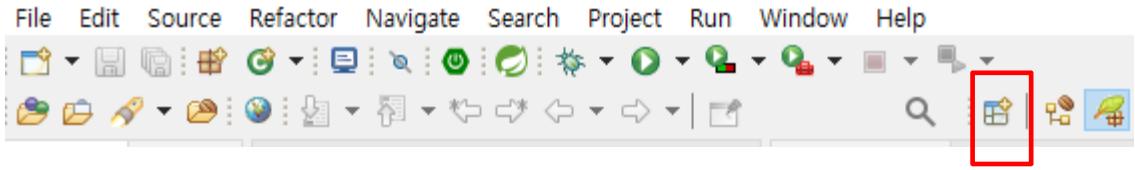
3. 스프링 MVC

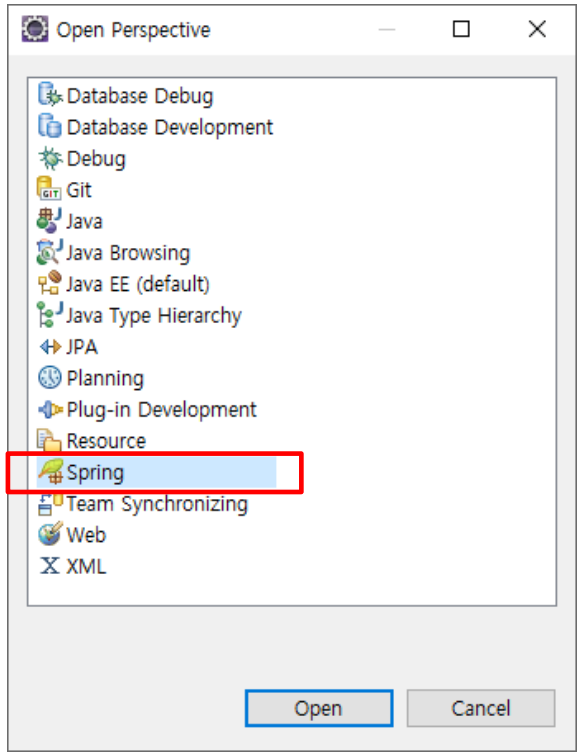
1. 스프링 MVC 프로젝트
2. 스프링 MVC 테스트
3. 스프링 MVC 구조

1.1 Spring MVC 프로젝트 생성

- perspective를 spring으로 변경하기
  - Open Perspective

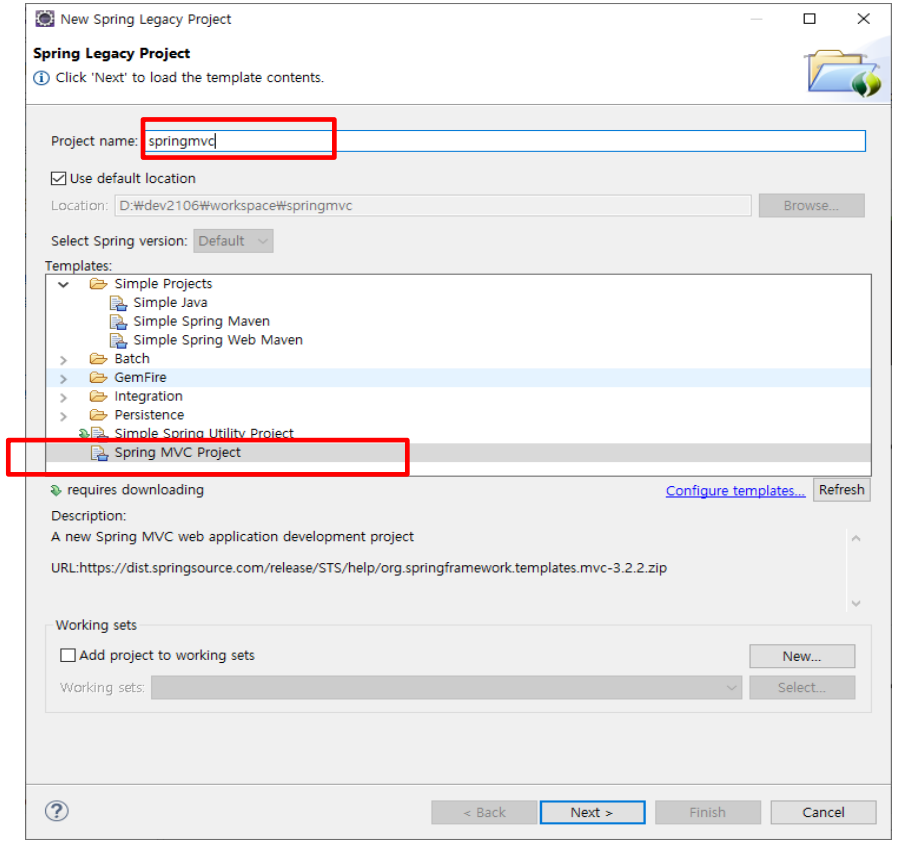


  - spring 선택

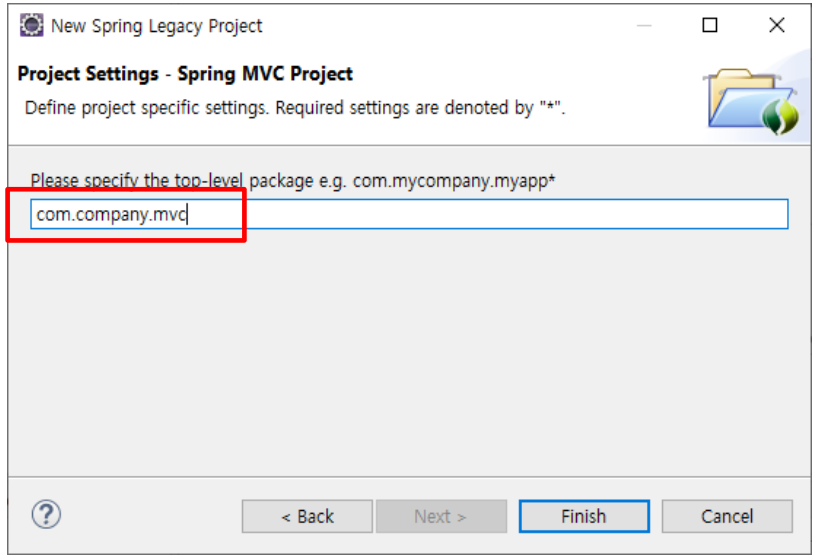


1.1 Spring MVC 프로젝트 생성

- 프로젝트 생성
  - File -> New -> spring Legacy Project
  - 템플릿에서 Spring MVC Project 선택

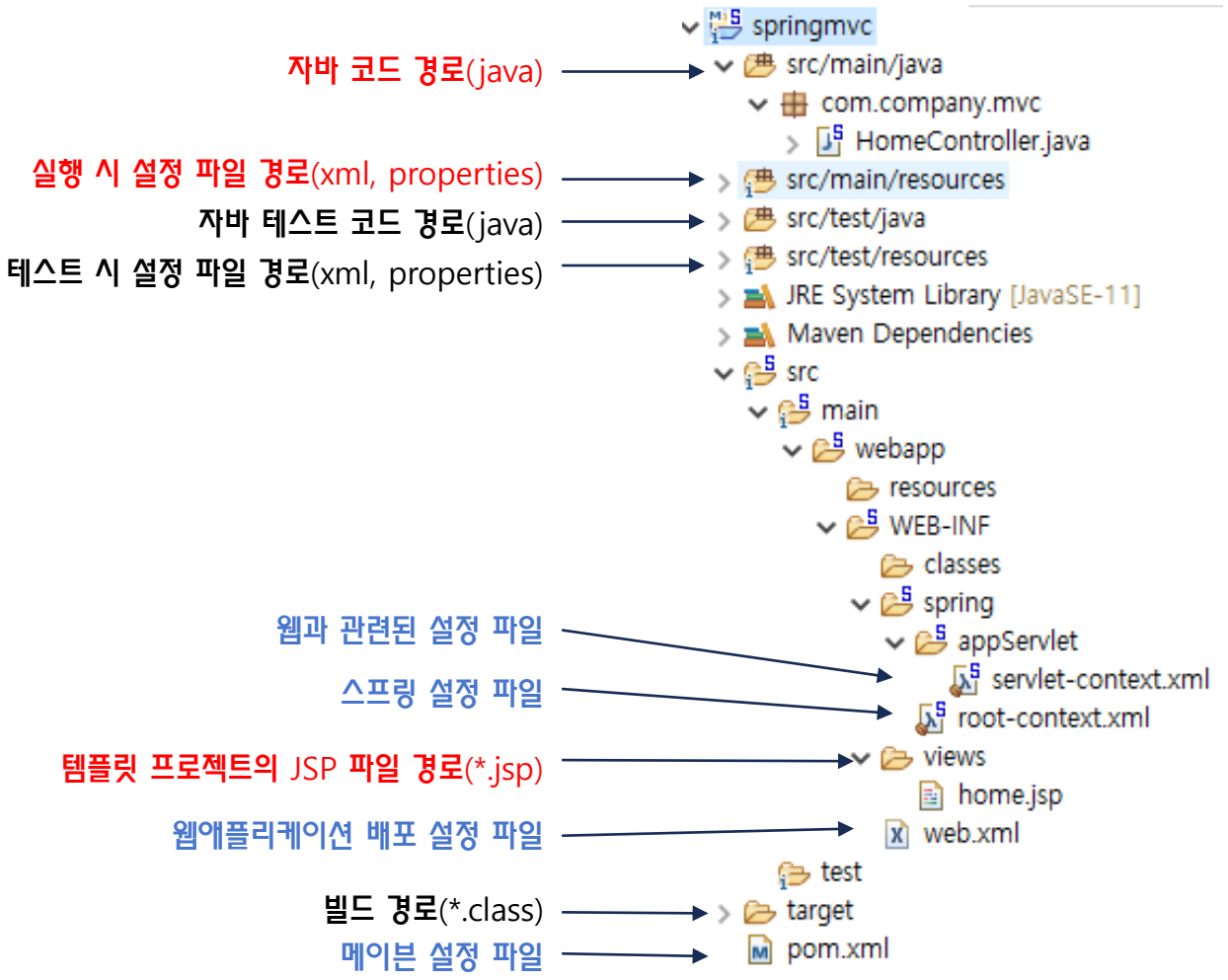


  - 패키지명 입력



1.1 Spring MVC 프로젝트 생성

- 프로젝트 구조



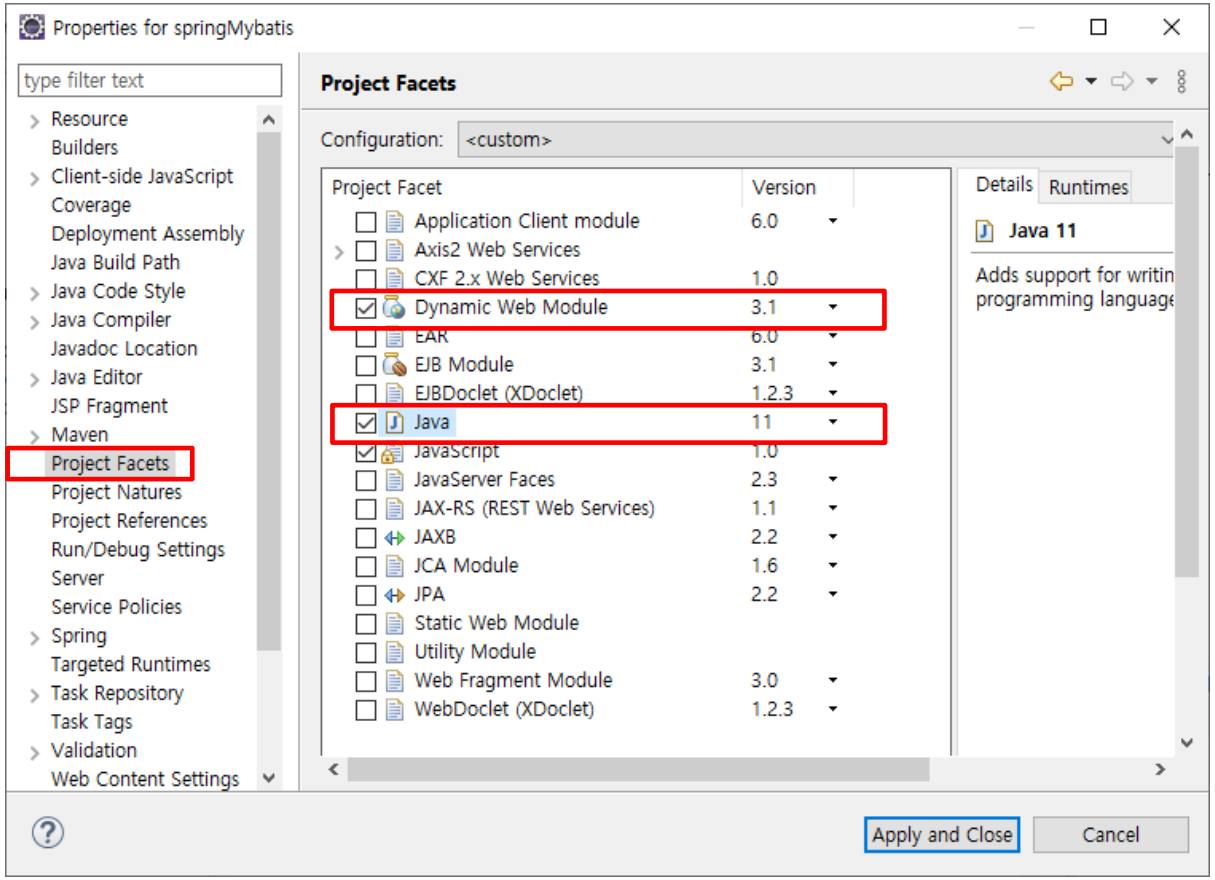
1.2 버전 변경

- Web Module 버전 변경
  - src₩main₩webapp₩WEB-INF₩web.xml

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"    version="3.1">
```

1.2 버전 변경

- java version 변경
  - 프로젝트 컨텍스트 메뉴 -> Properties 메뉴 -> Project Facets
  - JAVA 11로 변경
  - dynamic Web Module 3.1로 변경



1.3 CharacterEncodingFilter 등록

- src₩main₩webapp₩WEB-INF₩web.xml
  - Encoding 파라미터 정보를 읽어 인코딩 방식을 설정
  - <url-pattern> 설정의 요청에 대해서 일괄적으로 한글 처리

```
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>utf-8</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

1.4 log4j.xml dtd 경로 변경

- src/main/resourecs/log4j.xml
  - dtd 변경

http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/xml/doc-files/log4j.dtd

1.5 라이브러리 의존성 설정

- pom.xml
  - version 변경
    - java version 1.8 -> 11
    - org.springframework-version 3.1.1.RELEASE -> 5.3.16
    - org.aspectj-version 1.6 -> 1.9.0
    - log4j version 1.2.15 -> 1.2.17
    - junit version 4.7 -> 4.12
  - 교체
    - servlet-api 2.5 -> 3.1.0
  - 추가
    - spring-test
    - Lombok
    - Jackson

```
<!- 기존의 servlet-api를 교체 -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>
```

```
<!-- spring-test -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-test</artifactId>
<version>${org.springframework-version}</version>
</dependency>

<!-- lombok -->
<dependency>
<groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
<version>1.18.24</version>
<scope>provided</scope>
</dependency>

<!-- jackson -->
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>2.13.2.2</version>
</dependency>
```

1.6 커넥션 풀 설정

- pom.xml
  - HikariCP
  - spring-jdbc (spring-tx 포함)
  - ojdbc8

```
<!-- Database connection pool -->
<dependency>
<groupId>com.zaxxer</groupId>
<artifactId>HikariCP</artifactId>
<version>5.0.1</version>
</dependency>

<!-- spring-jdbc -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-jdbc</artifactId>
<version>${org.sprigframework-version}</version>
</dependency>

<!-- ojdbc8 -->
<dependency>
<groupId>com.oracle.database.jdbc</groupId>
<artifactId>ojdbc8</artifactId>
<version>19.3.0.0</version>
</dependency>
```

- src₩main₩webapp₩WEB-INF₩spring₩root-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
xsi:schemaLocation="http://mybatis.org/schema/mybatis-spring
http://mybatis.org/schema/mybatis-spring-1.2.xsd
http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- datasource connection pool -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
  <property name="jdbcUrl"     value="jdbc:oracle:thin:@127.0.0.1:1521:xe" />
  <property name="username"    value="hr" />
  <property name="password"    value="hr" />
</bean>

<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource"
      destroy-method="close">
  <constructor-arg ref="hikariConfig" />
</bean>

</beans>
```

1.7 Mybatis 설정

- pom.xml
  - mybatis-spring
  - mybatis

```
<!-- mybatis -->
<dependency>
<groupId>org.mybatis</groupId>
<artifactId>mybatis</artifactId>
<version>3.5.9</version>
</dependency>

<!-- mybatis-spring -->
<dependency>
<groupId>org.mybatis</groupId>
<artifactId>mybatis-spring</artifactId>
<version>2.0.6</version>
</dependency>
```

- src₩main₩webapp₩WEB-INF₩spring₩root-context.xml

```
<!-- mybatis  SqlSessionFactory -->
<bean class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource"/>
</bean>

<!– mapper scan -->
<mybatis-spring:scan base-package="com.company.**.mapper" />
```

1.8 sql 로그 설정

- PreparedStatement에서 파라미터가 대입된 쿼리 내용과 실행결과를 볼 수 있다.

```
Console
Tomcat v9.0 Server at localhost [Apache Tomcat] C:₩Program Files₩Java₩jdk-11.0.12₩bin₩javaw.exe  (2022. 7. 21. 오후 1:41:06)
INFO : jdbc.sqlonly - SELECT employee_id, first_name, last_name, email, hire_date, job_id, salary FROM employees
WHERE employee_id = '101'

INFO : jdbc.sqltiming - SELECT employee_id, first_name, last_name, email, hire_date, job_id, salary FROM employees
WHERE employee_id = '101'
 {executed in 0 msec}
INFO : jdbc.resultsettable -
|------------|-----------|----------|---------|--------------------|-------|-------|
|employee_id |first_name |last_name |email    |hire_date           |job_id |salary |
|------------|-----------|----------|---------|--------------------|-------|-------|
|101         |Neena      |Kochhar   |NKOCHHAR |2005-09-21 00:00:00 |AD_VP  |17000  |
|------------|-----------|----------|---------|--------------------|-------|-------|
```

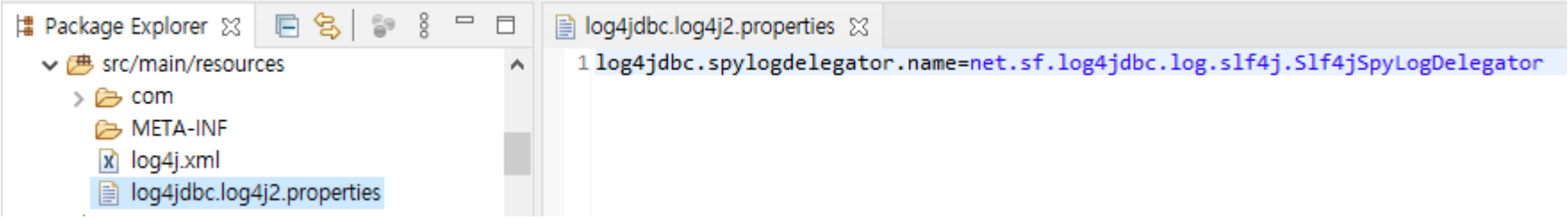
1.8 sql 로그 설정

- pom.xml 라이브러리 추가
  - log4jdbc-log4j2

```
<dependency>
  <groupId>org.bgee.log4jdbc-log4j2</groupId>
  <artifactId>log4jdbc-log4j2-jdbc4.1</artifactId>
  <version>1.16</version>
</dependency>
```

- 로그 설정파일 추가
  - src/main/resources/log4jdbc.log4j2.properties

```
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
```



1.8 sql 로그 설정

- JDBC 드라이버와 URL 정보 수정
  - src/main/webapp/WEB-INF/spring/root-context.xml

```
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
<!–
  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
  <property name="jdbcUrl"     value="jdbc:oracle:thin:@127.0.0.1:1521:xe" />
-->
  <property name="driverClassName" value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy" />
  <property name="jdbcUrl"         value="jdbc:log4jdbc:oracle:thin:@127.0.0.1:1521:xe" />
  <property name="username"        value="hr" />
  <property name="password"        value="hr" />
</bean>
```

1.8 sql 로그 설정

- src/main/resources/log4j.xml
  - 로그 레벨
    - trace < debug < info < warn < error < fetal
    - 지정된 레벨 이하는 출력 안됨
  - 루트 로그 레벨 설정
    - 패키지별 별도 지정이 없으면 루트 레벨을 적용함

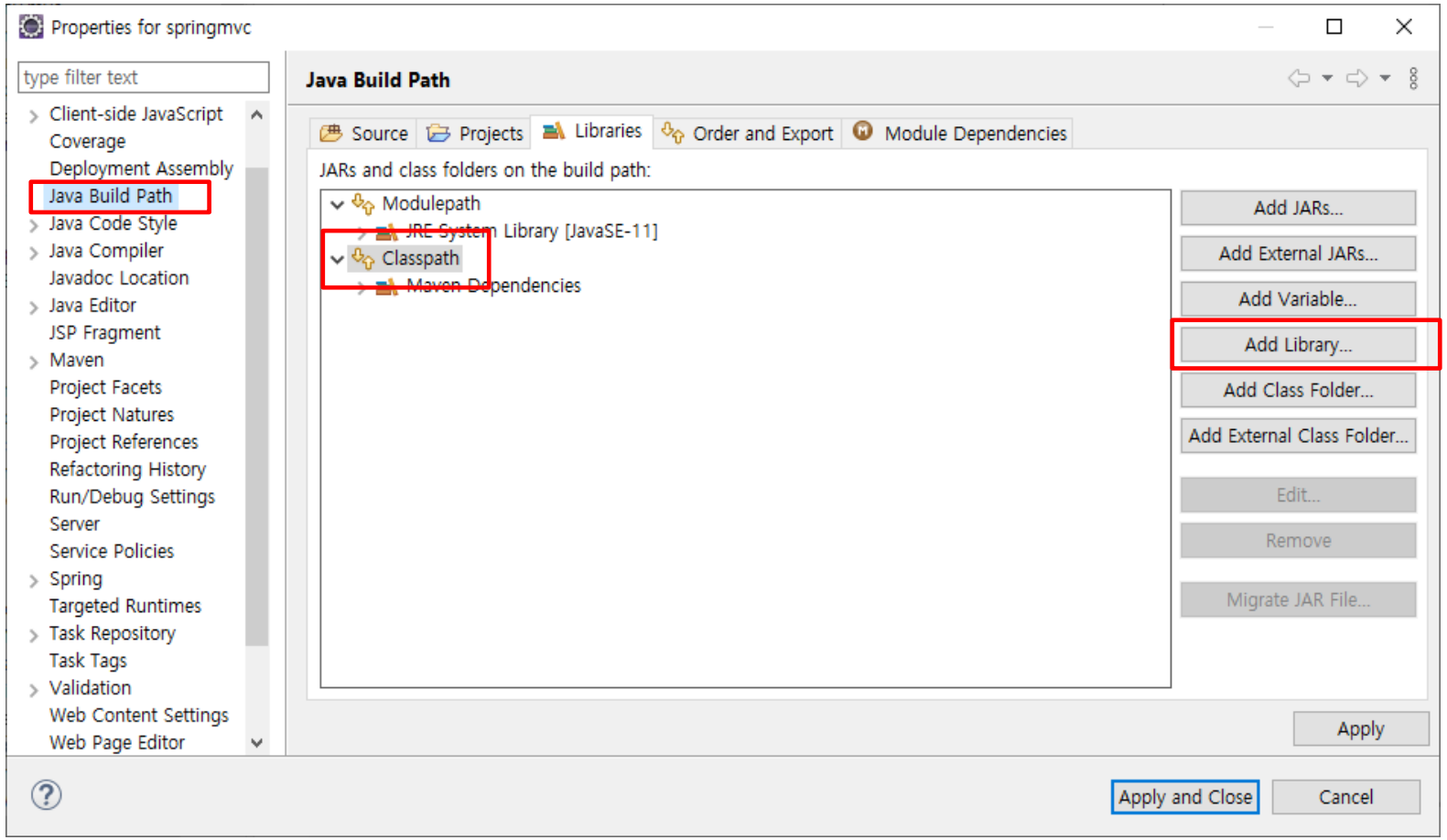
```
<!-- Root Logger -->
<root>
  <priority value="info" />
  <appender-ref ref="console" />
</root>
```

- 패키지별로 로그 레벨 설정

```
<logger name="jdbc.sqlonly">
  <level value="info" />
</logger>

<logger name="jdbc.sqltiming">
  <level value="info" />
</logger>

<logger name="jdbc.resultsettable">
  <level value="info" />
</logger>

<logger name="jdbc.audit">
  <level value="warn" />
</logger>

<logger name="jdbc.resultset">
  <level value="warn" />
</logger>
```

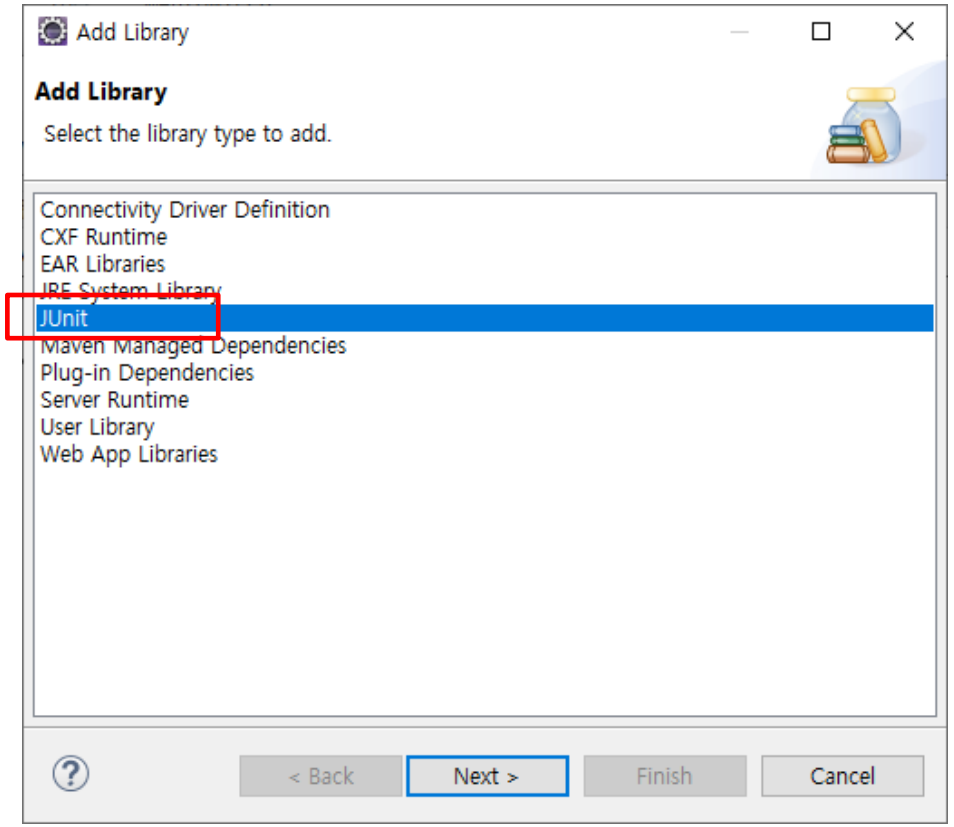
1.9 junit 설정

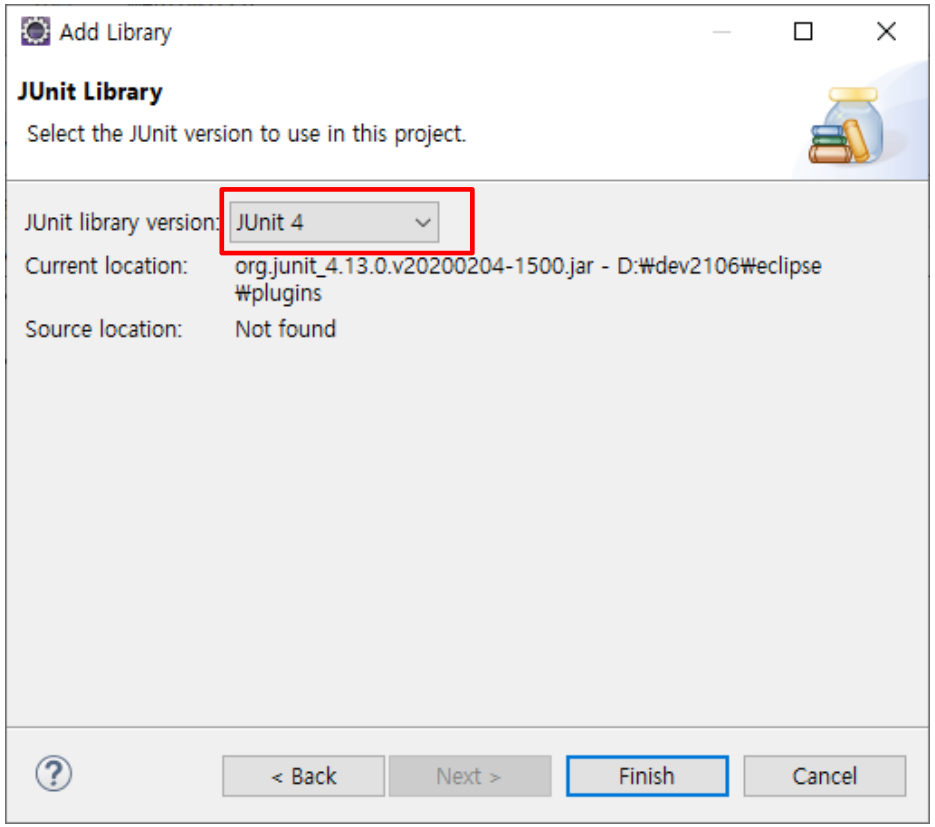
- junit 라이브러리 추가



1.9 junit 설정

- junit 라이브러리 추가





2.1 Mybatis 테스트

- EmpVO

```
@Data
public class EmpVO {
    String employee_id;
    String first_name;
    String last_name;
    String email;
    String hire_date;
    String job_id;
    String department_id;
    String salary;
}
```

- mapper 인터페이스

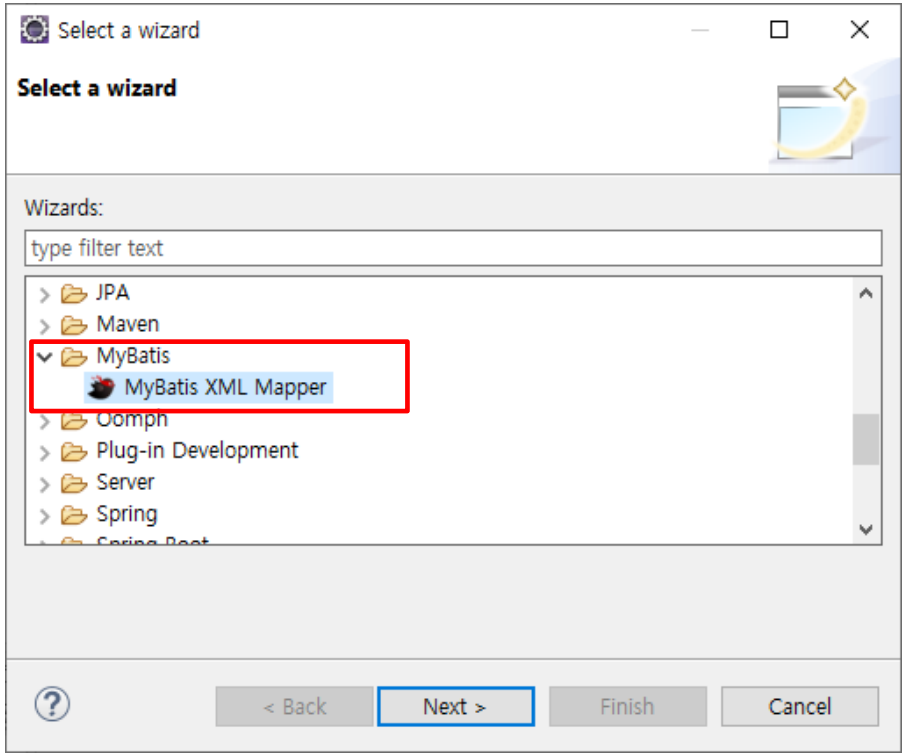
```
public interface EmpMapper {
   public EmpVO getEmp(EmpVO empVO);
}
```

- sql statmement xml 파일

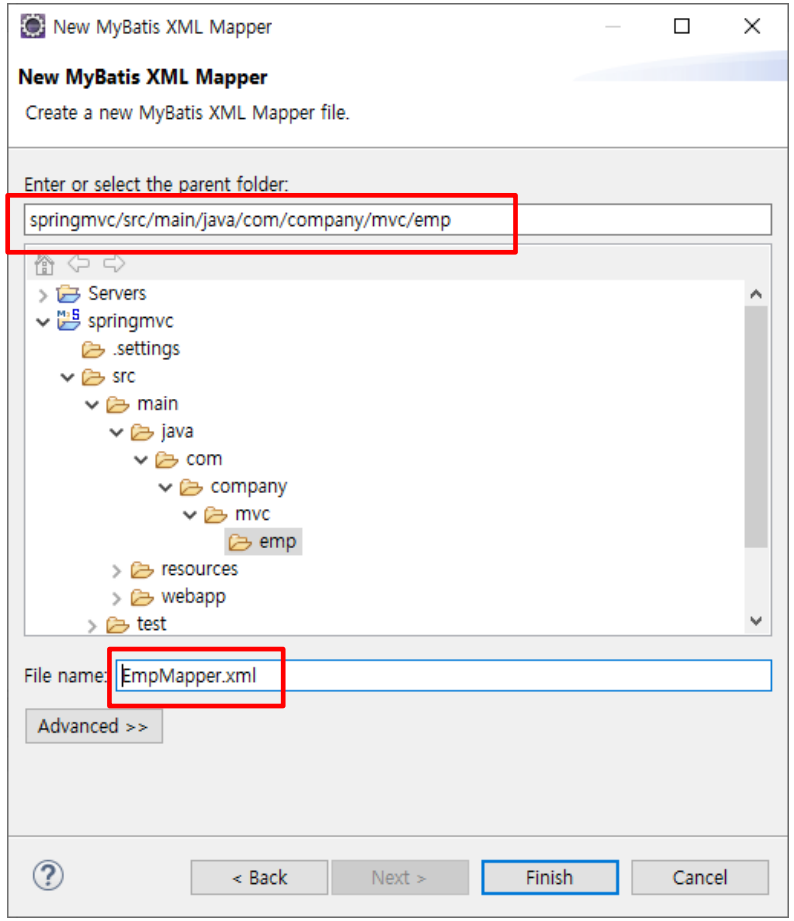
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD
Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper
namespace="com.company.mvc.emp.EmpMapper">
<select id="getEmp"
        parameterType="com.company.mvc.emp.EmpVO"
        resultType="com.company.mvc.emp.EmpVO">
SELECT  employee_id,
        first_name,
        last_name,
        email,
        hire_date,
        job_id,
        salary
  FROM employees
 WHERE employee_id = #{employee_id}
</select>
</mapper>
```

2.1 Mybatis 테스트

- Sql statement xml 파일 생성
  - File -> New -> Other...



- 생성위치와 파일명 입력



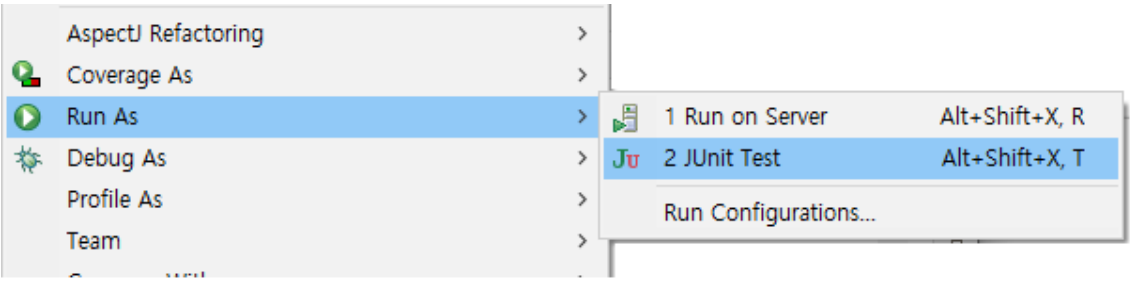
2.1 Mybatis 테스트

- 테스트 코드 작성
  - src/test/java/EmpMapperTest.java

```
package com.company.mvc.emp;

import static org.junit.Assert.assertEquals;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "file:src/main/webapp/WEB-INF/spring/root-context.xml")
public class EmpMapperClient {

    @Autowired EmpMapper empMapper;

    @Test
    public void getEmp() {
        EmpVO vo = new EmpVO();
        vo.setEmployee_id("100");
        EmpVO findVO = empMapper.getEmp(vo);
        System.out.println(findVO.getLast_name());
        assertEquals(findVO.getLast_name(),"King");
    }
}
```

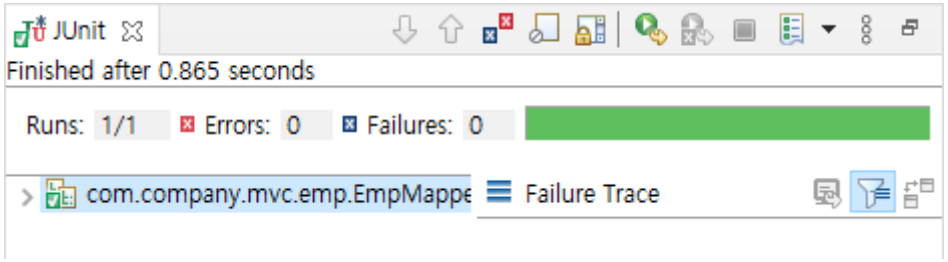
2.1 Mybatis 테스트

- jUnit Test

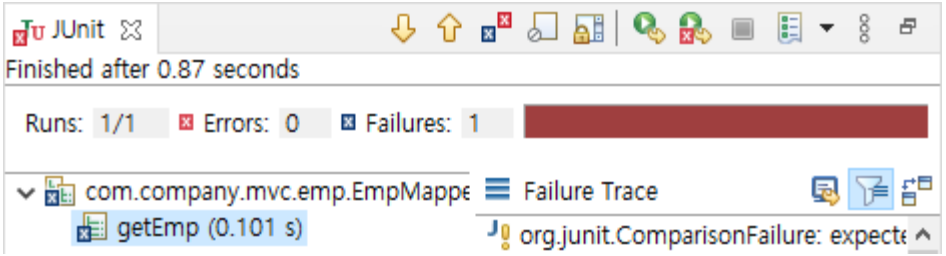


- 실행결과

테스트 성공



테스트 실패



2.2 컨트롤러 테스트

- 컨트롤러 작성
  - src/main/hava/com/company/controller/EmpController.java

```
@Controller
public class EmpController {

  @Autowired EmpMapper empMapper;

  @GetMapping("/emp")
  public String emp(Model model, EmpVO empVO) {
     model.addAttribute("emp", empMapper.getEmp(empVO));
     return "emp";
  }
}
```

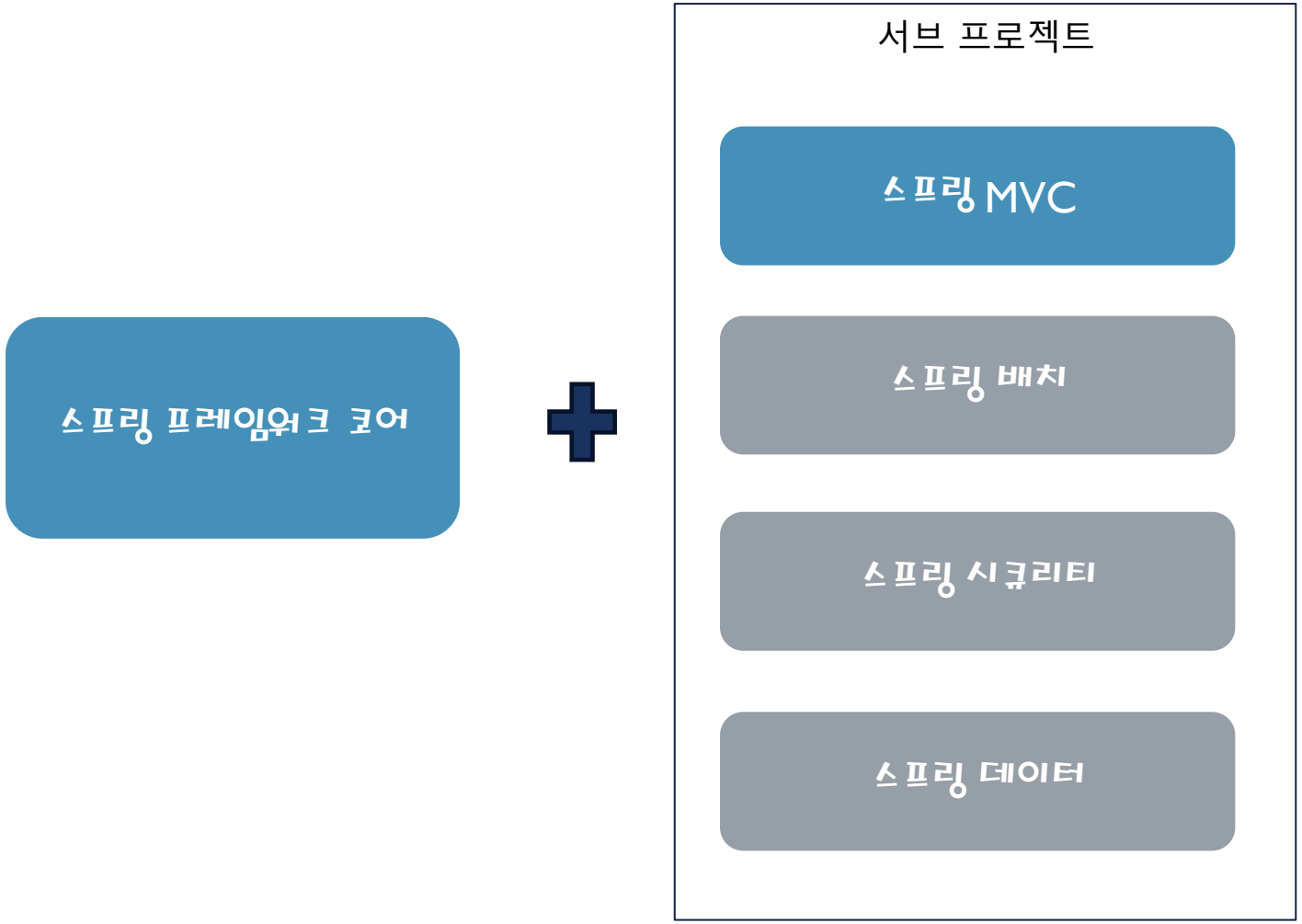
- 뷰페이지 작성
  - src/main/webapp/WEB-INF/views/emp.jsp

```
<body>
<h3>사원조회</h3>
<div>사번: ${emp.employee_id}</div>
<div>이름: ${emp.first_name}</div>
<div>입사일자: ${emp.hire_date}</div>
<div>급여: ${emp.salary}</div>
</body>
```

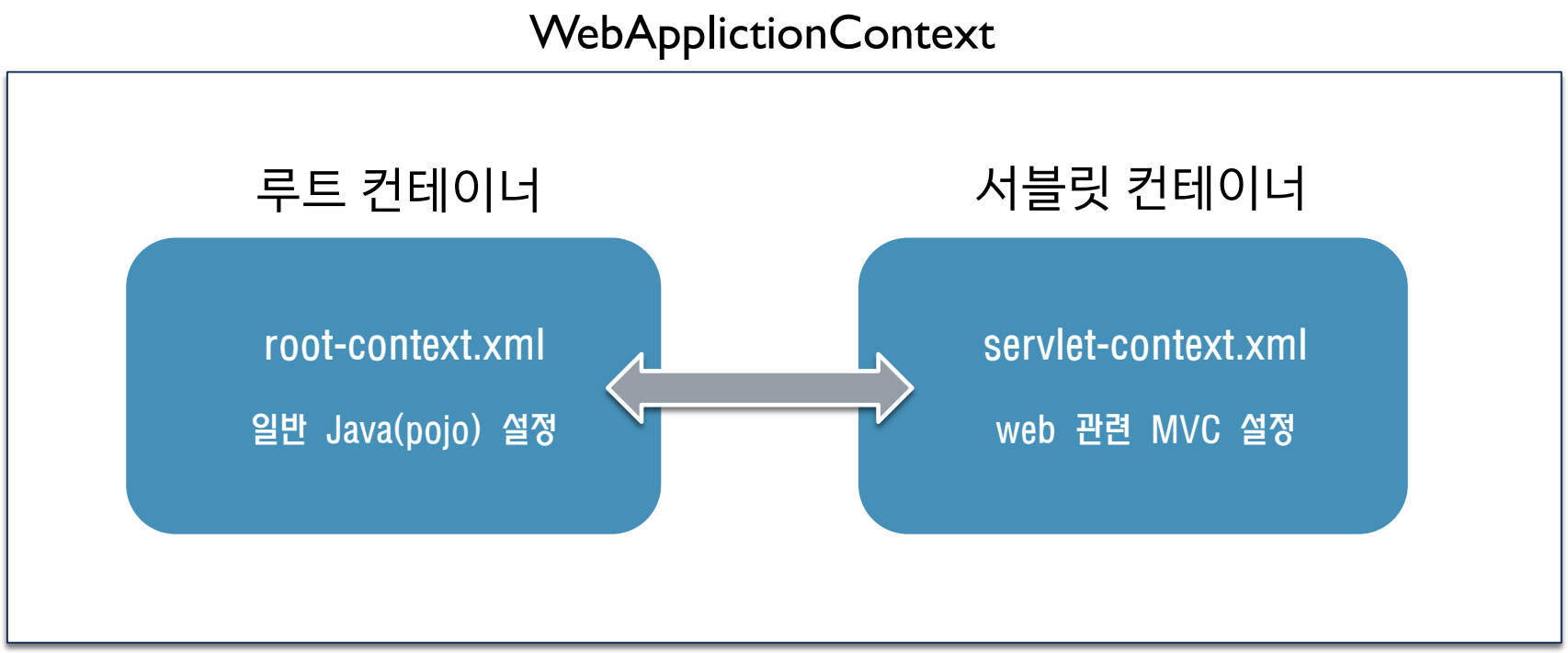
- 테스트
  - tomcat 서버 시작하고 브라우저에서 URL 입력

```
http://localhost/web/emp?employee_id=100
```

3.1 스프링 프레임워크



3.2 스프링 MVC 내부구조



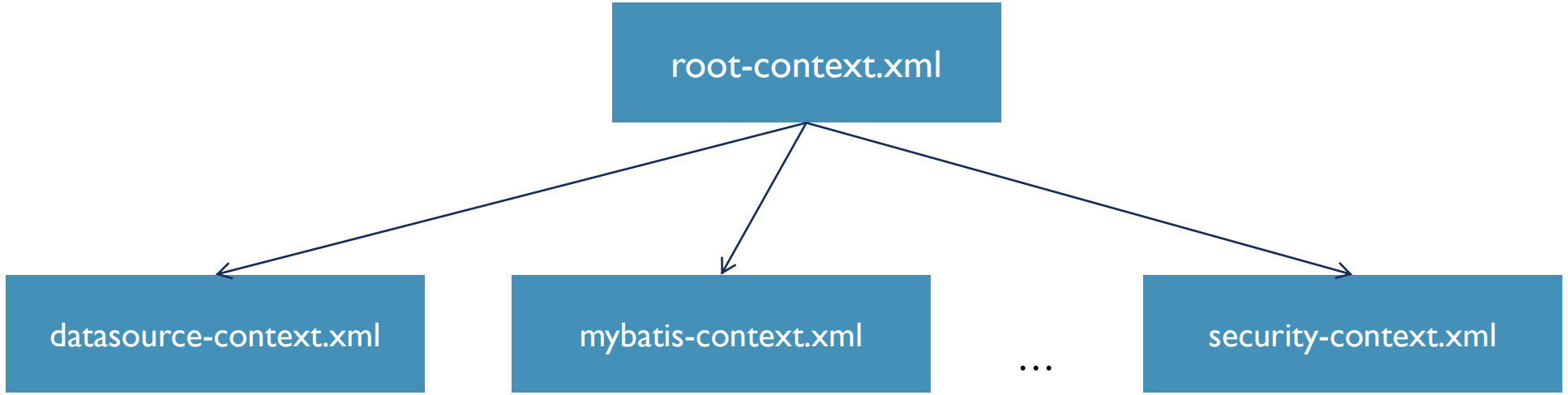
- 스프링이 웹어플리케이션을 목적으로 나온 프레임워크가 아니기 때문에 완전히 분리하고 연동하는 방식으로 구현됨
- root-context.xml에 정의된 객체(Bean)들은 컨텍스트 안에 생성되고 객체들 간의 의존성이 처리되고 나서 스프링 MVC 에서 사용하는 DispatcherServlet 관련 설정이 동작

3.2 스프링 MVC 내부구조

- src/main/webapp/WEB-INF/web.xml

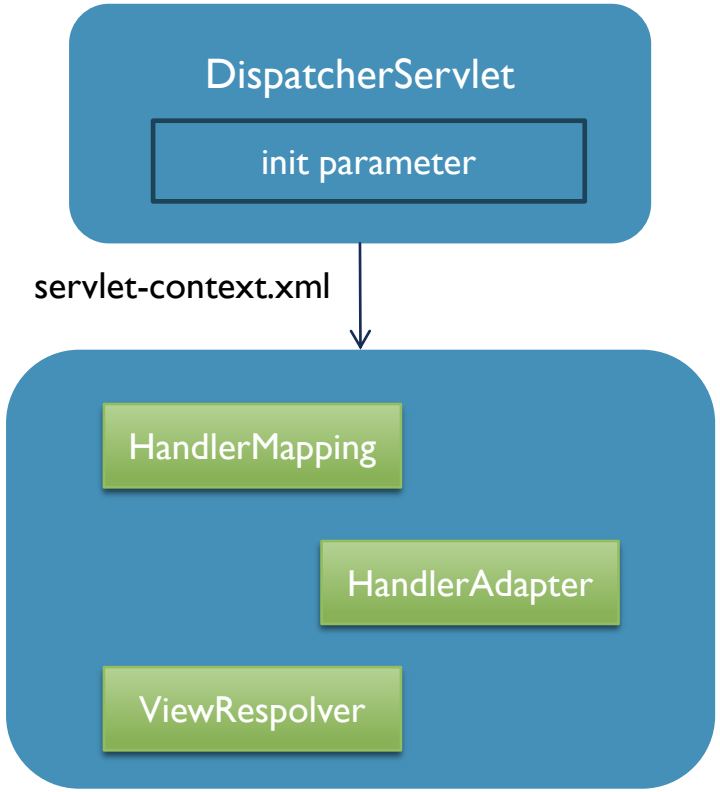
```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- Processes application requests -->
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

3.3 설정파일 분리

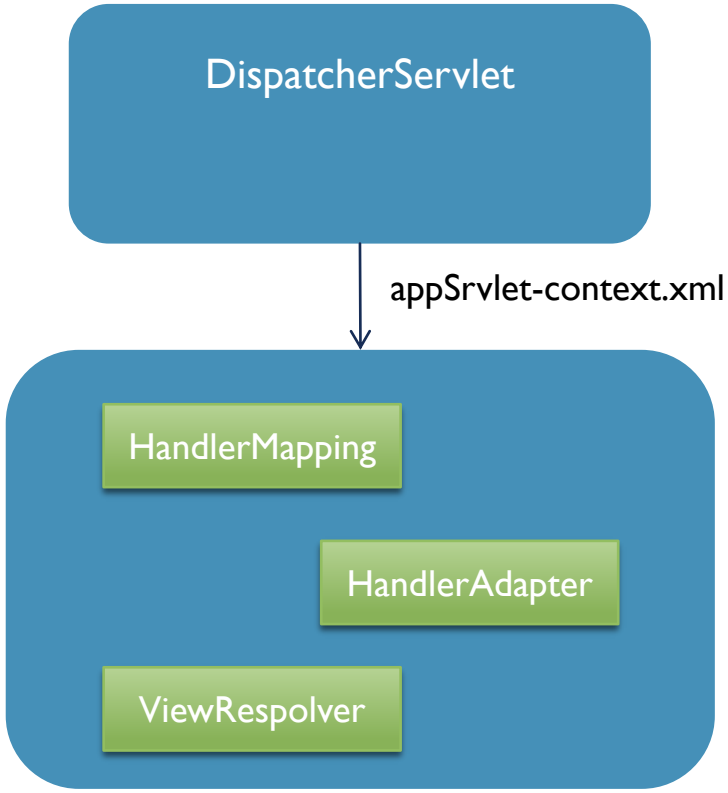



```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/*-context.xml</param-value>
</context-param>
```

3.5 DispatcherServlet



서블릿 컨테이너



서블릿 컨테이너

3.6 servlet-context.xml

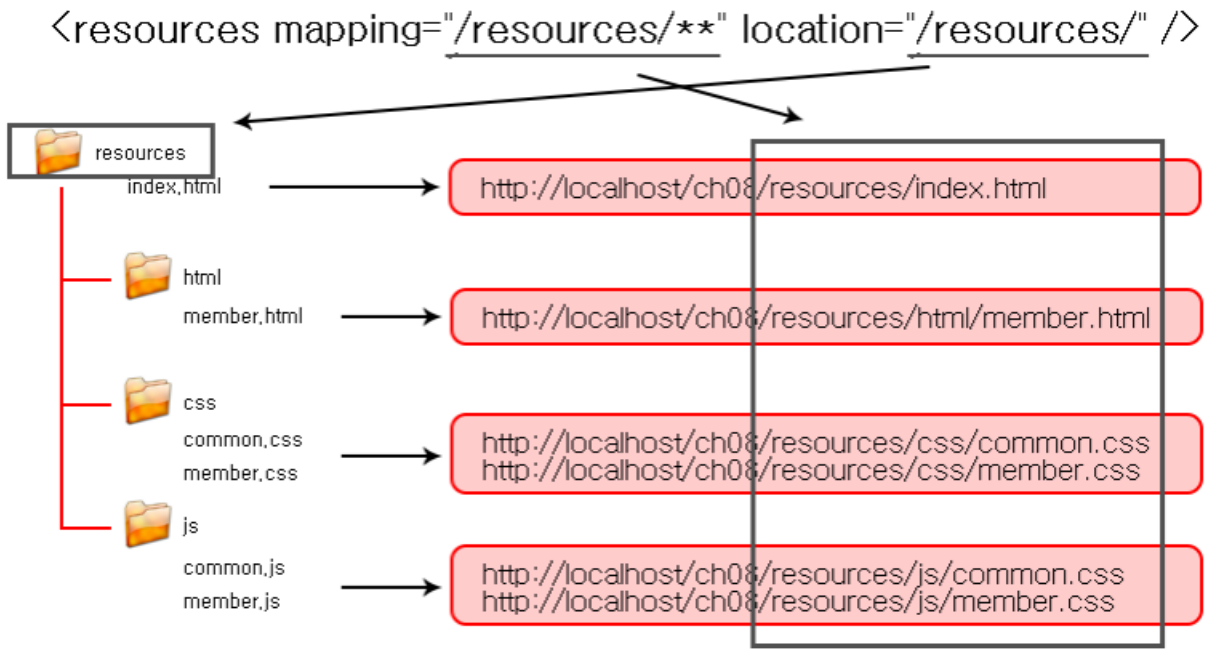
- ViewResolver

```
<!-- Enables the Spring MVC @Controller programming model -->
<annotation-driven />

<!-- Handles HTTP GET requests for /resources/** by efficiently serving
up static resources in the ${webappRoot}/resources directory -->
<resources mapping="/resources/**" location="/resources/" />

<!-- Resolves views selected for rendering by @Controllers to .jsp resourcesin the /WEB-INF/views directory -->
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <beans:property name="prefix" value="/WEB-INF/views/" />
  <beans:property name="suffix" value=".jsp" />
</beans:bean>

<context:component-scan base-package="co.company.mvc" />
```

3.6 servlet-context.xml

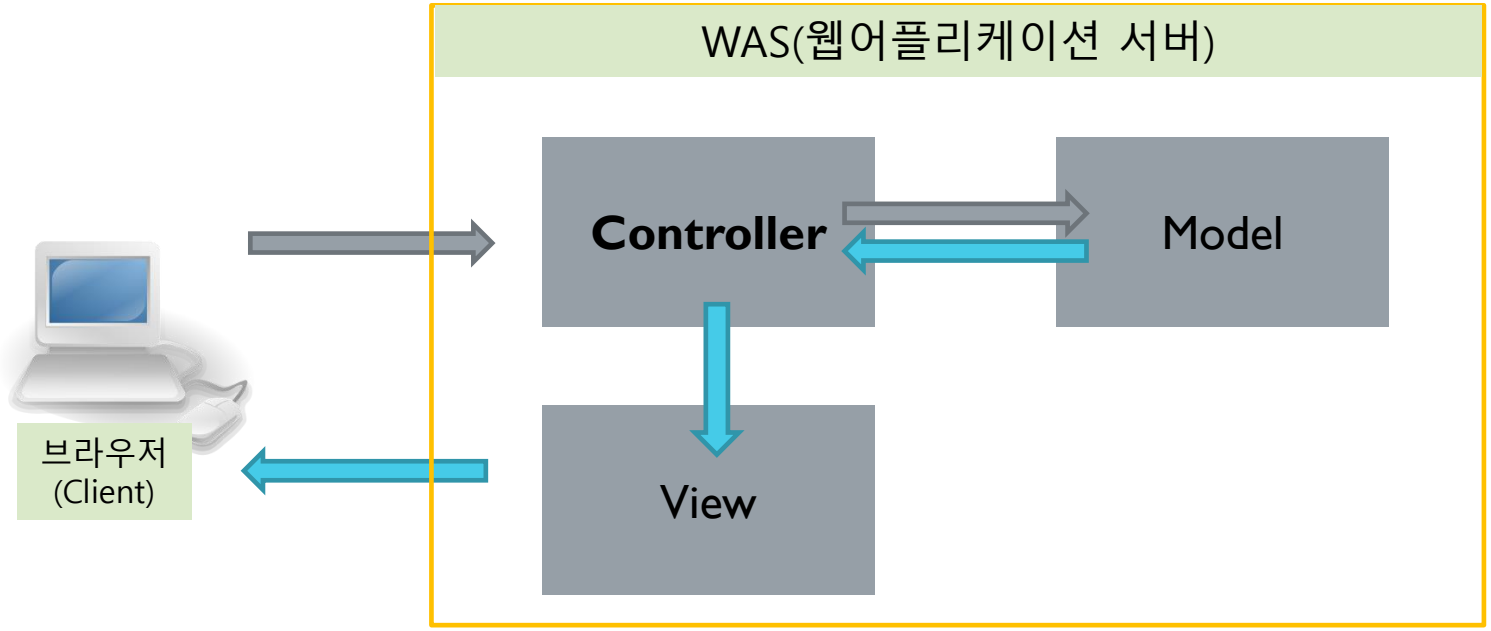
- 정적 리소스 경로 지정

<resources mapping=" " location=" " />



3.7 모델2와 Spring MVC 구조

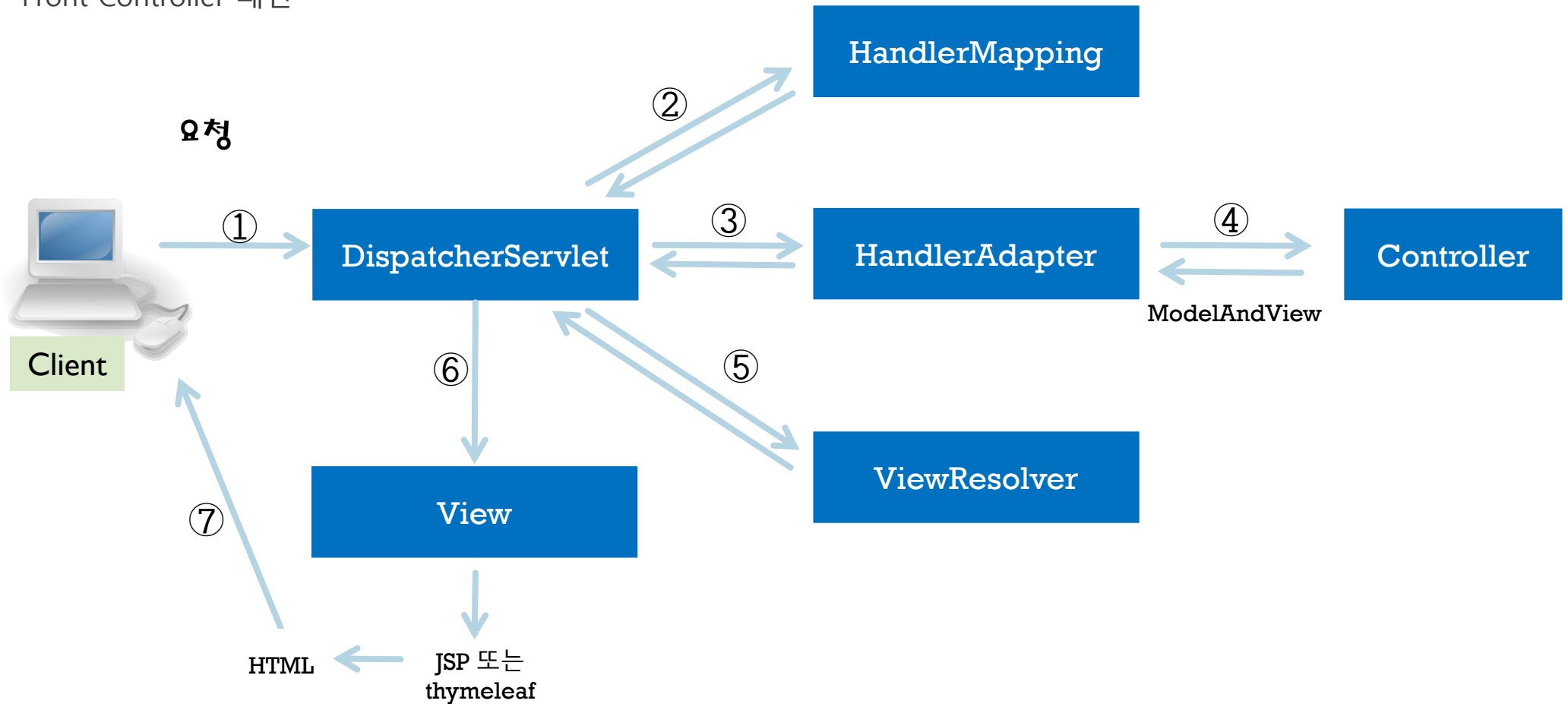
- 모델2 방식
  - 화면과 데이터 처리를 분리해서 재사용이 가능하도록 하는 구조



  - Model: 데이터 혹은 데이터를 처리하는 영역
  - View: 결과화면을 만들어 내는데 사용하는 자원
  - Controller: 웹의 요청(request)를 처리하는 영역으로 뷰와 모델 사이의 중간통신 역할

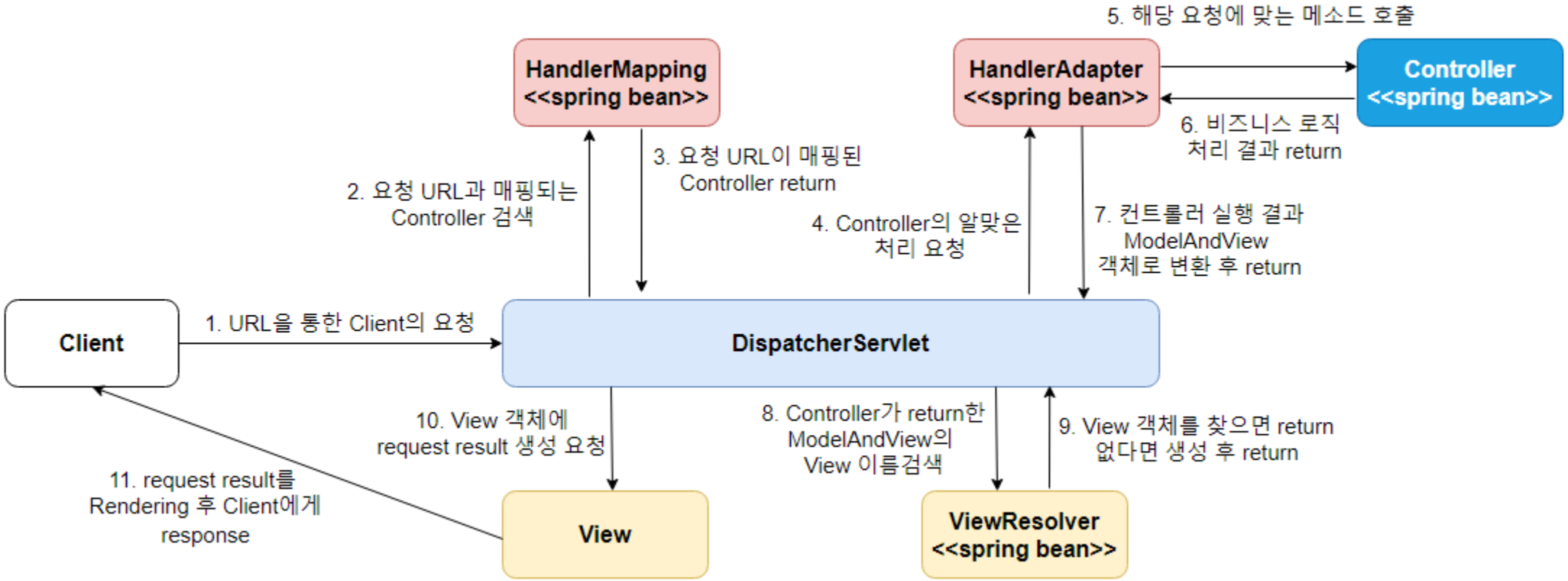
3.7 모델2와 Spring MVC 구조

- 스프링 MVC 구조
  - Front-Controller 패턴



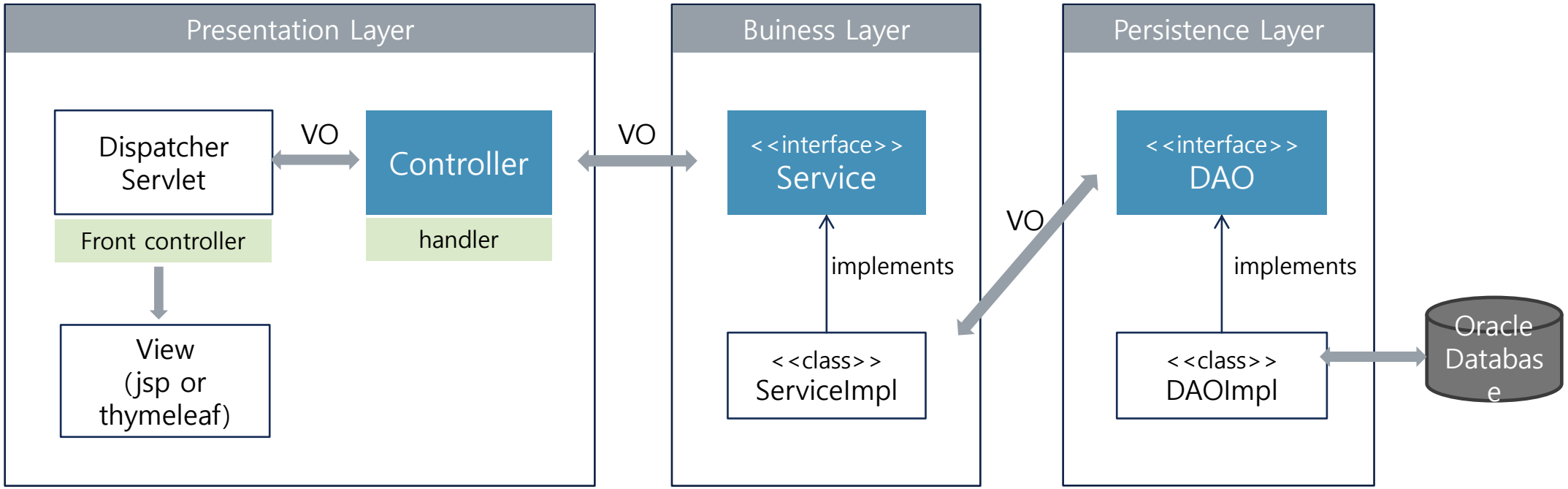
3.7 모델2와 Spring MVC 구조

- 스프링 MVC 구조
  - Front-Controller 패턴



3.8 spring Layer 아키텍쳐

- 3 layer 구조



- 화면흐름결정
- 사용자 입력 값에 대한 검증
- 서비스계층의 호출과 전달되는 값의 포맷의 변화를 처리

- 핵심 업무 로직 구현
- 트랜잭션 처리

- 데이터 CRUD 처리

3.8 spring Layer 아키텍쳐

- 패키지 구성