Spring MVC (Model-View-Controller)

설계 방식 중 하나

Spring MVC Framework

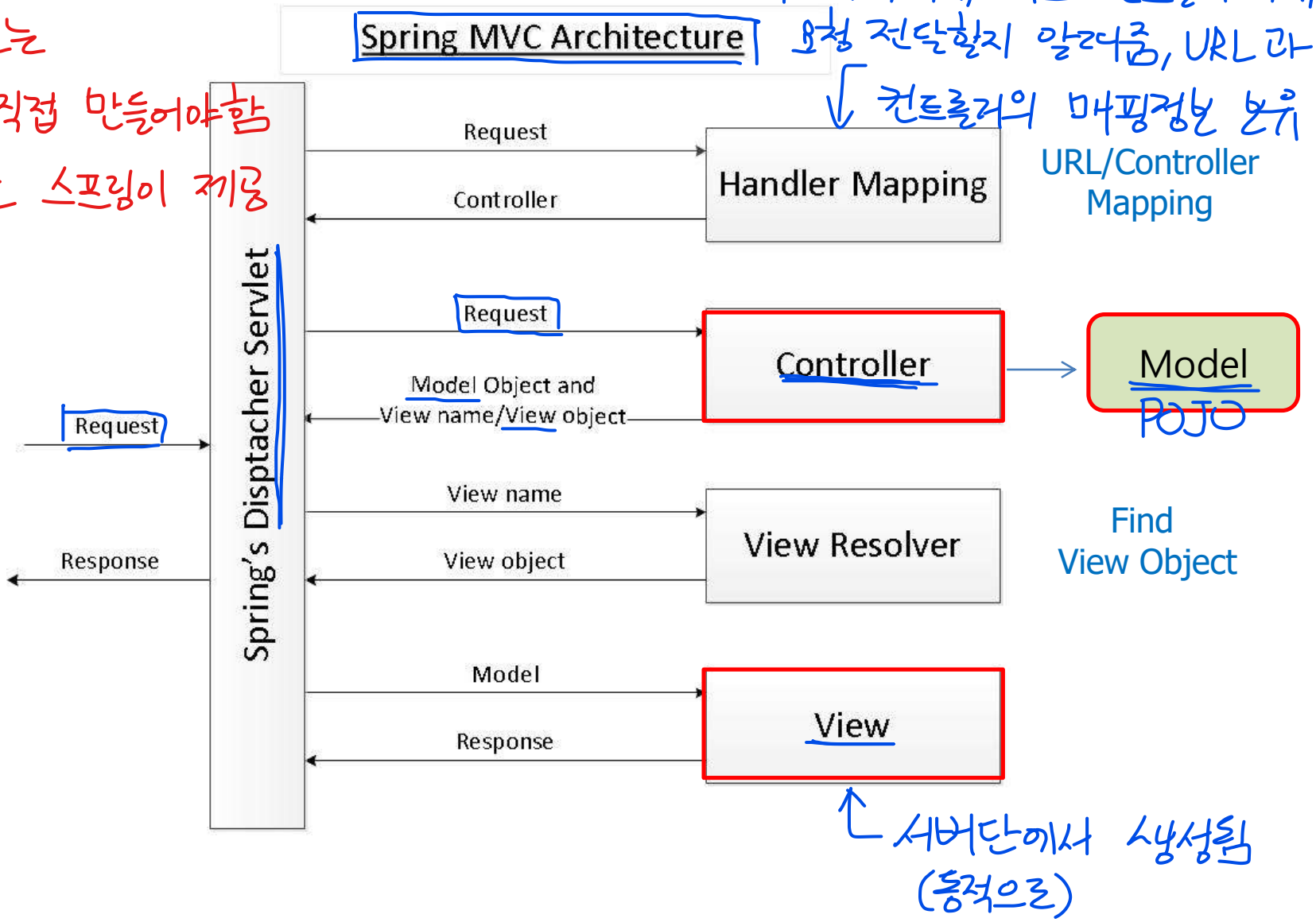
얘가 다 해줄 거니까 MVC 구조가 어떻게 동작하는지만 알면됨

- The Spring MVC framework provides model-view-controller architecture for developing web applications

  - Model 애플리케이션 데이터를 POJO에 저장
    - encapsulates the application data and in general they will consist of POJO
  - View 모델 데이터를 렌더링
    - responsible for rendering the model data and in general it generates HTML output
  - Controller 사용자의 요청 처리, 모델 만들고 View에 전달
    - responsible for processing user requests and building appropriate model and passes it to the view for rendering

1. Spring MVC Overview

빨간네모는
우리가 직접 만들어야함
나머지는 스프링이 제공



디스패처에게 어느 컨트롤러에게 요청 전달할지 알려줌, URL과 컨트롤러의 매핑정보 보유

URL/Controller Mapping

Find View Object

서버단에서 생성됨 (동적으로)

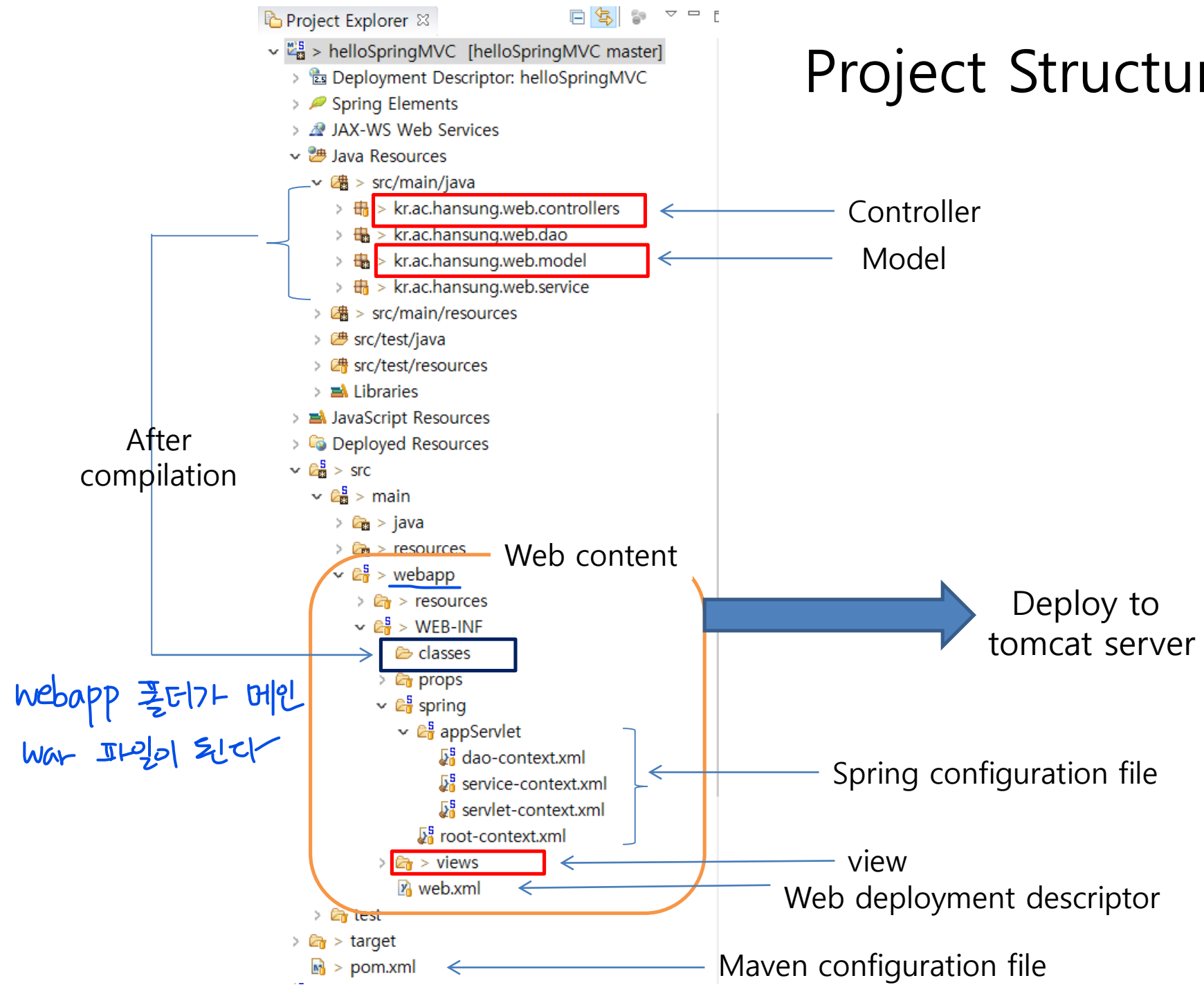
Overview

- Dispatcher servlet
  - acts as a front controller
  - intercepts all requests coming to the application and dispatches request to handlers (i.e., controllers)
  - consults the Handler Mapping for which controller to be invoked to handle the requests
- Handler Mapping
  - responsible to find appropriate controllers that handle specific requests
  - The mapping between request URLs and controller classes is done via XML configuration or annotations

Overview

- Controller
  - responsible to process the requests by calling other business/service classes
  - The output can be attached to model objects which will be sent to the view
- View Resolver
  - finds the physical view files from the logical names
- View
  - physical view files which can be JSP, HTML, XML, Velocity template, etc

Project Structure



3. Required Configuration

설정 파일들

- Maven Configuration
  - POM.xml
- Web deployment descriptor
  - Web.xml
- Spring MVC Configuration
  - root-context.xml
  - servlet-context.xml, dao-context.xml, service-context.xml

Maven을 설정하는 건 POM.xml
근데 maven 왜 쓰나? : 자동 빌드, 레포지토리에서 다운받기 해줌

1) Maven configuration (pom.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/X
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/mav
    <modelVersion>4.0.0</modelVersion>
    <groupId>kr.ac.hansung</groupId>
    <artifactId>helloSpringMVC</artifactId>
    <name>helloSpringMVC</name>
    <packaging>war</packaging>
    <version>1.0.0-BUILD-SNAPSHOT</version>
    <properties>
        <java-version>1.11</java-version>
        <org.springframework-version>5.1.5.RELEASE</org.springframework-version>
        <org.aspectj-version>1.6.10</org.aspectj-version>
        <org.slf4j-version>1.6.6</org.slf4j-version>
    </properties>
    <dependencies>
        <!-- Spring -->
        <dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${org.springframework-version}</version>
        </dependency>

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

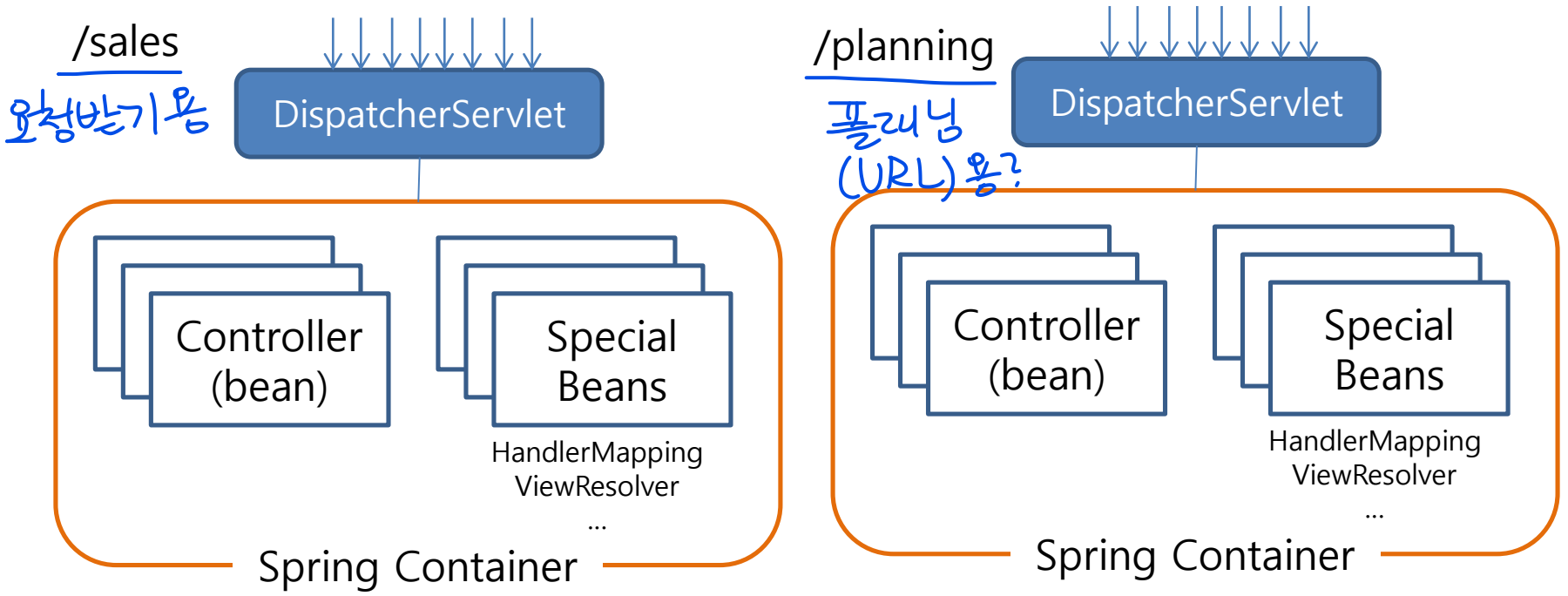
≒ 폰 모델명

Spring MVC dependency

maven의 레포지토리 다운로드

2) Web deployment descriptor (web.xml)

- DispatcherServlet 디스패처 2개 있다
  - instantiates a Spring Container(WebApplicationContext)
  - As with any DI container, WebApplicationContext has to be provided with some configuration metadata



①

```
<servlet>
    <servlet-name>salesServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/sales-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name> salesServlet </servlet-name>
    <url-pattern>/sales</url-pattern>
</servlet-mapping>
```

configuration metadata

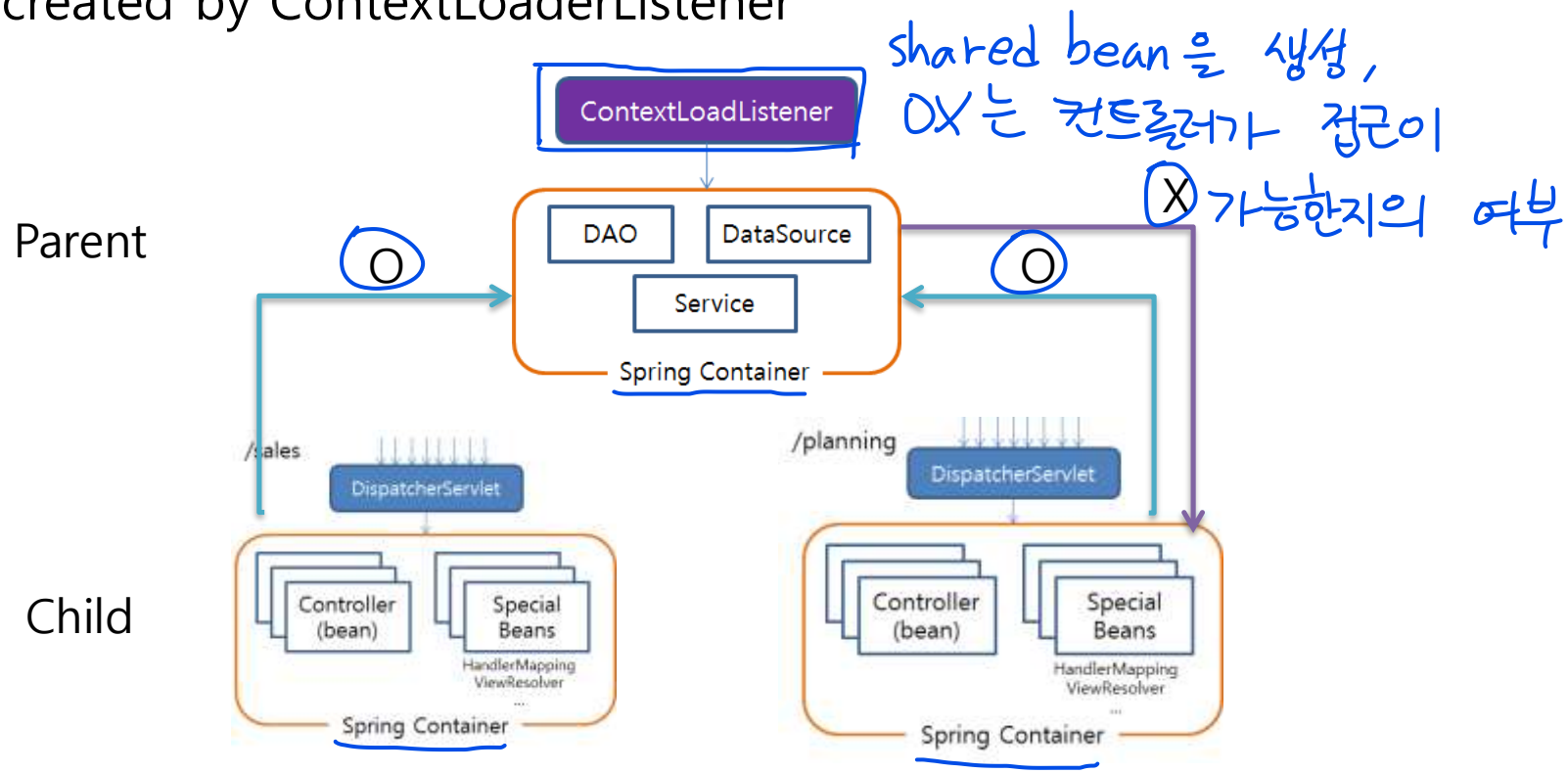
②

```
<servlet>
    <servlet-name>planningServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/planning-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name> planningServlet </servlet-name>
    <url-pattern>/planning</url-pattern>
</servlet-mapping>
```

Web deployment descriptor (web.xml)

- ContextLoadListener
  - instantiate a Spring Container which contains shared beans
  - Beans created by DispatcherServlet can reference beans of created by ContextLoaderListener



Web deployment descriptor (web.xml Example)

```xml
<!-- 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>
```

(xml은 직접 만들어야 함)

shared bean

```xml
<!-- 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) Spring MVC Configuration

STS created two Spring configuration files

① 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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframew

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

</beans>
```

This specifies configuration for root Spring container
which are shared by all servlets and filters
The root-context.xml file is loaded by the Spring's ContextLoaderListener
upon application's startup. This file is empty by default

Spring MVC Configuration

②

servlet-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mv
    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 http://www.springframework.org/schema/mvc/spring-
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.

    <!-- 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 in the ${webappRoot
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in 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="kr.ac.hansung" />

</beans:beans>
```

This file is loaded by the Spring's DispatcherServlet

동적 말고 정적인 자원을 처리가능 (이미지)

정적인 자원이 있는 경로

base-package로 경로를 명시하여 그곳을 스캔하고 bean을 찾고

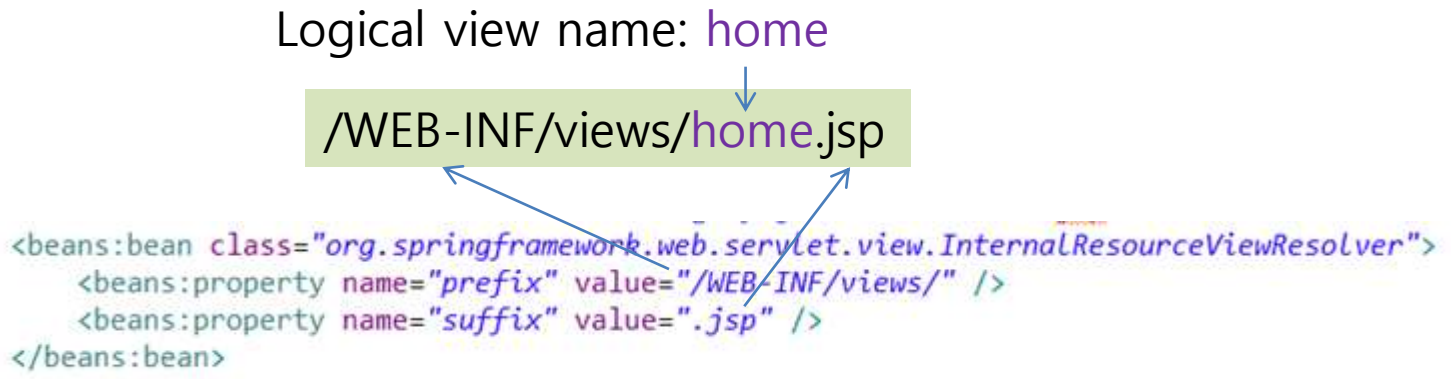
Spring MVC Configuration

- <annotation-driven />
  - tells the framework to use annotations-based approach to scan files in the specified packages

- <resources mapping=… />
  - maps static resources directly with HTTP GET requests. For example images, javascript, CSS,.. resources do not have to go through controllers

Spring MVC Configuration

- Bean InternalResourceViewResolver
  - Tells the framework how to find physical JSP files according to logical view names returned by the controllers



Spring MVC Configuration

- <context:component-scan .../>
  - tells the spring container which packages to be scanned when using annotation-based strategy
  - allows the container to auto-detect annotated controllers
  - here, the container will scan all classes under the package *kr.ac.hansung*

4. Model

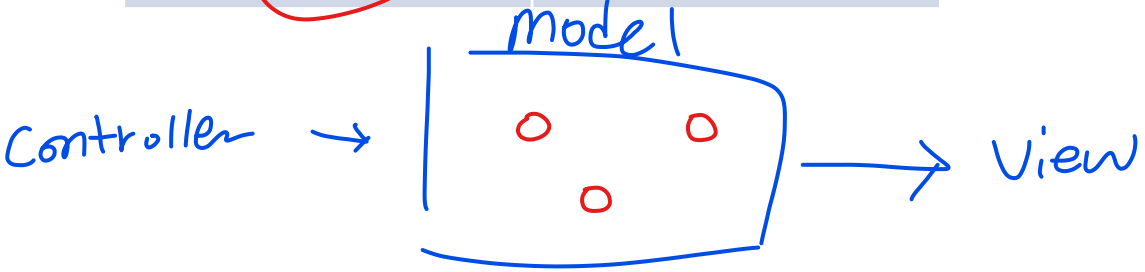
컨트롤러가 수행한 결과물을 저장함

- The model will contain the fraction of outcome (결과물)
- It is used to pass objects from the controller to the view
- The collection of named objects

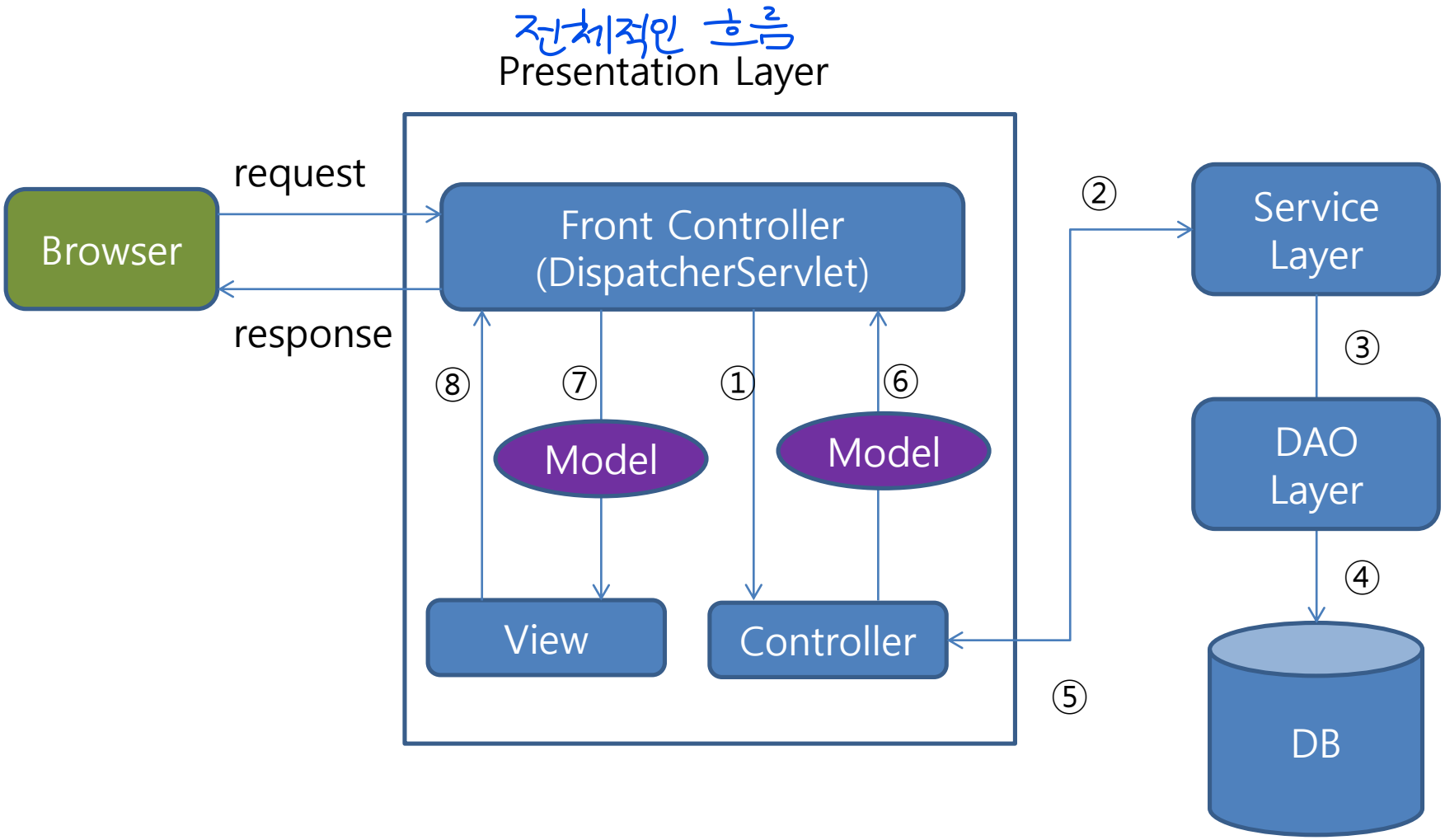
Model은 이름이 부여된 객체의 집합

Key(name)	Value
key1	value1
Key2	value2
key3	value3

Named object (called *model attributes*)



Model


전체적인 흐름
Presentation Layer
request
response
Browser
Front Controller
(DispatcherServlet)
Model
Model
View
Controller
①
②
③
④
⑤
⑥
⑦
⑧
Service
Layer
DAO
Layer
DB

Model 구현하는 방법

Model Implementations

- Several data structures can be used for representing a model

1) Any implementation of java.util.map by Java
2) Any implementation of Model interface provided by Spring
3) A ModelMap object provided by Spring

Model as an input argument

- It is common practice to let the framework provide the model directly as an *input argument*

```
public String getGreeting(Map<String, Object> model ) { … }

public String getGreeting (Model model ) { … }

public String getGreeting (ModelMap model ) { … }
```

1) Model as a Map

- The only requirement when using *java.util.Map* to store model attributes is that the key has to be of type *String*

key의 타입은 문자열이어야 함

```
@RequestMapping("/greeting")
                                    key
public String getGreeting(Map<String, Object> model ) {

        String greeting = service.getRandomGreeting();
        model.put("greeting", greeting)
                  key
        return "home"
}
                                    name    value
```

2) Model as a Model

- The downside of using a Map is that we have to decide the name of each model attribute

```
// deciding names... boring(and difficult !)

model.put("greeting", greeting);
```

- The Model interface provides convenient methods for populating the Model, such as addAttribute()
- addAttribute() is the same as Map's put(), except that it *automatically* generates the name for the model attribute 맵과 같지만 이름을 자동으로 생성해줌
- It is still possible to decide the name of each model attribute

Model as a Model

```
@RequestMapping("/special-deals")

public String getSpecialDeals(Model model) {

        List<SpecialDial> specialDeals = service.getSpecialDeals();
        model.addAttribute(specialDeals);

        return "home"
}
```

key 없이 value만 있음

3) Model as ModelMap

- The ModelMap provides some additional conveniences
  - Supports chained calls of the addAttribute( ) method

```
@RequestMapping("/fullname")

public String getFullname(ModelMap model ) {

        // chained calls are handy!
        model.addAttribute("name", "Jon")
             .addAttribute("surname", "Snow")

        return "home"
}
```

model 생략해주는 기능

@Component : 그냥 bean / 나머지는 추가정보 포함 → @Controller @Service @Repository

5. Controller

HomeController.java

```java
package kr.ac.hansung;

import java.text.DateFormat;

/**
 * Handles requests for the application home page.
 */
@Controller
public class HomeController {

	private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

	/**
	 * Simply selects the home view to render by returning its name.
	 */
	@RequestMapping(value = "/", method = RequestMethod.GET)
	public String home(Locale locale, Model model) {
		logger.info("Welcome home! The client locale is {}.", locale);

		Date date = new Date();
		DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale);

		String formattedDate = dateFormat.format(date);

		model.addAttribute("serverTime", formattedDate );

		return "home";
	}

}
```

2개

URL

2개

그리고는 model에 저장

Controller

- Spring controllers are *beans* annotated with @Controller

- @Controller annotation
  - Indicates that a class serves the role of controller
    클래스가 컨트롤러의 역할을 하도록 함
- @RequestMapping annotation
  - used to map a URL to either an entire class or a particular handler method URL을 클래스에 매핑 혹은 메서드에 매핑
  - In this case, it specifies that the home() method will handle a GET request with the URL / 26p 에서는 home()에 매핑

Controller

- Inside the home() method,
  - It creates a String object(formatted date) to hold the current date based on the current locale
  - It adds this object to the model with the name "serverTime":
    - model.addAttribute("serverTime", formattedDate); 모델 생성
  - The method returns a view named "home", which will be resolved by the view resolver specified in the servlet-context.xml file, to find the actual view file

View의 이름인 "home" 리턴

최종 파일 결정

Controller
(Mapping Requests)

- **Class** level mapping

```
@Controller
@RequestMapping("/appointments")
public class AppointmentsController {

    // an HTTP POST for /appointments
    @RequestMapping(method = RequestMethod.POST)
    public Map<String, Appointment> get() {
        return appointmentBook.getAppointmentsForToday();
    }

    // an HTTP GET for /appointments/new
    @RequestMapping(value="/new", method = RequestMethod.GET)
    public AppointmentForm getNewForm() {
        return new AppointmentForm();
    }
}
```

appointments 가 공통 루트

Controller
(Mapping Requests)

- **Handler** level mapping

```
@Controller
public class EmployeeController
{
    @RequestMapping("/employee-management/employees")
    public String getAllEmployees(Model model)
    {
        //application code
        return "employeesList";
    }

    @RequestMapping("/employee-management/employees/add")
    public String addEmployee(EmployeeVO employee)
    {
        //application code
        return "employeesDetail";
    }
}
```

반복, 즉 29p가 더 좋다

Controller
(Example)

① Add attributes to the model

```
@RequestMapping(value = "/", method = RequestMethod.GET)
public String home(Locale locale, Model model) {
   Date date = new Date();
   DateFormat dateFormat =
        DateFormat.getDateTimeInstance(DateFormat.LONG,
              DateFormat.LONG, locale);

   String formattedDate = dateFormat.format(date);

   model.addAttribute("serverTime", formattedDate );

   return "home";
}
```

key

String attributeName

Object attributeValue

Logical view name

Controller
(Example)

② Retrieve request parameters as regular parameters

```
@RequestMapping(value = "/login", method = RequestMethod.GET)

public String doLogin(@RequestParam String username,
                      @RequestParam String password) {
        ...
        return success;
}
```

http://localhost:8080/spring/login?username=scott&password=tiger

@RequestParam이 있으면

스프링 한테 매칭시켜 달라고 함 → request 파라미터를 알아서 넣어줌

6. View
(JSP)

home.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
    <title>Home</title>
</head>
<body>
<h1>
    Hello world!
</h1>

<P>  The time on the server is ${serverTime}. </P>
</body>
</html>
```

jsp key값 넣으면 value 표시됨

It just prints out value of the variable "serverTime"
which is passed by the controller, using an EL expression:

```
The time on the server is ${serverTime}.
```

JSTL
(JSP Standard Tag Library)

- It extends the JSP specification by adding a tag library of JSP tags JSP 확장 (태그)
- Include JSTL as part of your maven dependency

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
    <scope>provided</scope>
</dependency>
```

JSTL

- prefix “c” can be used for core language
- prefix “fn” for using JSTL functions

c 와 fn 은 이름인데, 마음대로 설정해도됨

```
<?xml version="1.0" encoding="UTF-8" ?>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>

<c:forEach var="i" begin="0" end="5">
        Item <c:out value="${i}"/><p>
</c:forEach>

<c:if test="${fn:length(friends) > 0}" >
        <%@include file="welcome.jsp" %>
</c:if>
```