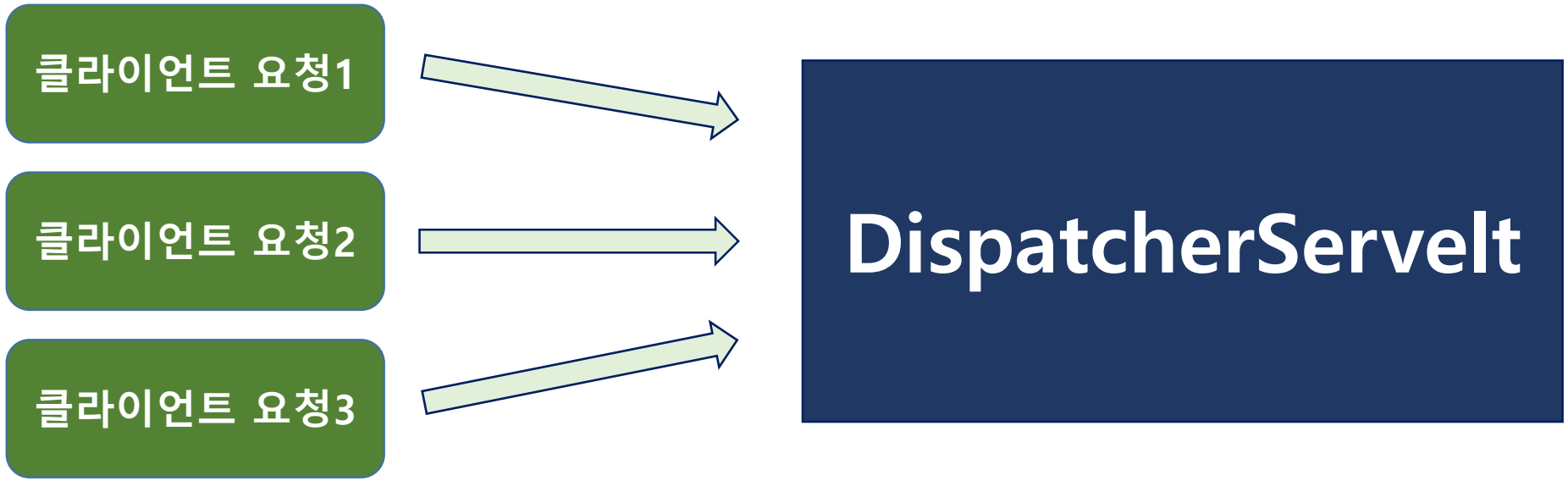
Spring MVC 란

Spring MVC는 Spring에서 제공하는 웹 모듈로 Model, View, Controller 이 세가지 요소를 사용해 사용자의 다양한 요청을 처리하고 응답해주는 프레임워크 이다.
MVC패턴을 사용하면 어플리케이션의 시각적 요소와 비지니스 로직을 분리하여, 서로 영향을 미치지 않게 할 수 있는 어플리케이션을 만들 수 있다.

MVC 패턴의 개념	
모델(Model)	데이터베이스와 연동하여 CRUD 데이터를 처리하는 컴포넌트이다.
뷰(View)	모델이 처리한 데이터 결과를 가지고 사용자에게 보여줄 화면을 만드는 컴포넌트이다.
컨트롤러 (Controller)	클라이언트의 요청을 받아서 요청에 대해 실제 업무를 수행하는 Model에게 작업 수행을 전달하고, 그 결과를 가지고 화면을 생성하도록 뷰에게 전달하는 컴포넌트이다.

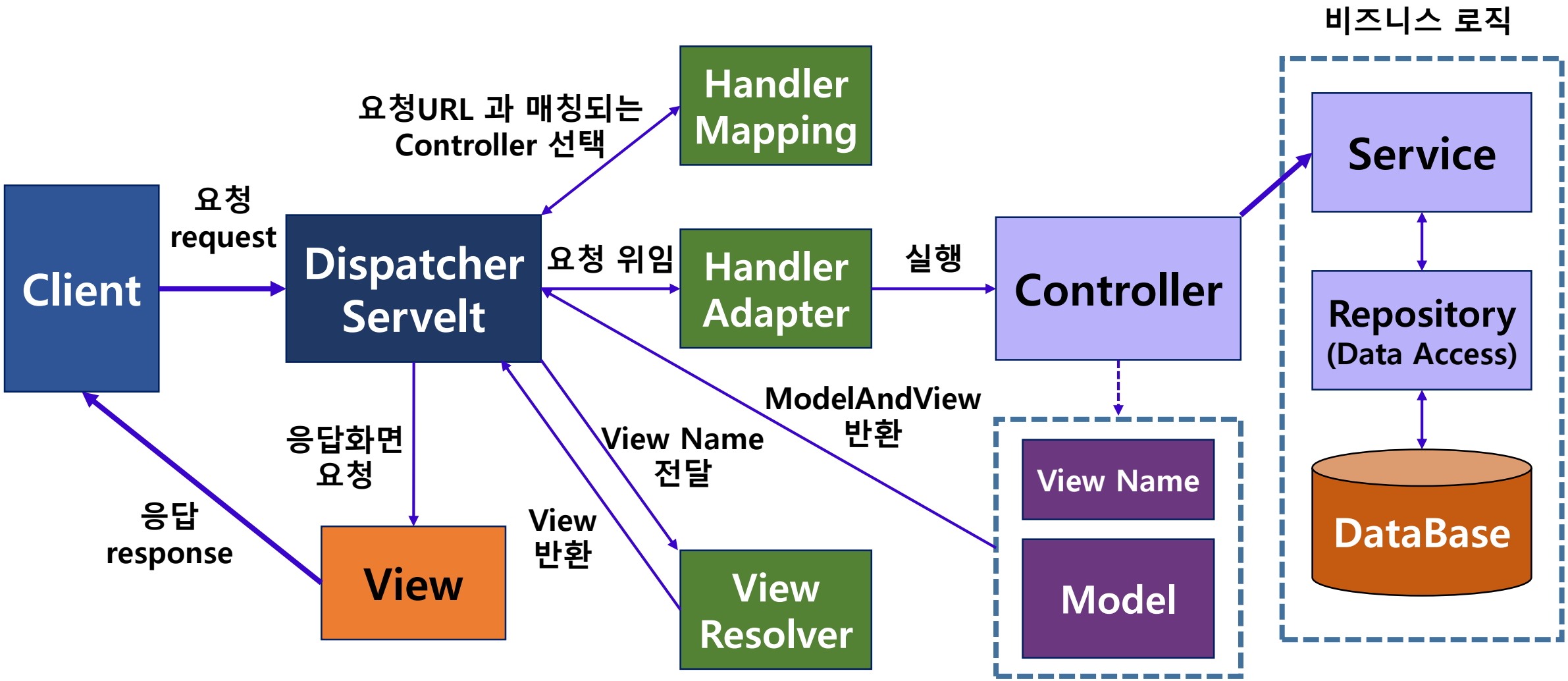
DispatcherServelt



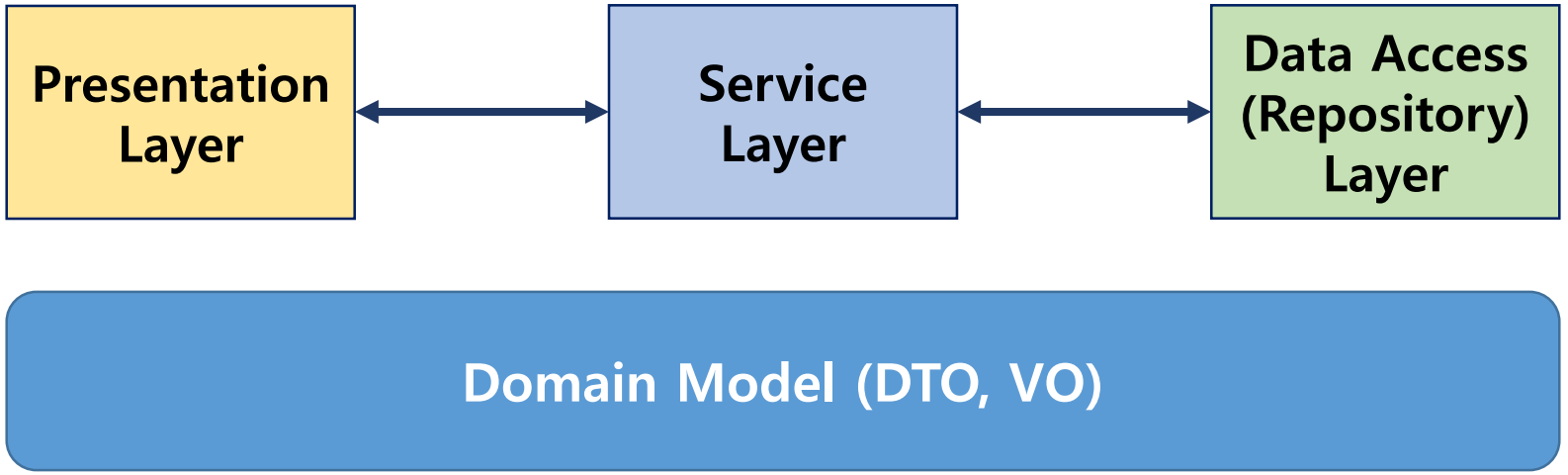
스프링은 클라이언트의 모든 요청을 DispatcherServelt이 받는다.

가장 앞단의 Front Controller 로의 역할을 한다.

Spring MVC 흐름

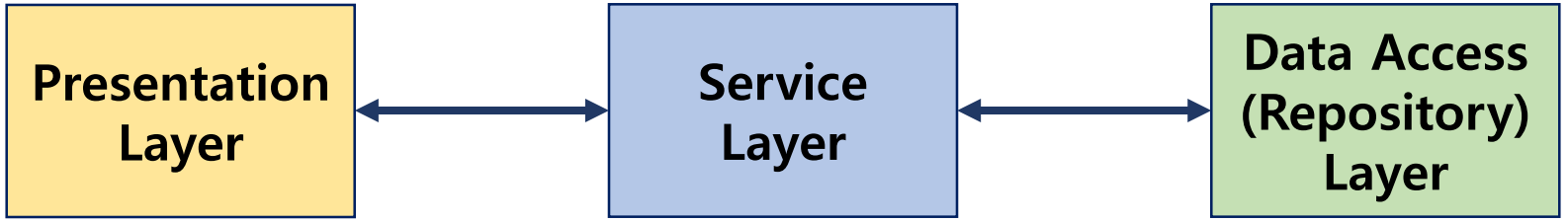


Spring 3가지 계층



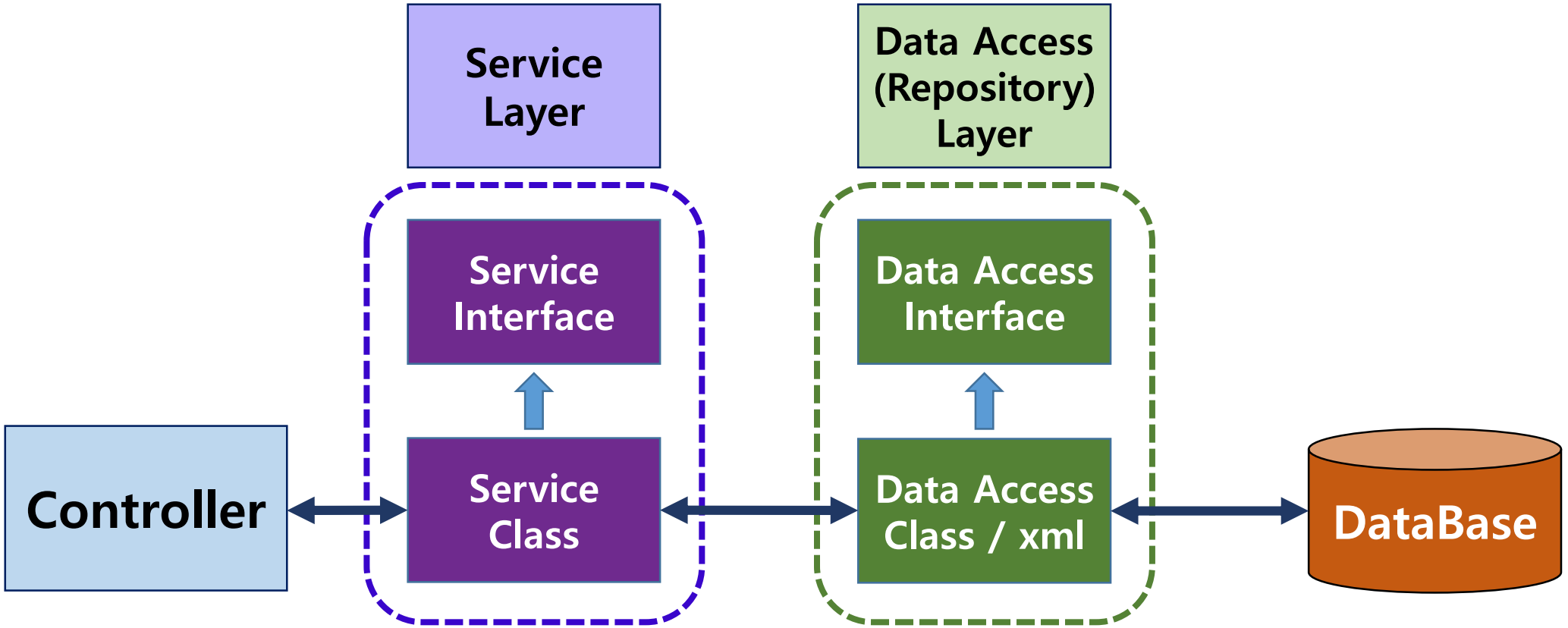
스프링은 프리젠테이션 계층(Presentation Layer) 서비스 계층(Service Layer)
데이터액세스 계층(Data Access Layer) 3가지 계층과
모든 계층에서 사용되는 도메인 모델로 구성되어 진다.

Spring 3가지 계층

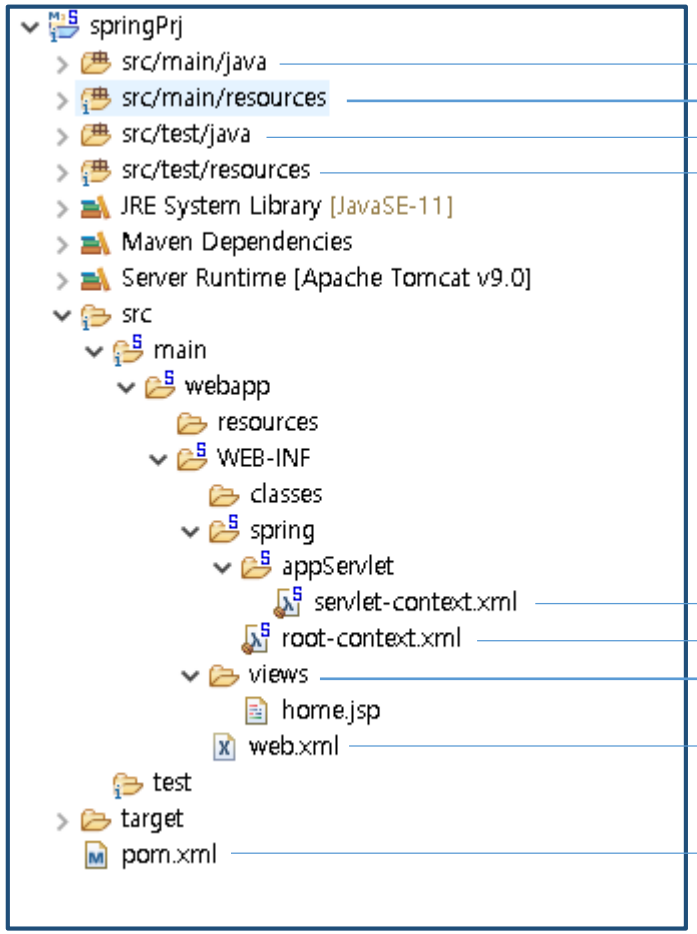


효율적인 개발을 위해 계층구조로 나누는데 Spring은 3가지 계층 구조가 있다.		
프리젠테이션 계층	서비스 계층	데이터 엑세스 계층
- 클라이언트의 요청 및 응답을 처리 - 서비스계층, 데이터 엑세스 계층에서 발생하는 Exception을 처리 - @Controller 어노테이션을 사용	- 비즈니스 로직 처리와 비즈니스와 관련된 도메인 모델의 적합성 검증 트랜잭션 관리 및 처리 - 프리젠테이션 계층과 데이터 엑세스 계층 사이를 연결하는 역할 - @Service 어노테이션을 사용	- ORM (Mybatis, Hibernate)를 주로 사용하는 계층 - Dabase에 data를 CRUD(Create, Read, Update, Delete)하는 계층 - @Repostiory 어노테이션 사용

Service / Data Access 계층



Spring MVC 프로젝트 구조



- JAVA source 파일
- 실행 시 설정(참고) 파일
- JAVA 테스트 파일
- 테스트 관련 설정파일
- 웹 관련 스프링 설정파일
- (DB)관련 스프링 설정파일
- JSP 파일 경로
- 서버(Tomcat)의 web.xml
- Maven이 사용하는 pom.xml

pom.xml 설정

pom.xml 파일에서 spring 버전과 java 버전을 변경해준다.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.java</groupId>
    <artifactId>spring</artifactId>
    <name>Spring_prj</name>
    <packaging>war</packaging>
    <version>1.0.0-BUILD-SNAPSHOT</version>
    <properties>
        <java-version>1.6</java-version>
        <org.springframework-version>3.1.1.RELEASE</org.springframework-
        <org.aspectj-version>1.6.10</org.aspectj-version>
        <org.slf4j-version>1.6.6</org.slf4j-version>
    </properties>
    <dependencies>
        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 ht
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.java</groupId>
    <artifactId>spring</artifactId>
    <name>Spring_prj</name>
    <packaging>war</packaging>
    <version>1.0.0-BUILD-SNAPSHOT</version>
    <properties>
        <java-version>11</java-version>
        <org.springframework-version>5.3.23</org.springframe
        <org.aspectj-version>1.6.10</org.aspectj-version>
        <org.slf4j-version>1.6.6</org.slf4j-version>
    </properties>
    <dependencies>
        <!-- Spring -->
```

pom.xml 설정

Servlet 버전 변경을 해준다.

```xml
<!-- Servlet -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>
```

```xml
<!-- Servlet -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

pom.xml 설정

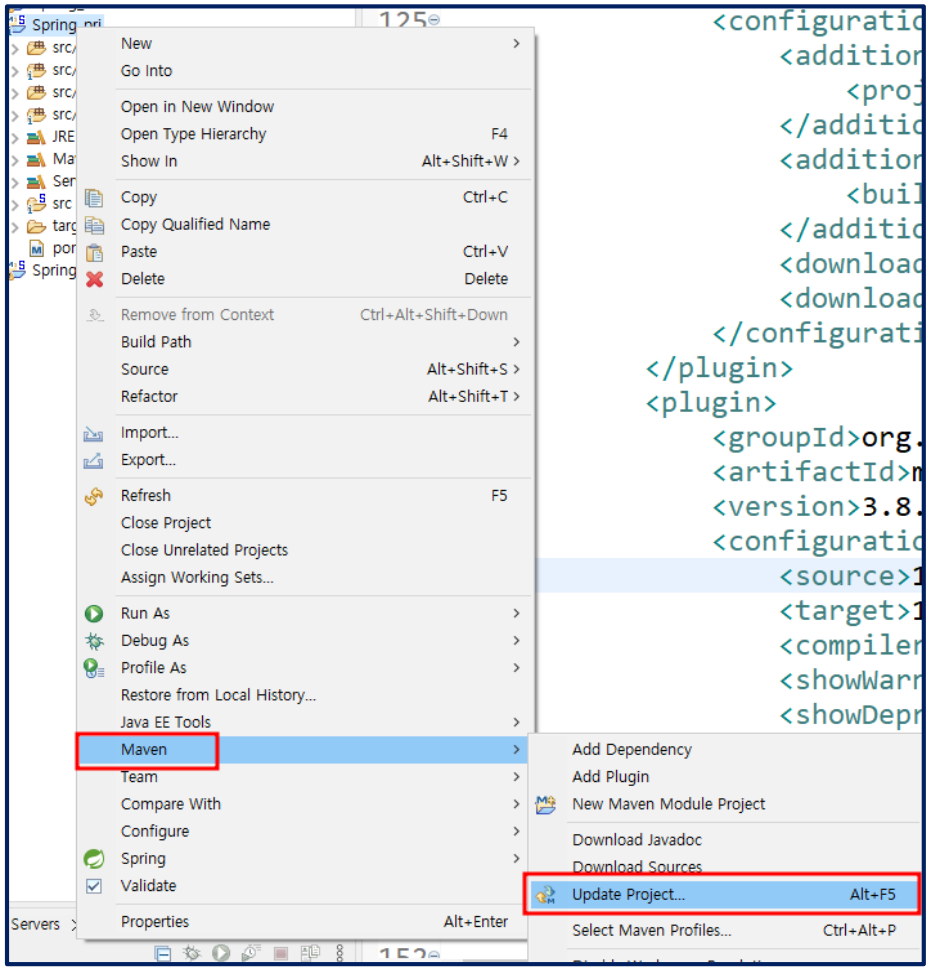
pom.xml 의 maven-compiler-plugin 의 버전과 자바버전 변경을 해준다.

```
129                 <additionalBuildcommands>
130                     <buildcommand>org.springframework.ide.eclipse.core
131                 </additionalBuildcommands>
132                 <downloadSources>true</downloadSources>
133                 <downloadJavadocs>true</downloadJavadocs>
134             </configuration>
135         </plugin>
136         <plugin>
137             <groupId>org.apache.maven.plugins</groupId>
138             <artifactId>maven-compiler-plugin</artifactId>
139             <version>2.5.1</version>
140             <configuration>
141                 <source>1.6</source>
142                 <target>1.6</target>
143                 <compilerArgument>-Xlint:all</compilerArgument>
144                 <showWarnings>true</showWarnings>
145                 <showDeprecation>true</showDeprecation>
146             </configuration>
147         </plugin>
148         <plugin>
149             <groupId>org.codehaus.mojo</groupId>
150             <artifactId>exec-maven-plugin</artifactId>
```

```
130                     <buildcommand>org.springframework.ide.eclipse.core.s
131                 </additionalBuildcommands>
132                 <downloadSources>true</downloadSources>
133                 <downloadJavadocs>true</downloadJavadocs>
134             </configuration>
135         </plugin>
136         <plugin>
137             <groupId>org.apache.maven.plugins</groupId>
138             <artifactId>maven-compiler-plugin</artifactId>
139             <version>3.8.1</version>
140             <configuration>
141                 <source>11</source>
142                 <target>11</target>
143                 <compilerArgument>-Xlint:all</compilerArgument>
144                 <showWarnings>true</showWarnings>
145                 <showDeprecation>true</showDeprecation>
146             </configuration>
147         </plugin>
148         <plugin>
149             <groupId>org.codehaus.mojo</groupId>
150             <artifactId>exec-maven-plugin</artifactId>
```

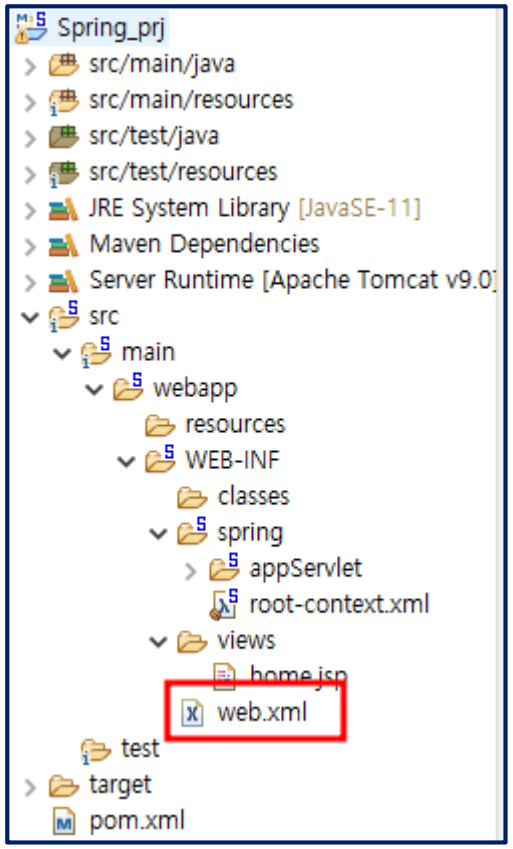
pom.xml 설정

설정을 마쳤으면 pom.xml 에 정의한 라이브러리들을 프로젝트에 적용하기 위해 프로젝트 업데이트를 해주자.



web.xml

web.xml은 웹어플리케이션의 Deployment Descriptor (배포 설명자) 로써 XML형식의 파일이다. 모든 웹어플리케이션은 반드시 하나의 web.xml 파일을 가져야하고 위치는 WEB-INF폴더 아래에있다. web.xml 파일의 설정들은 웹어플리케이션 시작시 메모리에 로딩된다.



web.xml

web.xml에 작성되는 내용
ServletContext 초기 파라미터
Servlet 매핑
리스너 / 필터 설정 등

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee https://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
    <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>

</web-app>
```

web.xml

```
<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>
```

DispatcherServlet 서블릿 등록 설정이다.

url 패턴을 / 로 지정하여 클라이언트의 모든 요청을 DispacherServlet 이 받는다.

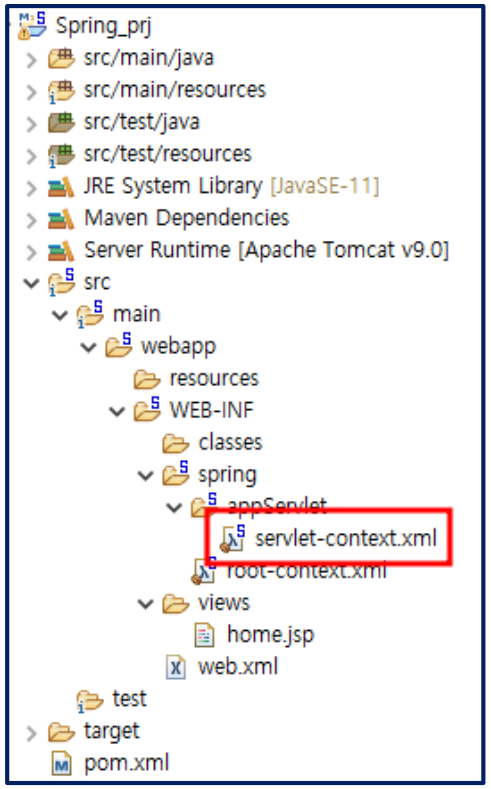
web.xml

한글처리를 위해 web.xml에 추가적인 인코딩 설정을 해주자

```
<!-- 한글 인코딩 필터 설정 -->
<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>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>

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

servlet-context.xml

servlet-context.xml 은 요청과 관련된 객체를 정의한다.
웹 어플리케이션에서 클라이언트의 요청을 받기 위한 컨텍스트 설정이다.




```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc https://www.springframework.or
        http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans
        http://www.springframework.org/schema/context https://www.springframework.org/schema/cor

    <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

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

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF
    <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="com.java.spring" />


</beans:beans>
```

servlet-context.xml

```
<!-- 뷰 리졸버 빈 등록설정 -->
<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>
```

servlet-content.xml 에 뷰 리졸버 빈 등록설정이다.

뷰리졸버는 컨트롤러가 지정한 jsp 파일이름(논리적 이름)을 접두사와 접미사를 붙여주어 클라이언트에게 실제로 응답을 해줄 자원의 경로로 바꾸어 준다.

root-context.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans https://www.springf

    <!-- Root Context: defines shared resources visible to all other web components -->

</beans>
```

root-context.xml 은 모든 웹 구성 요소에 표시되는 공유 리소스를 정의한다.

Repository(DAO), DB등 비즈니스 로직과 관련된 설정을 한다.

Spring (Annotation)

Spring 에서 빈 등록으로 사용되는 대표적인 어노테이션	
@Component	해당 클래스를 스프링의 Bean으로 등록할 때 사용하는 어노테이션. 스프링은 해당 클래스를 스프링의 Bean으로 등록한다.
@Controller	해당 클래스가 Controller의 역할을 명시하는 어노테이션
@Service	비지니스 로직이 들어가는 Service 역할을 명시하는 어노테이션
@Repository	DB연동 작업을 하는 DAO 역할을 명시하는 어노테이션

Spring (Annotation)

클라이언트의 요청에 대해 해당 메소드와 연결시켜주는 어노테이션	
@RequestMapping	클라이언트가 요청한 주소를 메소드와 연결시켜주는 어노테이션
@GetMapping	요청방식이 GET 방식일때 클라이언트가 요청한 주소를 메소드와 연결시켜주는 어노테이션
@PostMapping	요청방식이 POST 방식일때 클라이언트가 요청한 주소를 메소드와 연결시켜주는 어노테이션

Spring (Annotation)

```java
@RequestMapping("/list")
public String list() {
    System.out.println("list() 실행");

    return "page/listPage";
}
```

@RequestMapping : 클라이언트의 요청에 대하여 어떤 Controller 가 실행할지 어떤 메소드를 실행하여 처리할지 맵핑하기 위한 어노테이션 이다.

예) /list 로 클라이언트의 요청이 들어오면 해당 list() 메소드를 실행시켜준다.

[GET / POST] 요청방식 모두 받을 수 있다.

Spring (Annotation)

```java
@RequestMapping(value = "/list", method = RequestMethod.GET)
public String list() {
    System.out.println("list() 실행");

    return "page/listPage";
}
```

```java
@RequestMapping(value = "/list", method = RequestMethod.POST)
public String list() {
    System.out.println("list() 실행");

    return "page/listPage";
}
```

@RequestMapping 어노테이션은 클라이언트의 요청방식에 따라 해당 메소드를 실행할 수 있게 한다. value="요청을 받을 url" 을 넣어준다.
method=RequestMethod.요청방식(GET, POST) 로 작성을 해준다.

Spring (Annotation)

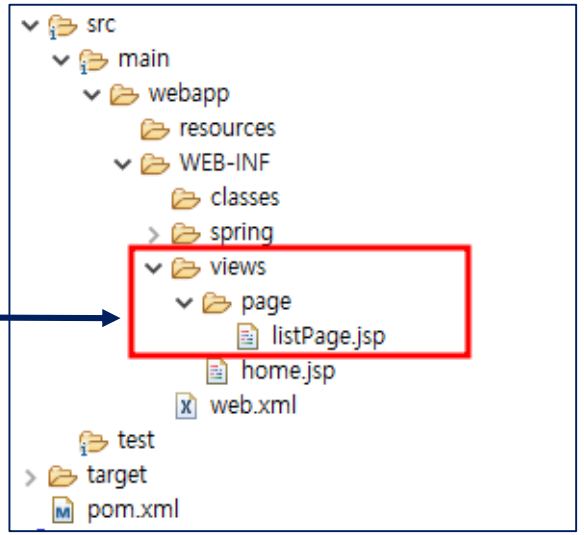
```java
@GetMapping("/list")
public String list() {
    System.out.println("list() 실행");

    return "page/listPage";
}
```

```java
@PostMapping("/list")
public String list() {
    System.out.println("list() 실행");

    return "page/listPage";
}
```

@RequestMapping 어노테이션으로 사용하여 클라이언트의 해당 요청을 받을 메소드를 지정할 수 있지만, 간편하게 요청방식에 따라서 클라이언트의 요청을 받는 @GetMapping 어노테이션과 @PostMapping 어노테이션이 있다.

Controller View 페이지 지정

```
@RequestMapping(value = "/list", method = RequestMethod.GET)
public String list() {
    System.out.println("list() 실행");

    return "page/listPage";
}
```



컨트롤러의 메소드는 클라이언트의 요청을 받아 로직등을 처리한 다음 클라이언트의 요청결과에 응답해주기 위한 JSP 페이지를 지정한다. JSP 페이지를 지정하는 방식은 반환타입을 String 타입으로 하여 문자열로 JSP 페이지를 지정한다. WEB-INF 폴더아래 views 폴더 다음부터의 경로를 문자열로 지정을 해주면 된다.

Controller View 페이지 지정

Controller에서 문자열로 /WEB-INF/views 폴더안에 있는 확장자를 제외한 jsp 파일의 경로를 문자열로 DispatcherServlet 에게 전송한다.
DispatcherServlet은 ViewResolver에게 전달받은 View 정보[이름]를 전달한다.
ViewResolver는 응답할 View에 대한 JSP를 찾아 DispatcherServlet 에게 전달하고 최종적으로 클라이언트에게 View를 응답하여 요청을 마친다.

```
<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>
```

```
return "home";
```

```
/WEB-INF/views/home.jsp
```

폼 데이터 받기

```java
@RequestMapping(value="/data", method=RequestMethod.POST)
public String paramData(HttpServletRequest request) throws Exception{

    request.setCharacterEncoding("UTF-8");

    String name = request.getParameter("name");
    String[] hobby = request.getParameterValues("hobby");
```

컨트롤러의 메소드에 HttpServletRequest 매개변수를 작성하여 넘어온 데이터를 getParameter, getParameterValues 메소드를 사용하여 데이터를 받을 수 있다.

폼 데이터 받기

```
이름 : <input type="text" name="name"><br>
나이 : <input type="number" name="age"><br>
```

```
@RequestMapping(value="/data", method=RequestMethod.POST)
public String paramData(@RequestParam("name")String name,
                        @RequestParam("age")int age) throws Exception{

    System.out.println("이름 : " + name);
    System.out.println("나이 : " + age);
```

@RequestParam 어노테이션을 사용하여 폼에서 넘어온 데이터를 받을 수 있다. 폼에서 넘어오는 파라미터의 이름을 문자열로 작성하여 해당하는 파라미터의 Value 값을 매개변수에 바인딩 해주어 간편하게 데이터를 받을 수 있다.

폼 데이터 받기

```
@RequestMapping(value="/data1", method=RequestMethod.POST)
public String paramData02(@RequestParam(value="name", required = false, defaultValue = "홍길동") String name,
                          @RequestParam(value="age", required = false, defaultValue = "20") Integer age ) {
```

@RequestParam 어노테이션에 추가적인 속성을 작성할 수 있다.	
value 속성	문자열로 넘어오는 파라미터 이름을 지정한다.
required 속성	속성값을 false 로 작성을 하면 넘어오는 파라미터 데이터가 없는 경우에 예외를 발생시키지 않고 null 값으로 저장이 되어진다.
defaultValue 속성	문자열로 기본값을 지정할 수 있다.

폼 데이터 받기

```java
public class PageMember {

    private String name;
    private Integer age;

    public PageMember() {}

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

}
```

```java
@RequestMapping(value="/data", method=RequestMethod.POST)
public String paramData(PageMember member) throws Exception{

    System.out.println("이름 : " + member.getName());
    System.out.println("나이 : " + member.getAge());
```

Command 객체를 통하여 폼에서 넘어온 데이터를 받을 수 있다. 메소드의 매개변수로 Command 타입의 변수를 작성하면 해당하는 Command 객체의 Setter 메소드를 호출하여 데이터를 자동으로 바인딩 해준다.

넘어오는 파라미터의 이름과 일치하는 프로퍼티의 Setter 메소드를 호출시켜 바인딩하기 때문에 이름이 꼭 일치되어야 한다.

Model 객체

```java
@PostMapping("/data")
public String paramData(Model model) {

    model.addAttribute("name", "홍길동");
    model.addAttribute("age", 20);

    return "/param/paramPrint";
}
```

```jsp
<title>Insert title here</title>
</head>
<body>

    이름 : ${name}<br>
    나이 : ${age}<br>

</body>
</html>
```

Model : 서버에서 생성한 데이터를 담아 JSP에 전달하는 역할을 하는 객체이다.

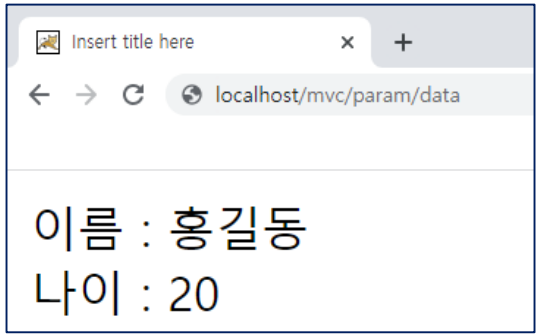
Controller의 메소드의 Model 매개변수를 선언해주면 Spring 은 자동으로 Model 객체를 생성하여 바인딩 해준다. addAttribute 메소드로 데이터를 저장한다.

JSP 에서는 EL 표현식으로 Model 데이터 안에 저장된 객체를 사용할 수 있다.

@ModelAttribute

```java
@PostMapping("/data")
public String paramData02( @ModelAttribute("name") String name,
                           @ModelAttribute("age") Integer age,
                           Model model) {

    return "/param/paramPrint";

}
```

```
이름 : ${name}<br>
나이 : ${age}<br>
```



@ModelAttribute : 전달받은 파라미터의 값을 Model 객체에 바인딩까지 해준다.

@ModelAttribute

```java
@PostMapping("/data")
public String paramData03(@ModelAttribute("member") PageMember member, Model model) {

    return "/param/paramPrint";
}
```

```
이름 : ${member.name }<br>
나이 : ${member.age }<br>
```

이름 : 홍길동
나이 : 20

파라미터 값을 바인딩한 Command 객체를 Model 객체에 바인딩까지 해준다.

ModelAndView 객체

```java
@PostMapping("/data")
public ModelAndView paramData(PageMember member) {

    ModelAndView mv = new ModelAndView();
    mv.setViewName("/param/paramPrint");
    mv.addObject("member", member);

    return mv;
}
```

ModelAndView : View 로 전송할 데이터와 JSP Page 를 같이 지정을 하는 객체
반환타입은 ModelAndView 로 지정을 한다. 메소드에서 ModelAndView 객체를 선언 및 생성을 한 후 View의 이름을 setViewName 메소드로 설정하고 전송할 데이터를 addObject 메소드로 설정 한 후 ModelAndView 객체를 반환한다.