JAVA 프레임워크 SPRING을 이용한 회원관리 시스템

장안대학교
인터넷정보통신과

Framework 란?

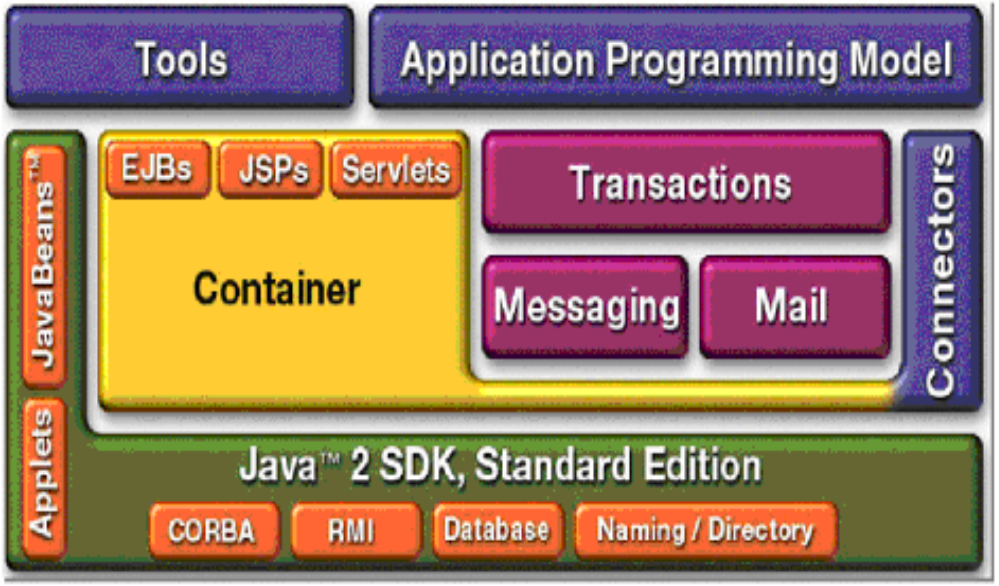
- Framework
  - 뼈대, 뼈대 공사, 골격
  - 랄프 존슨(Ralph Johnson) 교수
    - 소프트웨어의 구체적인 부분에 해당하는 설계와 구현을 재사용이 가능하게끔 일련의 협업화된 형태로 클래스들을 제공하는 것
  - 소프트웨어의 특정문제 해결 또는 소프트웨어 제작을 편리하게 할 수 있도록 미리 뼈대를 이루는 클래스와 인터페이스를 제작하여 모아둔 것
  - 특정문제를 해결하기 위한 디자인 패턴들을 모아 놓고 적용하기 쉽도록 구성하여 개발자들이 쉽게 문제를 해결할 수 있도록 뼈대를 제공해주는 것

Enterprice Application

- **Enterprice Application 란?**
  - 기업의 비즈니스활동에 사용되는 응용프로그램
  - 기업용 응용 프로그램은 복잡한 확장성, 분산, 컴포넌트 기반에서 작동.
  - 기업용 응용 프로그램은 매우 복잡한 시스템이며 데이터 중심, 사용자 친화적, 보안, 관리 및 유지 보수에 대한 엄격한 요구 사항을 충족해야 함.
- 웹 기반 어플리케이션에서 '구조가 복잡한 대규모 분산 객체 환경'을 쉽게 구현하기 위해서 자바 EJB(Enterprise JavaBean)가 퍼지기 시작했고 자바는 엔터프라이즈 어플리케이션을 구축하는 데 필요한 기본 기술이 됨

J2EE(Java 2 Enterprise Edition)

- 클라이언트나 서버 환경에서 서버 단에서 수행되는 프로그램을 JAVA로 구현할 때 쓰는 기술
- 엔터프라이즈 어플리케이션은 표준화되고 모듈화된 컴포넌트로 구성되며 J2EE는 서비스 개발에 필요한 완벽한 컴포넌트 군을 제공하여
- 개발자에게 복잡한 개발 과정 없이 해당 컴포넌트를 자동적으로 처리해주는 환경을 제공
- J2EE은 멀티티어(multi-tier) 엔터프라이즈 어플리케이션의 개발을 위한 엔터프라이즈 환경 표준



스트럿츠(STRUTS) framework

- 스트럿츠(STRUTS) - 지주, 버팀목, 받침대.
- Java EE 웹 애플리케이션을 개발하기 위한 오픈 소스 프레임워크
- 자바를 기반으로 한 JSP만을 위한 프레임워크이며 따라서 운영체제에 구애받지 않고 독립된 플랫폼 사용하며, 오픈 소스이므로 개발에 필요한 부분만을 수정하여 사용 할 수 있음.
- 스트럿츠 프레임워크는 MVC(Model-View-Controller) 패턴 기반을 이용하여 개발
  - Model - 로직을 처리 해주는 부분
  - View - 사용자에게 보여주는 부분
  - Controller 부분 - Model 부분과 View 부분을 연결하고 제어
- 크레이그 맥클라나한(Craig McClanahan) 에 의해 최초로 만들어 짐

Spring framework

- Rod Johnson이 2002년에 발표한 Expert One-on-One J2EE Design and Developement에 선보인 코드를 기반으로 만들어진 프레임워크를 오픈소스 생태계를 통해 효과적으로 검증하고 발전시킨 결과물
- 2003년 6월에 최초로 아파치 2.0 라이선스로 공개됨

- 어플리케이션 프레임워크는 애플리케이션 개발을 빠르고 효율적으로 할 수 있도록 애플리케이션의 바탕이 되는 틀과 공통 프로그래밍 모델, 기술 API 등을 제공 하는 것.
- 자바 엔터프라이즈(Enterprise) 애플리케이션(Application) 개발에 사용되는 경량형 어플리케이션 프레임워크(Framework)
- J2EE(Java 2 Enterprise Edition) 에서 제공하는 대부분의 기능을 지원하기 때문에J2EE를 대체하는 프레임워크로 자리매김

- 자바를 기반으로 한 기술이며 객체지향 설계와 구현에 관해 객체를 어떻게 효과적으로 설계하고 구현하고 사용하고, 이를 개선해 나갈 것인가에 대한 모델과 기법 기준제공

Spring framework 특징

- **1. 스프링은 경량 컨테이너이다.**
스프링은 자바 객체를 담고 있는 컨테이너이다. 스프링은 이들 자바 객체의 생성, 소멸과 같은 라이프 사이클을 관리하며, 스프링으로부터 필요한 객체를 가져와 사용할 수 있다.

2. 스프링은 DI(Dependency Injection)패턴을 지원한다.
스프링은 설정파일을 통해서 객체간의 의존관계를 설정할 수 있도록 하고 있다. 따라서 객체는 직접 의존하고 있는 객체를 생성하거나 검색할 필요가 없다.

3. 스프링은 AOP(Aspect Oriented Programming)를 지원한다.
스프링은 자체적으로 AOP를 지원하고 있기 때문에 트랜잭션이나 로깅, 보안과 같이 여러 모듈에서 공통으로 필요로 하지만 실제 모듈의 핵심은 아닌 기능들을 분리해서 각 모듈에 적용할 수 있다.

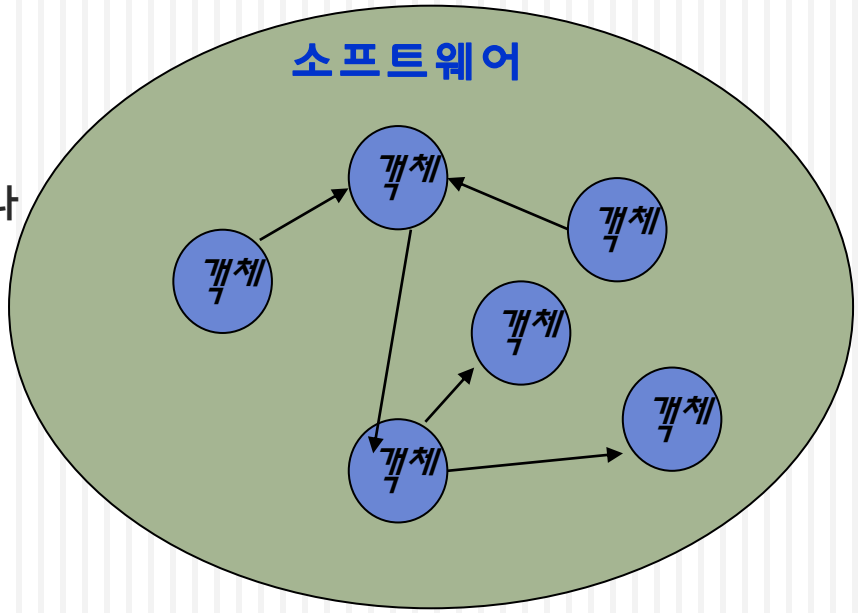
4. 스프링은 POJO(Plain Old Java Object)를 지원한다.
스프링 컨테이너에 저장되는 자바 객체는 특정한 인터페이스를 구현하거나 클래스를 상속받지 않아도 된다. 따라서 기존에 작성한 코드를 수정할 필요 없이 스프링에서 사용할 수 있다.

- **5. 트랜잭션 처리를 위한 일관된 방법을 제공한다.**
JDBC를 사용하든, JTA를 사용하든, 또는 컨테이너가 제공하는 트랜잭션을 사용하든, 설정 파일을 통해 트랜잭션 관련 정보를 입력하기 때문에, 트랜잭션 구현에 상관없이 동일한 코드를 여러 환경에서 사용할 수 있다.

6. 영속성과 관련된 다양한 API를 지원한다.
스프링은 JDBC를 비롯하여 iBatis, 하이버네이트, JPA, JDO등 데이터베이스 처리와 관련하여 널리 사용되는 라이브러리와의 연동을 지원하고 있다.

7. 다양한 API에 대한 연동을 지원한다.
스프링은 JMS, 메일, 스케쥴링 등 엔터프라이즈 어플리케이션을 개발하는데 필요한 다양한 API를 설정파일을 통해서 손쉽게 사용할 수 있도록 하고 있다.

객체 지향(Object-Oriented)과 의존관계

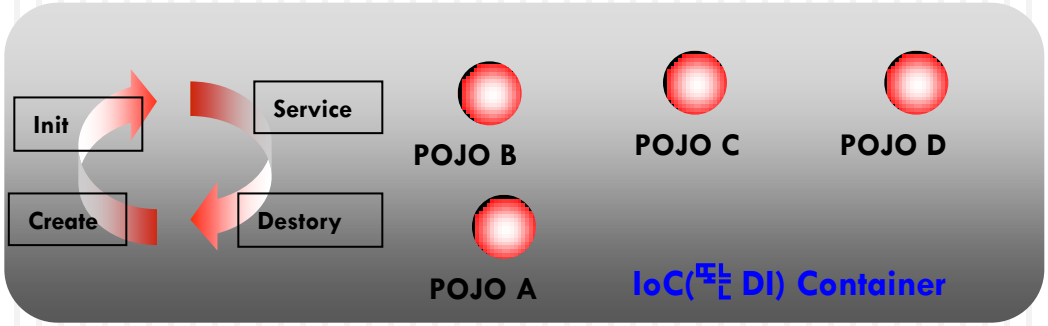


- 소프트웨어는 객체의 모임으로 이루어진다.
- 객체 - 컴포넌트 , java bean
- 하나의 클래스(객체)는 다른 객체를 사용하거나 포함하는 등의 관계를 가지고 작동한다.
- 클래스가 다른 클래스를 사용하는 관계를 의존관계(Dependency)라 한다.
- 의존관계에서는 사용되는 클래스가 변경되면 사용하는 클래스까지 영향을 받는다.

- 유지보수- 사용자의 비즈니스 프로세스변경에 따라 요구사항은 끊임없이 바뀌고 발전한다.
- 유지 보수시에 변화의 폭을 최소화 하는 것은 중요한 문제이다.
- 관심사의 분리 – 관심이 같은 것끼리는 하나의 객체 안으로 또는 친한 객체로 모이게 하고 관심이 다른 것은 가능한 따로 떨어져 서로 영향을 주지 않도록 분리하는 것

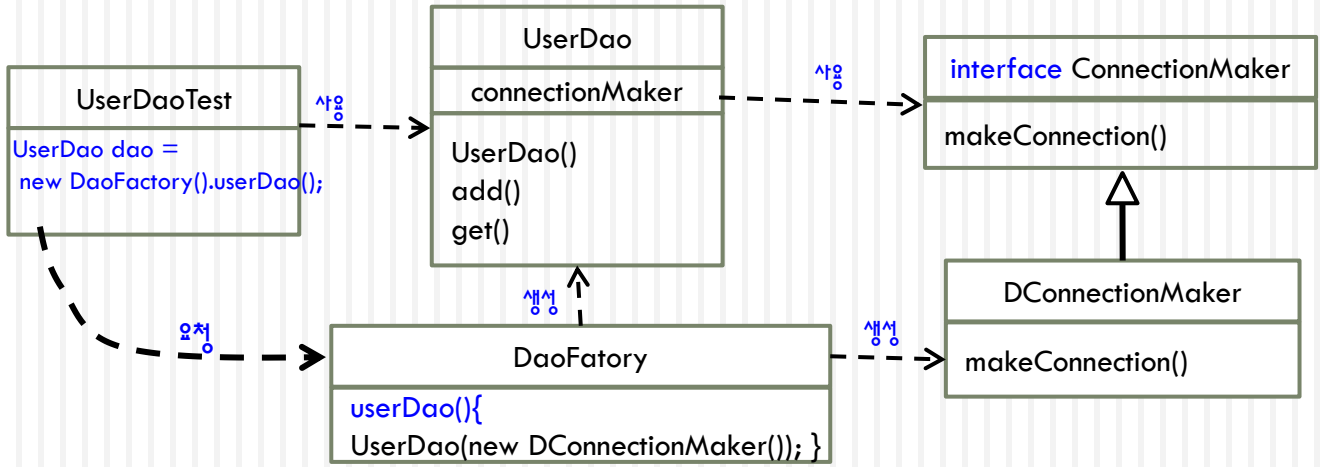
IoC(Spring ,DI) Container

- **Ioc (Inversion of Control) 컨테이너 :제어의 역전**
  - 개발자가 객체(POJO)를 생성하고 각 객체간의 의존관계를 제어하였으나 Spring(IoC) 컨테이너가 객체의 생성, 초기화, 서비스 소멸에 관한 모든 제어 권한을 가지면서 객체의 생명주기를 관리.
  - 개발자들이 직접 POJO를 생성할 수도 있지만, 모든 권한을 Container에게 위임
  - Transaction, Security 추가적인 기능을 제공한다. AOP 기능을 이용하여 새로운 Container 기능을 추가 가능
- **DI(Dependency Injection) : 의존관계 주입**
  - 변화의 폭을 최소화하도록 클래스 사이의 의존관계를 빈 설정(Bean Definition)정보를 바탕으로 컨테이너가 자동적으로 연결해주는 것.



Framework 관련 용어

- **Refactoring :**
  - 기능에는 변화없이 내부구조를 변경해서 재구성하는 작업 – 코드 내부설계 개선
- **전략 패턴(strategy pattern)**
  - 변경이 필요한 알고리즘 부분을 인터페이스로 만들어 외부로 분리시키고 그 구현 클래스의 알고리즘을 필요에 따라 바꿔서 사용
  - 클래스가 변경되어도 그 기능을 사용하는 클래스의 코드는 같이 수정할 필요가 없어지게하는 디자인패턴



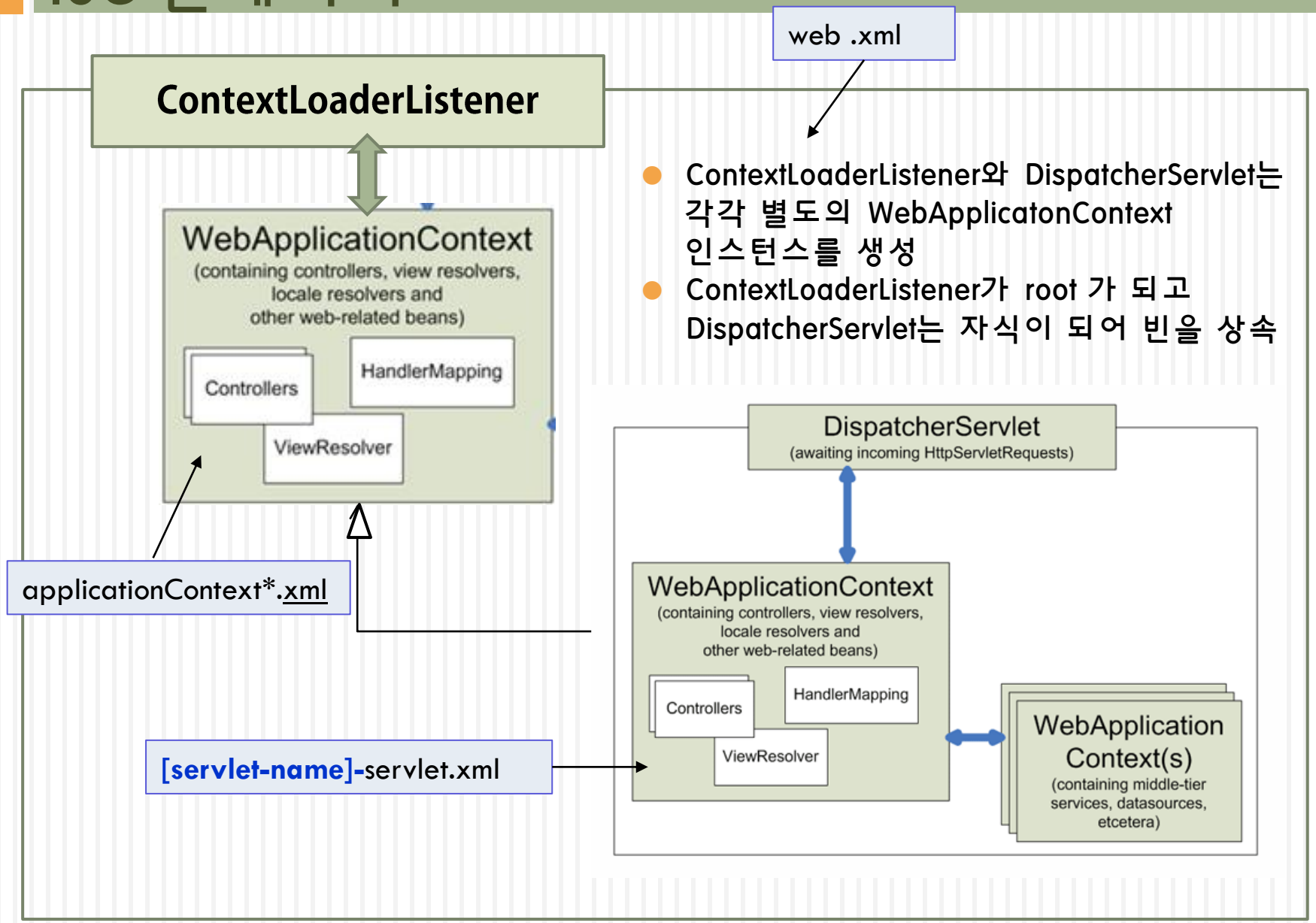
팩토리 클래스에서 UserDao 객체와 연결 객체 의존관계 설정

- **팩토리(factory):** 객체의 생성 방법을 결정하고 그 방법에 따라 만들어진 객체를 반환하는 객체
- **개방 폐쇄의 원칙** – 클래스나 모듈은 확장에는 열려 있어야 하고 변경에는 닫혀 있어야 한다.

Application Context(애플리케이션 컨텍스트)

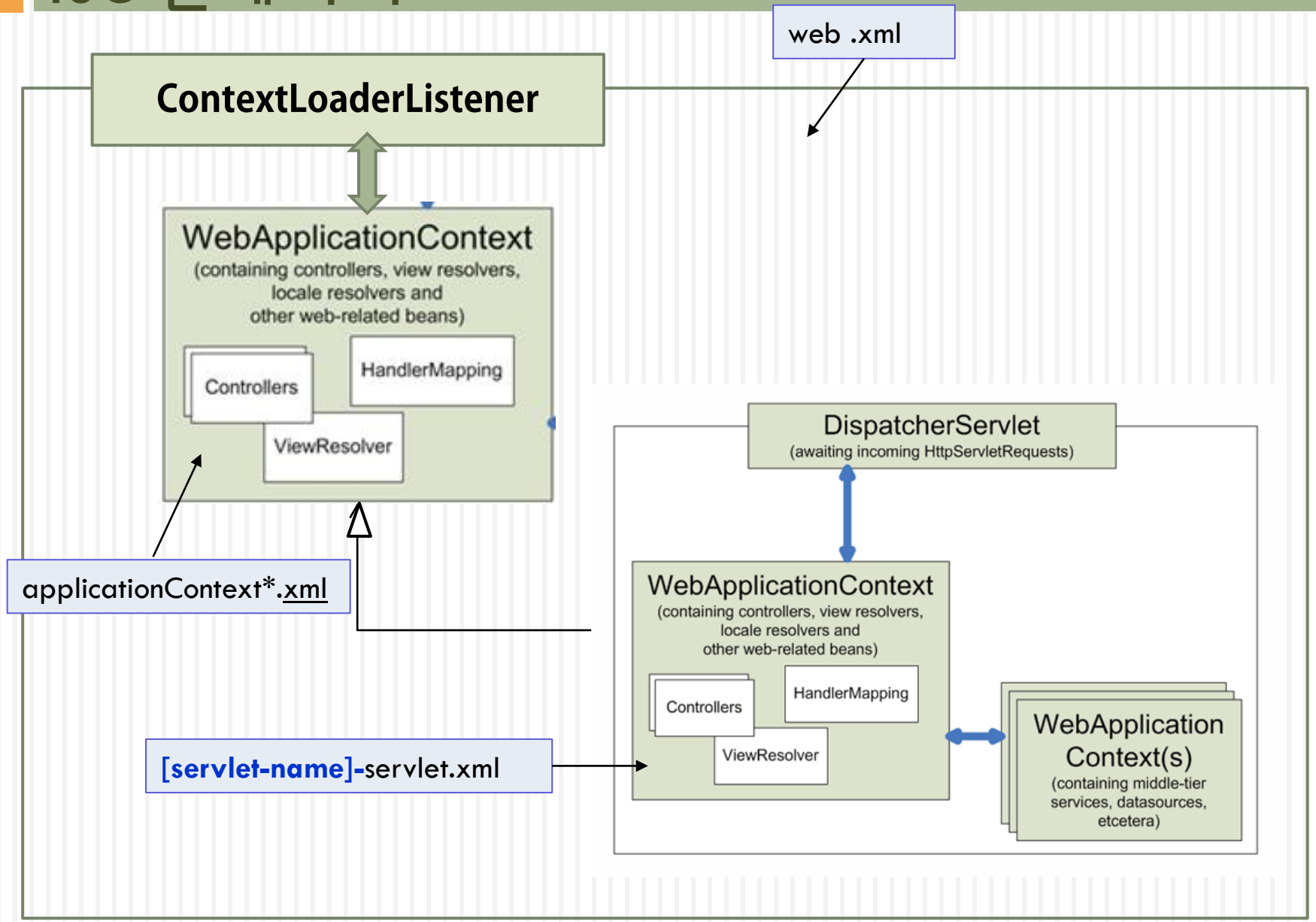
- 제어의 역전(IoC):
  - 자신이 사용할 객체를 자신이 선택하거나 생성하지 않고, 자신도 어떻게 만들어지고 사용될지 모른다.(UserDao)
- 빈 팩토리(bean factory) ➔ 애플리케이션 컨텍스트:
  - 빈의 생성과 관계설정과 같은 제어를 담당하는 Ioc 객체
  - 제어의 역전 개념을 스프링을 이용해서 구현
- Spring 컨테이너 : Ioc 방식으로 빈을 관리한다는 의미로 빈 팩토리, 애플리케이션 컨텍스트를 컨테이너라고 함
- 애플리케이션 컨텍스트(ApplicationContext)가 XML을 이용하여 DI 의존관계를 설정한다.
  - Context :사물의 서로 잇닿아 있는 관계나 연관

IoC 컨테이너

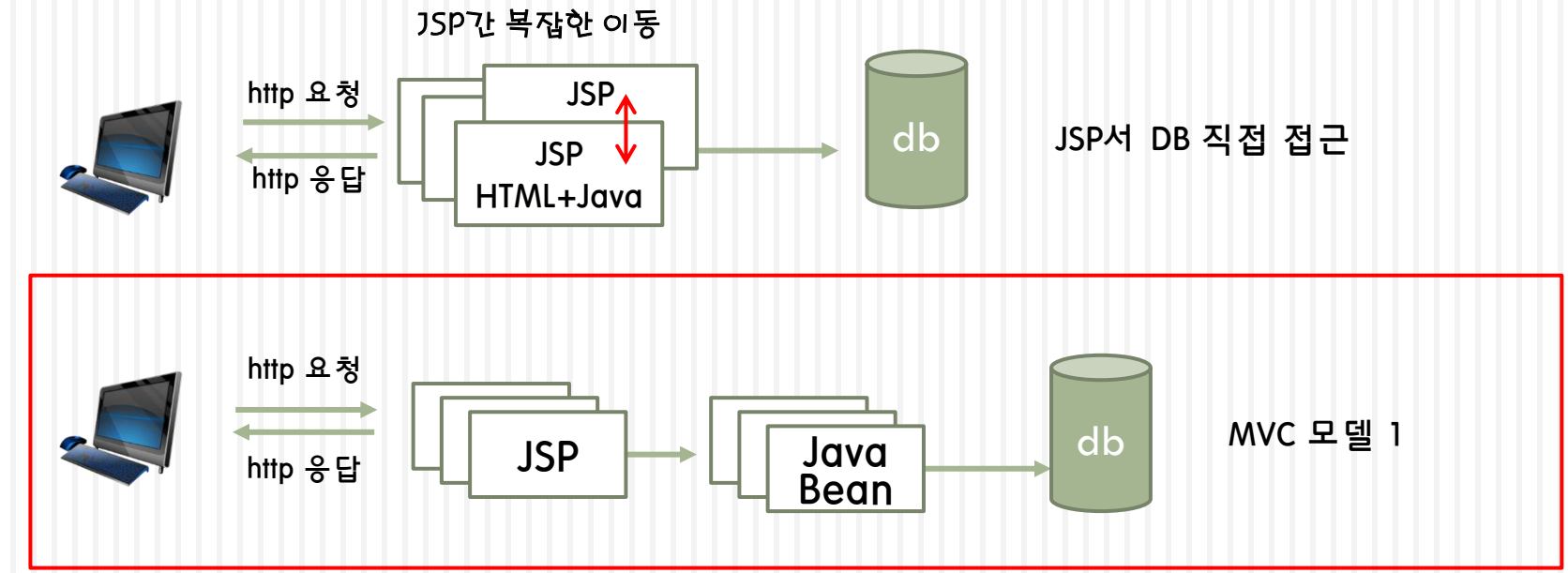


- ContextLoaderListener와 DispatcherServlet는 각각 별도의 WebApplicatonContext 인스턴스를 생성
- ContextLoaderListener가 root 가 되고 DispatcherServlet는 자식이 되어 빈을 상속

IoC 컨테이너


web .xml
ContextLoaderListener
WebApplicationContext
(containing controllers, view resolvers, locale resolvers and other web-related beans)
Controllers
HandlerMapping
ViewResolver
applicationContext*.xml
DispatcherServlet
(awaiting incoming HttpServletRequests)
WebApplicationContext
(containing controllers, view resolvers, locale resolvers and other web-related beans)
Controllers
HandlerMapping
ViewResolver
WebApplication Context(s)
(containing middle-tier services, datasources, etcetera)
[servlet-name]-servlet.xml

MVC 모델

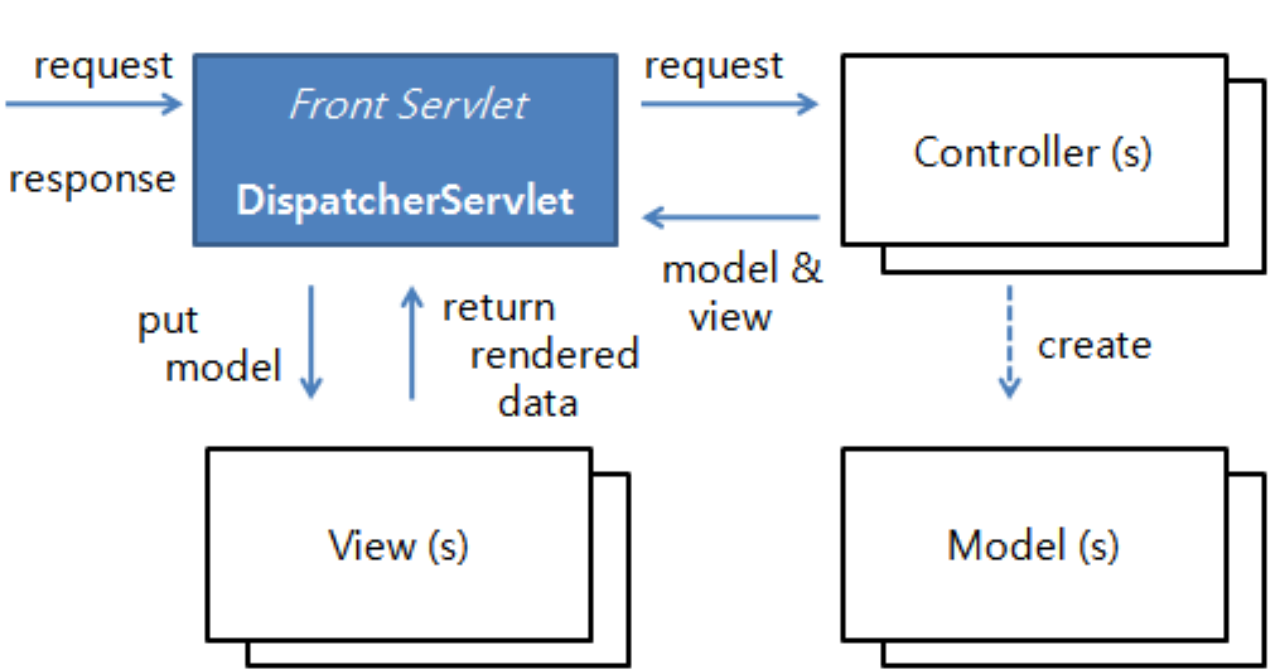

JSP간 복잡한 이동
http 요청
http 응답
JSP
JSP
HTML+Java
db
JSP서 DB 직접 접근
http 요청
http 응답
JSP
Java
Bean
db
MVC 모델 1
일관된 응답요청관리
http 요청
http 응답
Servlet
(Controller)
JSP
(View)
Bean
(Model)
db
MVC 모델 2

MVC 모델

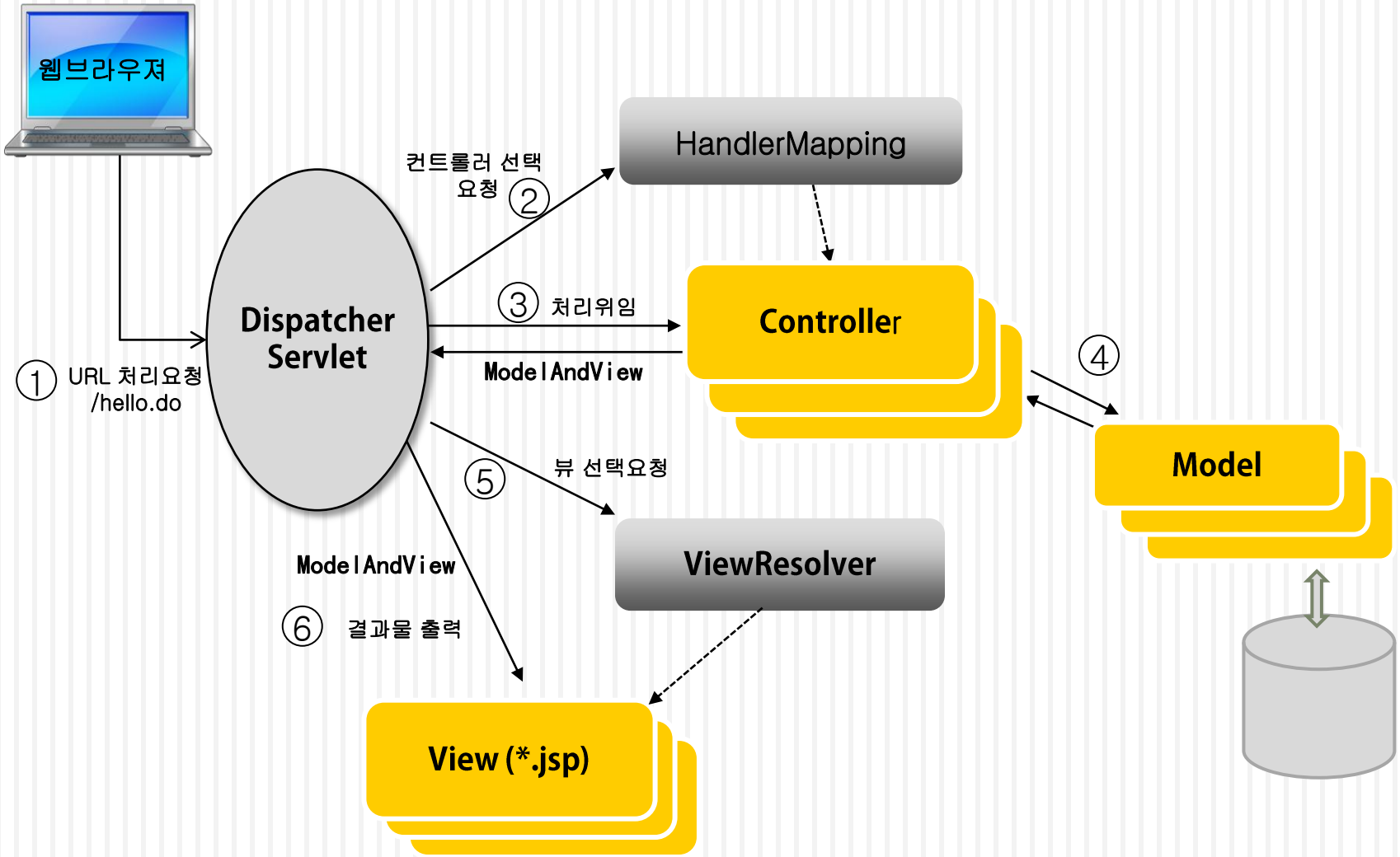
●Model: Business Logic과 데이터베이스 입출력을 담당하는 Layer. View와는 독립적
●View : 어플리케이션의 유저 인터페이스만 담당 최소한의 코드를 이용하여 단순하게 작성(JSP)
●Controller : 어플리케이션의 Flow를 담당,View와 Business Logic의 중재자 역할

- JSP Model 1 - 장점
  - 기본적인 HTML, JavaScript, JAVA만 알아도 배우기 쉽다.
  - 객체지향 원리를 잘 몰라도 기능을 구현 할 수 있다.
  - 처리 로직이 jsp 페이지에 JAVA 코드로 구현 되어있다.
  -
- JSP Model 1 – 단점
  - 디자인(HTML, CSS, JavaScript)과 로직(JAVA)이 혼합되어 있어 소스 수정시 예상하지 못한 곳에서 많은 시간이 지연된다.
  - 같은 JAVA코드가 여러 JSP페이지에 중복되어 있어 소스 수정이 어렵다.
  - 권한 등의 구현 시 모든 jsp페이지 상에 SSI등을 이용해 구현해야 한다.
- JSP Model 2 – 장점
  - Client의 요청을 받는 부분, 처리 로직 부분, 뷰 부분이 분리되어 있어 유지보수가 편리
  - 모든 기능이 세분화해 구조를 이루고 있음으로 기능 확장이 용이
  - 컨트롤러 서블릿에서 권한 및 인증 구현을 편리하게 할 수 있음
- JSP Model 2 – 단점
  - 디자인 패턴등의 개념과 자바언어를 잘 알고 있어야 함.
  - Application이 처리 파트별로 나뉘어져 있음으로 개발 작업의 양이 많음.
  - 디버깅 작업을 처리 파트 별로 살펴야 함으로 어려운 부분이 있음.
  - 개발자들이 MVC에대한 개념을 가지고 있어야 효과를 볼 수 있음.

Spring framework와 MVC 모델

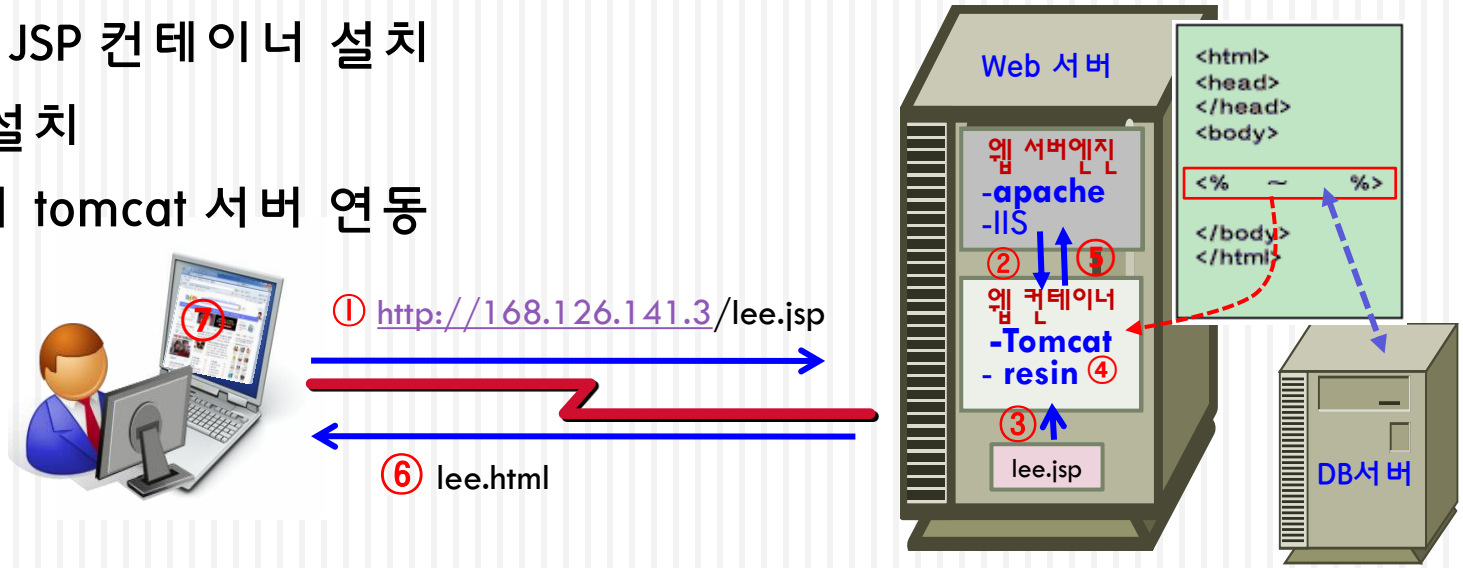


Spring MVC 처리의 흐름



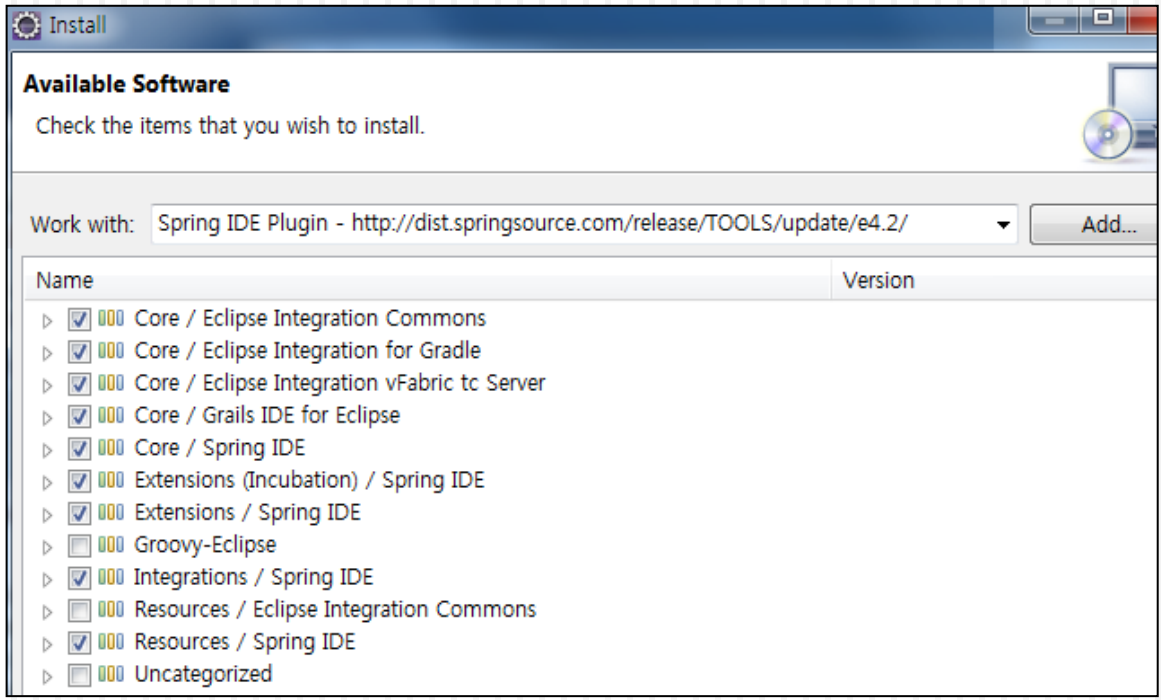
Spring 프로젝트 환경 구성

- JRE
- JDK(Java Development Kit)
  - PATH 설정 : C:\Program Files\Java\jdk1.7.0\bin
  - CLASSPATH 설정 : C:\Program Files\Java\jre7\lib
    - Jre는 바이트코드(xxx.class)를 찾아 실행한다. 바이트코드를 찾을 위치나 파일을 모두 등록한다.
  - JAVA_HOME 설정 : C:\Program Files\Java\jdk1.7.0\
- Apache 웹 서버 설치(개발 환경에서는 불필요)
- Tomcat JSP 컨테이너 설치
- Eclipse 설치
- Eclipse에 tomcat 서버 연동



Spring Framework Eclipse 환경구성

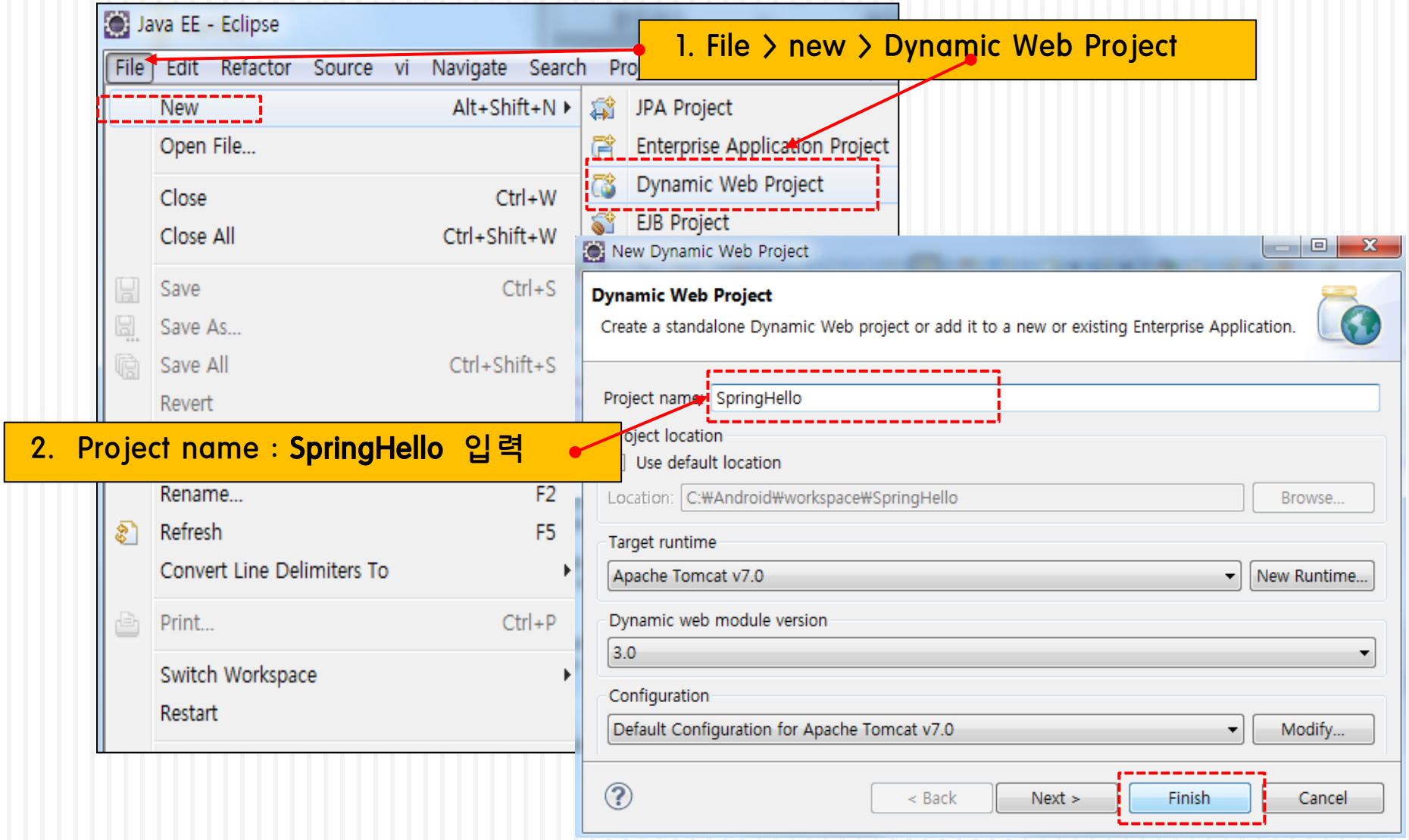
- Eclipse에 Spring IDE plugin
  - Spring IDE Plugin - http://dist.springsource.com/release/TOOLS/update/e4.2/



- http://www.springsource.org/download/community/에서 Spring Framework의 다운로드➔ 압축풀기
- Dynamic WebProject 생성
- WebContent/WEB_INF/lib 폴더에 Spring Framework 모듈 import

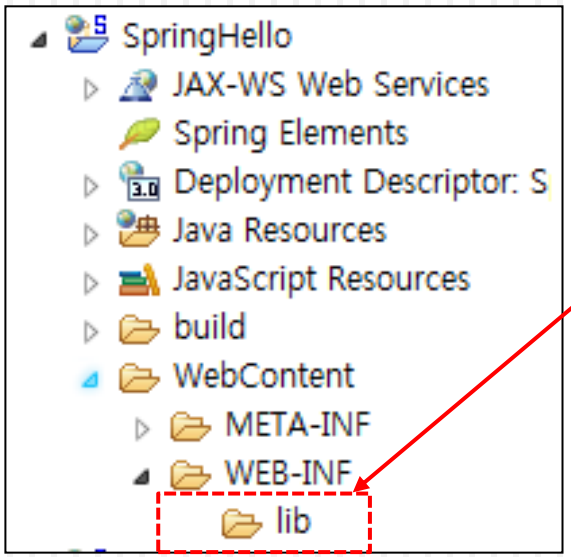
Spring MVC 프로젝트 시작

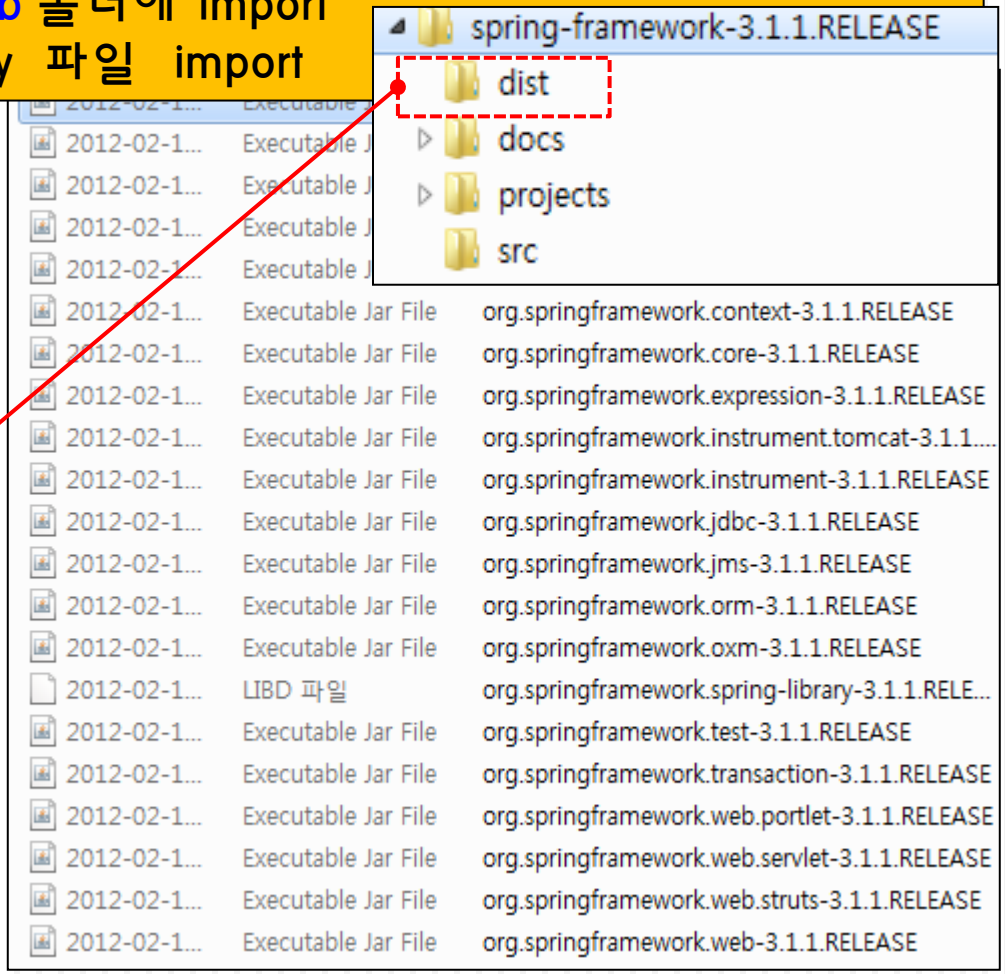
- helloworld! 프로젝트 - DB 연동(model)부분이 없음



Spring MVC의 환경 구성

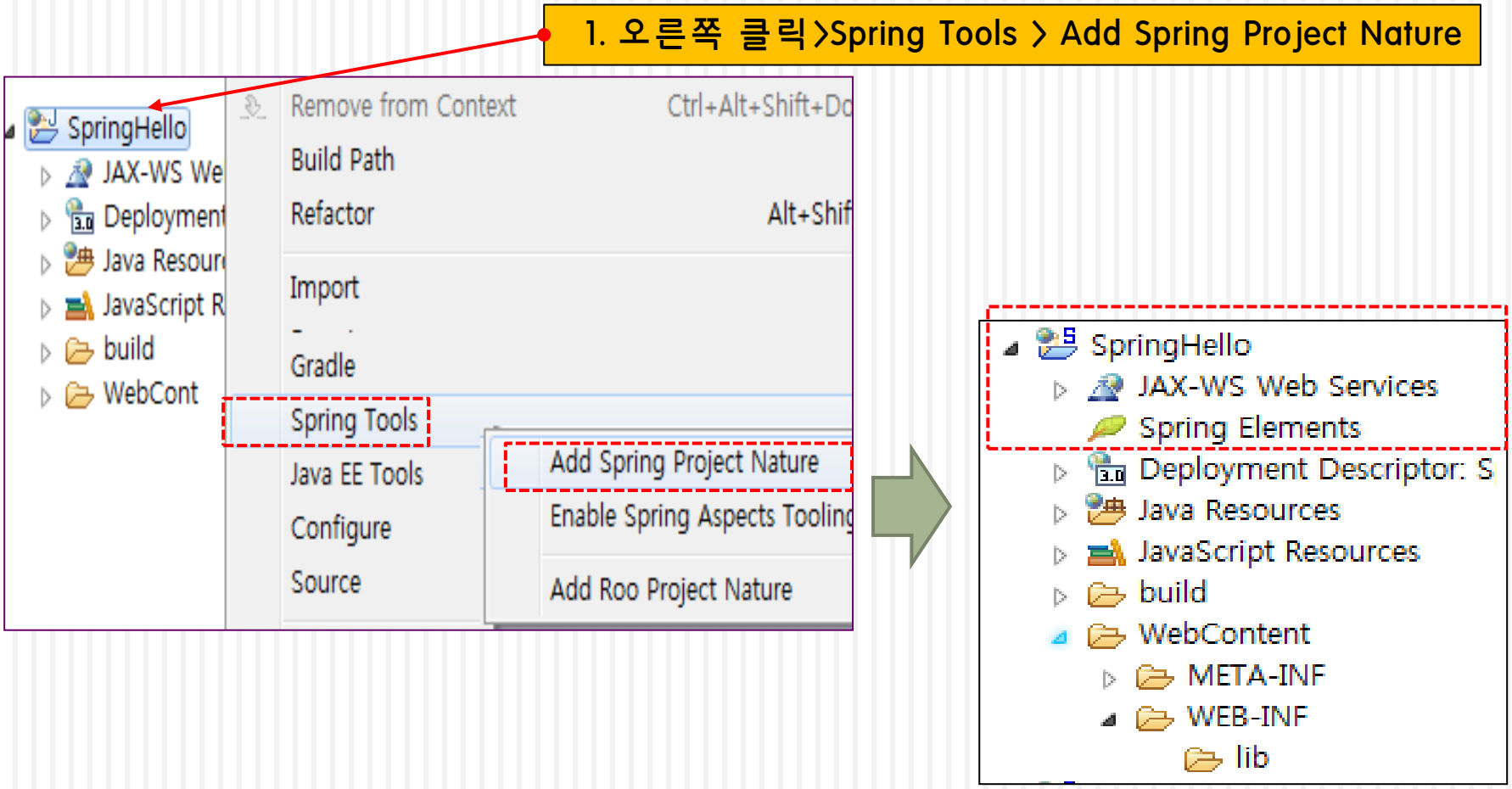
1.http://www.springsource.org/download/community/에서 Spring Framework의 다운로드
2.압축을 풀고 dist 폴더를 프로젝트 lib 폴더에 import
3.기타 의존관계에 있는 필요한 library 파일 import



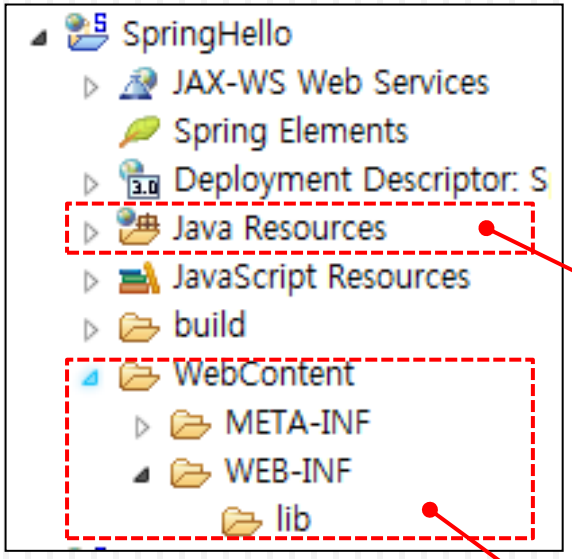


Spring MVC 프로젝트 생성

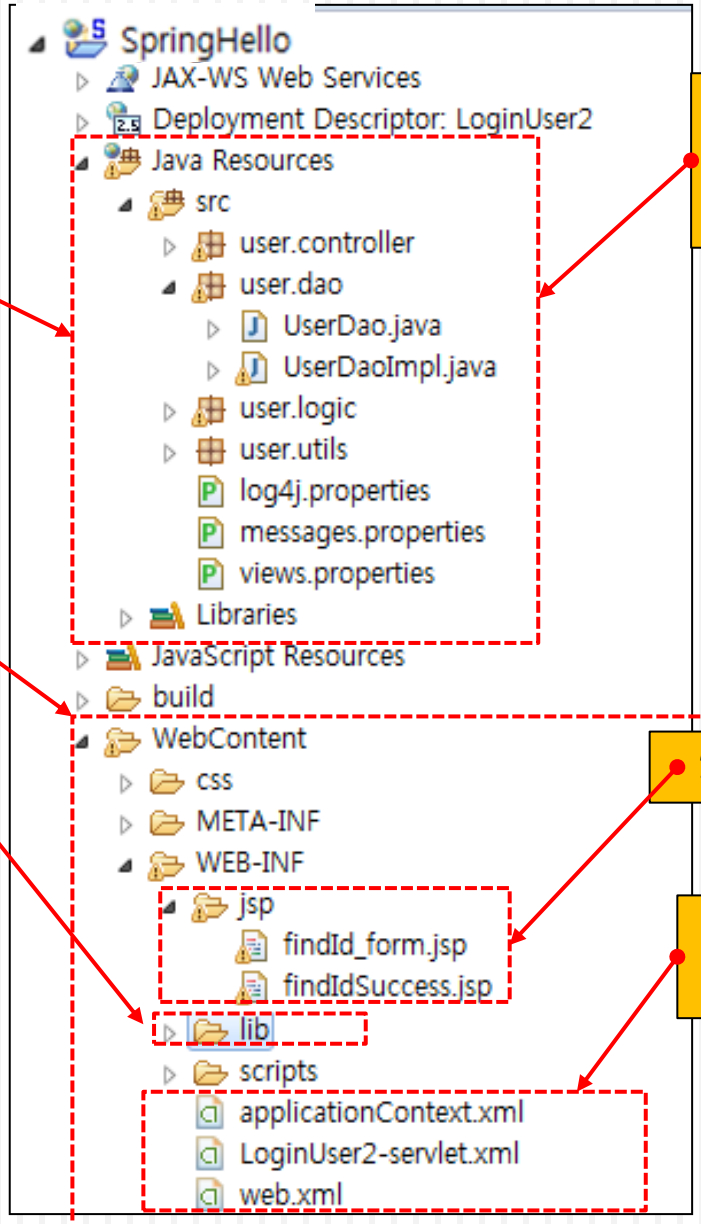
- Web project를 Spring Project로 전환



Spring Project의 전체 구성



web root



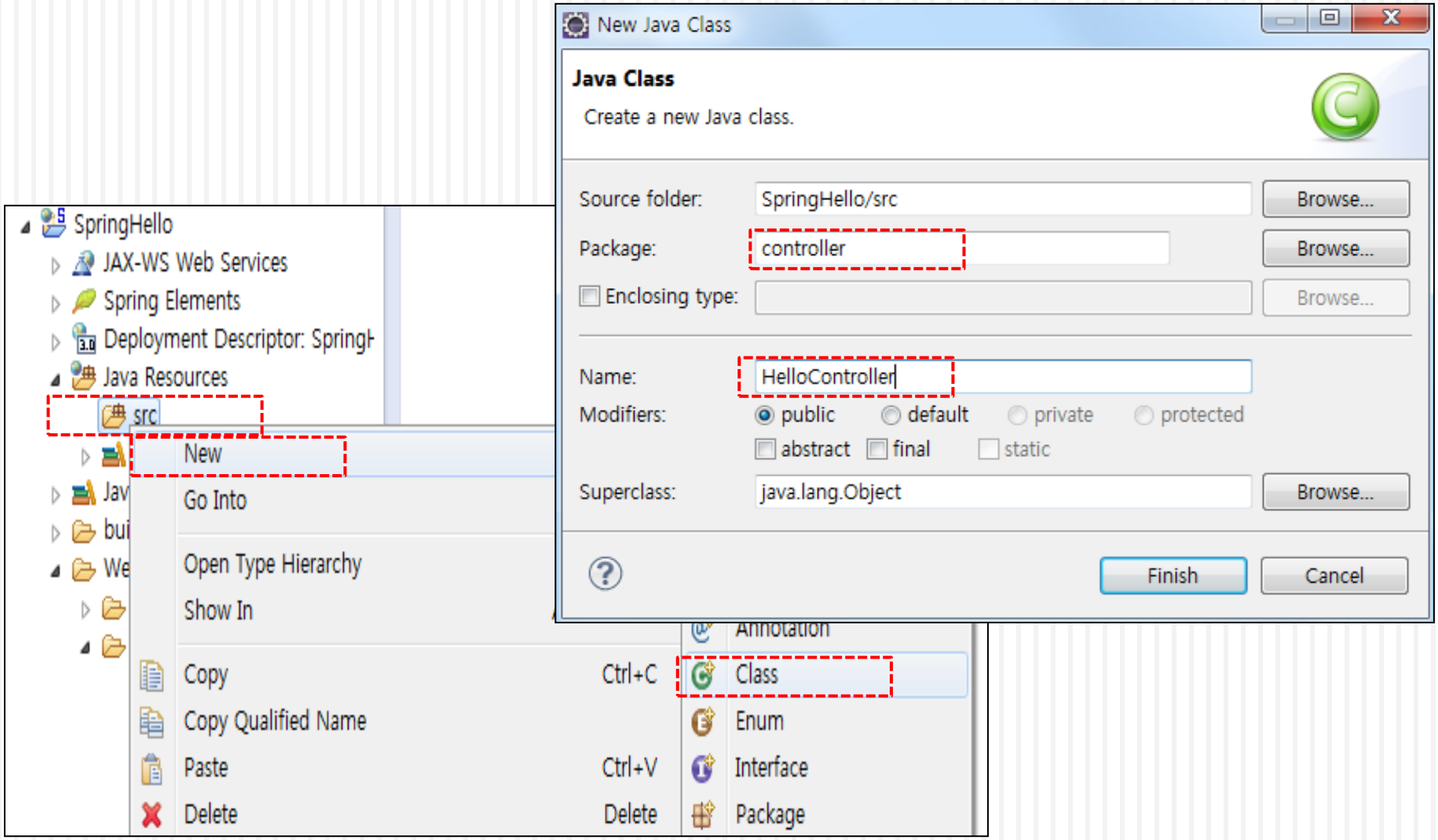
1. java Bean(소스코드)
   - Controller
   - Model

2. jsp 파일 - View

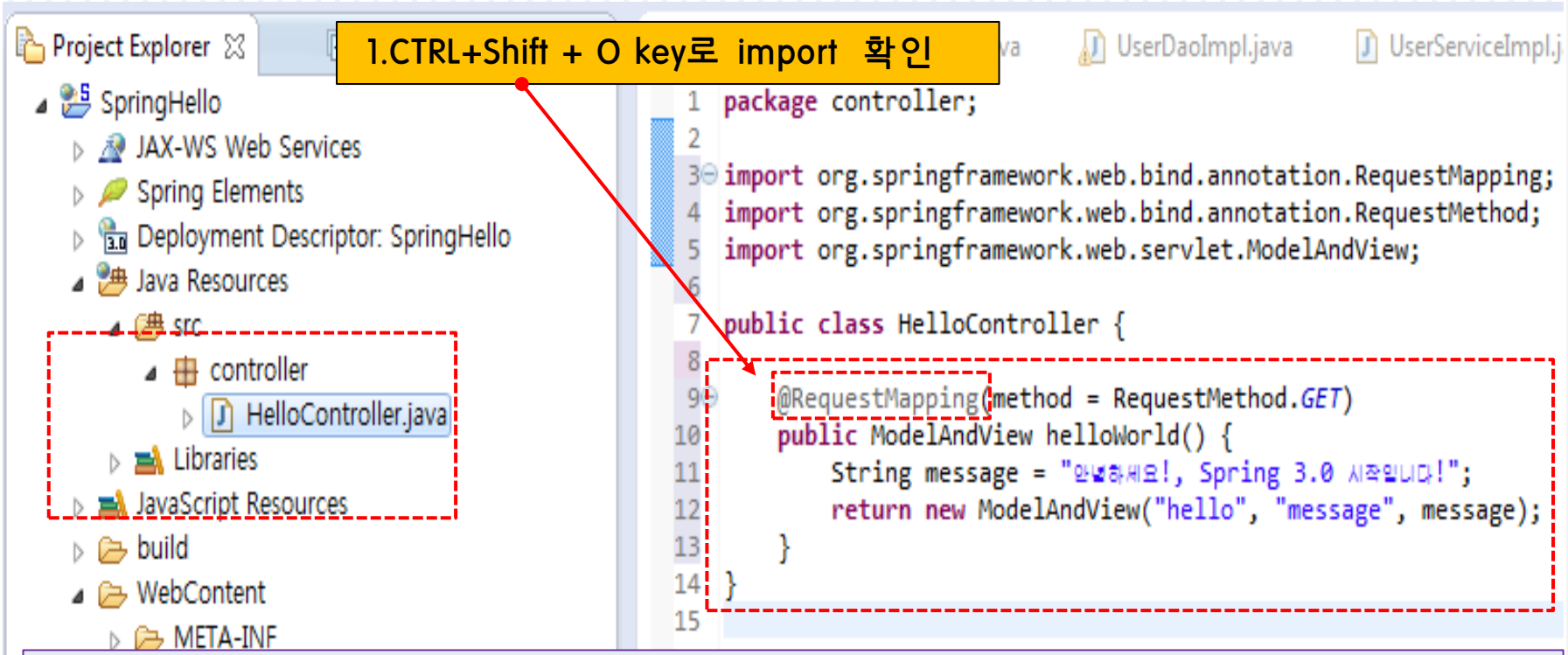
3. Spring(IoC) 컨테이너 구성 파일

4. Spring 3.0 Library 파일

Spring Controller Class 생성

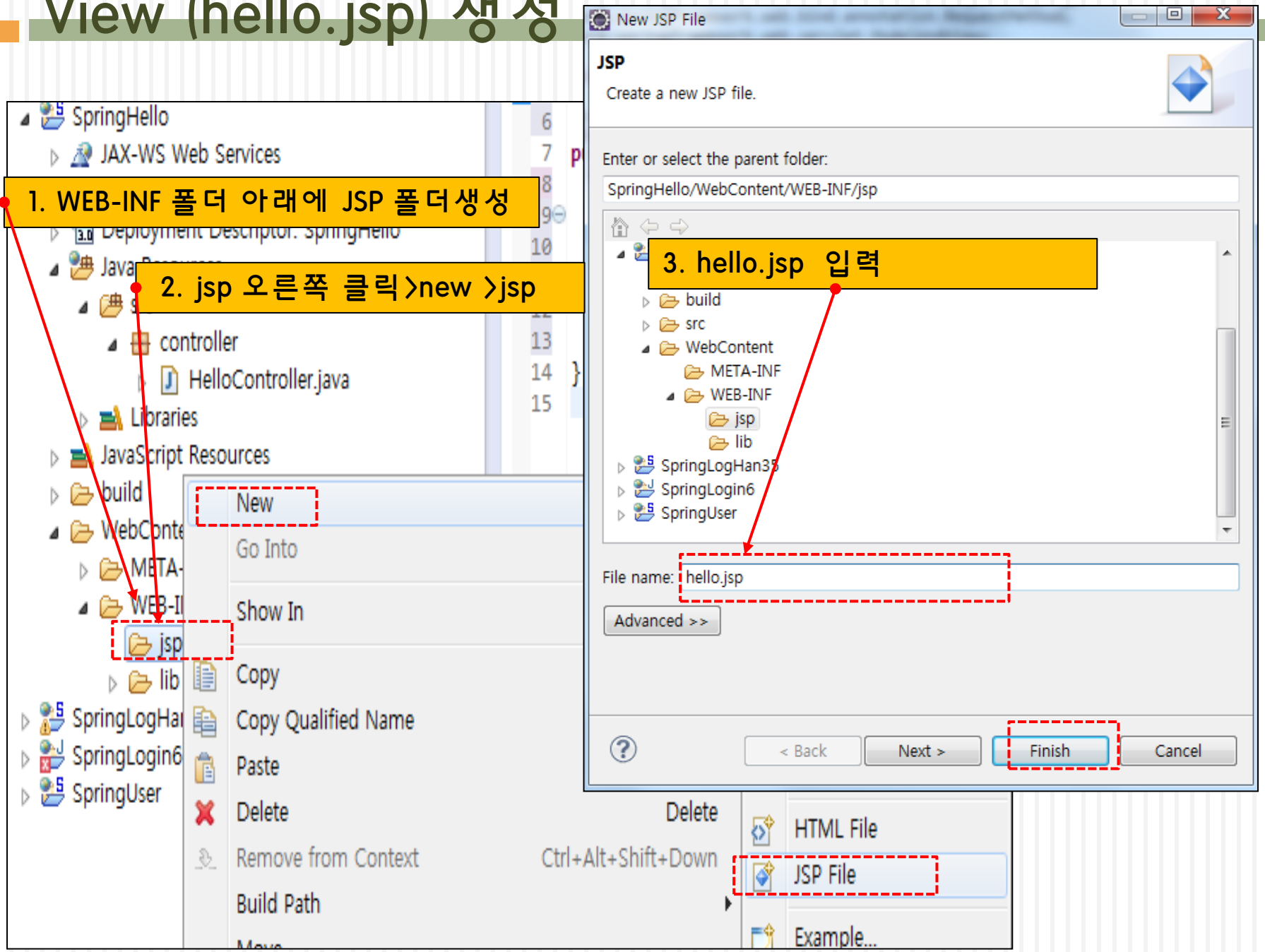


Controller 생성

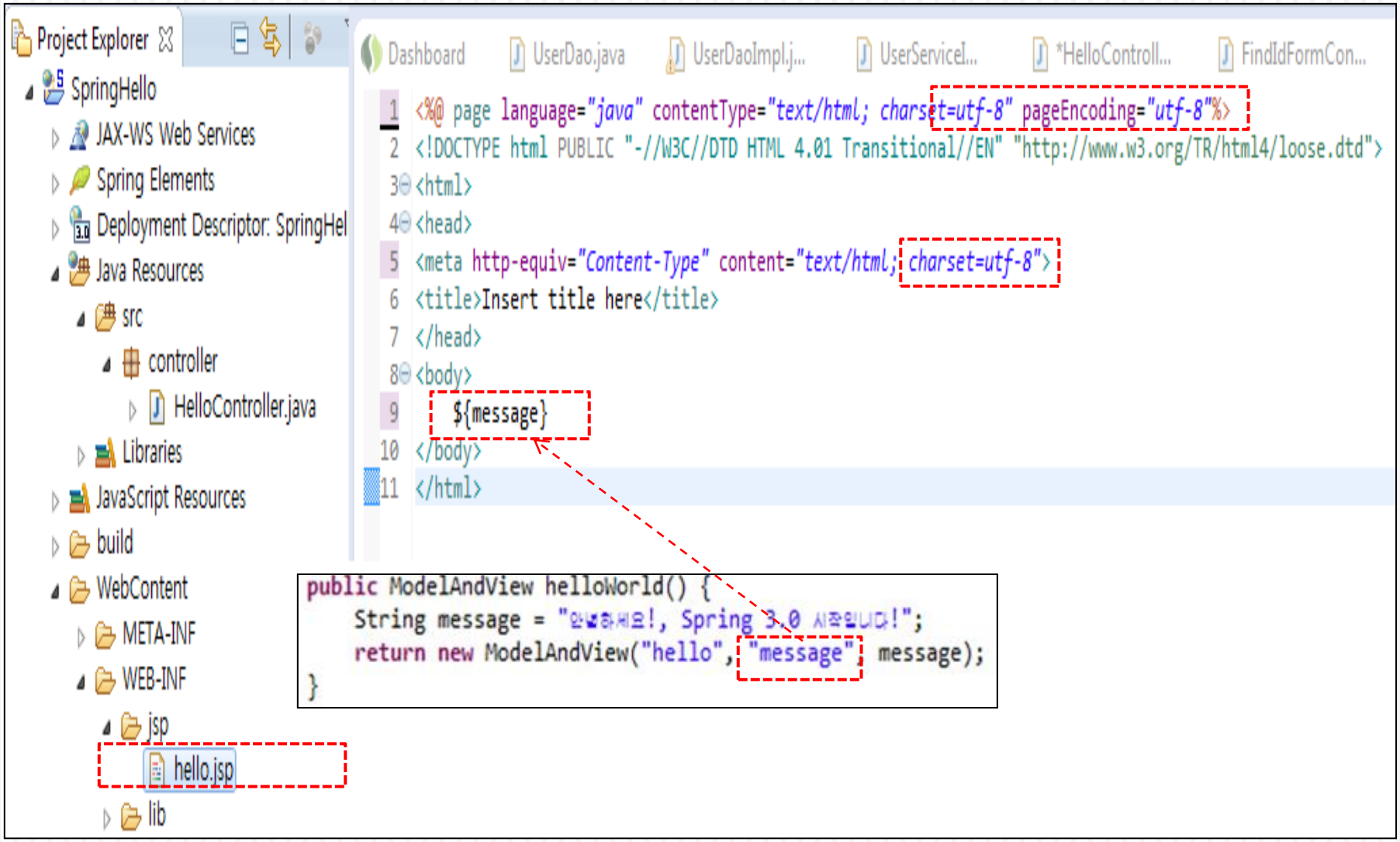


- @RequestMapping 에노테이션, RequestMethod 객체
  - 스프링에게 URL 요청이 get 방식일 때 helloWorld() 메서드가 모든 요청을 처리할 것 임을 알린다.
- ModelAndView 객체 생성
  - 모델을 출력할 뷰이름 : "hello"
  - ModelAndView 객체는 key가 "message" 이고 "안녕하세요…" 라는 String 형의 메시지를 포함
  - ModelAndView 객체는 View에 전달되어 출력됨

View (hello.jsp) 생성



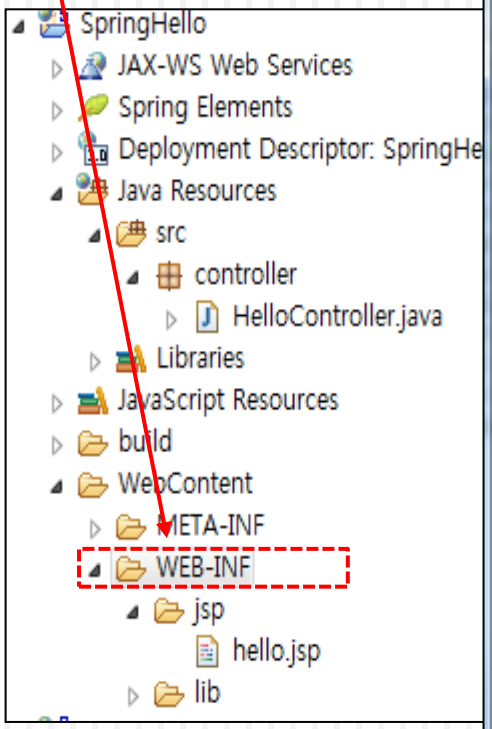
hello.jsp

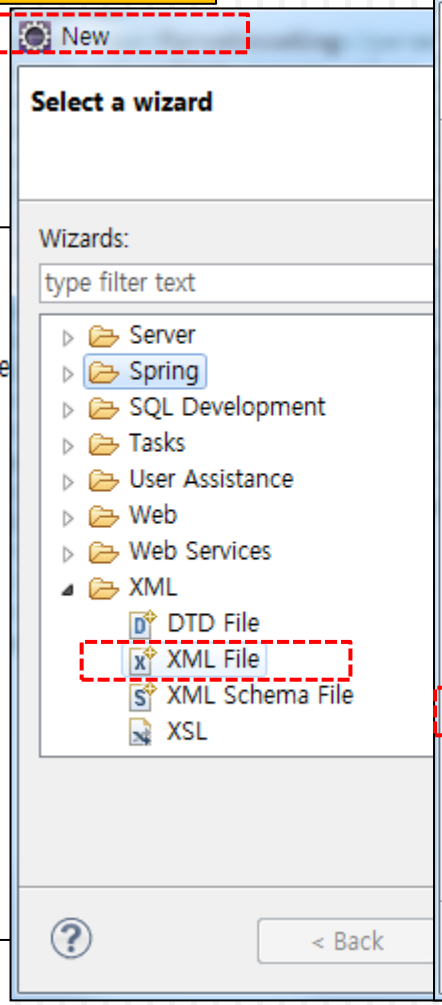

Project Explorer
SpringHello
JAX-WS Web Services
Spring Elements
Deployment Descriptor: SpringHel
Java Resources
src
controller
HelloController.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
jsp
hello.jsp
lib
Dashboard
UserDao.java
UserDaoImpl.j...
UserServiceI...
*HelloControll...
FindIdFormCon...
1 <%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
3 <html>
4 <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
6 <title>Insert title here</title>
7 </head>
8 <body>
9 ${message}
10 </body>
11 </html>
public ModelAndView helloWorld() {
String message = "안녕하세요!, Spring 3.0 시작합니다!";
return new ModelAndView("hello", "message", message);
}

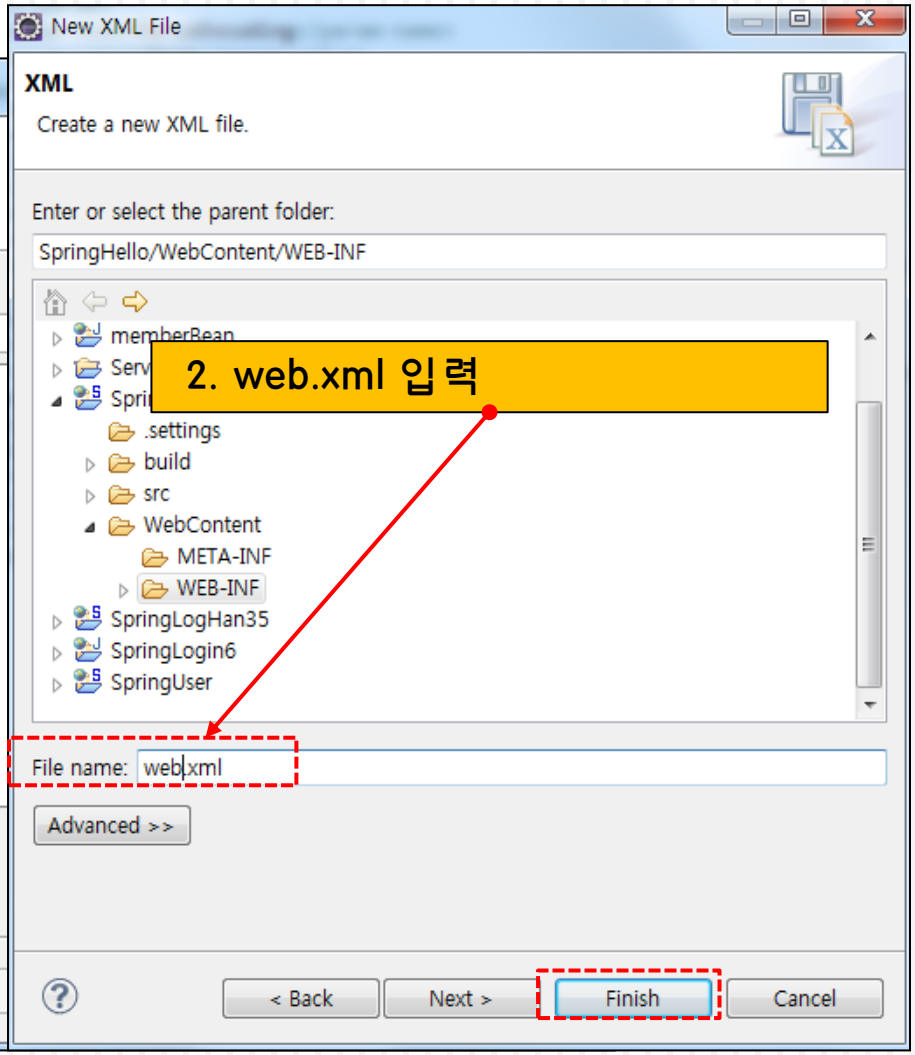
IoC 컨테이너 설정(web.xml)

- ContextLoaderListener 와 DispatcherServlet 설정

1. 오른쪽 클릭>new>XML파일

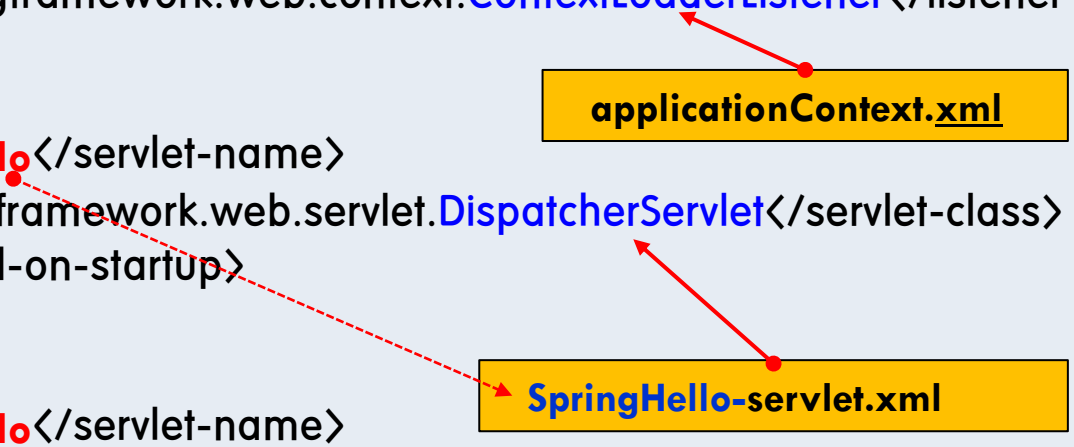






web.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<servlet>
    <servlet-name>SpringHello</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>SpringHello</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>
</web-app>
```



web.xml(추가)

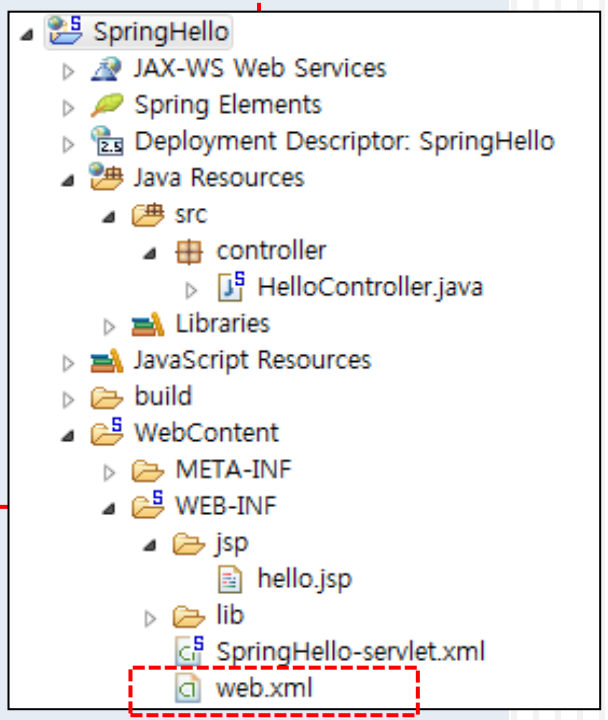
```
<web-app >
 <display-name>SpringHello</display-name>
 <welcome-file-list>
       <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>
<filter>
   <filter-name>CharacterEncodingFilter</filter-name>
   <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
   <init-param>
        <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>CharacterEncodingFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
<context-param>
   <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext*.xml</param-value>
</context-param>
 …
<web-app>
```

어플리케이션 이름

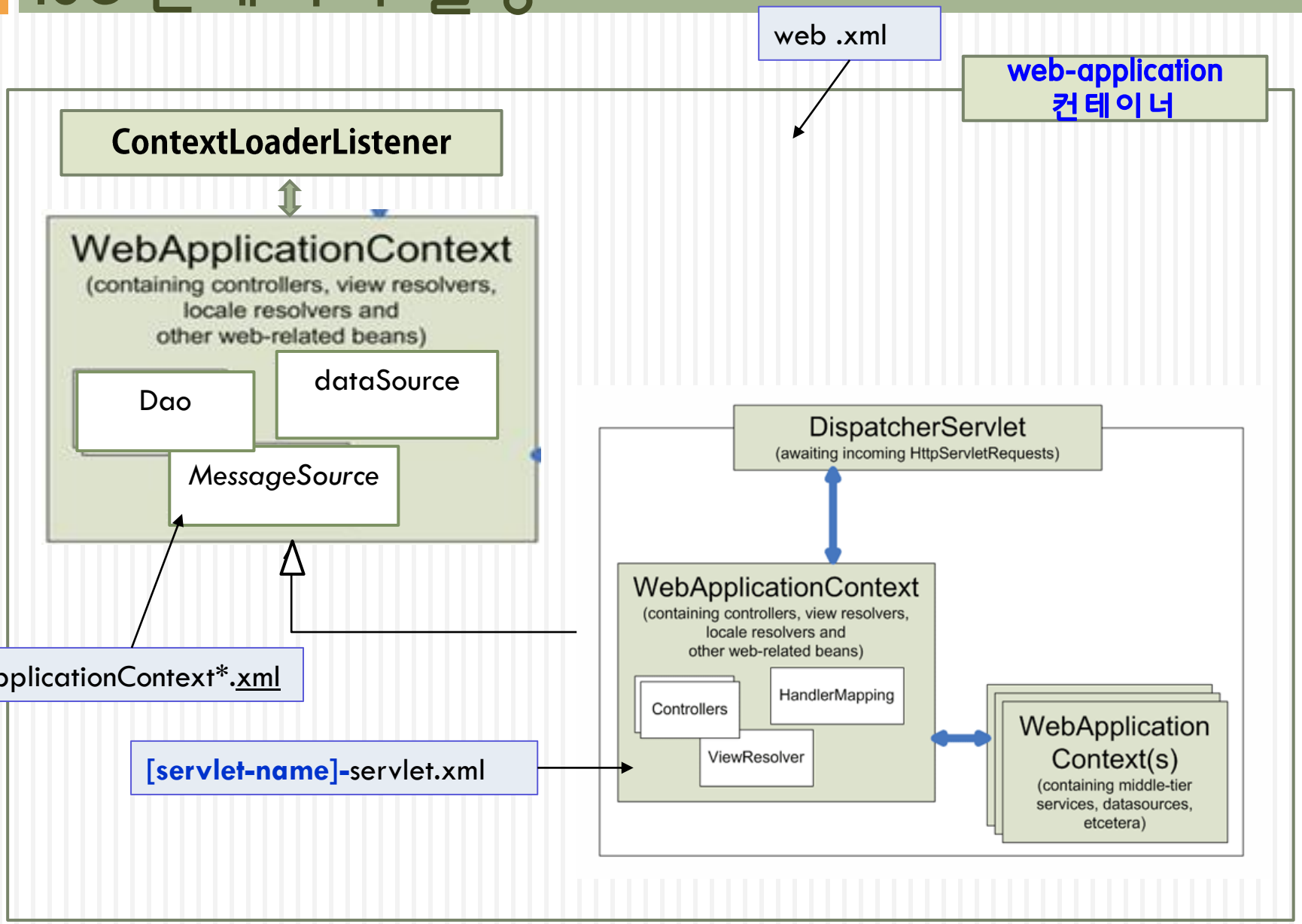
<!-- 시작페이지 설정 -->

문자 인코딩 처리 utf-8

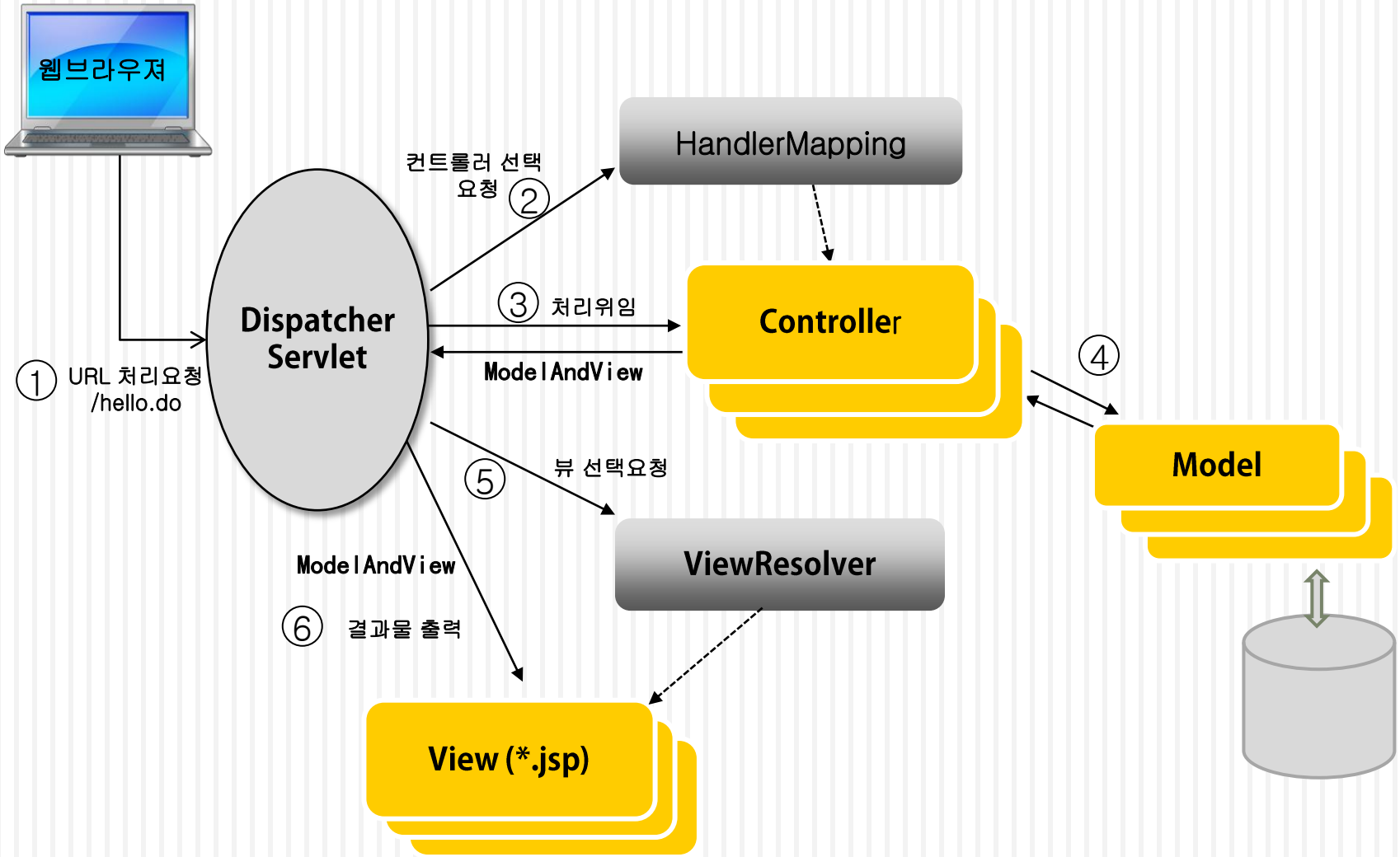
applicationContext.xml 을 나누어 작성할 때



IoC 컨테이너 설정

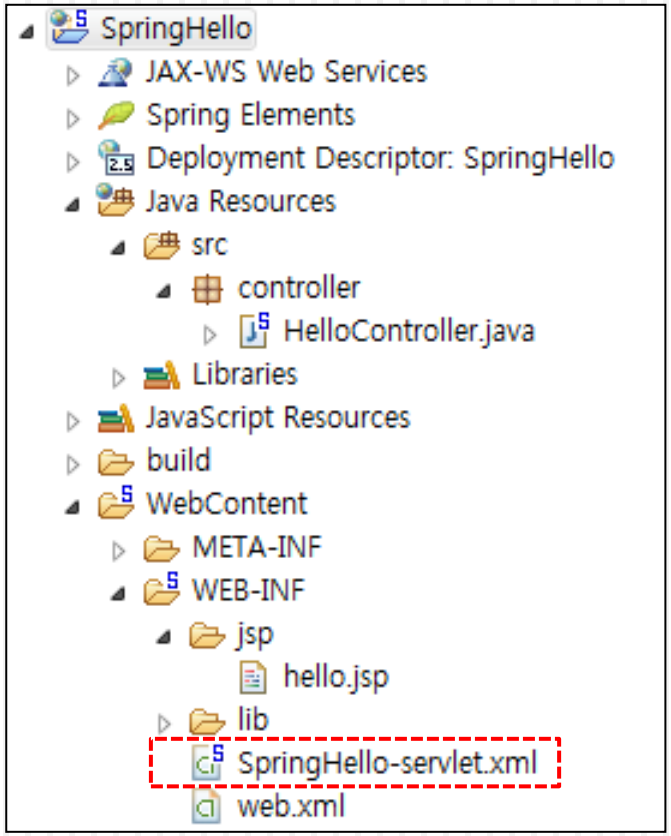

web .xml
web-application
컨테이너
ContextLoaderListener
WebApplicationContext
(containing controllers, view resolvers,
locale resolvers and
other web-related beans)
Dao
dataSource
MessageSource
applicationContext*.xml
[servlet-name]-servlet.xml
DispatcherServlet
(awaiting incoming HttpServletRequests)
WebApplicationContext
(containing controllers, view resolvers,
locale resolvers and
other web-related beans)
Controllers
HandlerMapping
ViewResolver
WebApplication
Context(s)
(containing middle-tier
services, datasources,
etcetera)

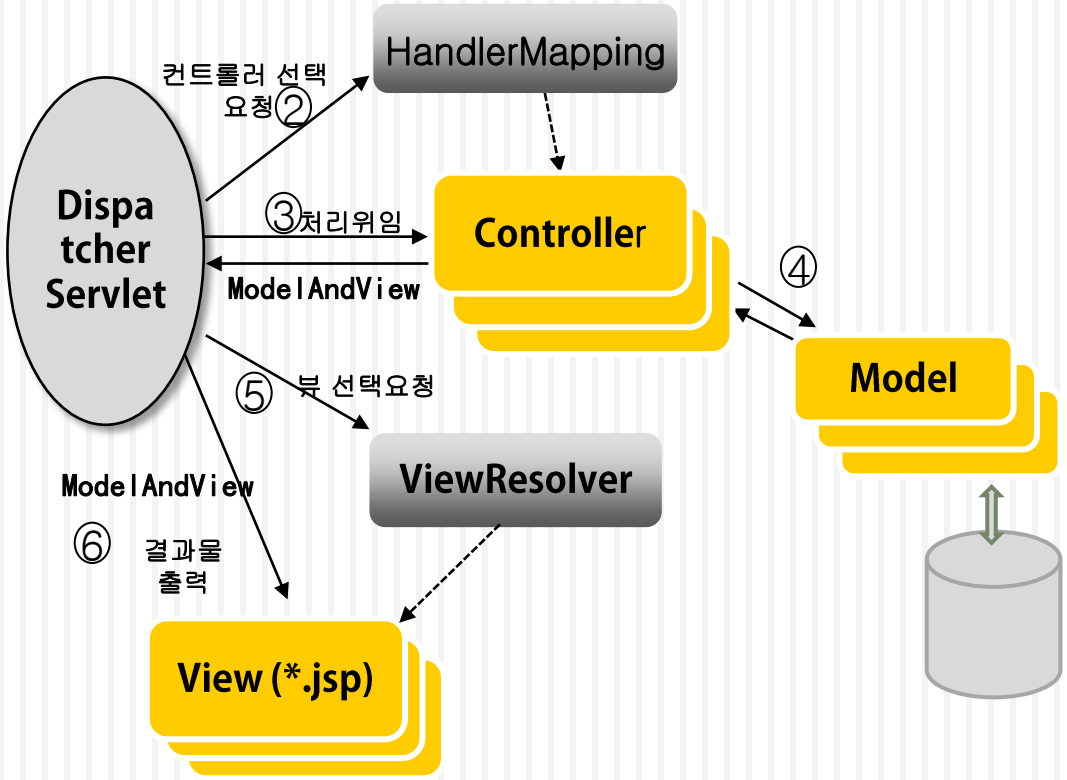
Spring MVC 처리의 흐름



DispatcherServlet 어플리케이션 컨텍스트 설정

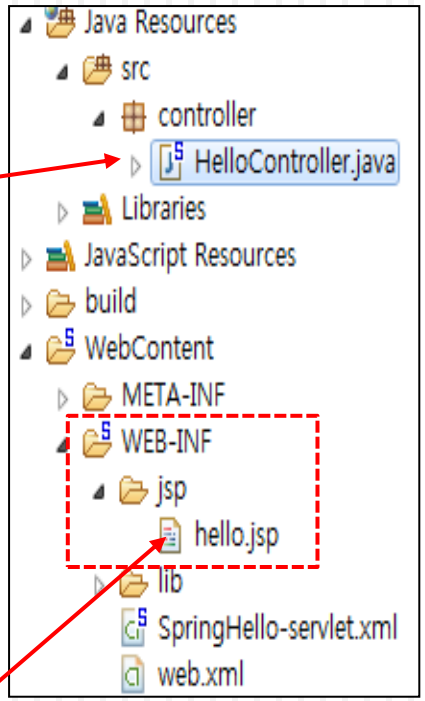
- [servlet-name]-servlet.xml 파일에 IoC 컨테이너(어플리케이션 컨텍스트)에 생성되어야하는 객체(beans)를 선언한다.





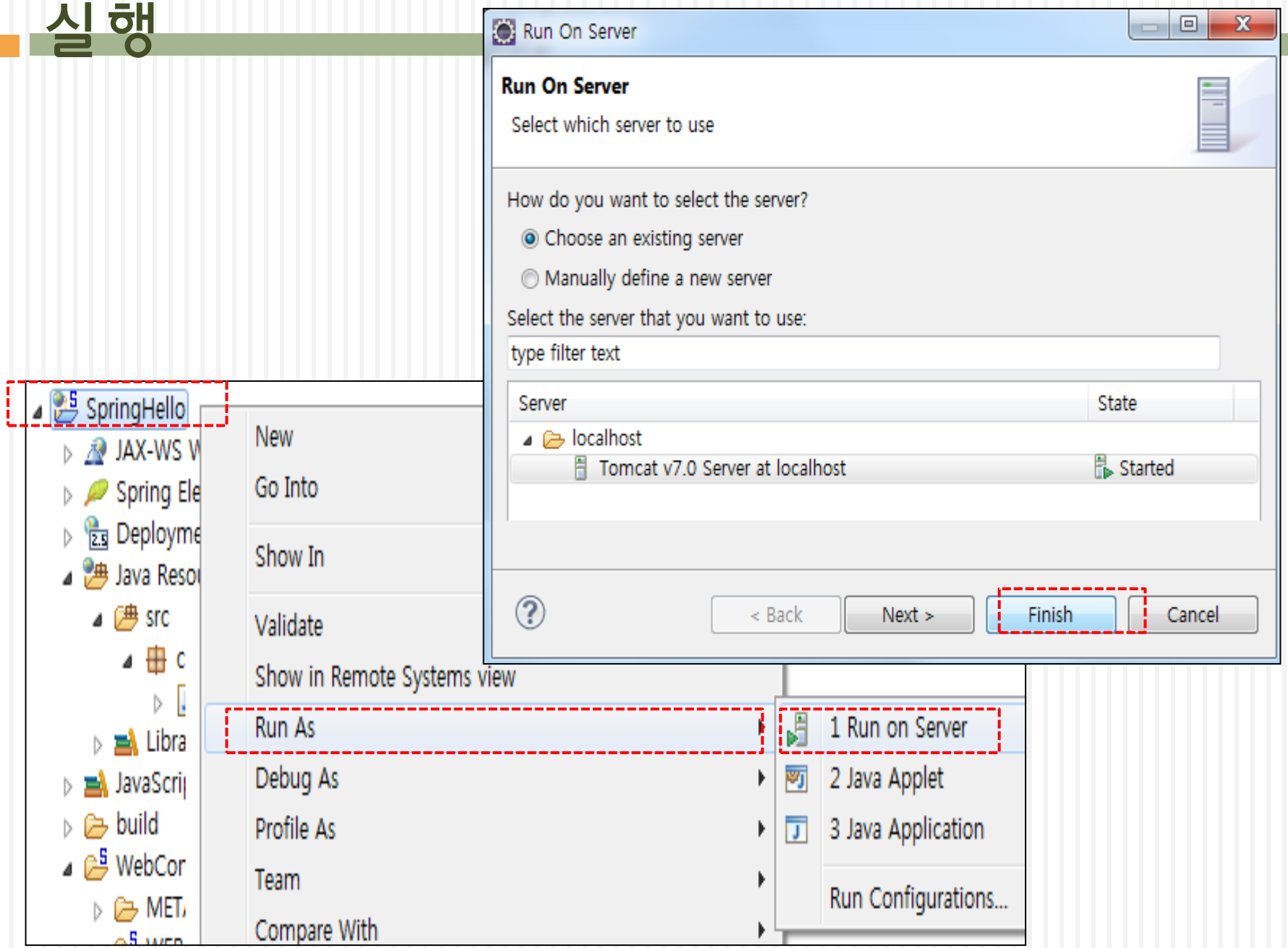
DispatcherServlet 설정 화일 (SpringHello-servlet.xml)

```
<?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:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

 <!-- HandlerMapping -->
 <bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
                <prop key="/hello.html">helloController</prop>
        </props>
    </property>
 </bean>

 <!-- Controller -->
 <bean id="helloController" class="controller.HelloController"></bean>

 <!-- ViewResolver -->
 <bean id="internalResourceViewResolver"
       class="org.springframework.web.servlet.view.InternalResourceViewResolver">
     <property name="viewClass">
          <value>org.springframework.web.servlet.view.JstlView</value>
     </property>
     <property name="prefix">
          <value> WEB-INF/jsp/</value>
     </property>
     <property name="suffix">
          <value>.jsp</value>
      </property>
 </bean>
</beans>
```

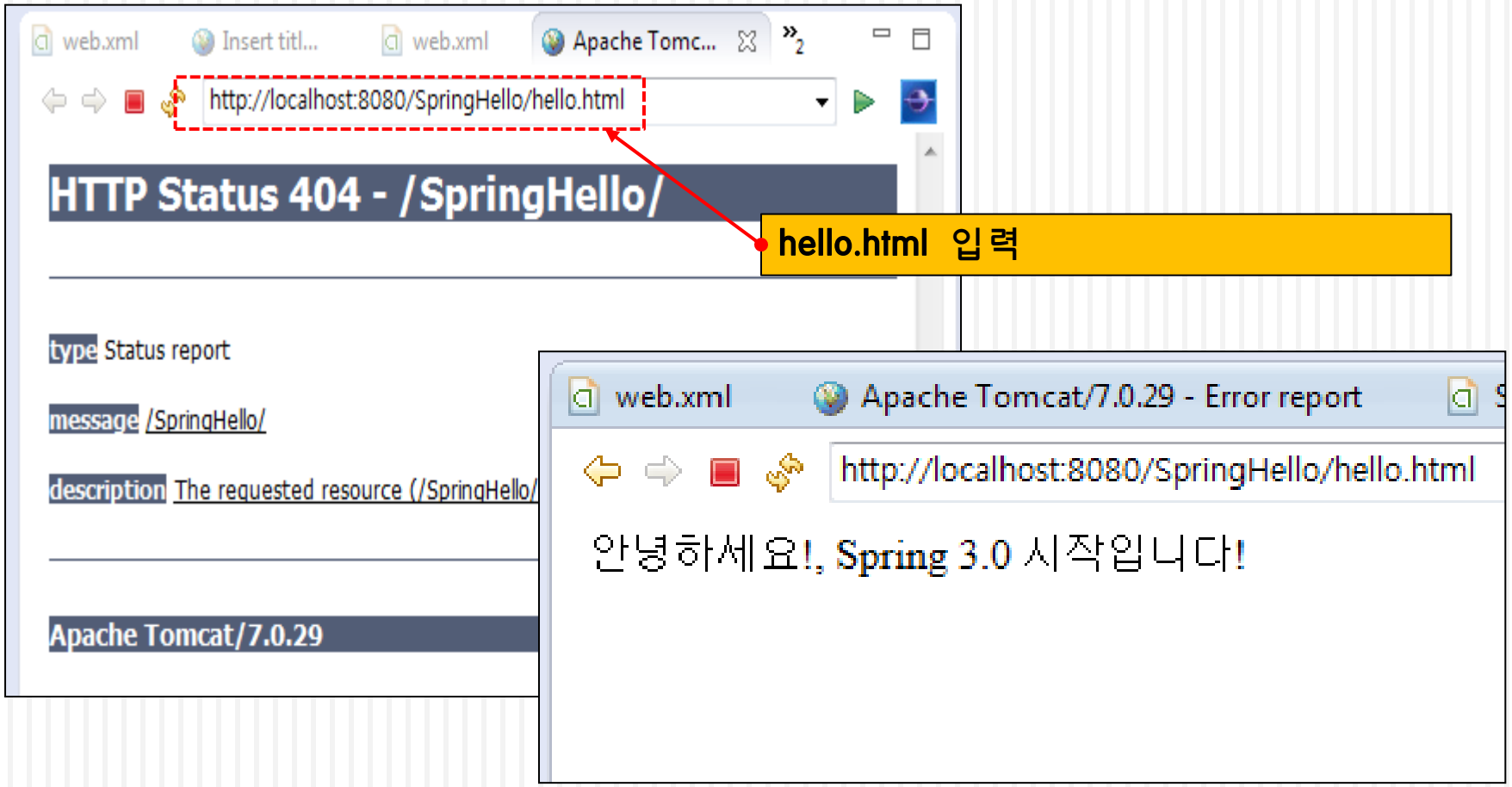


WEB_INF/jsp/hello.jsp

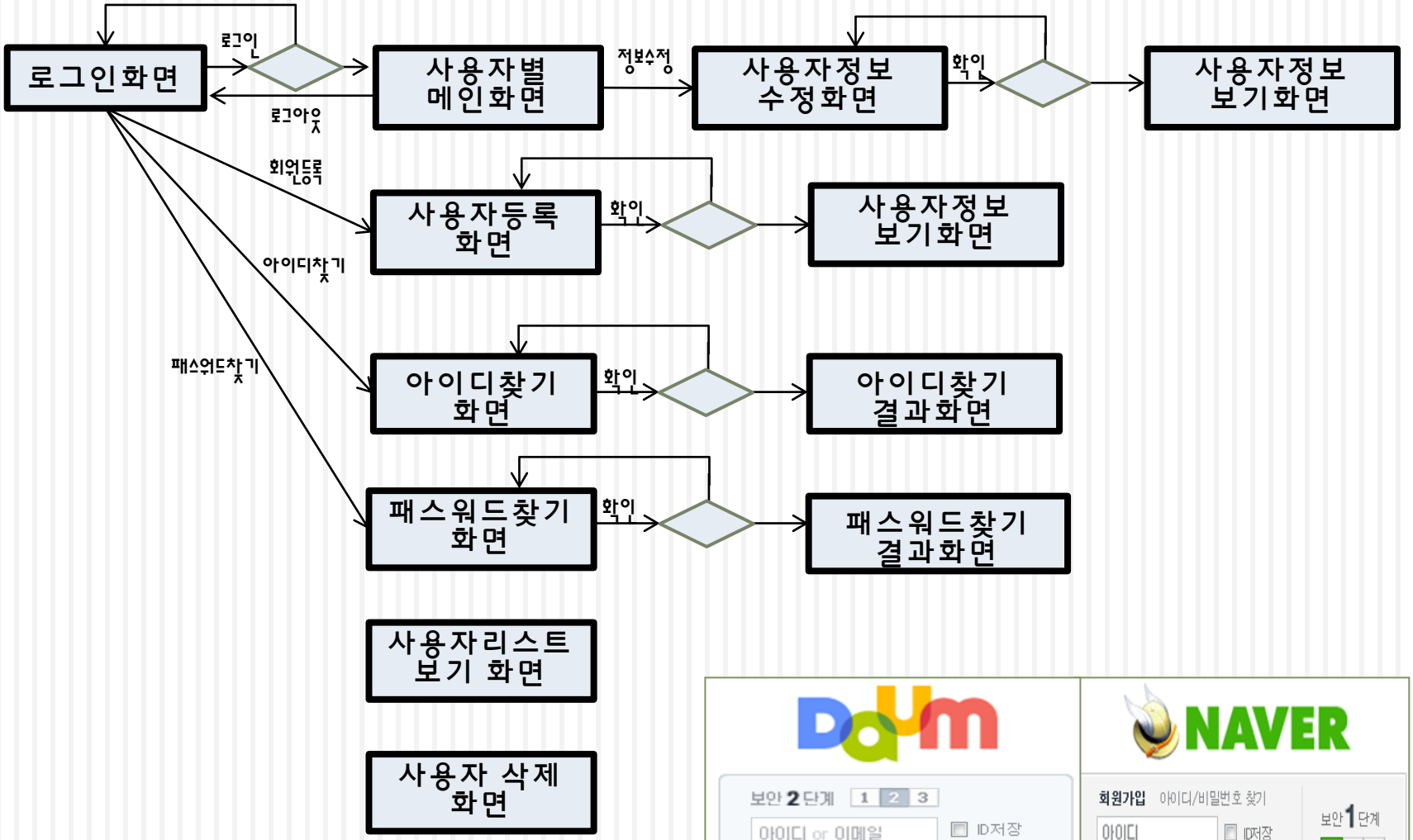
실행


Run On Server
Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server
Manually define a new server
Select the server that you want to use:
type filter text
Server
State
localhost
Tomcat v7.0 Server at localhost
Started
< Back
Next >
Finish
Cancel
SpringHello
New
Go Into
Show In
Validate
Show in Remote Systems view
Run As
Debug As
Profile As
Team
Compare With
1 Run on Server
2 Java Applet
3 Java Application
Run Configurations...

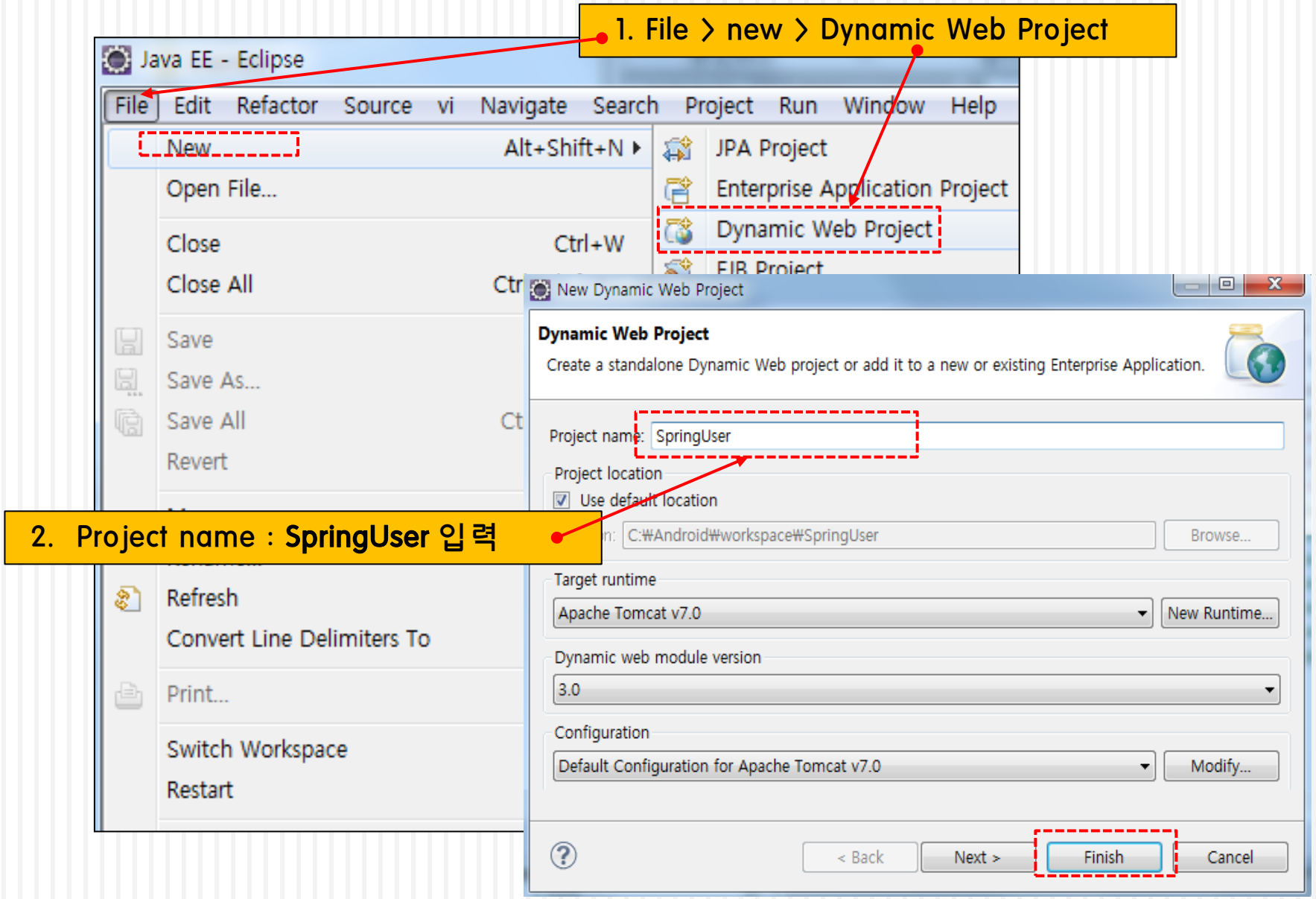
결과확인


web.xml
Insert titl...
web.xml
Apache Tomc...
http://localhost:8080/SpringHello/hello.html
HTTP Status 404 - /SpringHello/
type Status report
message /SpringHello/
description The requested resource (/SpringHello/
Apache Tomcat/7.0.29
hello.html 입력
web.xml
Apache Tomcat/7.0.29 - Error report
http://localhost:8080/SpringHello/hello.html
안녕하세요!, Spring 3.0 시작입니다!

Spring MVC를 이용한 사용자관리 시스템

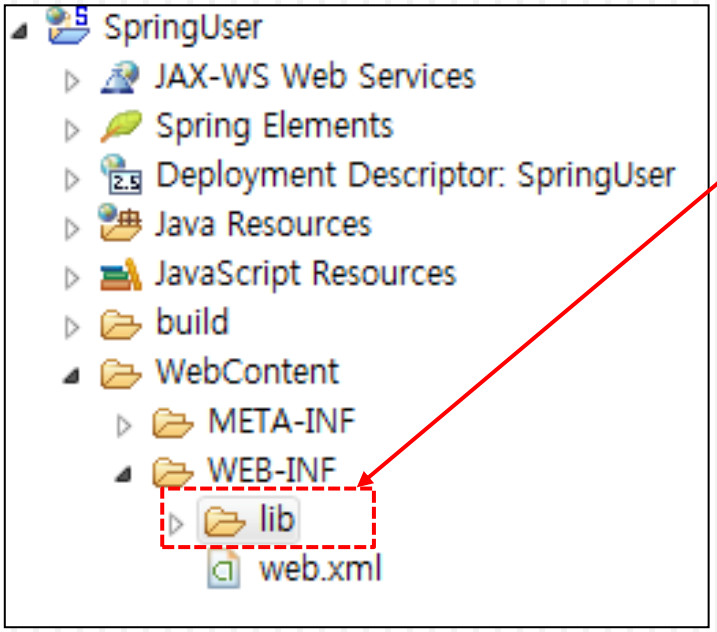


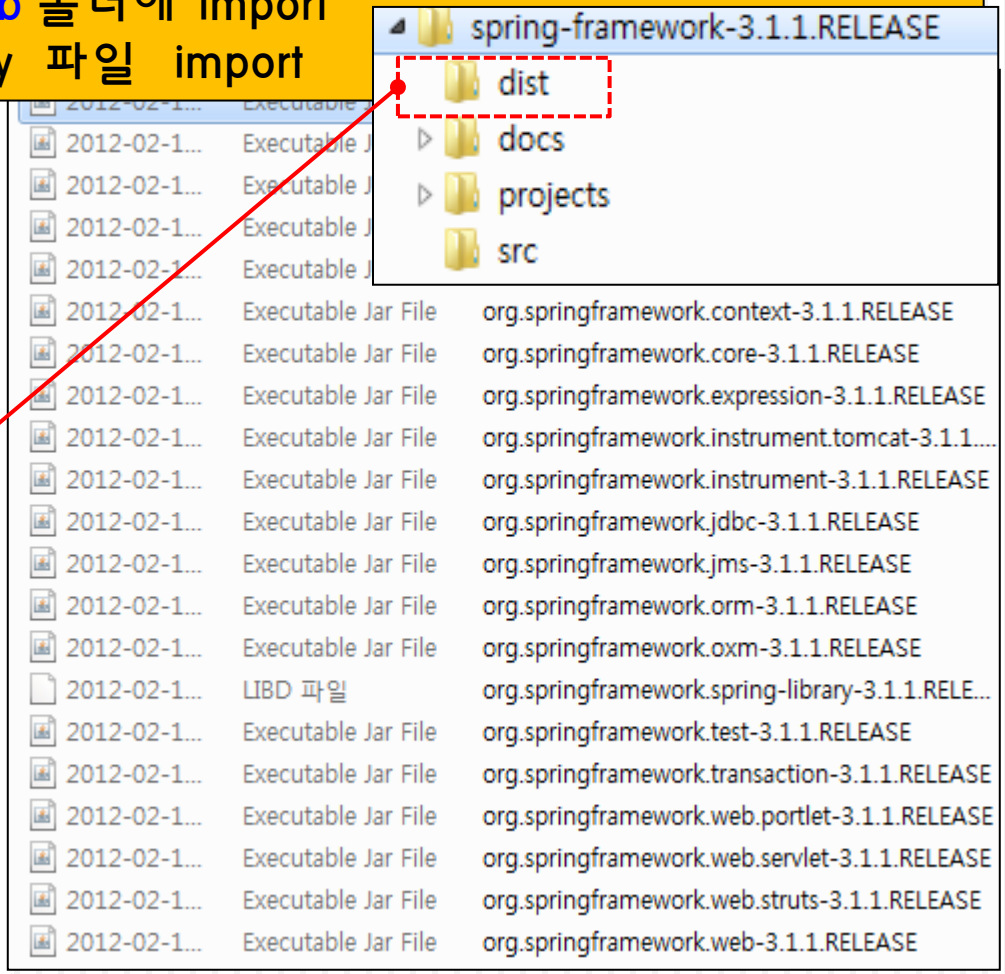
Spring MVC 프로젝트 생성



Spring MVC의 환경 구성

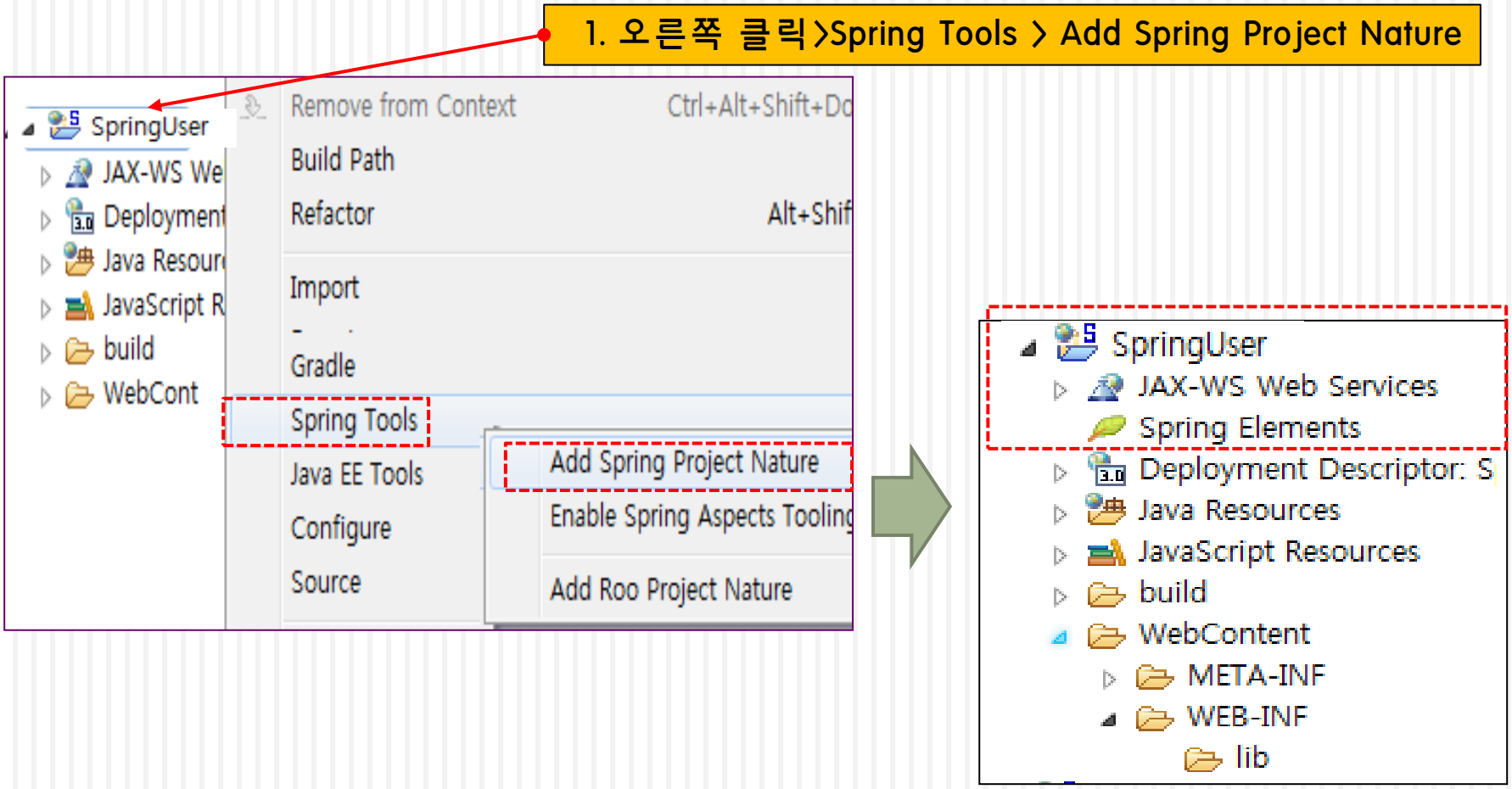
1.http://www.springsource.org/download/community/에서 Spring Framework의 다운로드
2.압축을 풀고 dist 폴더를 프로젝트 lib 폴더에 import
3.기타 의존관계에 있는 필요한 library 파일 import



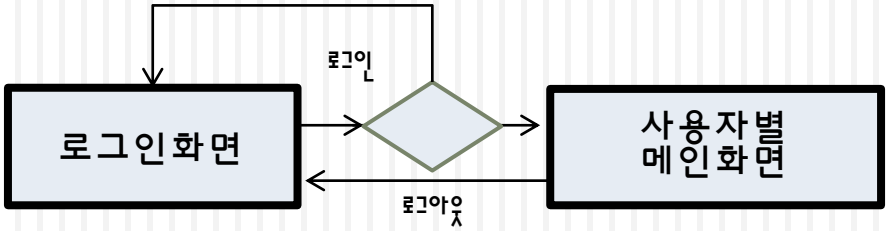


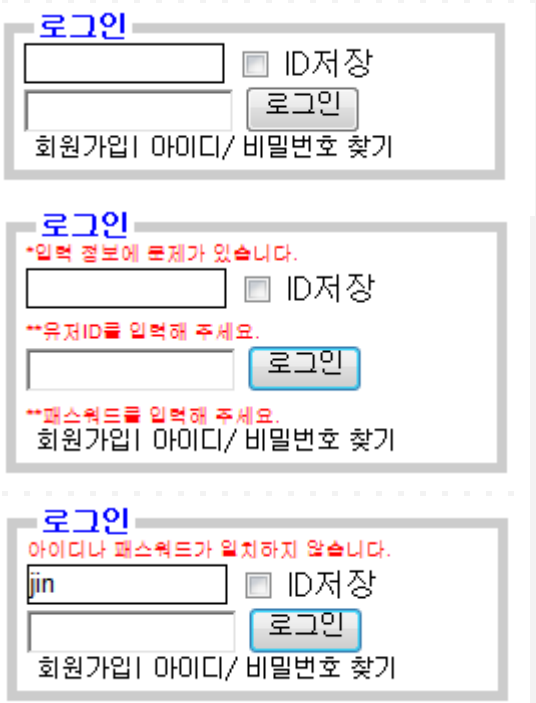
Spring MVC 프로젝트 생성

- Web project를 Spring Project로 전환



사용자 로그인 구성





login.jsp

LoginFormController

Model

LoginValidator

UserServise

UserDao

DataSource

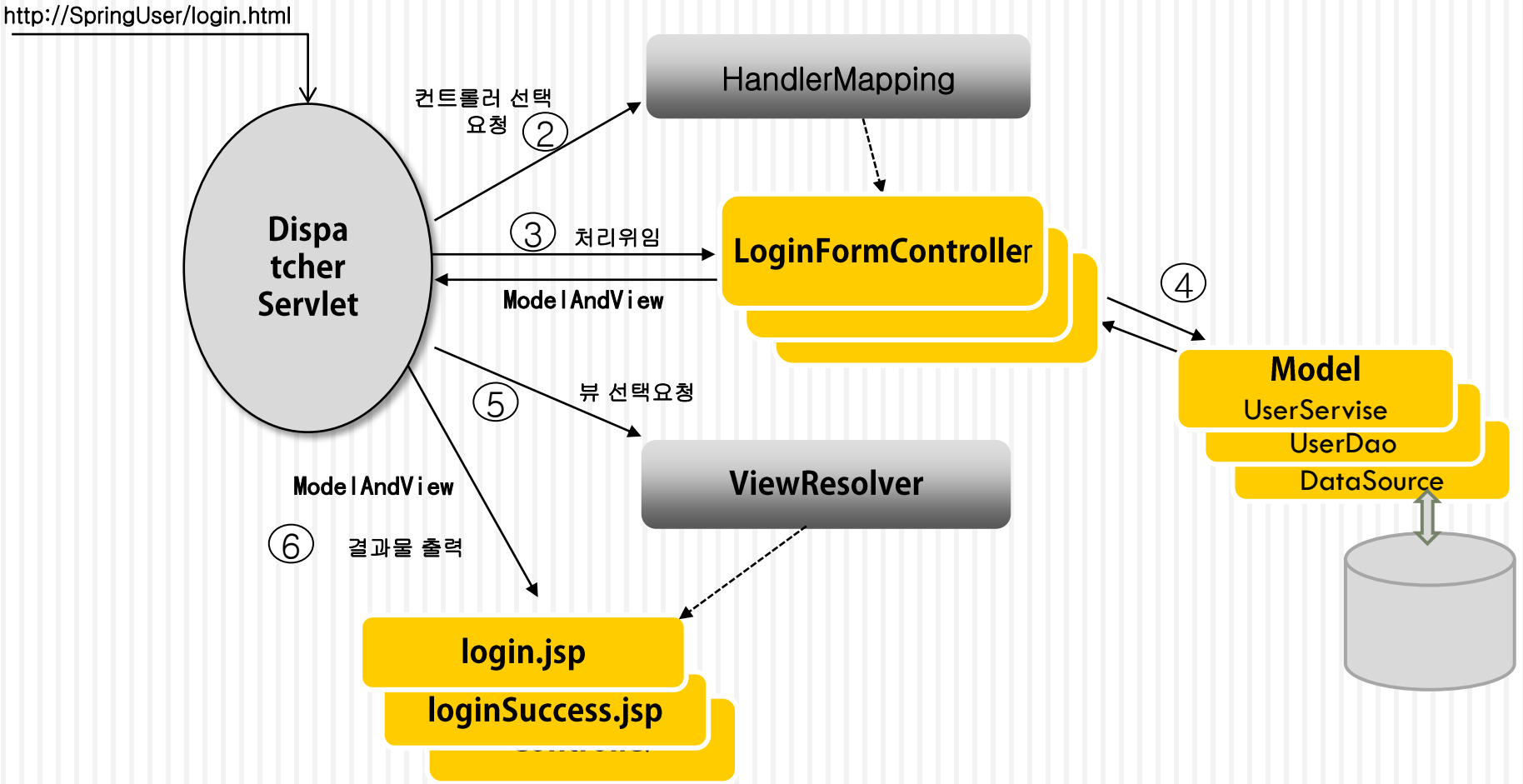
로그인화면

환영합니다,이현상씨 !

〈회원정보 수정〉

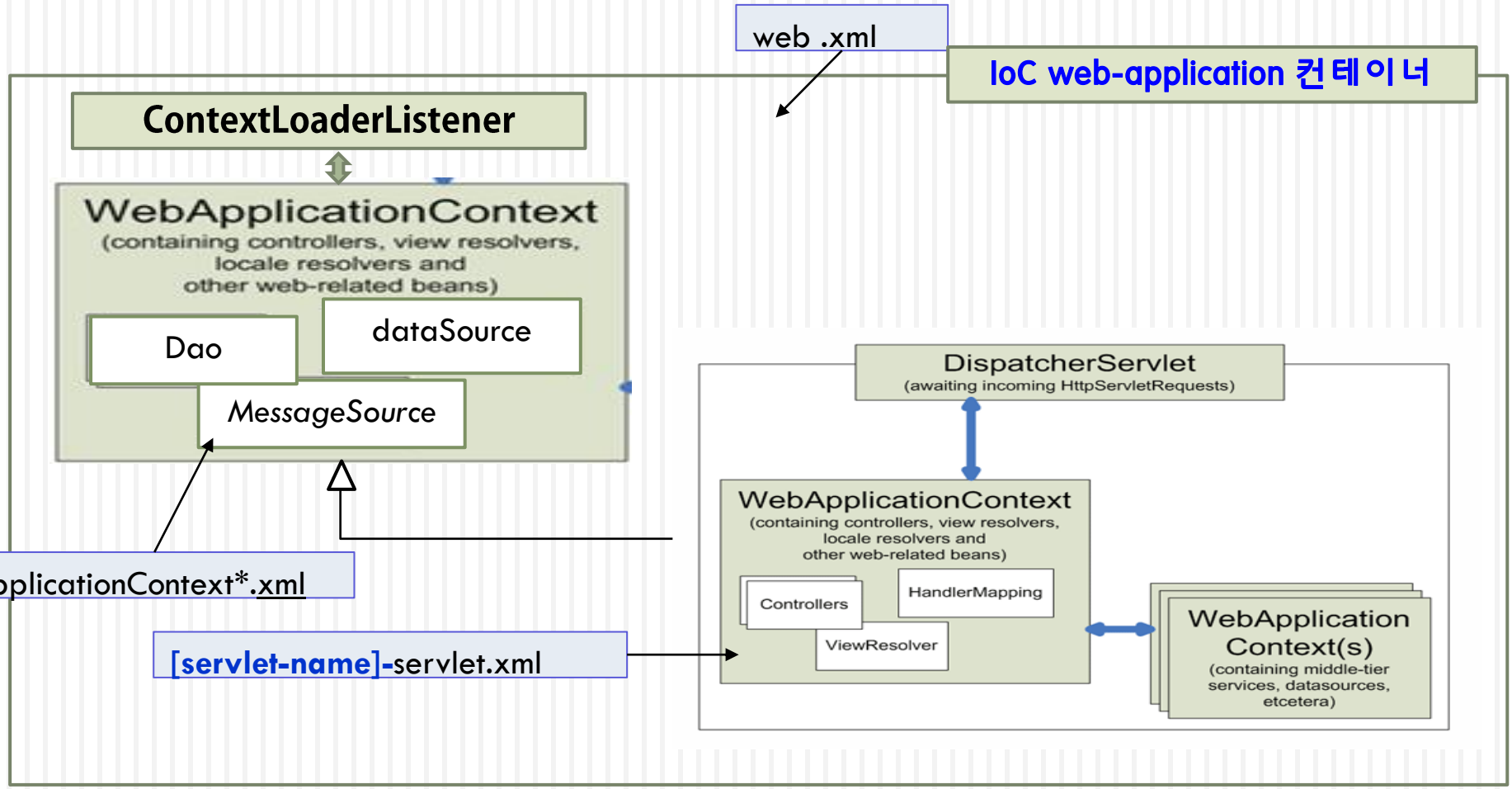
loginSuccess.jsp

로그인 MVC 처리흐름


http://SpringUser/login.html
Dispa
tcher
Servlet
컨트롤러 선택
요청 ②
HandlerMapping
③ 처리위임
ModelAndView
LoginFormController
④
Model
UserServise
UserDao
DataSource
⑤ 뷰 선택요청
ViewResolver
ModelAndView
⑥ 결과물 출력
login.jsp
loginSuccess.jsp

Spring MVC의 환경 구성

- Ioc 컨테이너 : 웹 애플리케이션 컨텍스트 구성
- Webroot의 /WEB-INF/web.xml 파일에 설정
  - root 애플리케이션 컨텍스트와 서블릿 애플리케이션 컨텍스트 설정

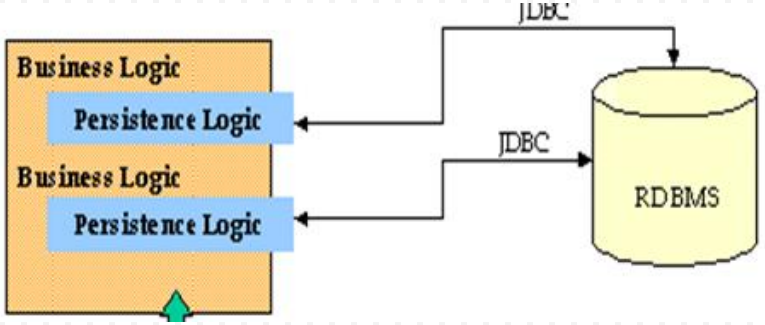


IoC 컨테이너 생성 파일 (web.xml) 작성

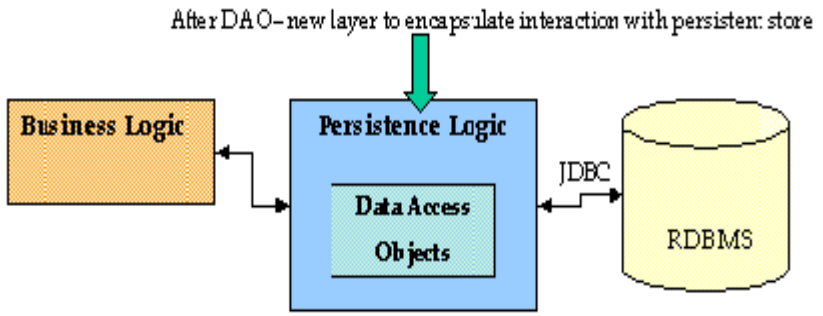
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
  <filter>
    <filter-name>CharacterEncodingFilter</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>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>SpringUser</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>SpringUser</servlet-name>
    <url-pattern>*.html</url-pattern>
  </servlet-mapping>
</web-app>
```

Spring MVC Model

- Business logic
  - 업무에 필요한 데이터를 얻기위해 수행하는 응용프로그램의 일부, 즉 데이터 입력, 수정, 조회,가공 및 보고서 처리 등을 수행하는 루틴

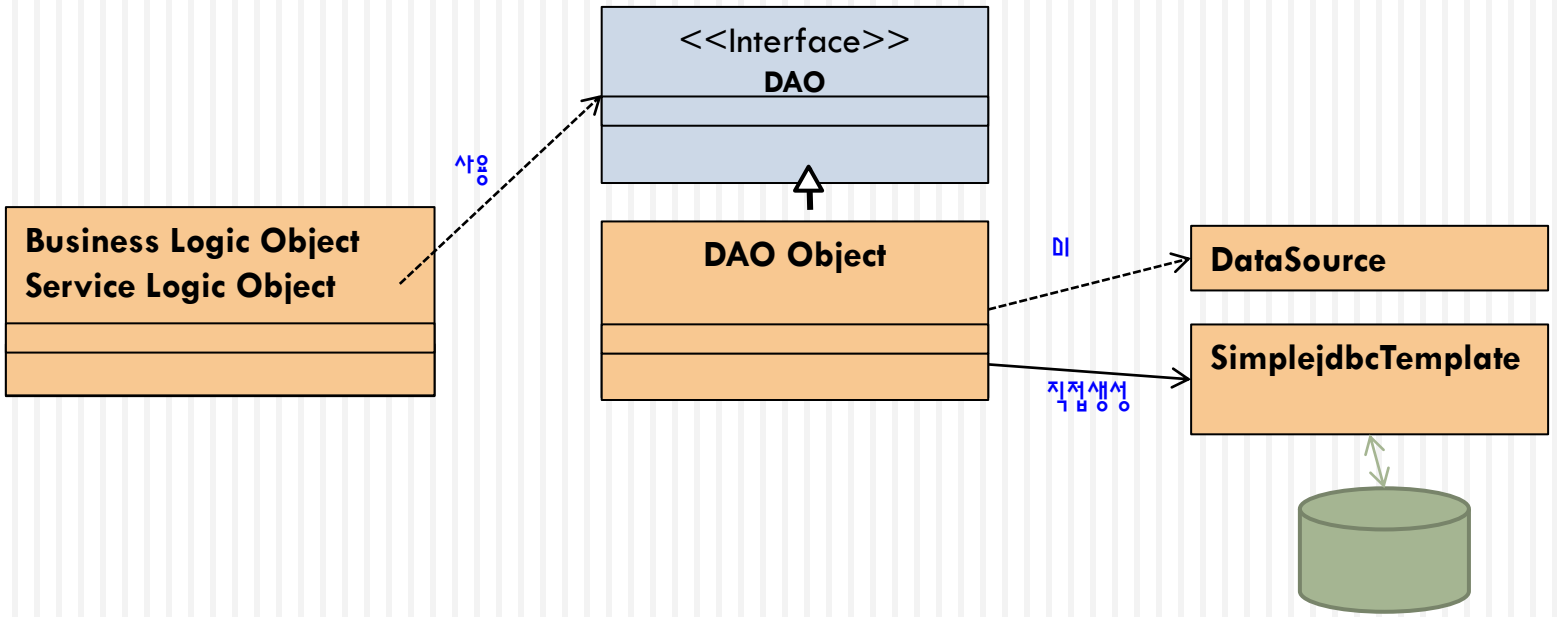


- DAO 디자인 패턴 - 데이터 접근 객체(DAO)를 따로 두어 데이터베이스 업체 변경이나 데이터베이스 유형변경과 같이 데이터 소스에 변경이 가해질 경우, DAO에만 수정이 요구되어 business 객체들에 가해지는 영향을 최소화 한다.



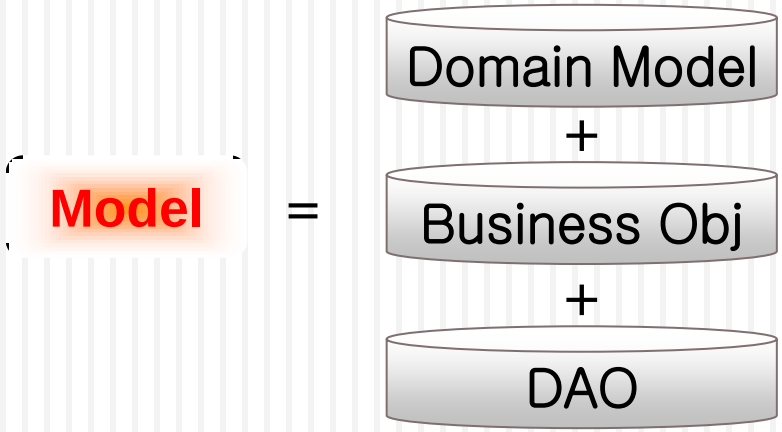
DAO 디자인 패턴

- 디자인 패턴을 사용하여 저수준의 데이터액세스(DAO) 로직과 고급 Business 로직을 분리한다.
- DAO 객체는 DAO 인터페이스를 상속받아 구현
- Business 서비스 객체들은 DAO 인터페이스 객체를 사용하며, 이 DAO 인터페이스를 구현한 클래스(DAO)에 대해서는 모른다.



Spring MVC Model

- Business 객체는 DAO 객체를 사용하여 업무에 필요한 데이터(model)를 얻고 Controller에 전달하는 기능을 수행하지만 데이터(Model)를 Controller가 원하는 형태로 가공하여 전해줄 수도 있다.
  - db -> (DAO) -> Model -> (Service) -> Model for Controller
  - 저수준의 데이터액세스(DAO) 로직과 고급 Business 로직 분리
  - 비지니스로직의 Model = 고객구매정보(고객정보(dao) + 구매목록(dao))
  - 서비스(비즈니스)로직의 data = 가격(DAO) + 부가세 계산



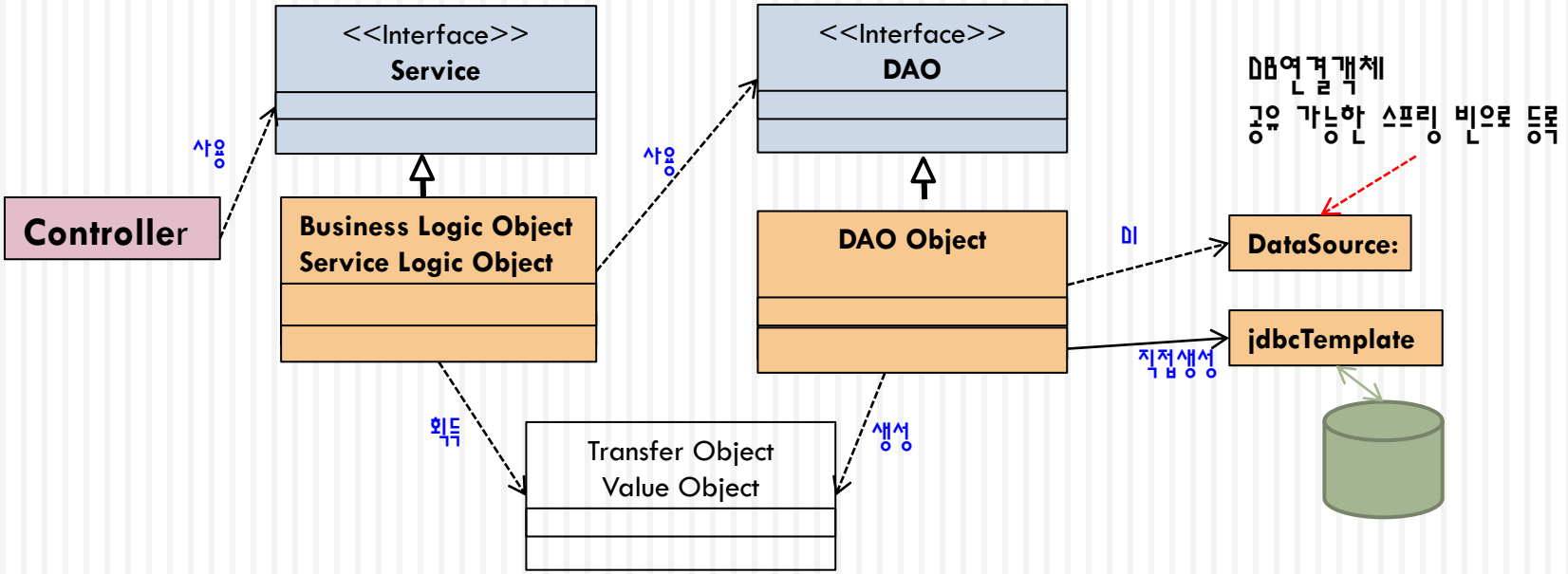
DAO design Pattern

Data Access Object(DAO)

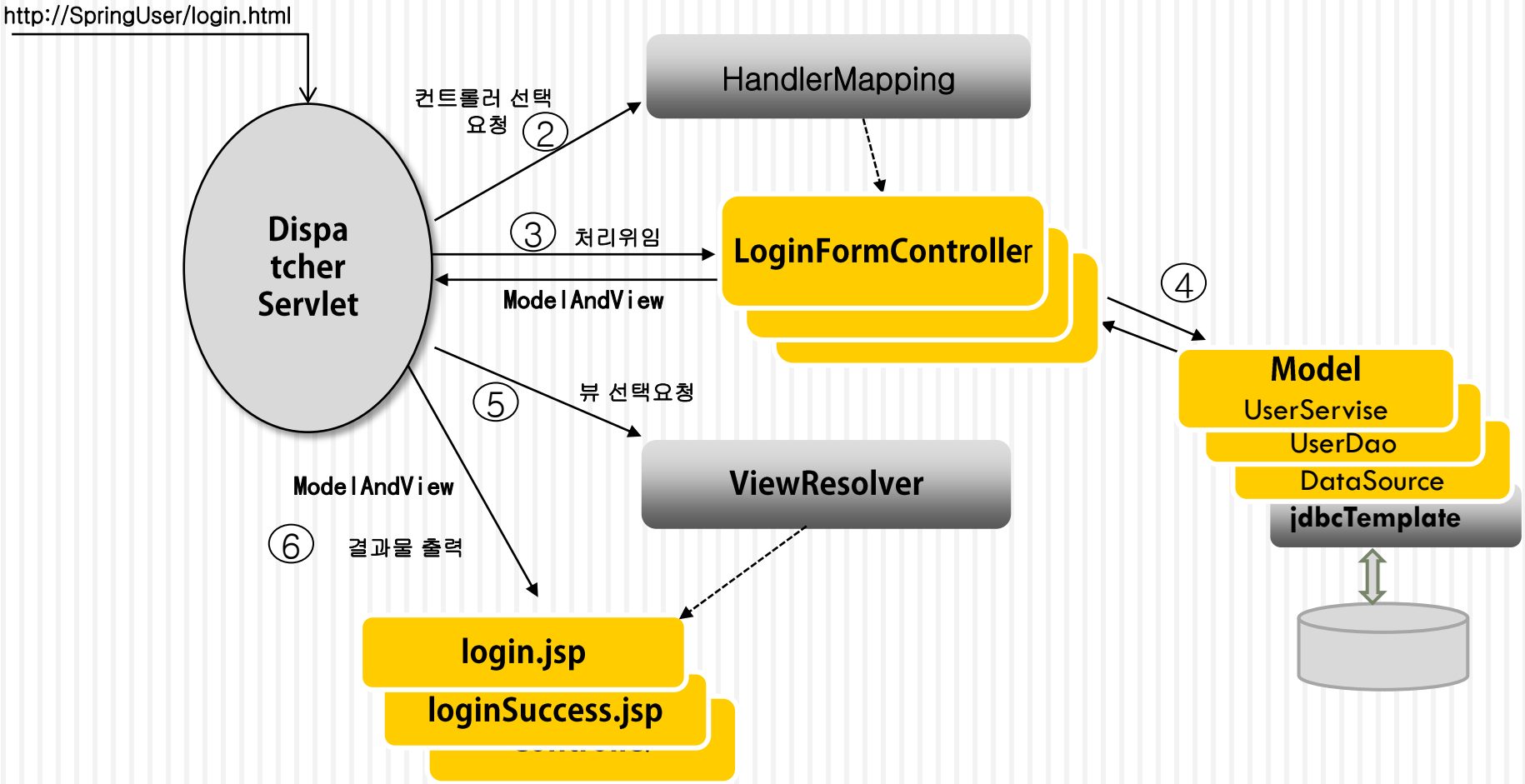
- DataSource에 접근하는 부분을 캡슐화 및 추상화
- 데이터를 저장하거나 가져오기 위해서 커넥션을 관리
- JdbcTemplate 객체에서 데이터를 검색해서 **전달객체**에 넣고 Business 객체에 전달하면 Business 객체는 **전달객체**를 통해서 DataSource에 있는 값을 얻을 수 있다.

jdbcTemplate 객체

- DataSource 로 부터 커넥션 추출
- 적절한 statement 객체 준비
- SQL CRUD 오퍼레이션 실행
- result set 탐색 및 표준 컬렉션 객체들에 결과를 채워 넣어주는 작업
- SQLException 예외를 처리하고 이를 오류에 특화된 예외 계층 구조로 변환하는 작업

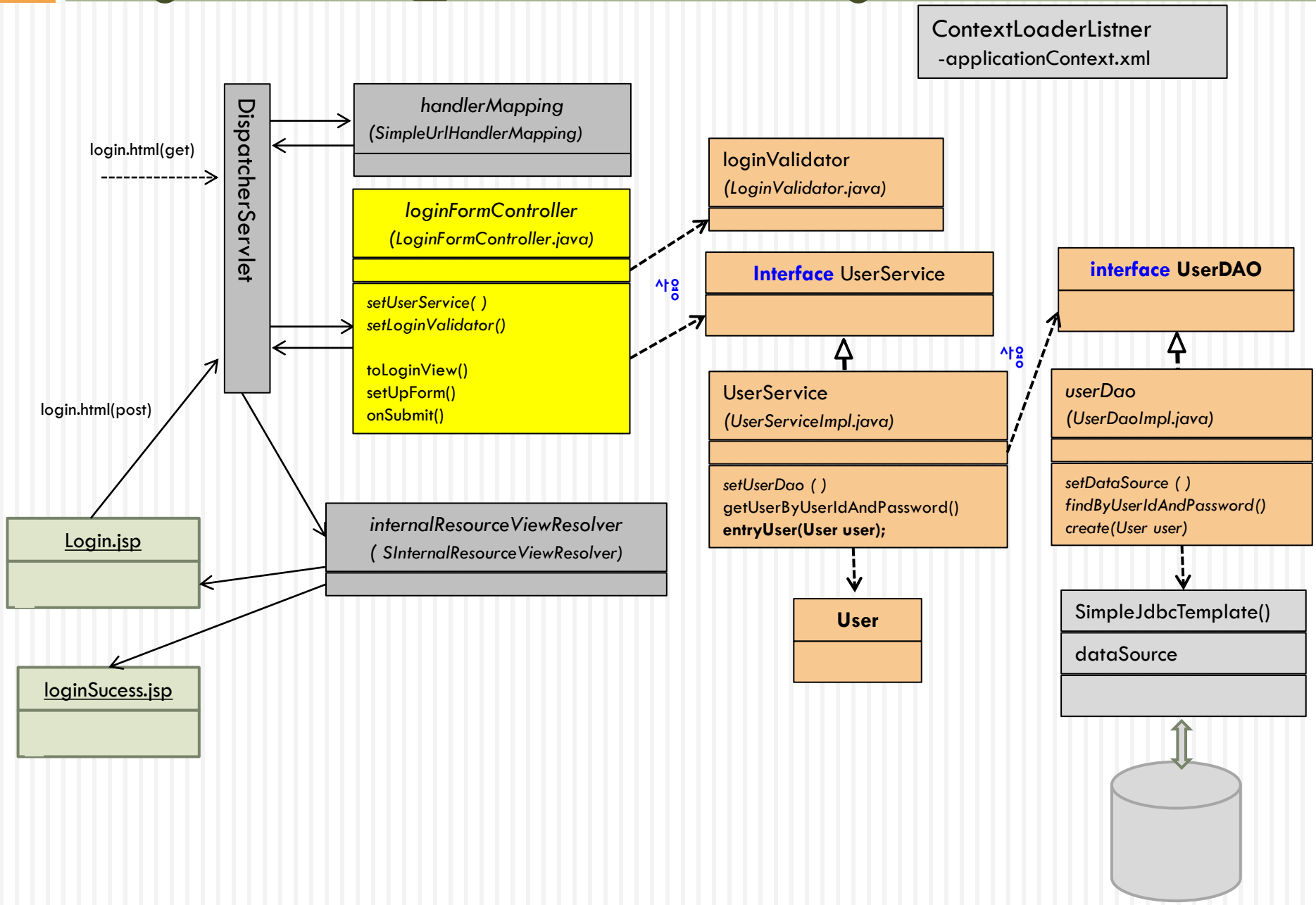


로그인 MVC 처리흐름



jdbcTemplate

사용자 로그인 MVC bean 구성



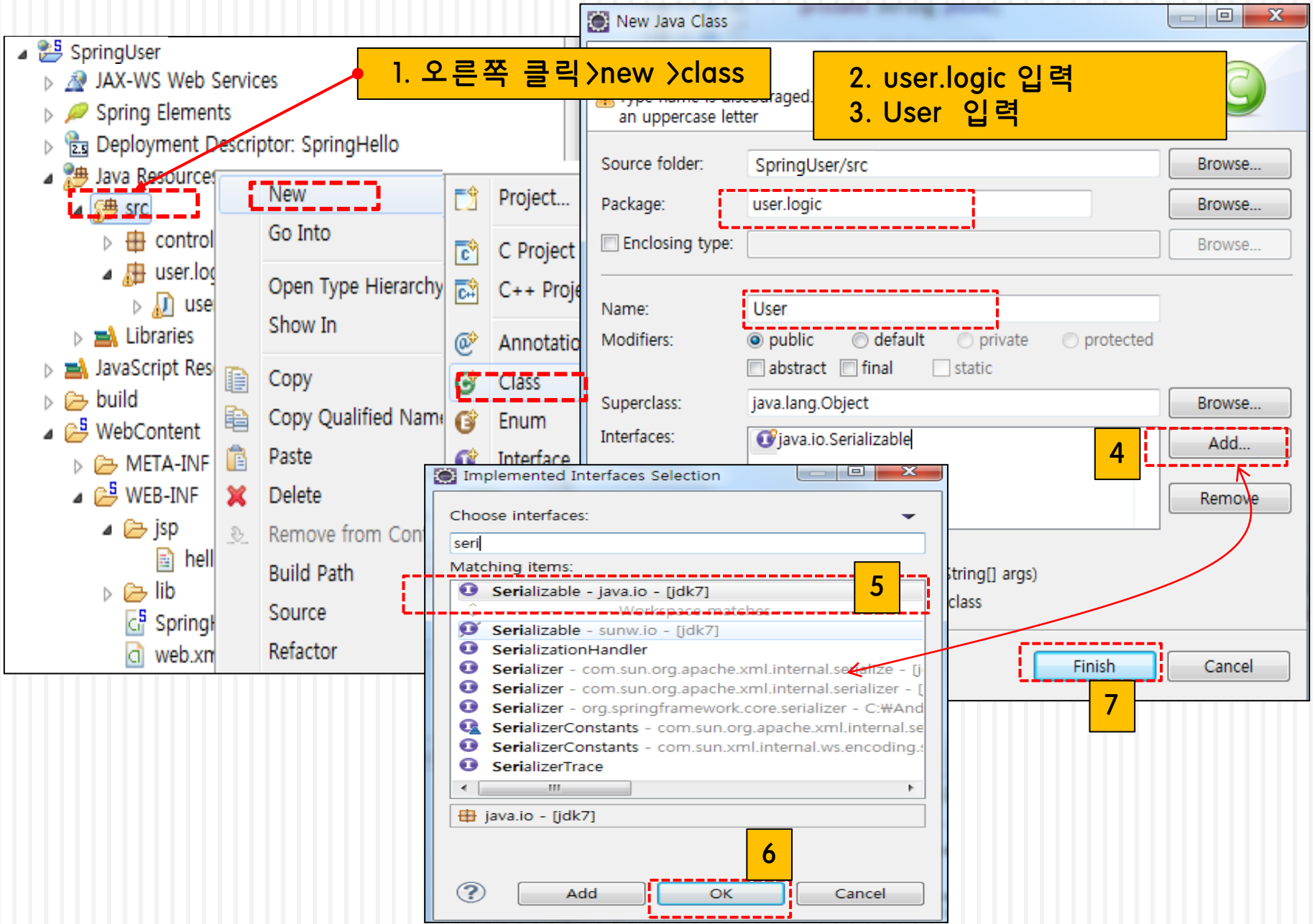
회원관리table 생성

```
CREATE TABLE  member_tbl  (
   no  int(8) NOT NULL AUTO_INCREMENT,
   id  varchar(10) NOT NULL,
   pass  varchar(10) NOT NULL,
   name  varchar(20) NOT NULL,
   jumin  char(13) NOT NULL,
   zip  char(7) DEFAULT NULL,
   addr1  varchar(100) DEFAULT NULL,
   addr2  varchar(50) DEFAULT NULL,
   phone  varchar(20) DEFAULT NULL,
   email  varchar(30) DEFAULT NULL,
  PRIMARY KEY ( no )
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

	PK	열	데이터 유형	기본값	NULL?	주석
1	PK	no	int(8)		NOT NULL	
2		id	varchar(10)		NOT NULL	
3		pass	varchar(10)		NOT NULL	
4		name	varchar(20)		NOT NULL	
5		jumin	char(13)		NOT NULL	
6		zip	char(7)			
7		addr1	varchar(100)			
8		addr2	varchar(50)			
9		phone	varchar(20)			
10		email	varchar(30)			

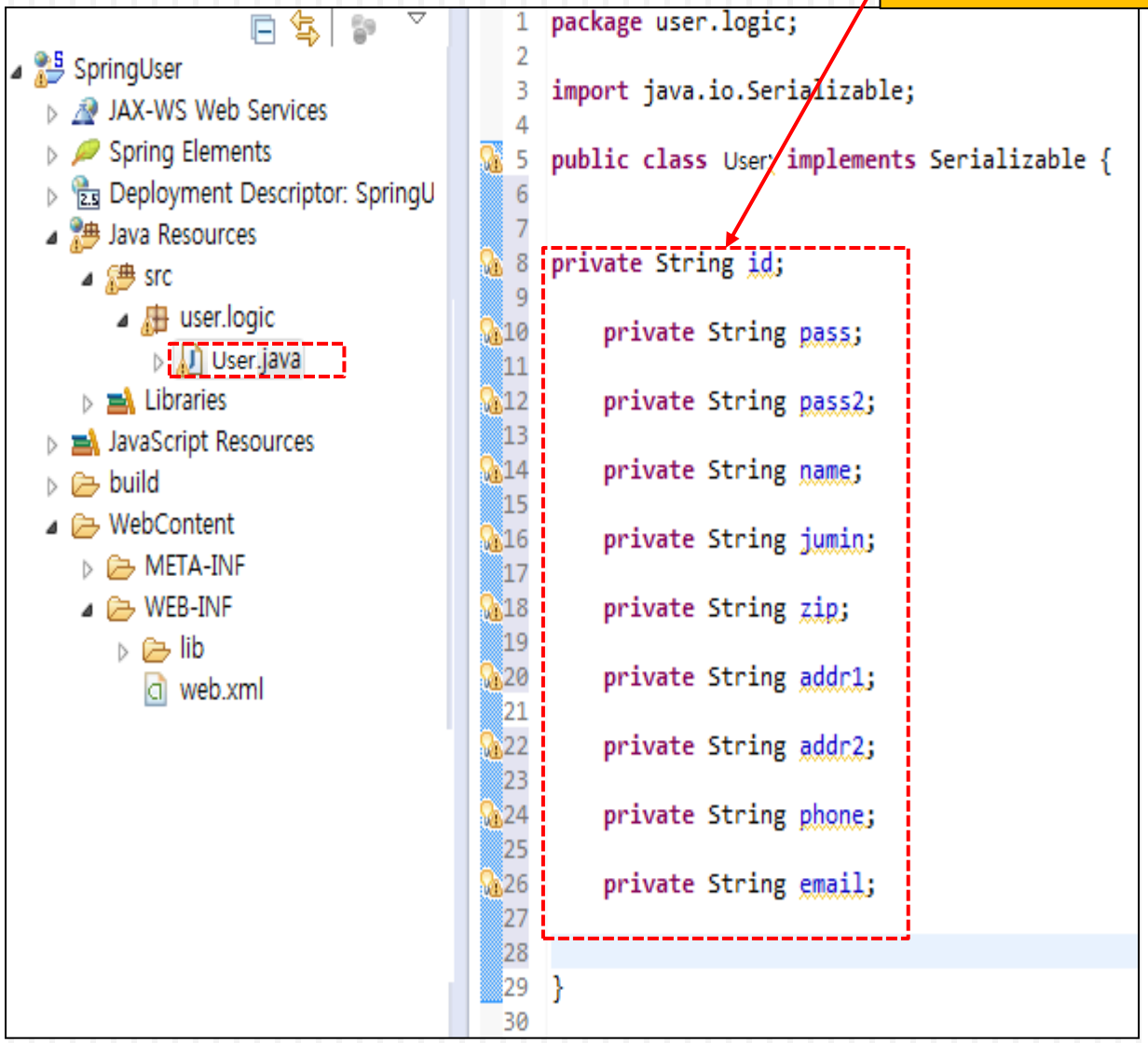
1	1	id	pass	name	jumin	zip	addr1	addr2	phone	email
2	2	jjin	1234	이현상	5708201566221	123-121	경기도 화성 봉담읍 장안대학	IT 학부 인터넷정보통신과	010-1234-1231	jjin@1234.aasd.qw
3	3	kim	1234	김고현	8905161566221	432-789	경기도 화성 봉담읍 장안대학	인터넷	010-5432-1231	kim@ds.as.aa
4	4	lee	1234	이하나	9209161566221	445-756	경기 화성시 봉담읍 상리 장인	인터넷	010-1234-5432	jjin@asd.asdf.asd

Value (Transfer:모델) 객체 생성



전달 객체 gatter & setter 생성

1. 입력



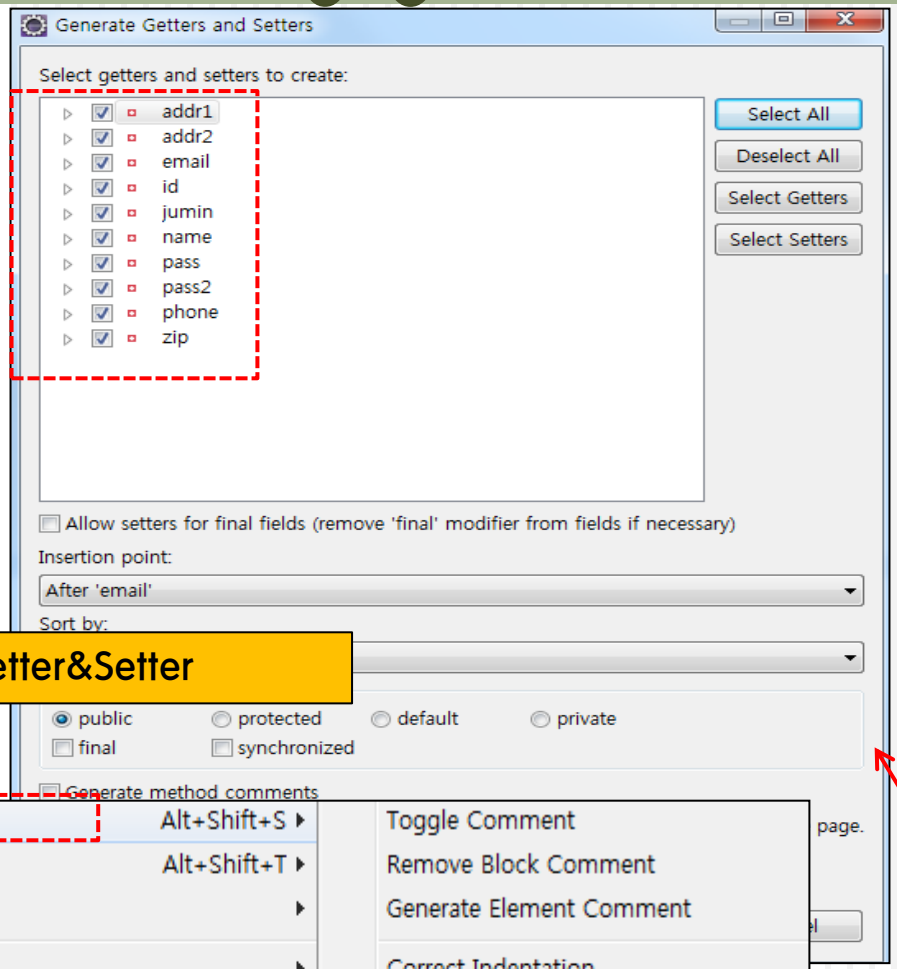

```
1  package user.logic;
2
3  import java.io.Serializable;
4
5  public class User implements Serializable {
6
7
8  private String id;
9
10     private String pass;
11
12     private String pass2;
13
14     private String name;
15
16     private String jumin;
17
18     private String zip;
19
20     private String addr1;
21
22     private String addr2;
23
24     private String phone;
25
26     private String email;
27
28
29 }
30
```

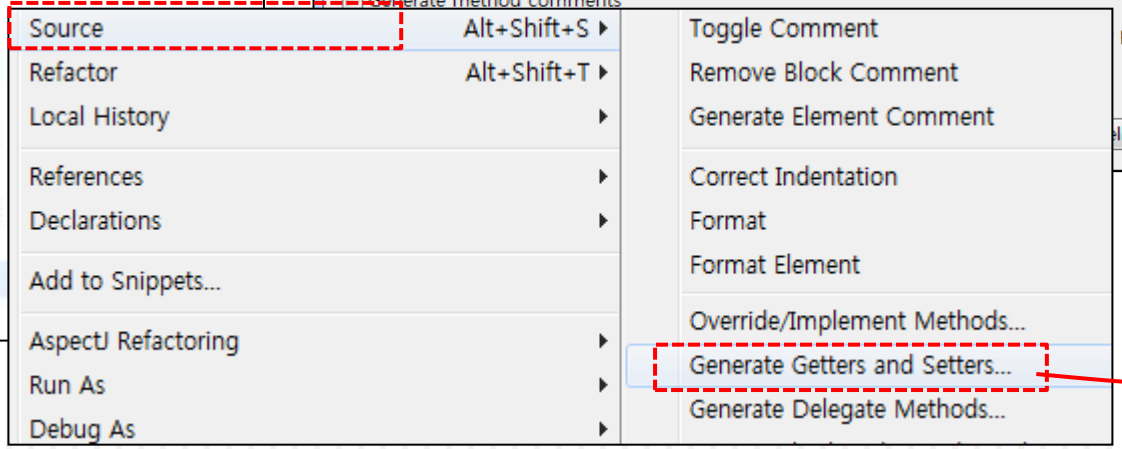
열	데이터 유형
no	int(8)
id	varchar(10)
pass	varchar(10)
name	varchar(20)
jumin	char(13)
zip	char(7)
addr1	varchar(100)
addr2	varchar(50)
phone	varchar(20)
email	varchar(30)

모델 객체 gatter & setter 생성

```
package user.logic;

import java.io.Serializable;

public class User implements Serializable {


private String id;

    private String pass;

    private String pass2;

    private String name;

    private String jumin;

    private String zip;

    private String addr1;

    private String addr2;

    private String phone;

    private String email;

}
```

1. 오른쪽클릭>Source>Generate Getter&Setter





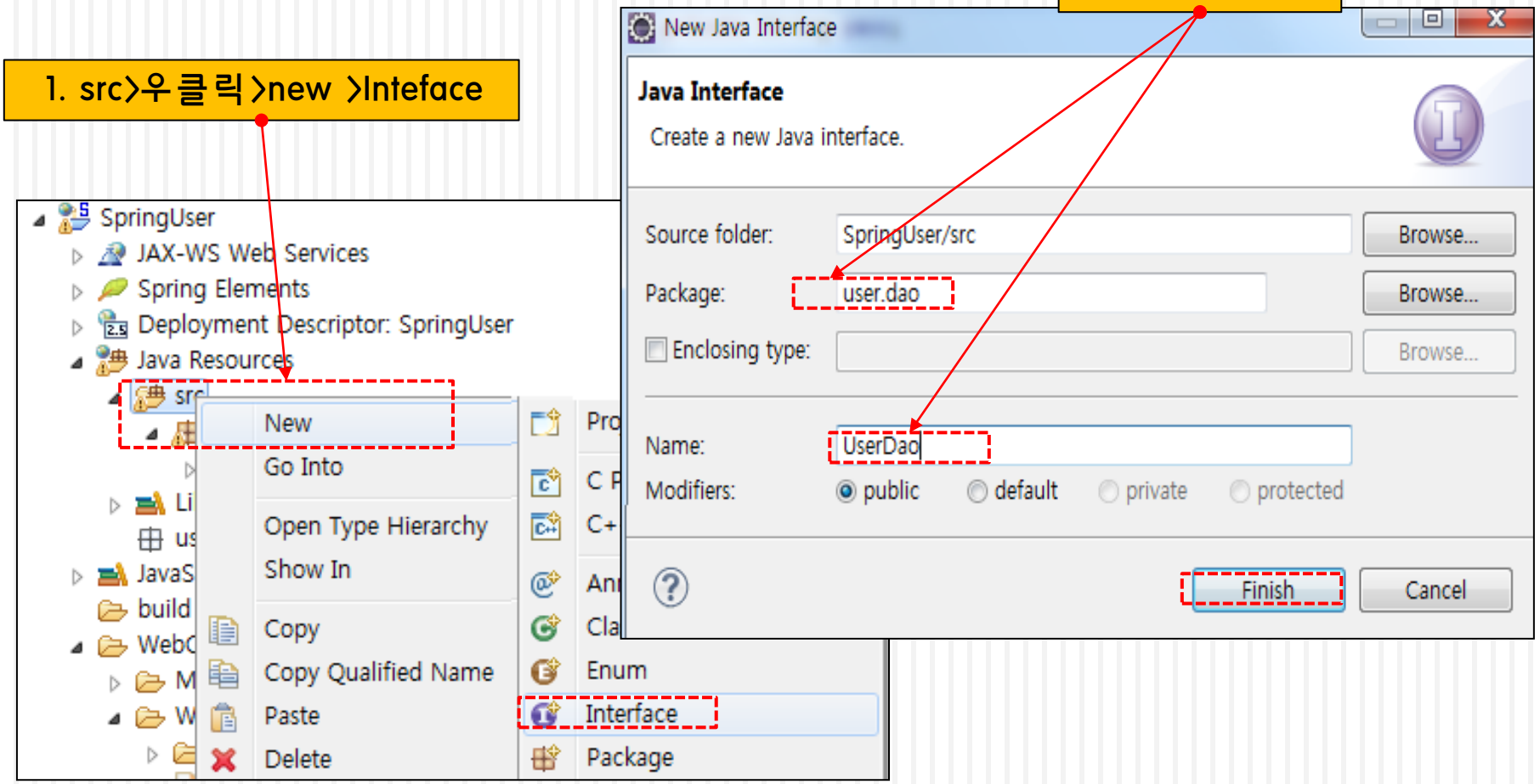
DAO 클래스 작성

- 일반적으로 DAO는 하나의 테이블에 CRUD 작업을 수행하며 다양한 형태의 검색 메서드를 구성할 수 있다.
- 조회의 경우만 반환값이 있다.
- Update() 메서드는 영향을 받은 레코드의 개수를 반환한다.
- DI(의존관계 주입 구현을 위하여 Interface 를 먼저 작성하고 이를 상속받아 DAO를 구현한다.

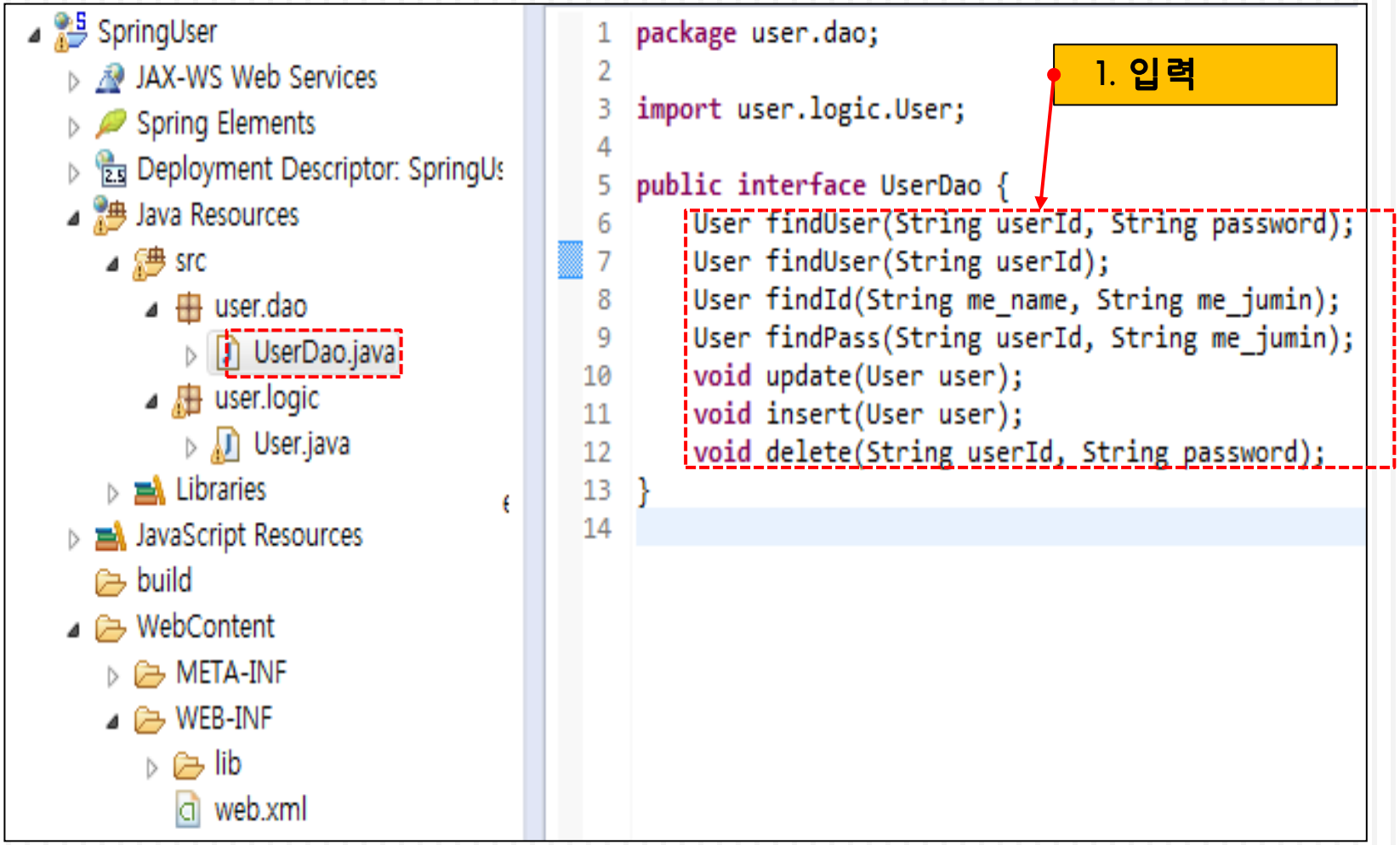
Dao Interface 생성

1. src>우클릭>new >Inteface

2. 입력



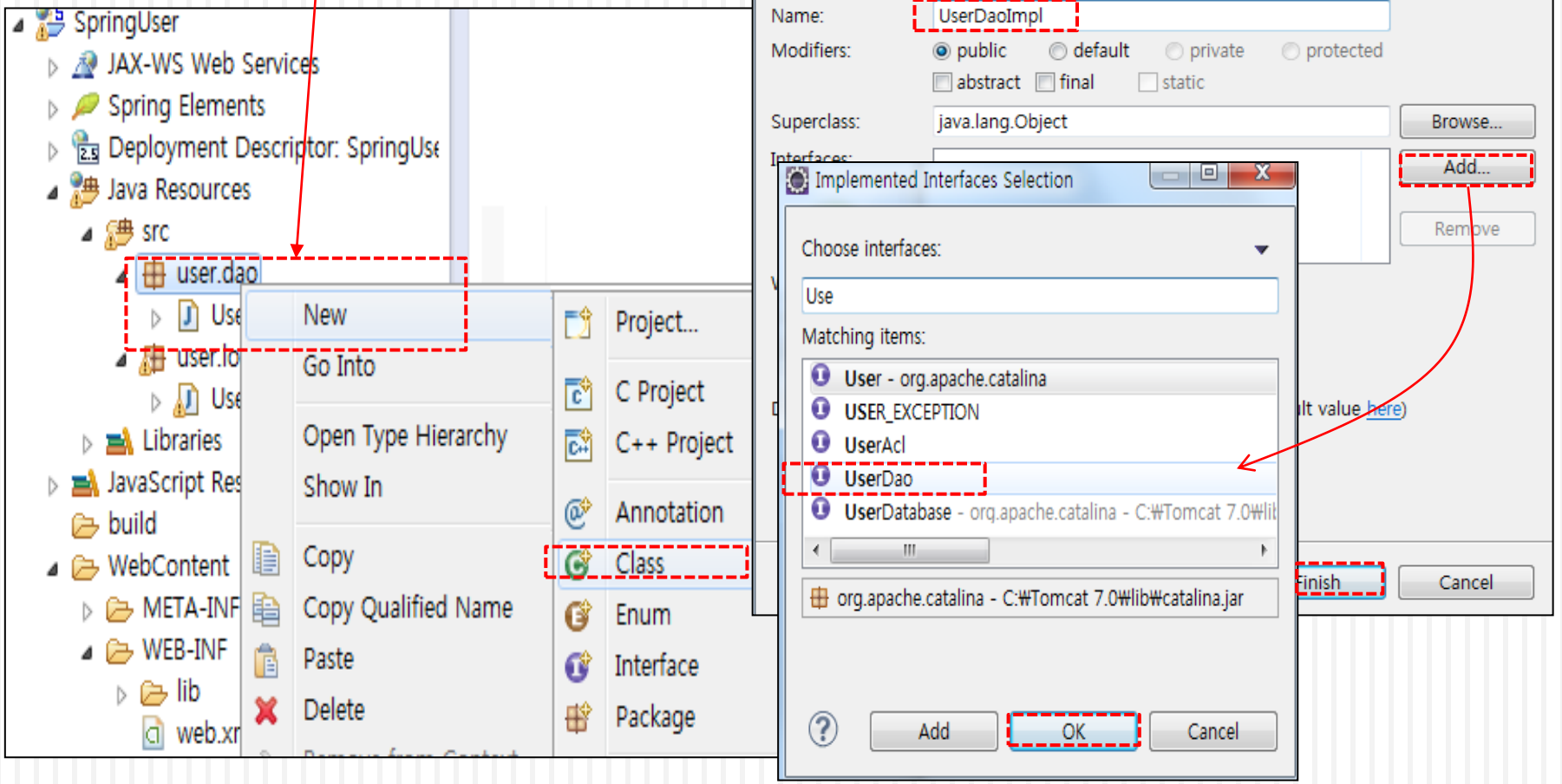
Dao Interface 생성



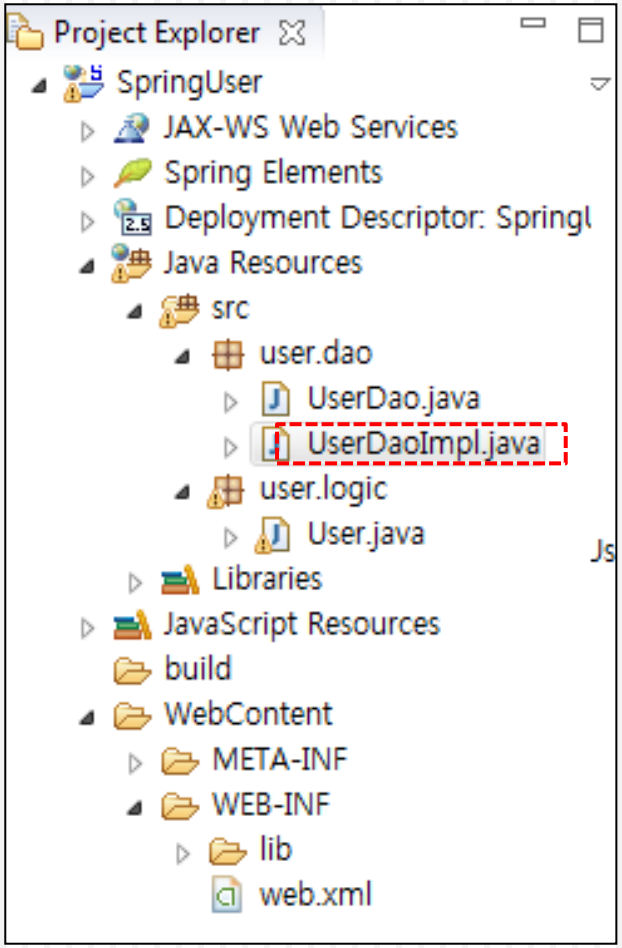
Dao 구현 클래스 생성

1. src>user.dao>우클릭>new >class

2. 입력



Dao구현 클래스생성




```java
package user.dao;

import user.logic.User;

public class UserDaoImpl implements UserDao {


    public User findUser(String userId, String password) {
        // TODO Auto-generated method stub
        return null;
    }

    public User findUser(String userId) {
        // TODO Auto-generated method stub
        return null;
    }

    public User findId(String me_name, String me_jumin) {
        // TODO Auto-generated method stub
        return null;
    }

    public User findPass(String userId, String me_jumin) {
        // TODO Auto-generated method stub
        return null;
    }

    public void update(User user) {
        // TODO Auto-generated method stub

    }

    public void insert(User user) {
        // TODO Auto-generated method stub

    }

    public void delete(String userId, String password) {
        // TODO Auto-generated method stub

    }

}
```

SQL 파라미터 바인딩 방법

- 위치 파라미터 치환자

```
public void delete(String userId, String password) {
    template.update("delete from member_tbl where id=? AND pass=?", userId, password );
}
```

- 이름 파라미터 치환자
  - *INSQL=* "Insert into member(id,name,pass) values (:id, :name, :pass)"

```
public void insert(User user) {
    SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(user);
    template.update(INSQL, parameterSource);
}
```

- SqlParameterSource 클래스
  - DB테이블의 필드 이름과 User 객체의 속성 이름을 매핑(이름이 동일시)
- RowMapper 클래스
  - ResultSet에서 값을 가져와 원하는 타입으로 매핑할 때 사용

SQL 작성

```
private static final String GETUSER_IDPASS = "SELECT id, pass, name, jumin, zip,"
+ " addr1, addr2, phone, email FROM member_tbl WHERE id = ? AND pass = ?";

private static final String GETUSER_ID = "SELECT id, pass, name, jumin, zip,"
+ " addr1, addr2, phone, email FROM member_tbl WHERE id = ?";

private static final String GETID  = "select id from member_tbl where name=? and jumin=?";

private static final String GETPASS  = "select pass from member_tbl where id=? and jumin=?";

private static final String INSERT = "INSERT INTO member_tbl (id, pass, name, jumin, zip, addr1,
addr2, phone, email)"
+ " VALUES(:Id, :pass, :Name, :Jumin, :zip, :addr1, :addr2, :phone, :email)";

private static final String UPDATE = "UPDATE member_tbl SET pass = :pass, zip = :zip, addr1
= :addr1, addr2 = :addr2, phone = :phone, email = :email WHERE id = :id";
```

Spring JDBC를 위한 템플릿 클래스

- Connection을 구하고, try-catch-finally로 자원을 관리하는 등 예외 처리의 중복된 코드를 제거.
- JDBC를 위한 세 개의 템플릿 클래스
  - jdbcTemplate
    - 기본적인 JDBC 템플릿 클래스.
    - JDBC를 이용해서 데이터에 대한 접근을 제공.
  - NamedParameterjdbcTemplate
    - PreparedStatement에서 위치 기반의 파라미터가 아닌 이름을 가진 파라미터를 사용할 수 있도록 지원하는 템플릿 클래스.
  - SimplejdbcTemplate
    - JdbcTemplate기능을 향상 - 위치와 이름 기반 파라메터 사용
- JdbcTemplate 생성과 DataSource 주입

```
private jdbcTemplate template;

public void setDataSource(DataSource dataSource) {
      this.template = new jdbcTemplate(dataSource);
}
```

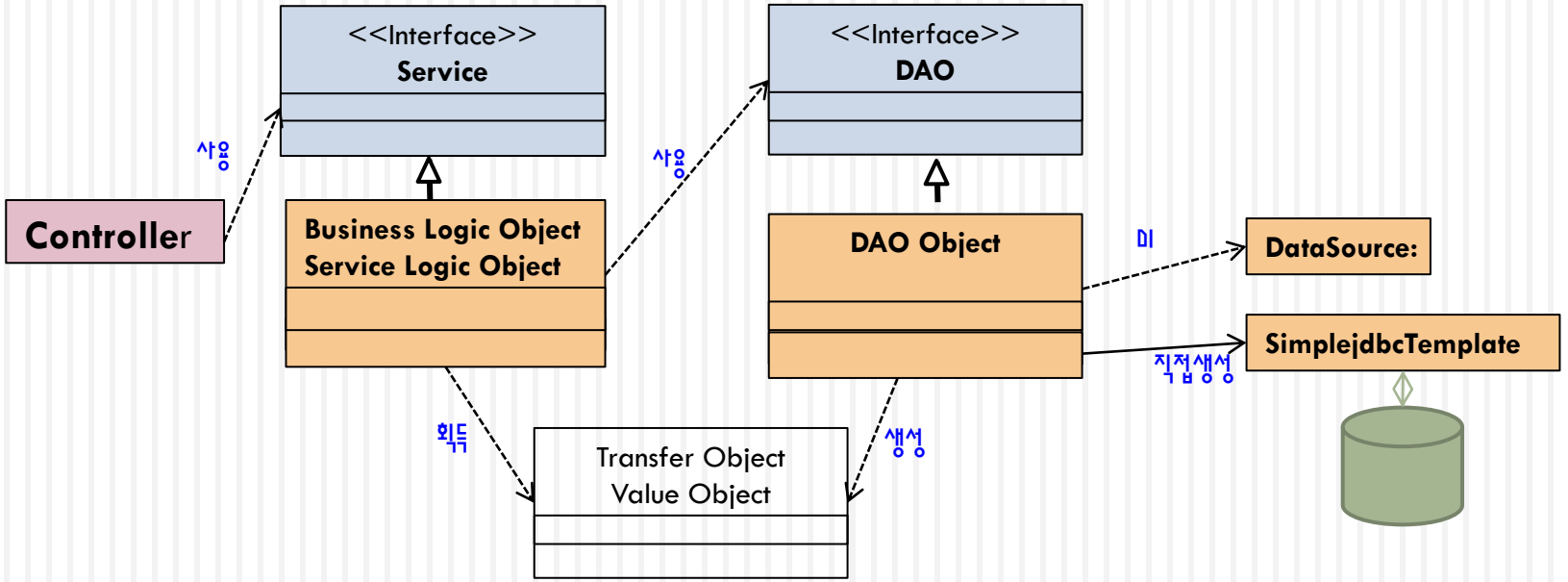
검색코드의 구현

```
@Override
public User findUser(String userId, String password) {
    RowMapper<User> mapper = new BeanPropertyRowMapper<User>(User.class);
    return  this.template.queryForObject(GETUSER_IDPASS, mapper, userId, password);
}

public User findUser(String userId) {
    RowMapper<User> mapper = new BeanPropertyRowMapper<User>(User.class);
    return  this.template.queryForObject(GETUSER_ID, mapper, userId);
}

public User findId(String me_name, String me_jumin) {
    RowMapper<User> mapper = new BeanPropertyRowMapper<User>(User.class);
    return  this.template.queryForObject(GETID, mapper, me_name, me_jumin);
}

public User findPass(String userId, String me_jumin) {
    RowMapper<User> mapper = new BeanPropertyRowMapper<User>(User.class);
    return  this.template.queryForObject(GETPASS, mapper, userId, me_jumin);
}
```

생성,수정,삭제 코드 구현

```
public void update(User user) {
    SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(user);
    this.template.update(UserDaoImpl.UPDATE, parameterSource);
}

public void insert(User user) {
    SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(user);
    this.template.update(UserDaoImpl.INSERT, parameterSource);

}

public void delete(String userId, String password) {
    this.template.update("delete from member_tbl where id = ? AND pass = ?", userId,
    password );
}
```

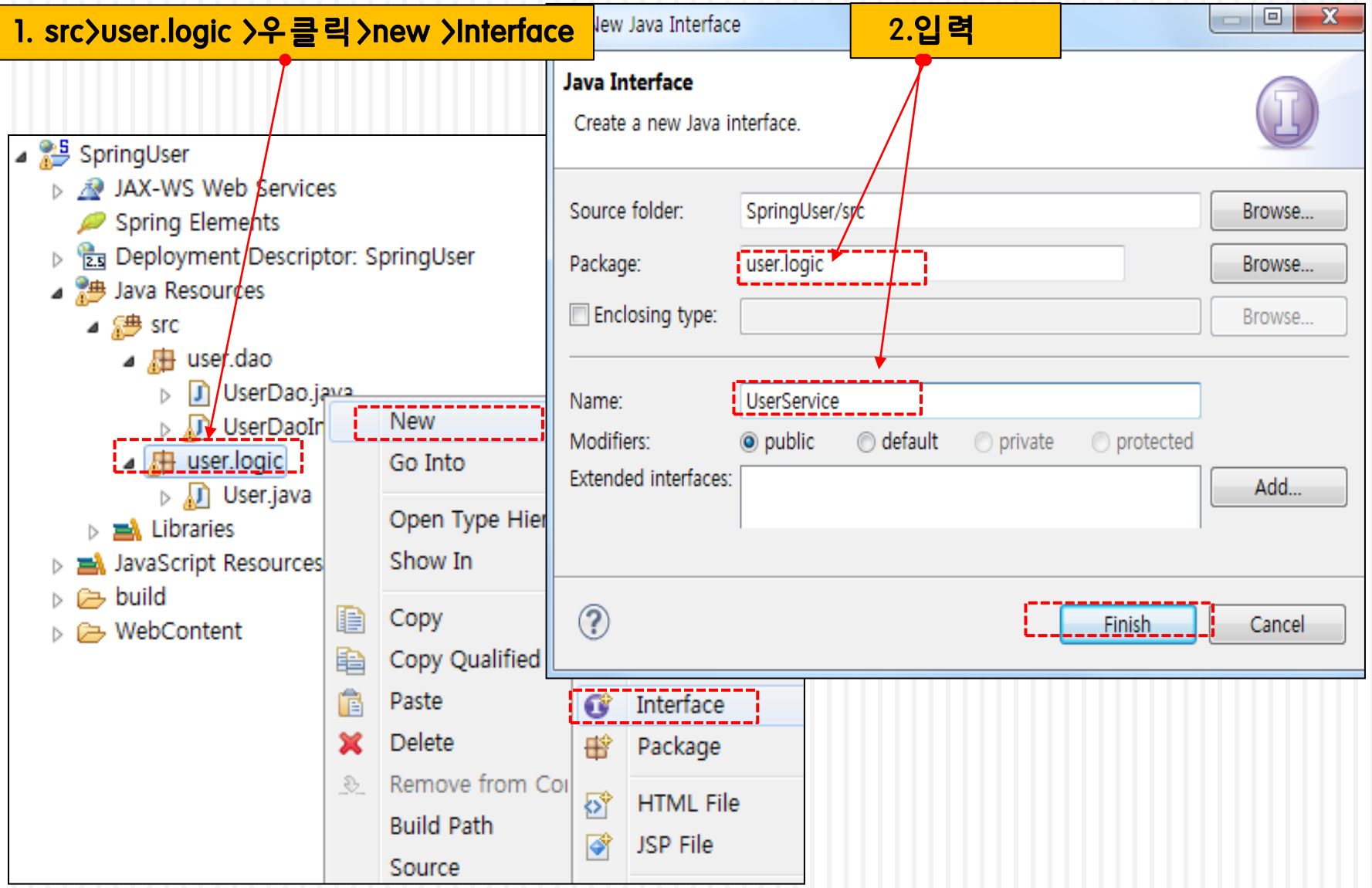
Service (Business)클래스 작성



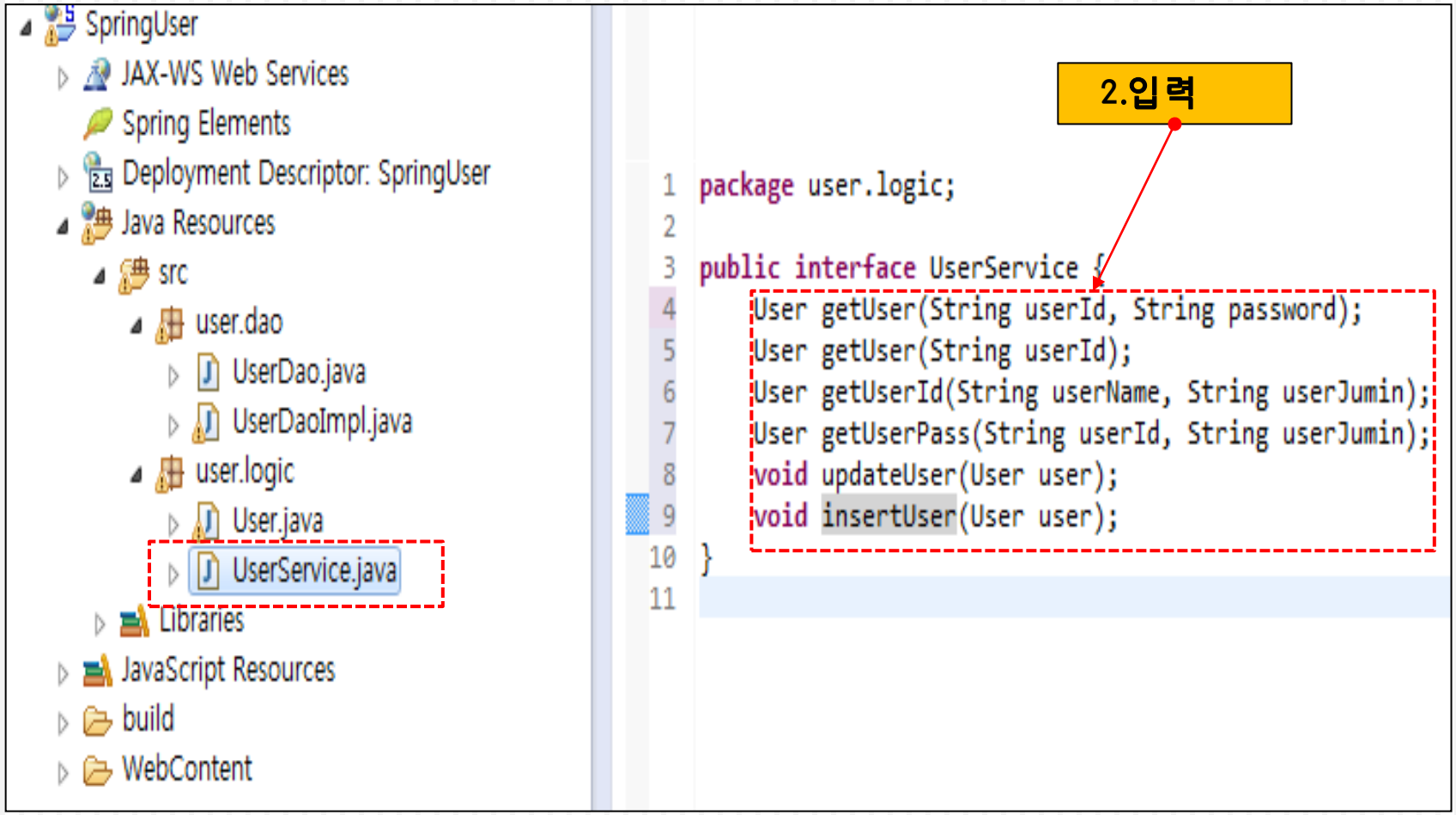
Business interface 생성

1. src>user.logic >우클릭>new >Interface

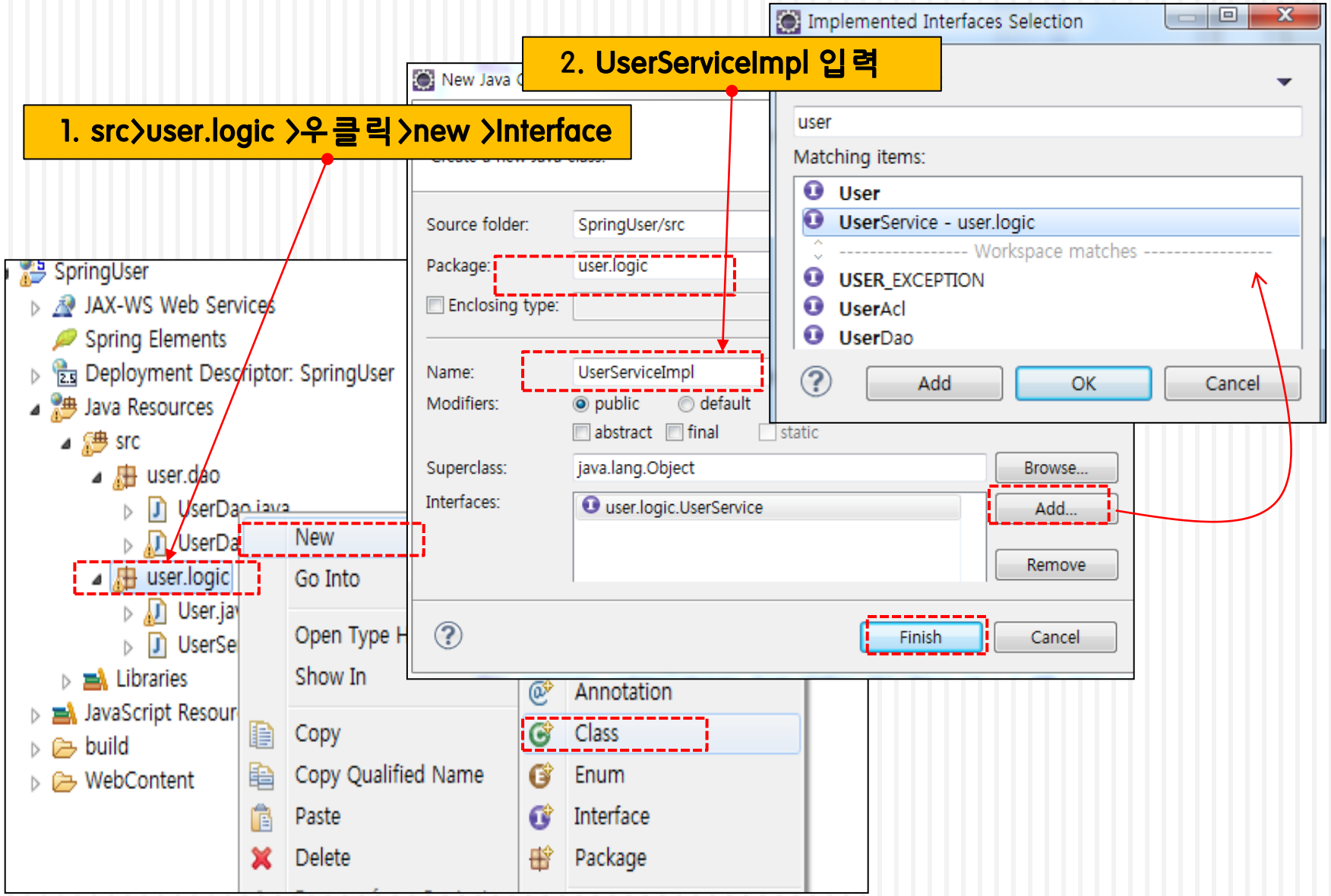
2.입력



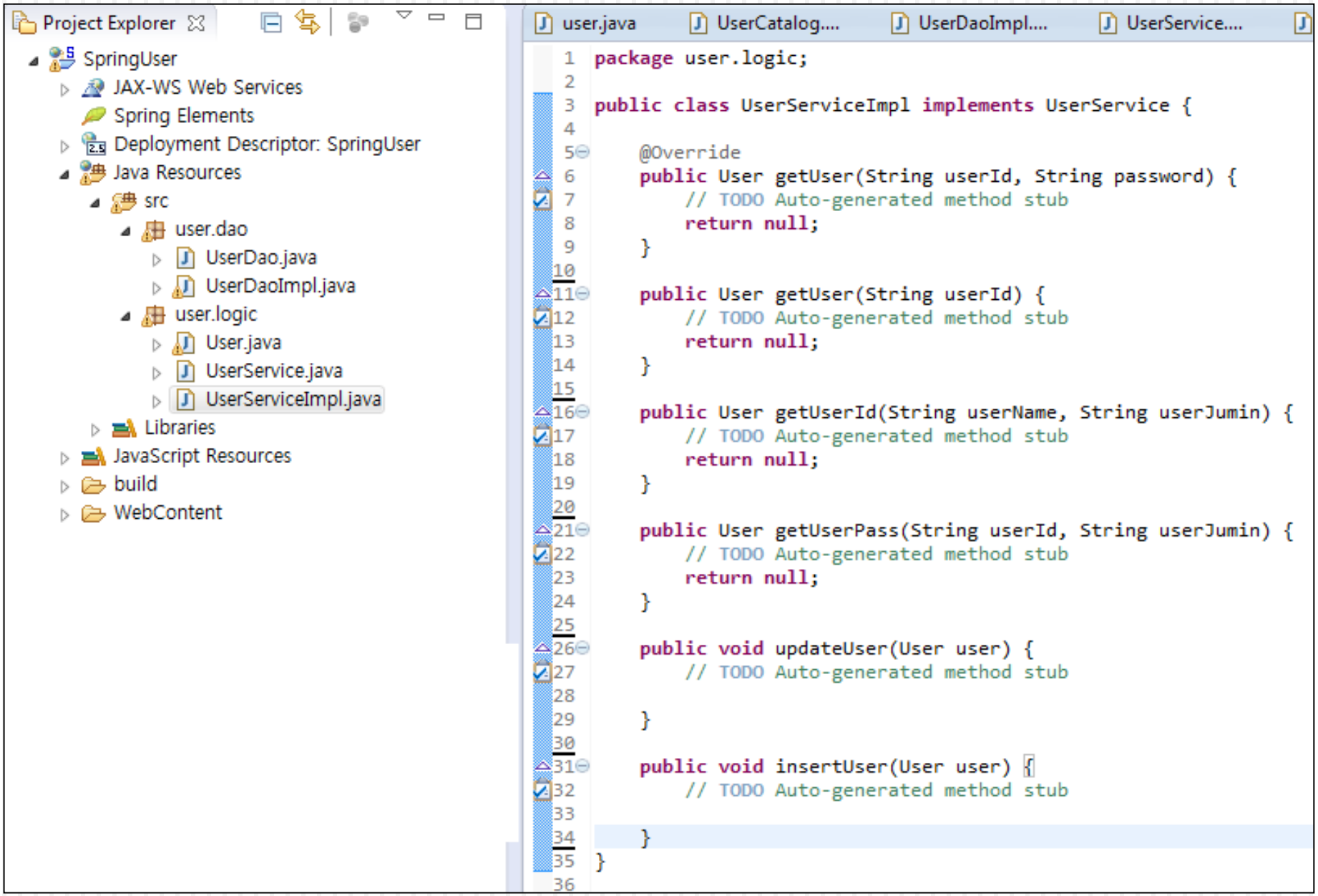
Business interface 생성



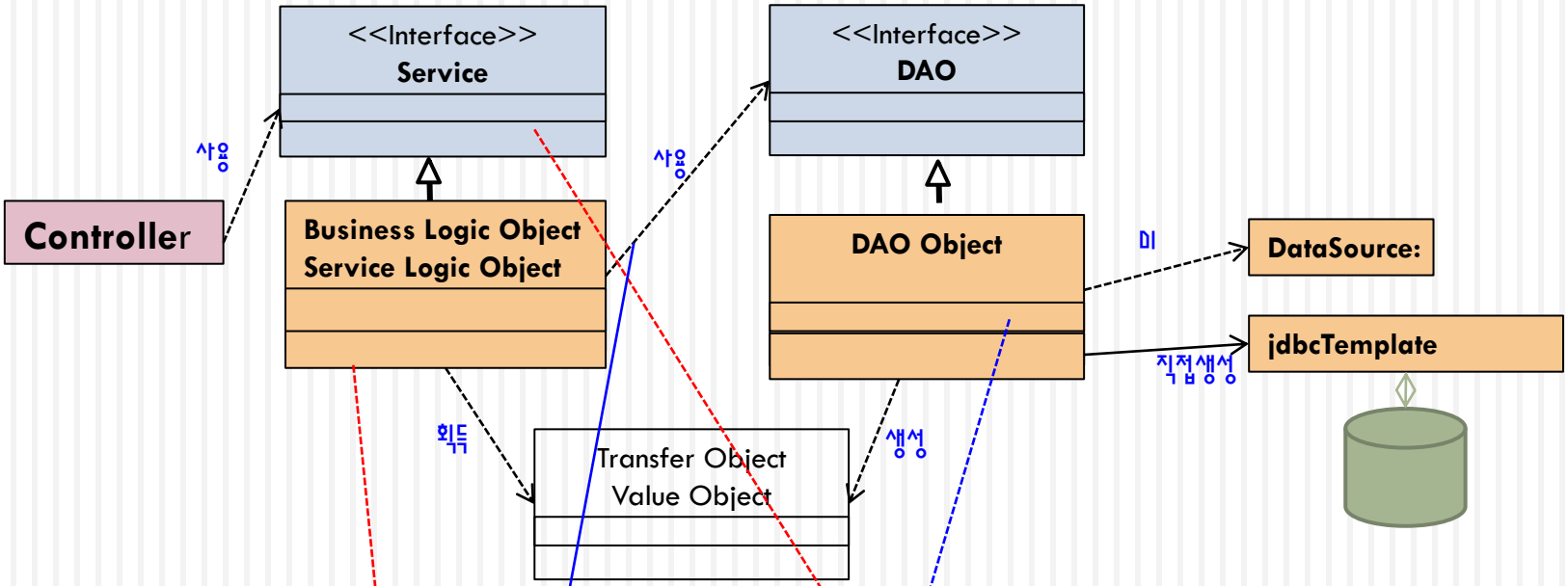
Business 구현 클래스 작성



Business 구현 객체 생성



Dao 구현객체 주입




```
public class UserServiceImpl implements UserService {

    private UserDao  userDao;

    public void setUserDao(UserDao  userDao) {
        this.userDao =  userDao;
    }
}
```

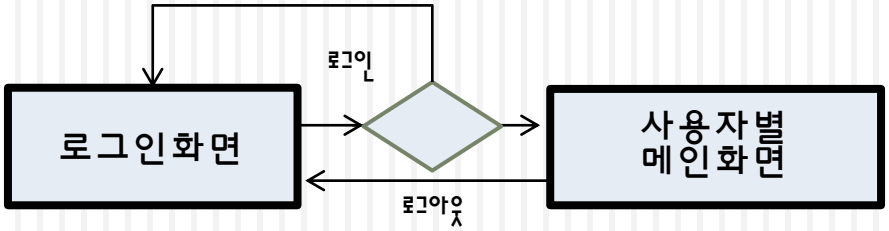
스프링 root 애플리케이션 컨텍스트에서 결정

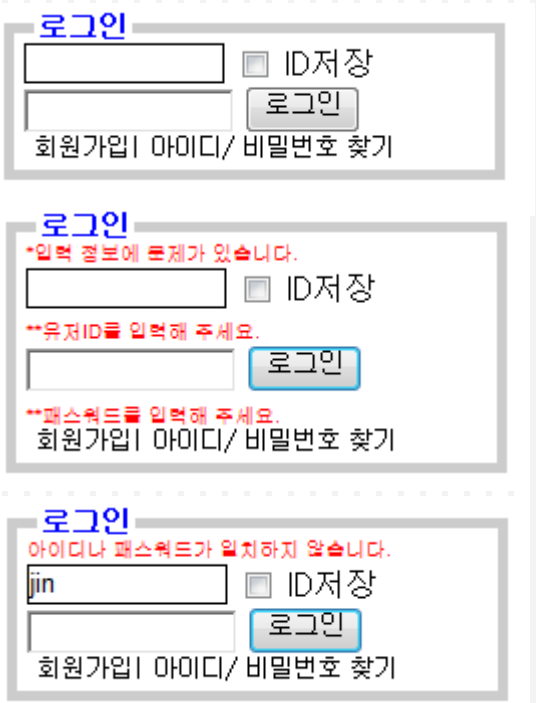
Business 구현 객체 완성

서비스 객체에서 데이터의 가공이나 변동 처리가 없음으로 이 객체를 생략하고 컨트롤러에서 직접 DAO 호출 가능

```java
package user.logic;

import user.dao.UserDao;

public class UserServiceImpl implements UserService {

    private UserDao userDao;

    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }

    @Override
    public User getUser(String userId, String password) {
        return this.userDao.findUser(userId, password);
    }

    public User getUser(String userId) {
        return this.userDao.findUser(userId);
    }

    public User getUserId(String userName, String userJumin) {
        return this.userDao.findId(userName,userJumin);
    }

    public User getUserPass(String userId, String userJumin) {
        return this.userDao.findPass(userId, userJumin);
    }

    public void updateUser(User user) {
        this.userDao.update(user);
    }

    public void insertUser(User user) {
        this.userDao.insert(user);
    }
}
```

View 와 Controller





login.jsp

LoginFormController

Model

LoginValidator

UserServise

UserDao

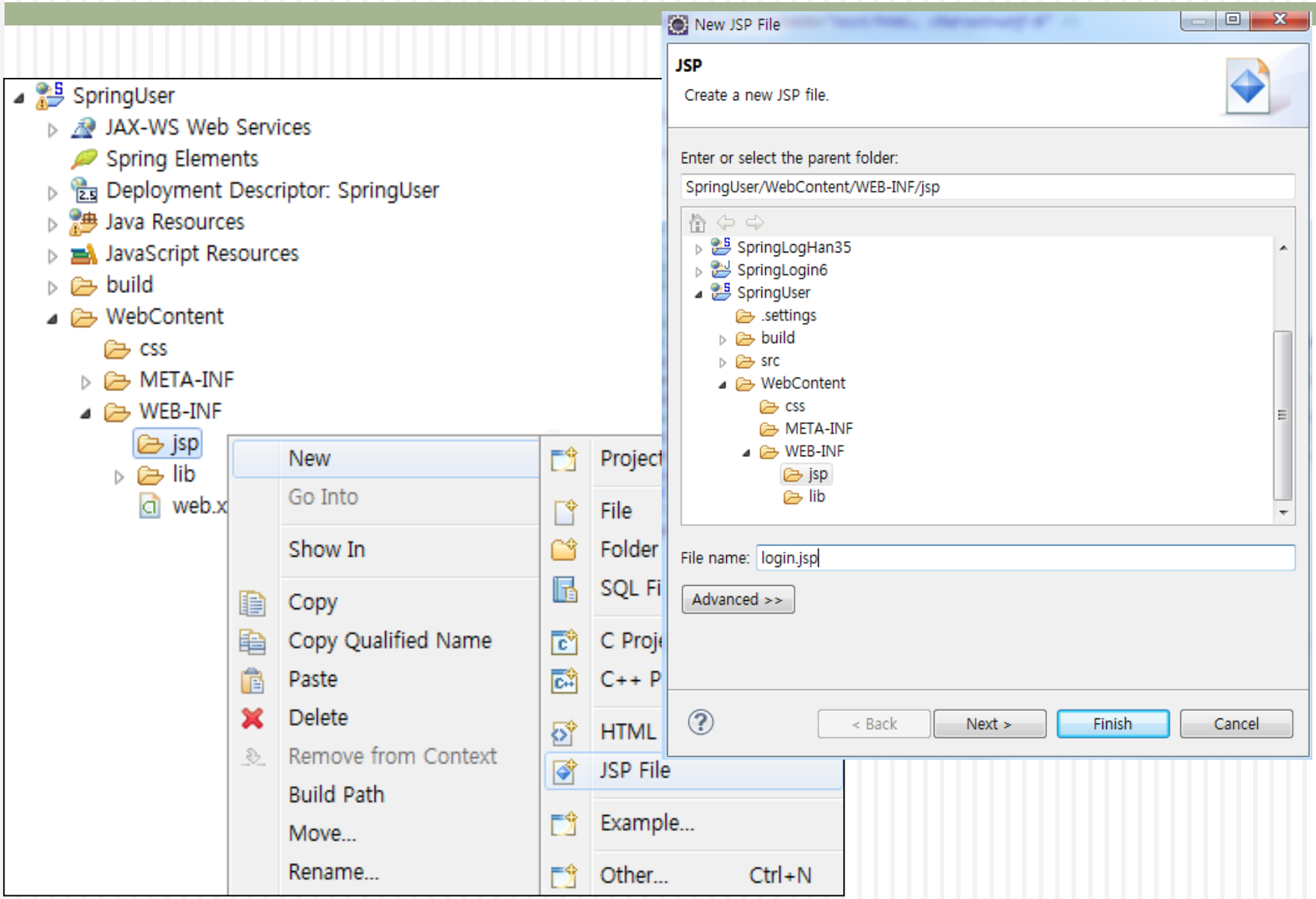
DataSource

로그인화면

환영합니다,이현상씨 !

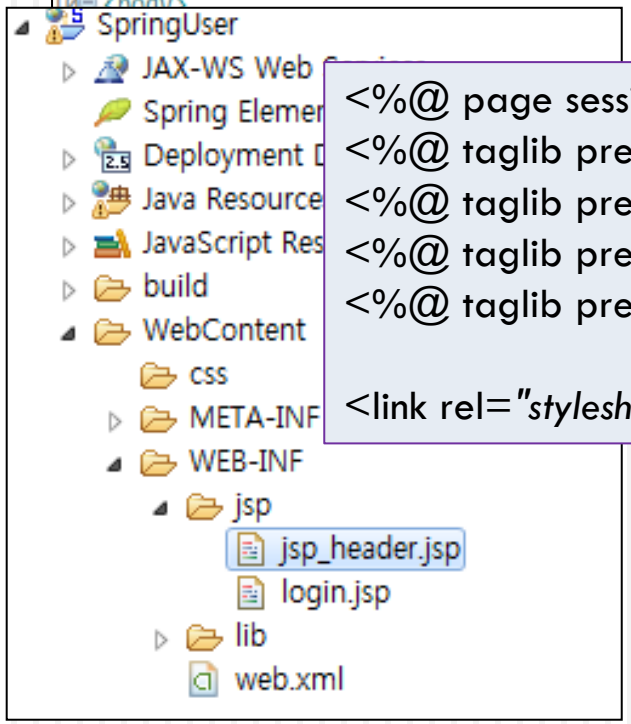
〈회원정보 수정〉

loginSuccess.jsp


SpringUser
JAX-WS Web Services
Spring Elements
Deployment Descriptor: SpringUser
Java Resources
JavaScript Resources
build
WebContent
css
META-INF
WEB-INF
jsp
lib
web.x
New
Go Into
Show In
Copy
Copy Qualified Name
Paste
Delete
Remove from Context
Build Path
Move...
Rename...
Project
File
Folder
SQL Fi
C Proje
C++ P
HTML
JSP File
Example...
Other... Ctrl+N
New JSP File
JSP
Create a new JSP file.
Enter or select the parent folder:
SpringUser/WebContent/WEB-INF/jsp
SpringLogHan35
SpringLogin6
SpringUser
.settings
build
src
WebContent
css
META-INF
WEB-INF
jsp
lib
File name: login.jsp
Advanced >>
< Back
Next >
Finish
Cancel

Include 파일과 CSS 파일

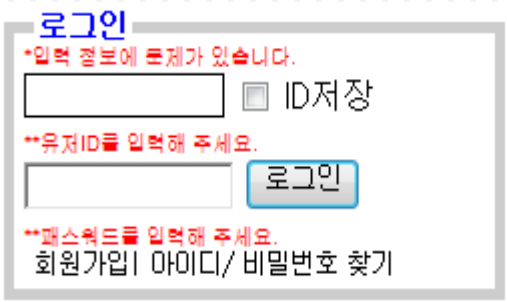
```jsp
<%@ page language="java" contentType="text/html; charset=utf-8"  pageEncoding="utf-8"%>
<%@ include file="/WEB-INF/jsp/jsp_header.jsp"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>회원인증</title>
</head>
<body>
```




```jsp
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="f" uri="http://java.sun.com/jsp/jstl/fmt" %>

<link rel="stylesheet" type="text/css" href="css/loginstyle.css">
```

Login .jsp

```
<body>
<br /><br />
<div id="loginArea" class="clear">
    <form:form modelAttribute="user" method="post" action="login.html">
      <fieldset>
        <legend><h4>로그인</h4></legend>
       <spring:hasBindErrors name="user">
         <span class="fieldError">
          <c:forEach var="error" items="${errors.globalErrors}" >
              <spring:message code="${error.code}"  />
          </c:forEach>
         </span>
      </spring:hasBindErrors>
```

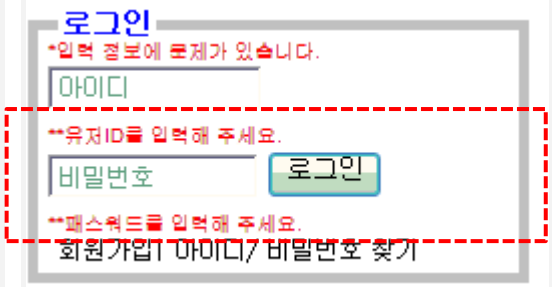



```
<p><ul id="ullog">
<li id="lilogb"><a href="/SpringLogHan/userEntry.html">회원가입</a></li>|
<li id="lilog"><a href="/SpringLogHan/findId_form.html">아이디/</a>
              <a href="/SpringLogHan/findPass_form.html" class="pwd">비밀번호 찾기</a></li>
          </ul></p>
    </fieldset>
 </form:form>
 </div>
</body>
```

Login .jsp

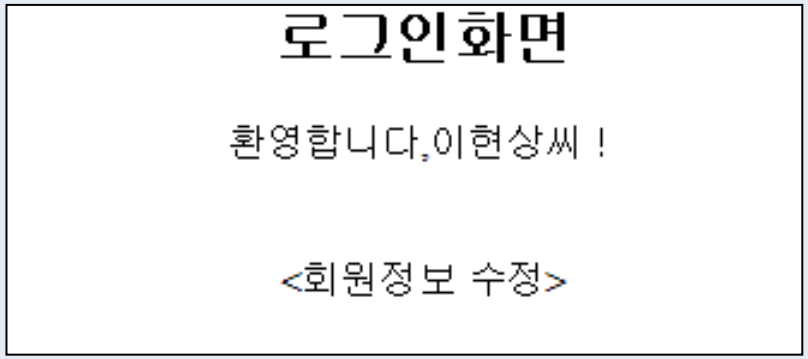
```
<p><form:input path="id"  size='12' maxlength='15'  title="아이디 입력"
    onfocus="this.className='id_focus'"
    onBlur="if ( this.value == '' ) { this.className='id_blur' }" class='id_blur' /></p>

<p><span class="fieldError"><form:errors path="id" /></span></p>

<p><form:password path="pass" size='13' maxlength='15' title="비밀번호 입력"
    onFocus="this.className='pw_focus'"
    onBlur="if ( this.value == '' ) { this.className='pw_blur' }" class='pw_blur' />
    <input type="submit" value="로그인" tabindex="4"  /></p>

<p><span class="fieldError"><form:errors path="pass" /></span></p>
```

바인딩 된 속성 아니면 error

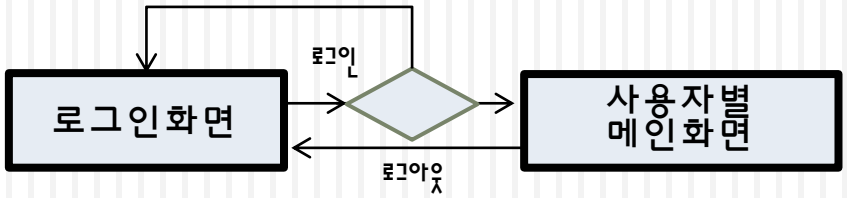


로그인 성공화면(loginSuccess.jsp)

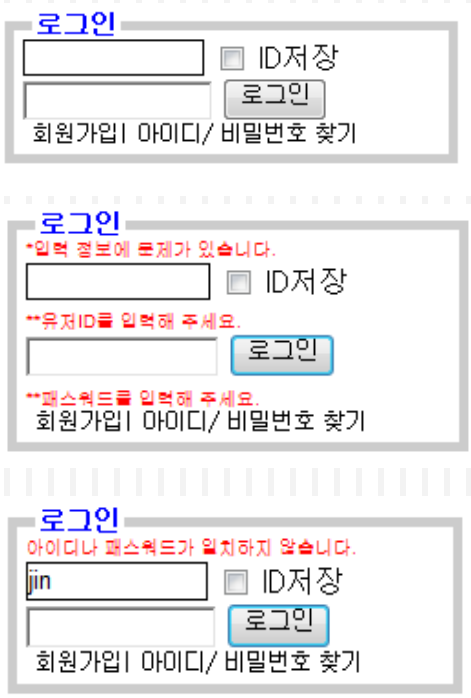
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ include file="/WEB-INF/jsp/jsp_header.jsp" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>로그인 성공화면</title>
</head>
<body>
<div align="center" class="body">
<h2>로그인화면</h2>
<br>
환영합니다,${loginUser.name}씨 !
<br><br><br>
<a href="#"><회원정보 수정></a>
</div>
</body>
</html>
```



View 와 Controller



http://SpringLogHan/login.html



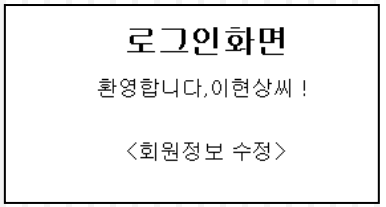
login.jsp

LoginFormController

LoginValidator

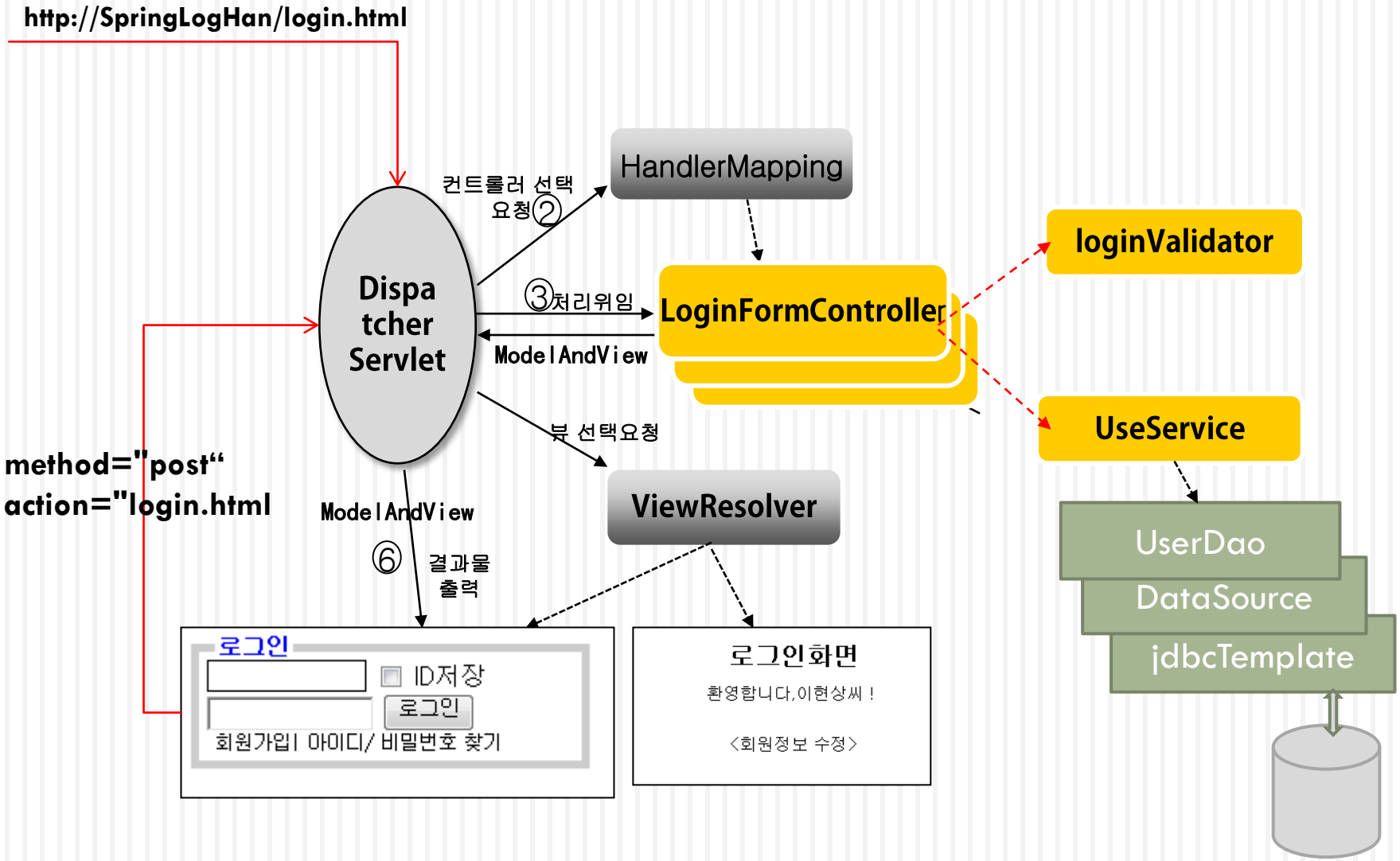
UserServise

Model



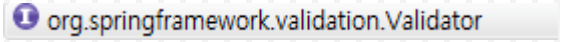
loginSuccess.jsp

Controller 설계

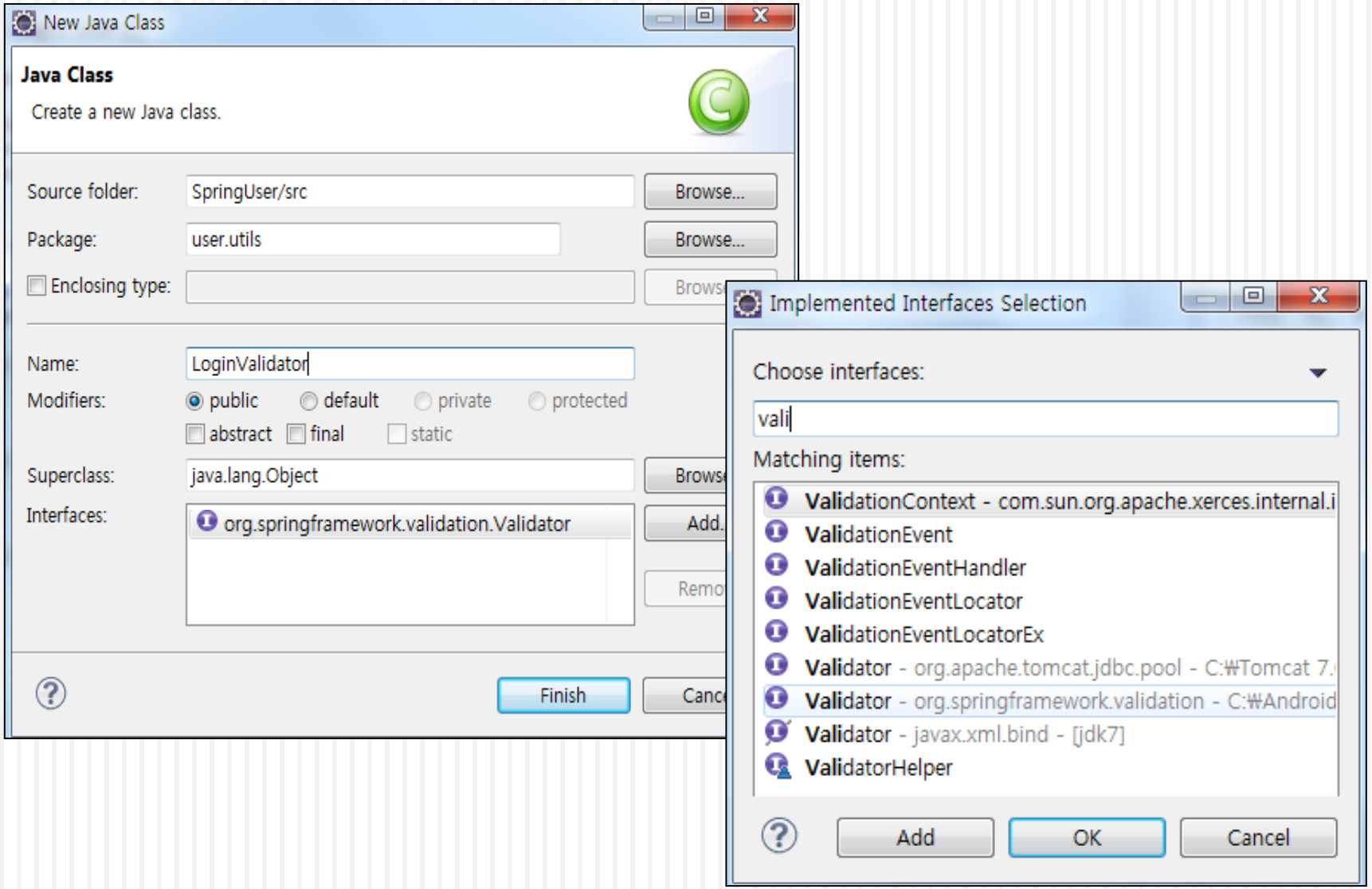


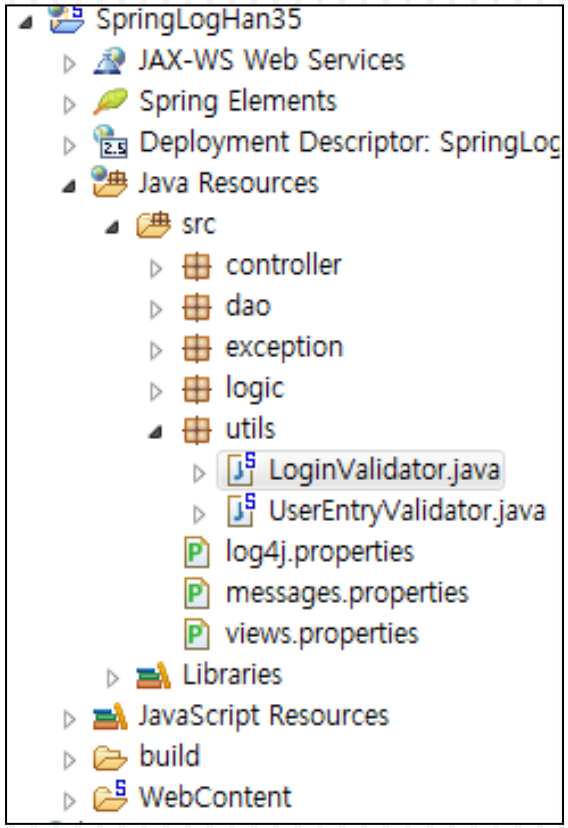
로그인 정보검증 클래스

- **Validator**
  - 로그인폼에서 입력한 항목을 검증한다.
  - Spring에서는 객체의 유효성을 체크하기 위해 Validator 인터페이스를 제공한다.
  - Validator인터페이스는 일관적이고 Errors객체를 사용하여 작동한다.
  - 유효성을 체크하는 동안, validator는 Errors객체에 대한 유효성체크 실패를 보고할 것이다.
  - 인터페이스를 제안하고 있다
  - **유효성체크(Validation생성**



유효성체크(Validation생성

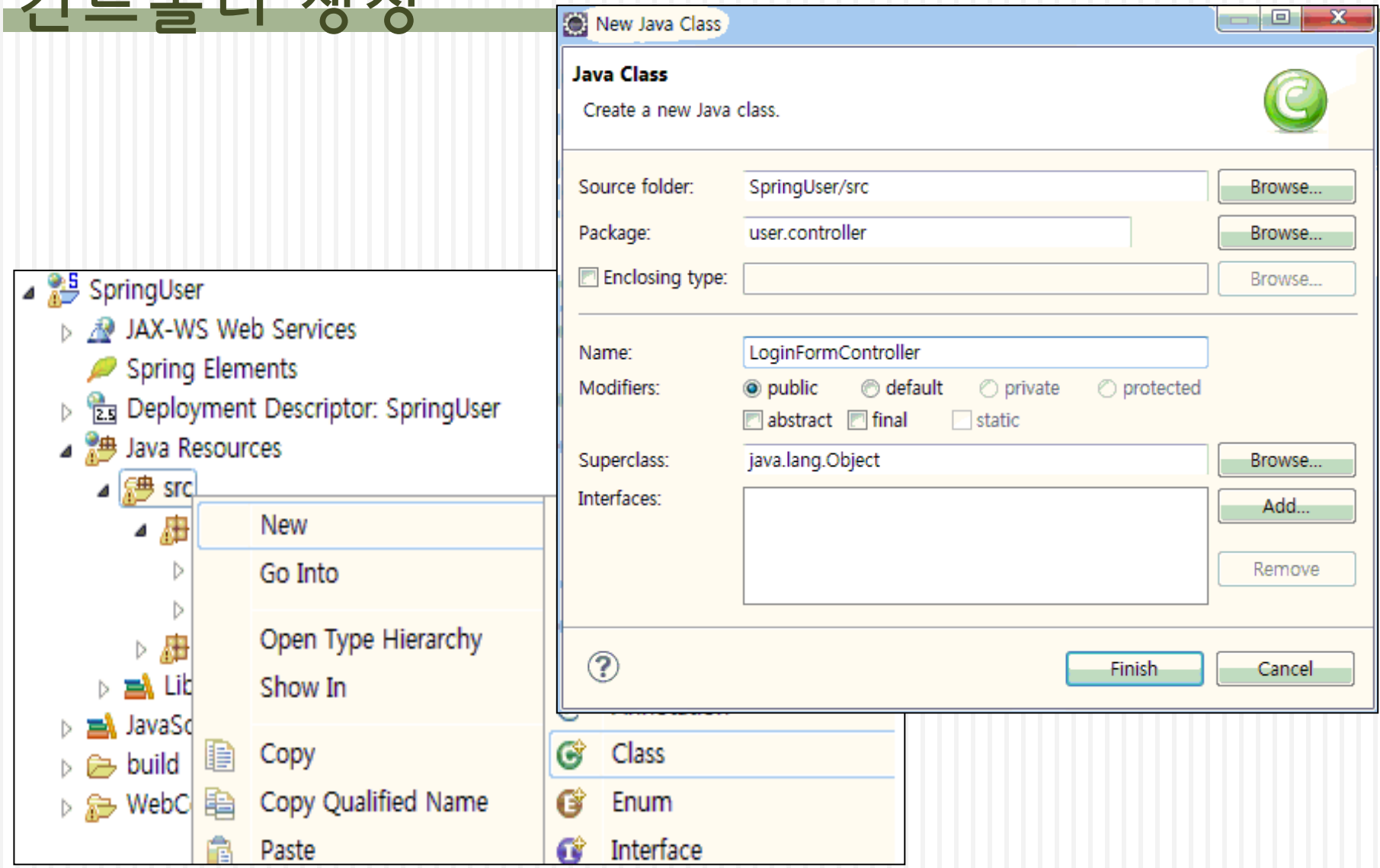




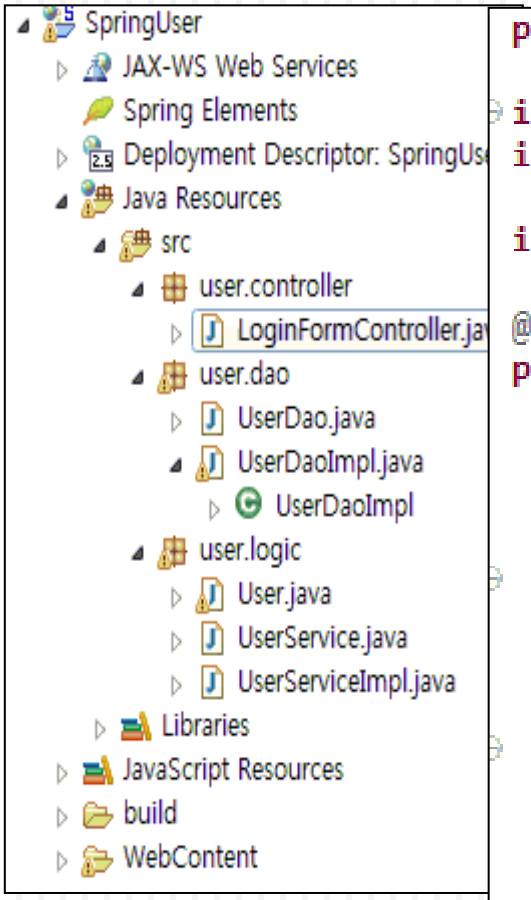

```java
package utils;

import logic.User;

import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

public class LoginValidator implements Validator {

    public boolean supports(Class<?> clazz) {
        return User.class.isAssignableFrom(clazz);
    }

    public void validate(Object command, Errors errors) {

        User user = (User) command;
        if (!StringUtils.hasLength(user.getUserId())) {
            errors.rejectValue("userId", "error.required");
        }

        if (!StringUtils.hasLength(user.getPassword())) {
            errors.rejectValue("password", "error.required");
        }

        if (errors.hasErrors()) {
            errors.reject("error.input.user");
        }
    }

}
```

messages.properties

컨트롤러 생성


New Java Class
Java Class
Create a new Java class.
Source folder: SpringUser/src
Browse...
Package: user.controller
Browse...
Enclosing type:
Browse...
Name: LoginFormController
Modifiers: public default private protected
abstract final static
Superclass: java.lang.Object
Browse...
Interfaces:
Add...
Remove
Finish
Cancel
SpringUser
JAX-WS Web Services
Spring Elements
Deployment Descriptor: SpringUser
Java Resources
src
New
Go Into
Open Type Hierarchy
Show In
Copy
Copy Qualified Name
Paste
Class
Enum
Interface

객체의 DI




```java
package user.controller;

import org.springframework.stereotype.Controller;
import org.springframework.validation.Validator;

import user.logic.UserService;

@Controller
public class LoginFormController {

    private UserService userService;
    private Validator loginValidator;

    public void setUserService(UserService userService) {
        this.userService = userService;
    }

    public void setLoginValidator(Validator loginValidator) {
        this.loginValidator = loginValidator;
    }

    ... 추가
}
```

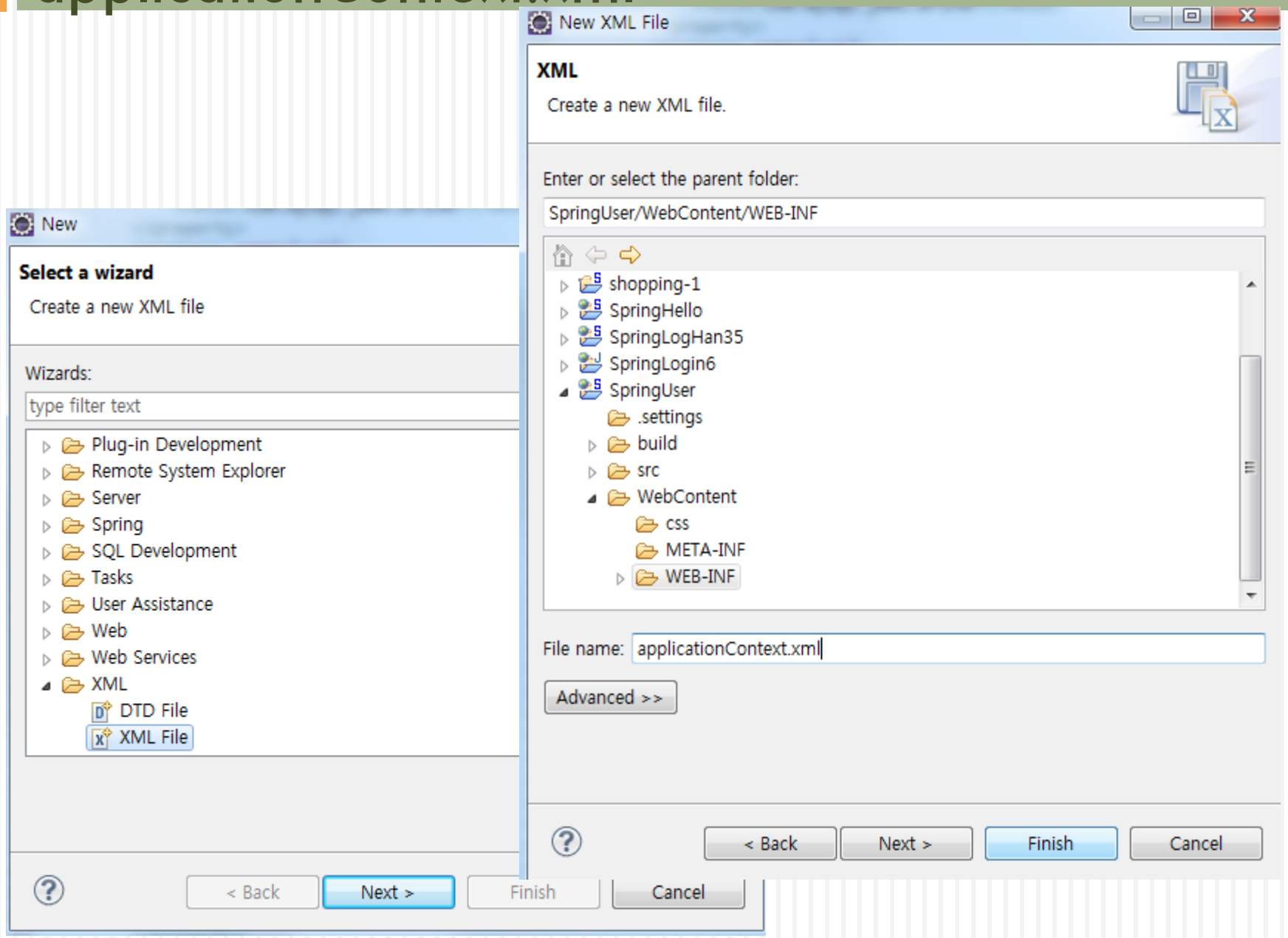
```
@RequestMapping(method = RequestMethod.GET)
public String toLoginView() {
       return "login";
}

@ModelAttribute
public User setUpForm() {
      return new User();
}
```

```
@RequestMapping(method = RequestMethod.POST)
public ModelAndView onSubmit(User user, BindingResult bindingResult) {

    this.loginValidator.validate(user, bindingResult);

    ModelAndView modelAndView = new ModelAndView();

    if (bindingResult.hasErrors()) {
            return modelAndView;
    }

    try {
            User loginUser = this.userService.getUser(user.getId(), user.getPass());
            modelAndView.setViewName("loginSuccess");
            modelAndView.addObject("loginUser", loginUser);

            return modelAndView;

    }catch (EmptyResultDataAccessException e) {
            bindingResult.reject("error.login.user");
            modelAndView.getModel().putAll(bindingResult.getModel());

            return modelAndView;
   }
}
```

ContextLoaderListener 설정화일

- Default BeanFactory - /WEB-INF/applicationContext.xml
- Root WebApplicationContext에서 필요한 객체 bean을 정의한다.
  - dataSource - 데이터베이스 연결 정보를 관리하는 객체
  - useDao, userService – model 구성을 위한 객체
  - loginValidator – 입력폼의 정보를 검증하기 위한 객체
  - messageSource – 오류메세지를 관리하기 위한 객체

applicationContext.xml



applicationContext.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:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

        … 추가

</beans>
```

dataSource Bean

```
<!-- Data Source -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName">
        <value>com.mysql.jdbc.Driver</value>
    </property>
    <property name="url">
        <value>jdbc:mysql://168.126.146.37:3306/jjindb?characterEncoding=utf8</value>
    </property>
    <property name="username">
        <value>jjin</value>
    </property>
    <property name="password">
        <value>jjinpang</value>
    </property>
    <property name="maxActive">
        <value>20</value>
    </property>
    <property name="maxIdle">
        <value>5</value>
    </property>
    <property name="maxWait">
        <value>10000</value>
    </property>
</bean>
```

userDao, userService, Validator,MessageSource Bean

```
<!-- UserDao -->
<bean id="userDao" class="user.dao.UserDaoImpl">
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
</bean>
```

의존관계정보 주입

```
public void setDataSource(DataSource dataSource) {
    this.template = new JdbcTemplate(dataSource);
}
```

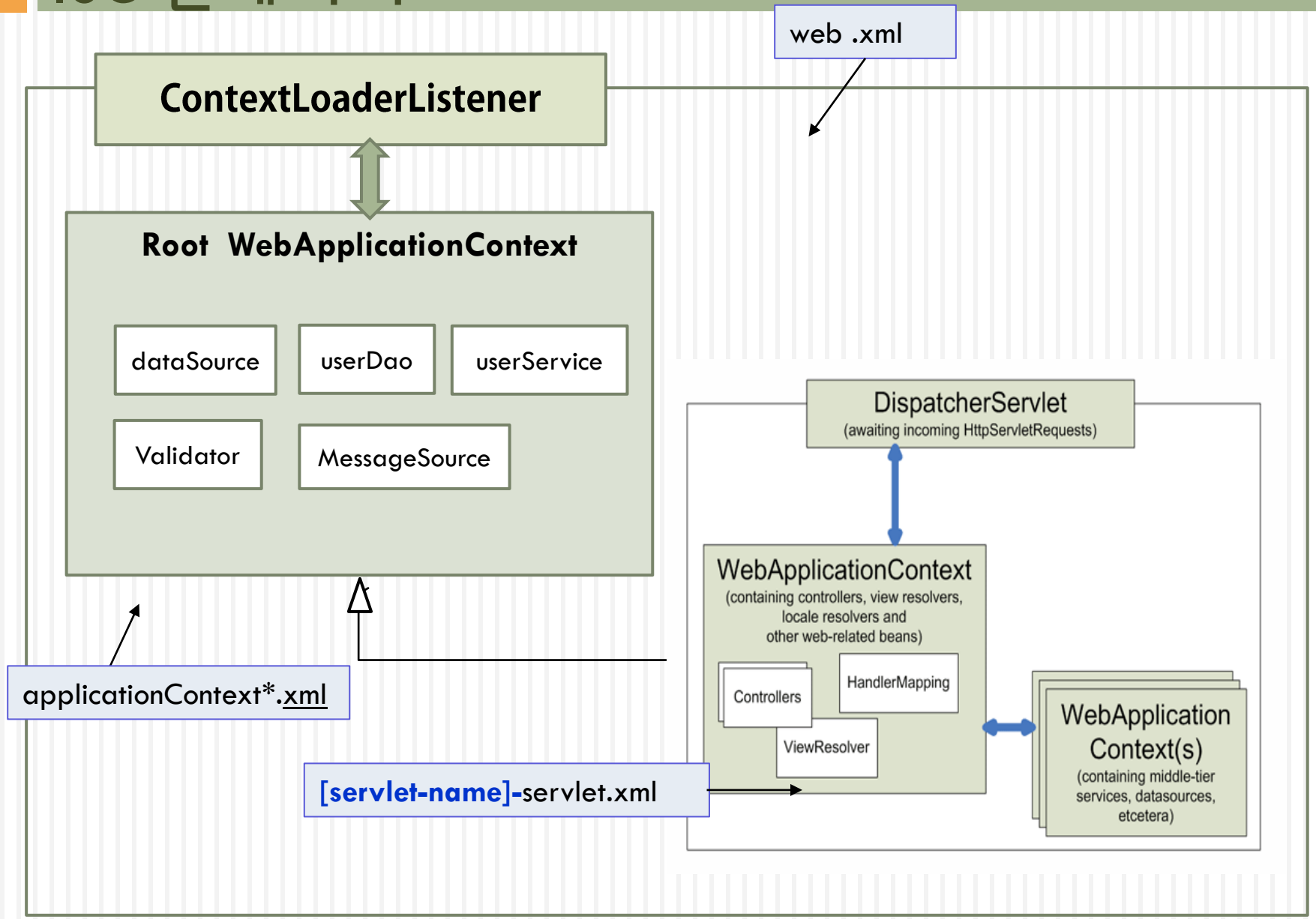
```
<!-- userService -->
<bean id="userService" class="user.logic.UserServiceImpl">
    <property name="userDao">
        <ref bean="userDao" />
    </property>
</bean>
```

의존관계정보 주입

```
public void setUserDao(UserDao userDao) {
    this.userDao = userDao;
}
```

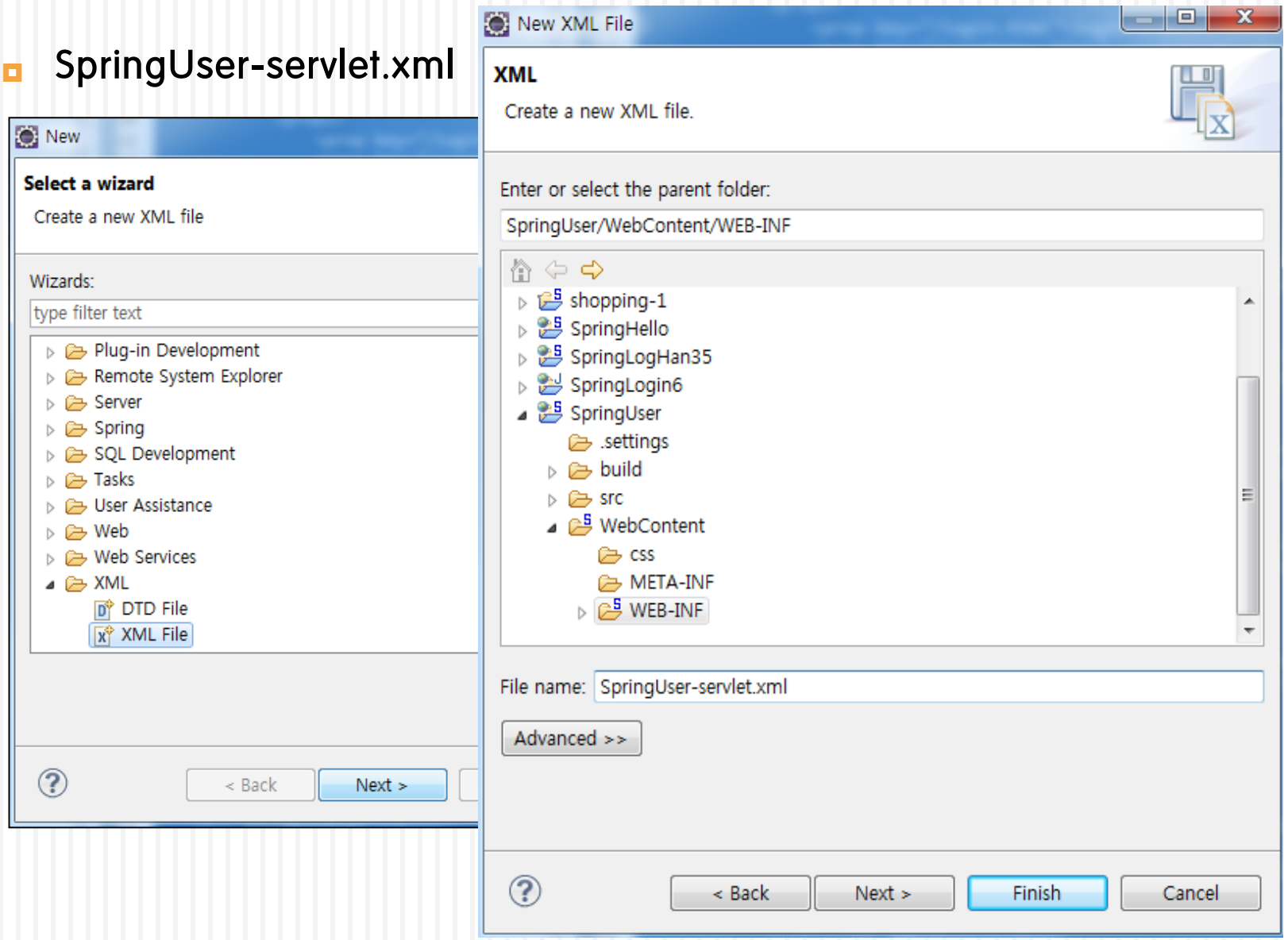
```
<!-- Validator -->
<bean id="loginValidator" class="user.utils.LoginValidator" />

<!-- MessageSource -->
<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
        <list><value>messages</value></list>
    </property>
</bean>
```

IoC 컨테이너

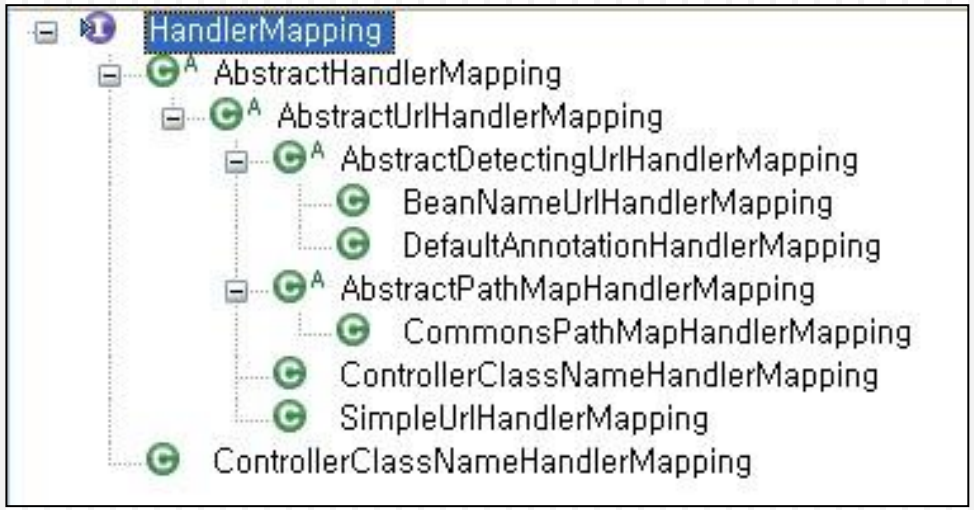

web .xml
ContextLoaderListener
Root WebApplicationContext
dataSource
userDao
userService
Validator
MessageSource
DispatcherServlet
(awaiting incoming HttpServletRequests)
WebApplicationContext
(containing controllers, view resolvers, locale resolvers and other web-related beans)
Controllers
HandlerMapping
ViewResolver
WebApplication Context(s)
(containing middle-tier services, datasources, etcetera)
applicationContext*.xml
[servlet-name]-servlet.xml

DispatcherServlet 설정화일

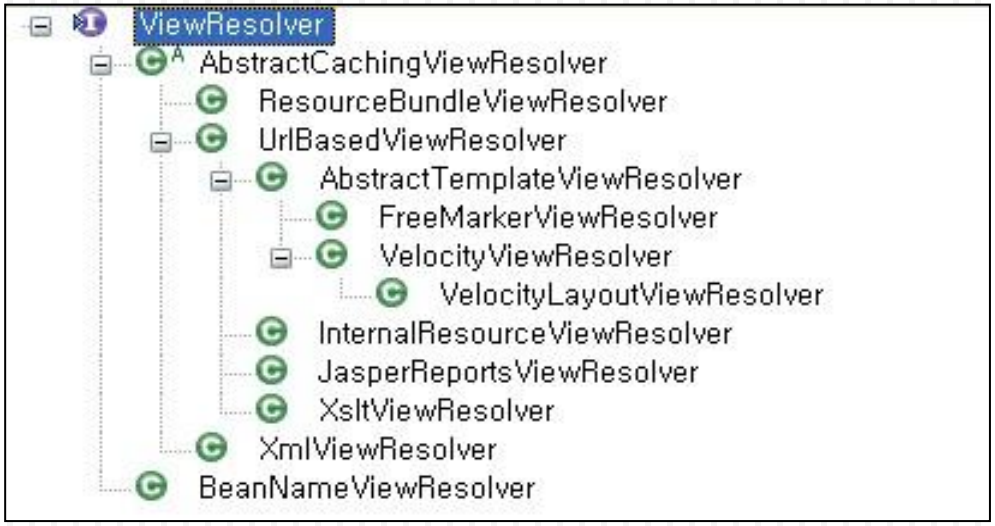
- SpringUser-servlet.xml



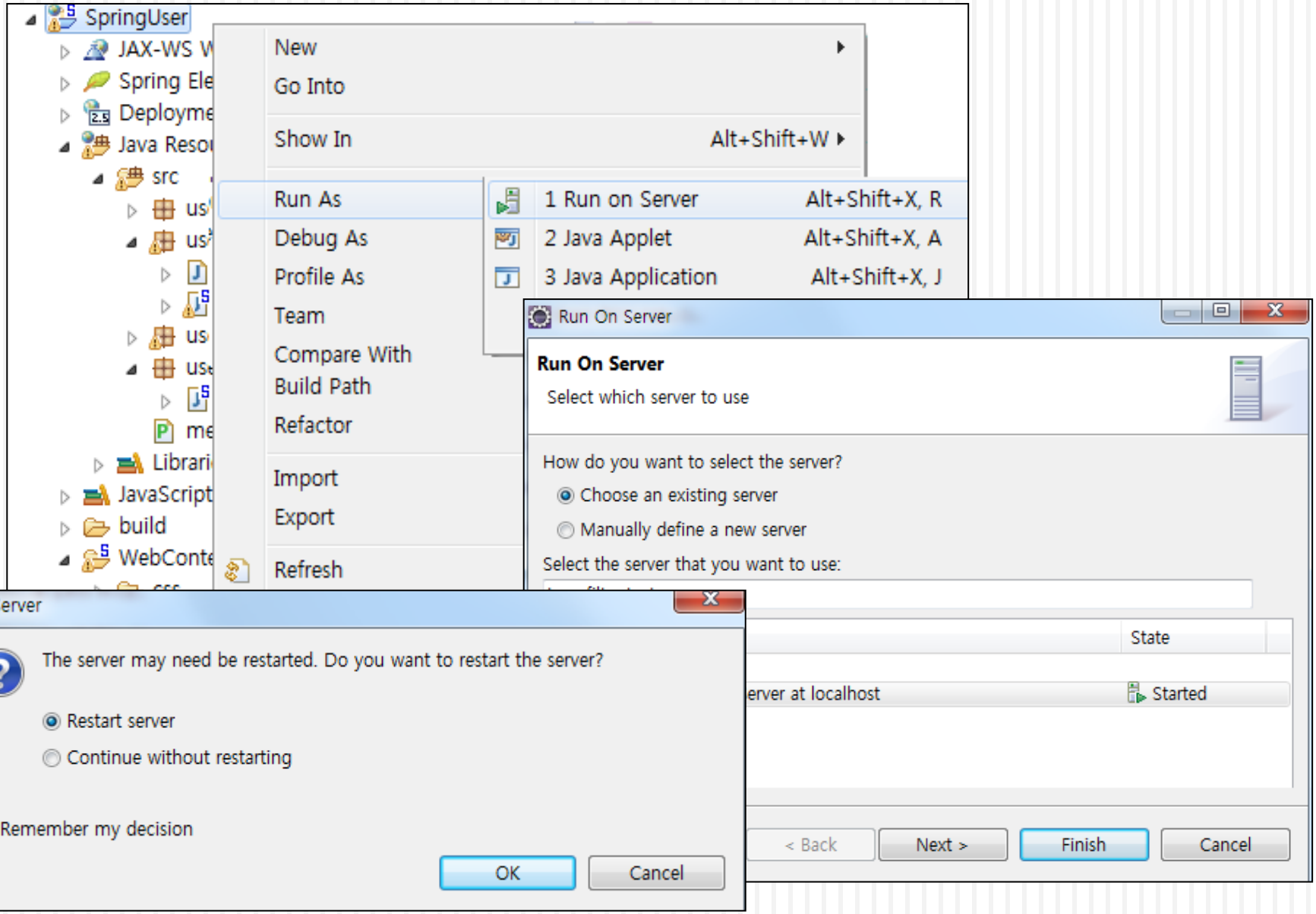
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- HandlerMapping -->
<bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
     <props>
        <prop key="/login.html">loginFormController</prop>
    </props>
    </property>
</bean>
    … 추가


</beans>
```

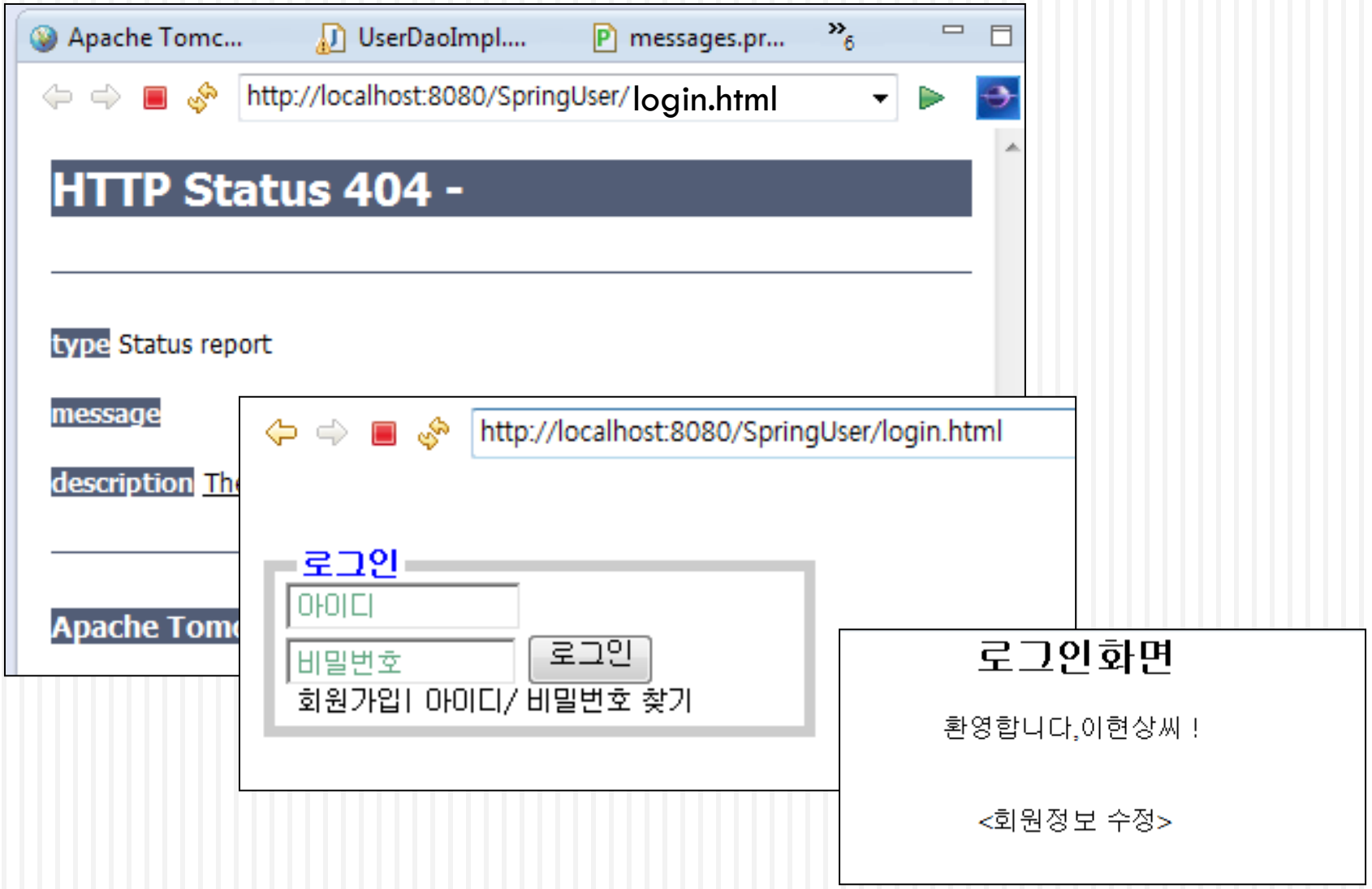


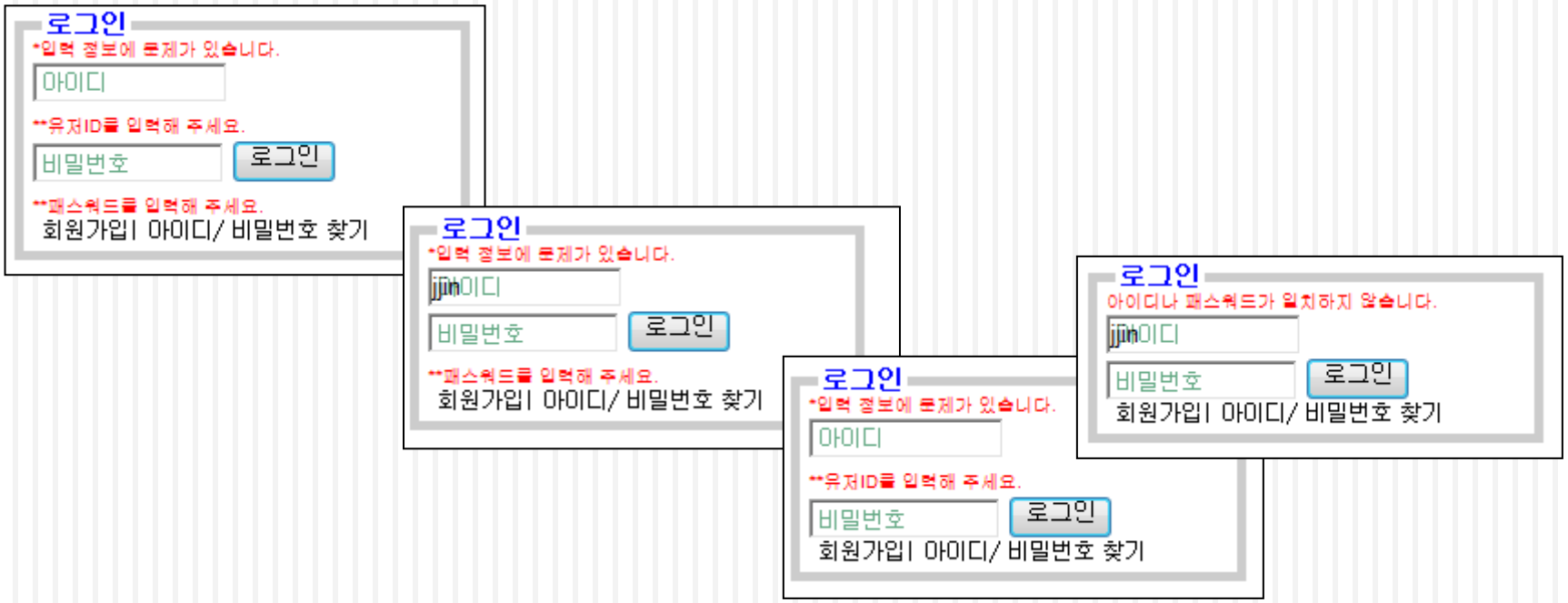
```
<!-- Controller -->
<bean id="loginFormController" class="user.controller.LoginFormController"
    p:userService-ref="userService" p:loginValidator-ref="loginValidator">
</bean>

<!-- ViewResolver -->
<bean id="internalResourceViewResolver"  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass">
        <value>org.springframework.web.servlet.view.JstlView</value>
    </property>
    <property name="prefix">
        <value>WEB-INF/jsp/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

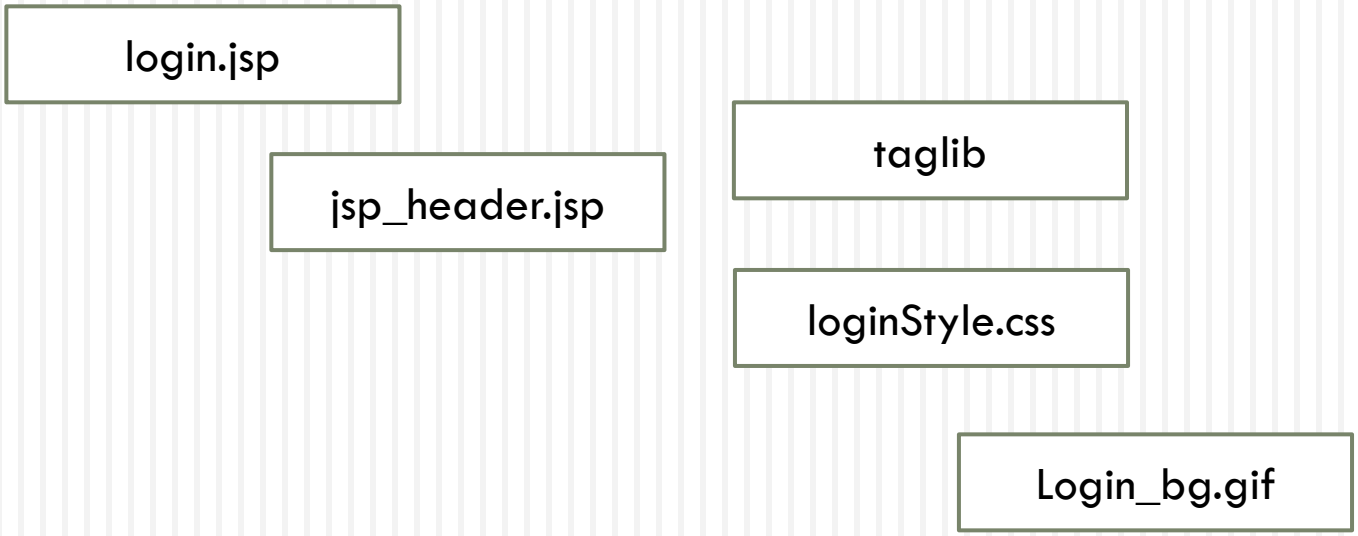


실 행


SpringUser
JAX-WS W
Spring Ele
Deployme
Java Reso
src
us
us
us
us
me
Librari
JavaScript
build
WebConte
New
Go Into
Show In
Alt+Shift+W
Run As
Debug As
Profile As
Team
Compare With
Build Path
Refactor
Import
Export
Refresh
1 Run on Server
Alt+Shift+X, R
2 Java Applet
Alt+Shift+X, A
3 Java Application
Alt+Shift+X, J
Run On Server
Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server
Manually define a new server
Select the server that you want to use:
State
erver at localhost
Started
< Back
Next >
Finish
Cancel
Server
The server may need be restarted. Do you want to restart the server?
Restart server
Continue without restarting
Remember my decision
OK
Cancel


Apache Tomc...
UserDaoImpl....
messages.pr...
http://localhost:8080/SpringUser/login.html
HTTP Status 404 -
type Status report
message
description
Apache Tomc
http://localhost:8080/SpringUser/login.html
로그인
아이디
비밀번호
로그인
회원가입| 아이디/ 비밀번호 찾기
로그인화면
환영합니다.이현상씨 !
<회원정보 수정>

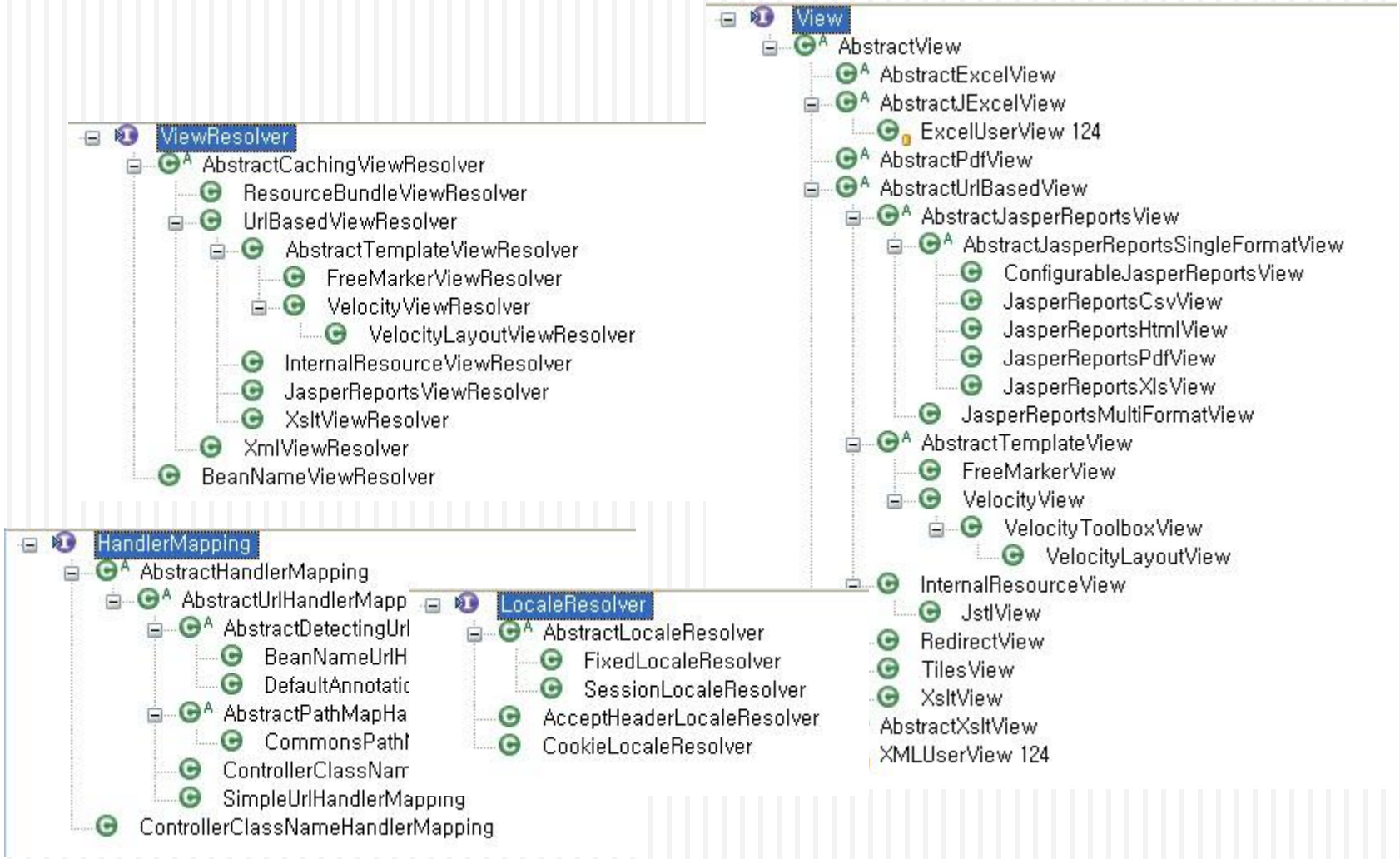

로그인
*입력 정보에 문제가 있습니다.
아이디
**유저ID를 입력해 주세요.
비밀번호
로그인
**패스워드를 입력해 주세요.
회원가입| 아이디/ 비밀번호 찾기
로그인
*입력 정보에 문제가 있습니다.
jjin아이디
비밀번호
로그인
**패스워드를 입력해 주세요.
회원가입| 아이디/ 비밀번호 찾기
로그인
*입력 정보에 문제가 있습니다.
아이디
**유저ID를 입력해 주세요.
비밀번호
로그인
회원가입| 아이디/ 비밀번호 찾기
로그인
아이디나 패스워드가 일치하지 않습니다.
jjin아이디
비밀번호
로그인
회원가입| 아이디/ 비밀번호 찾기


login.jsp
jsp_header.jsp
taglib
loginStyle.css
Login_bg.gif

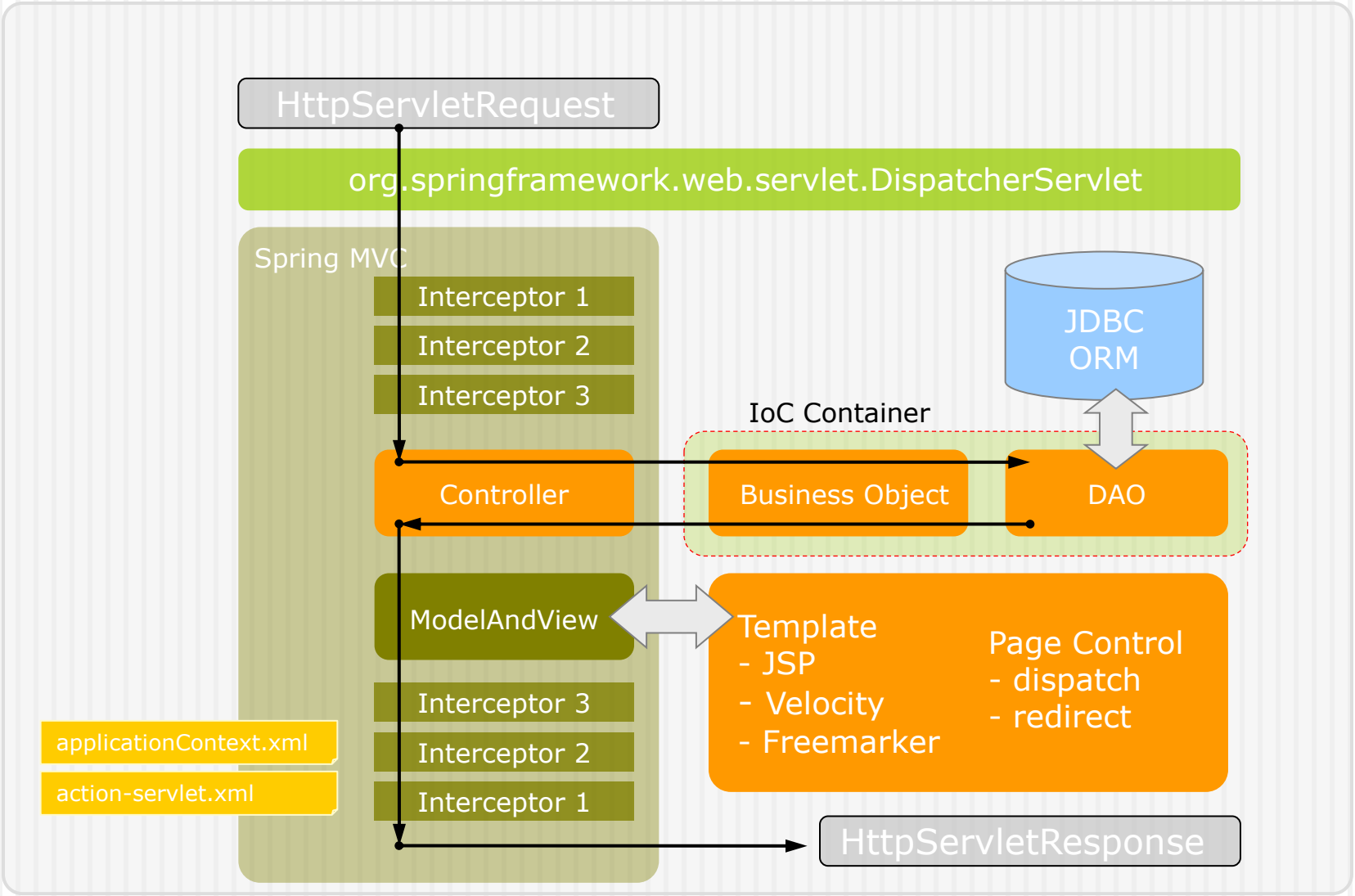
IoC 컨테이너 구성

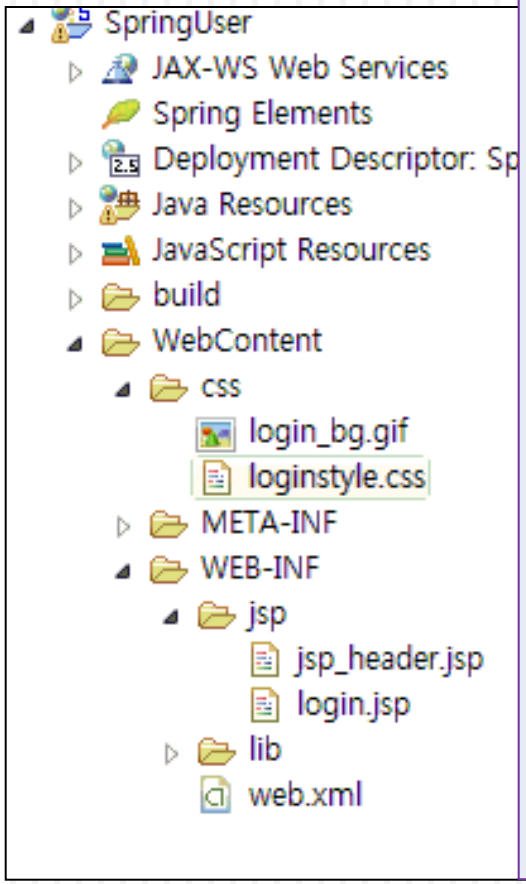
- ContextLoaderListener와 DispatcherServlet는 각각 별도의 WebApplicatonContext 인스턴스를 생성
- ContextLoaderListener가 root 가 되고 DispatcherServlet는 자식이 되어 빈을 상속
- ContextLoaderListener는 기본 applicationContext*.xml 설정 화일을 이용하여 빈을 생성, 공통빈으로 사용.
- DispatcherServlet는 기본 [servlet-name]-servlet.xml 파일로 부터 설정 정보를 읽어 자체 빈 생성

서블릿 애플리케이션 컨텍스트



b


HttpServletRequest
org.springframework.web.servlet.DispatcherServlet
Spring MVC
Interceptor 1
Interceptor 2
Interceptor 3
JDBC
ORM
IoC Container
Controller
Business Object
DAO
ModelAndView
Template
- JSP
- Velocity
- Freemarker
Page Control
- dispatch
- redirect
Interceptor 3
Interceptor 2
Interceptor 1
applicationContext.xml
action-servlet.xml
HttpServletResponse




```
a:link{font-family:"";color:black;text-decoration:none;}
a:visited{font-family:"";color:black;text-decoration:none;}
a:hover{font-family:"";color:black;text-decoration:underline;}
h4 {color:blue;font-family:굴림;font-size:14px;font-weight:bold;}
fieldset { border:5px solid #ccc; padding:0 5px 5px;}

#loginArea { margin-left:5px;  width:250px;}
#thisform { font-family:Georgia, serif; font-size:11px; color:#999;}
#thisform label { font-family:굴림, sans-serif; font-weight:bold;  color:#660000;}
#thisform p { margin:5px;}
#ullog { list-style-type:none; margin-left:5px; padding:0;font-family:굴림;font-size:12px;}
#lilog { display:inline; margin: 0px 0px 0px 0px; padding:0 0 0 0; border:1;font-
family:굴림; font-size:12px; }
#lilogb { display:inline; margin: 0px 0px 0px 0px; padding:0 0 0 0; border:1;font-
family:굴림; font-size:12px; }
* { padding: 0; margin: 0; }

.id_blur { background: transparent url("login_bg.gif") top left}
.id_focus { background: #ffffe0 ; color: #003300 }
.pw_blur { background: transparent url("login_bg.gif") bottom left}
.pw_focus { background: #ffffe0 ; color: #003300 }
.clear { clear: both; background: none; }
span.fieldError {
color: red;
font-size: 10px;
font-family:arial,sans-serif;
}
```

아이디

비밀번호