는 일에 집중하도록 하자. 과거 개발자들도 모두 비슷한 고민이 있었고, 그래서 MVC 패턴이 등장했다. 우리도 직접 MVC 패턴을 적용해서 프로젝트를 리팩터링 해보자.

한국IT진흥부설
정보보호교육학원 아이섹



JAVA 웹 개발자 양성과정

Spring: Spring MVC

2. MVC 패턴

MVC 패턴 - 개요

너무 많은 역할

하나의 서블릿이나 JSP만으로 비즈니스 로직과 뷰 렌더링까지 모두 처리하게 되면, 너무 많은 역할을 하게되고, 결과적으로 유지보수가 어려워진다. 비즈니스 로직을 호출하는 부분에 변경이 발생해도 해당 코드를 손대야 하고, UI를 변경할 일이 있어도 비즈니스 로직이 함께 있는 해당 파일을 수정해야 한다. HTML 코드 하나 수정해야 하는데, 수백줄의 자바 코드가 함께 있다고 상상해보라! 또는 비즈니스 로직을 하나 수정해야 하는데 수백 수천줄의 HTML 코드가 함께 있다고 상상해보라.

변경의 라이프 사이클

사실 이게 정말 중요한데, 진짜 문제는 둘 사이에 변경의 라이프 사이클이 다르다는 점이다. 예를 들어서 UI를 일부 수정하는 일과 비즈니스 로직을 수정하는 일은 각각 다르게 발생할 가능성이 매우 높고 대부분 서로에게 영향을 주지 않는다. 이렇게 변경의 라이프 사이클이 다른 부분을 하나의 코드로 관리하는 것은 유지보수하기 좋지 않다. (물론 UI가 많이 변하면 함께 변경될 가능성도 있다.)

기능 특화

특히 JSP 같은 뷰 템플릿은 화면을 렌더링 하는데 최적화 되어 있기 때문에 이 부분의 업무만 담당하는 것이 가장 효과적이다.

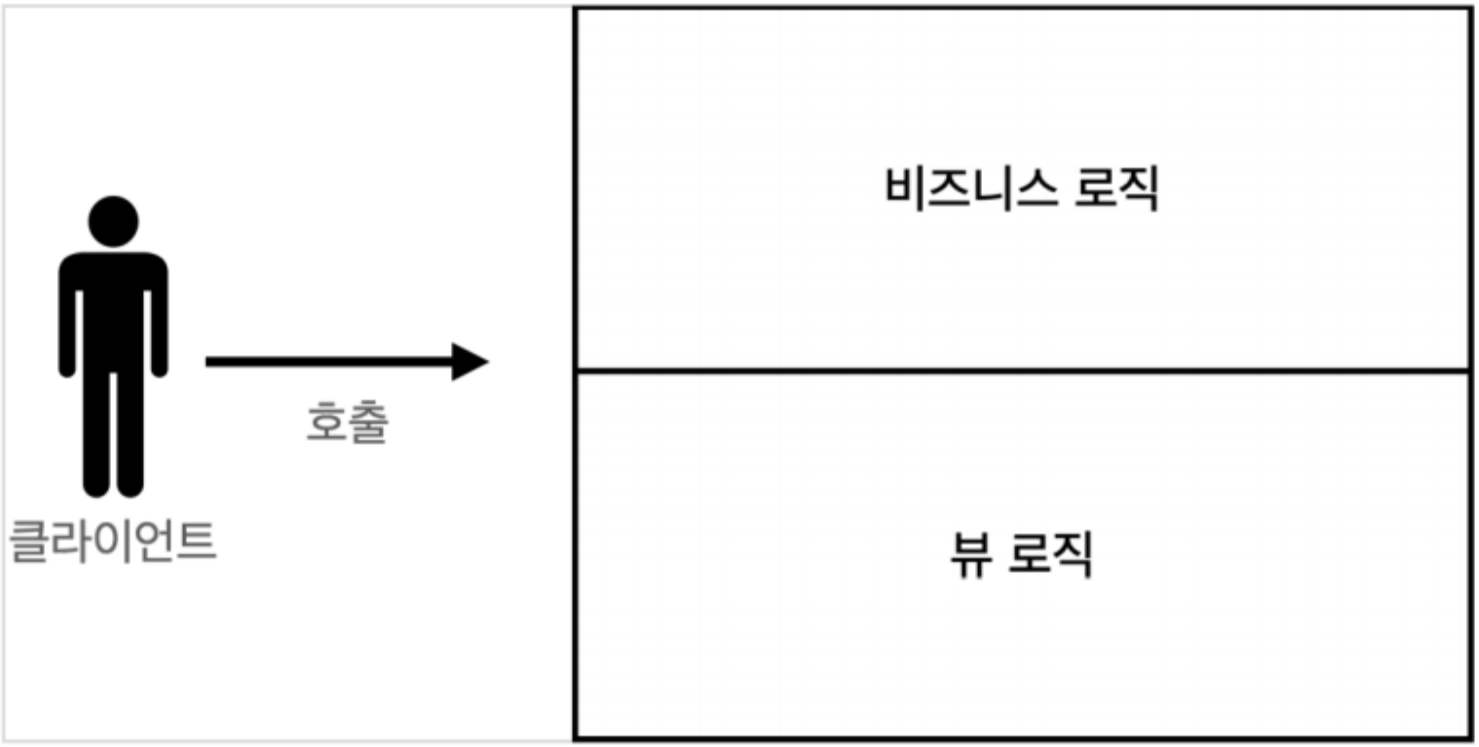
Model View Controller

MVC 패턴은 지금까지 학습한 것 처럼 하나의 서블릿이나, JSP로 처리하던 것을 컨트롤러(Controller)와 뷰(View)라는 영역으로 서로 역할을 나눈 것을 말한다. 웹 애플리케이션은 보통 이 MVC 패턴을 사용한다.

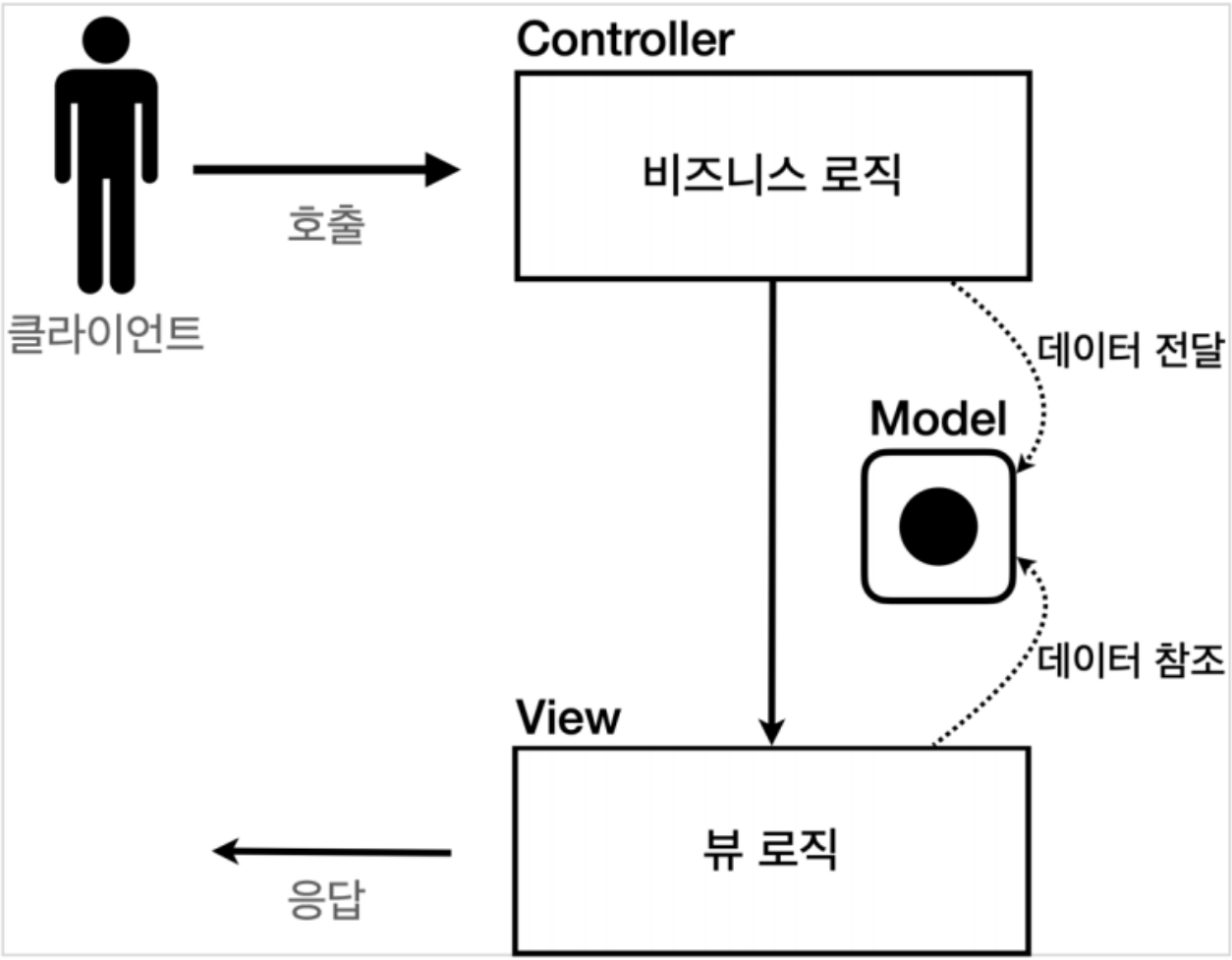
컨트롤러: HTTP 요청을 받아서 파라미터를 검증하고, 비즈니스 로직을 실행한다. 그리고 뷰에 전달할 결과 데이터를 조회해서 모델에 담는다.

모델: 뷰에 출력할 데이터를 담아둔다. 뷰가 필요한 데이터를 모두 모델에 담아서 전달해주는 덕분에 뷰는 비즈니스 로직이나 데이터 접근을 몰라도 되고, 화면을 렌더링 하는 일에 집중할 수 있다.
뷰: 모델에 담겨있는 데이터를 사용해서 화면을 그리는 일에 집중한다. 여기서는 HTML을 생성하는 부분을 말한다.

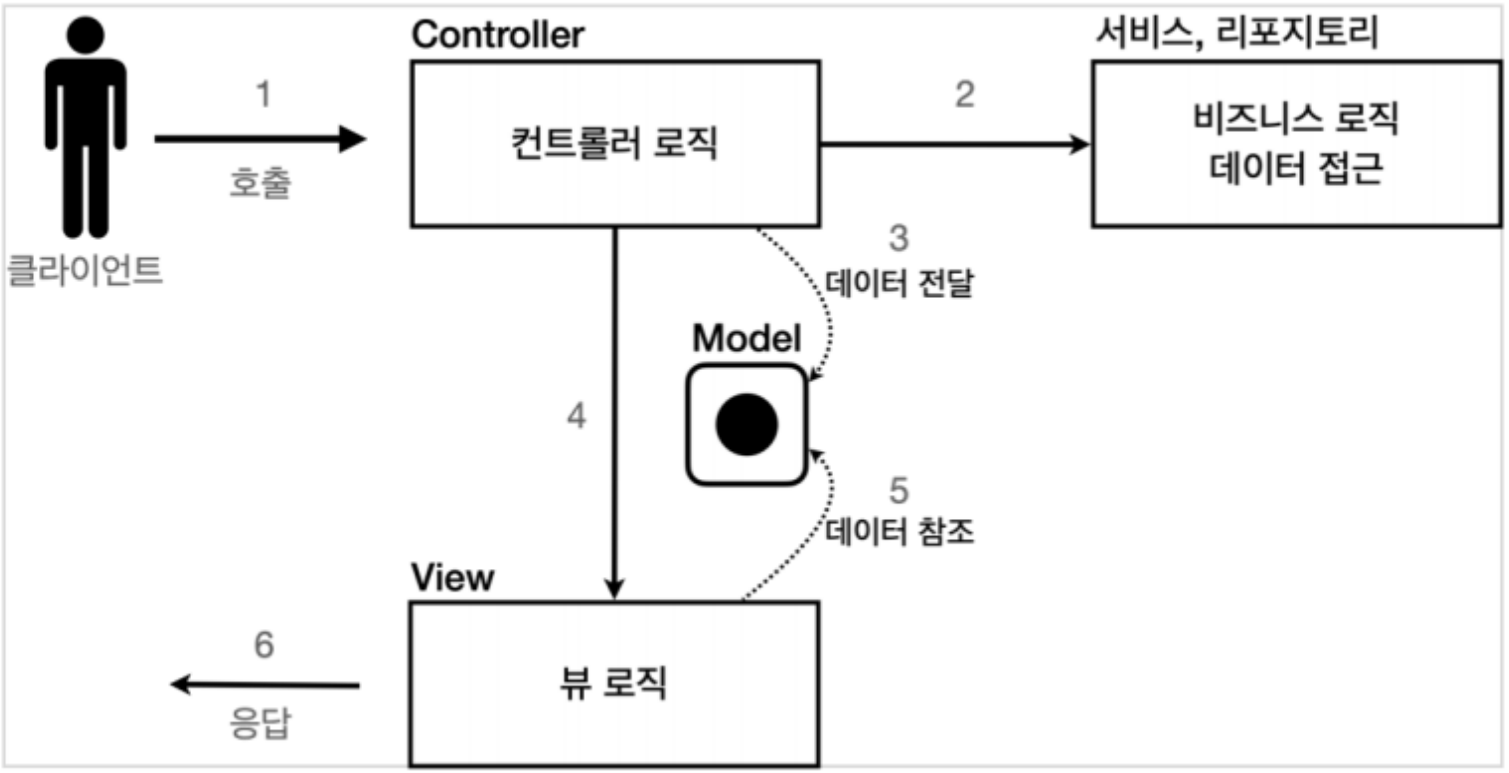
MVC 패턴 이전



MVC 패턴1



MVC 패턴2


Controller
서비스, 리포지토리
1
호출
클라이언트
컨트롤러 로직
2
비즈니스 로직
데이터 접근
3
데이터 전달
Model
4
5
데이터 참조
View
6
응답
뷰 로직

MVC 패턴 - 한계

MVC 패턴을 적용한 덕분에 컨트롤러의 역할과 뷰를 렌더링 하는 역할을 명확하게 구분할 수 있다.
특히 뷰는 화면을 그리는 역할에 충실한 덕분에, 코드가 깔끔하고 직관적이다. 단순하게 모델에서 필요한 데이터를 꺼내고, 화면을 만들면 된다.
그런데 컨트롤러는 딱 봐도 중복이 많고, 필요하지 않는 코드들도 많이 보인다.

MVC 컨트롤러의 단점

포워드 중복

View로 이동하는 코드가 항상 중복 호출되어야 한다. 물론 이 부분을 메서드로 공통화해도 되지만, 해당 메서드도 항상 직접 호출해야 한다.

```
RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
dispatcher.forward(request, response);
```

ViewPath에 중복

```
String viewPath = "/WEB-INF/views/new-form.jsp";
```

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

그리고 만약 jsp가 아닌 thymeleaf 같은 다른 뷰로 변경한다면 전체 코드를 다 변경해야 한다.

사용하지 않는 코드

다음 코드를 사용할 때도 있고, 사용하지 않을 때도 있다. 특히 response는 현재 코드에서 사용되지 않는다.

```
HttpServletRequest request, HttpServletResponse response
```

공통 처리가 어렵다.

기능이 복잡해질 수 록 컨트롤러에서 공통으로 처리해야 하는 부분이 점점 더 많이 증가할 것이다. 단순히 공통 기능을 메서드로 뽑으면 될 것 같지만, 결과적으로 해당 메서드를 항상 호출해야 하고, 실수로 호출하지 않으면 문제가 될 것이다. 그리고 호출하는 것 자체도 중복이다.

정리하면 공통 처리가 어렵다는 문제가 있다.

이 문제를 해결하려면 컨트롤러 호출 전에 먼저 공통 기능을 처리해야 한다. 소위 **수문장 역할**을 하는 기능이 필요하다. **프론트 컨트롤러(Front Controller) 패턴**을 도입하면 이런 문제를 깔끔하게 해결할 수 있다.
(입구를 하나로!)
스프링 MVC의 핵심도 바로 이 프론트 컨트롤러에 있다.

감사합니다
THANK YOU