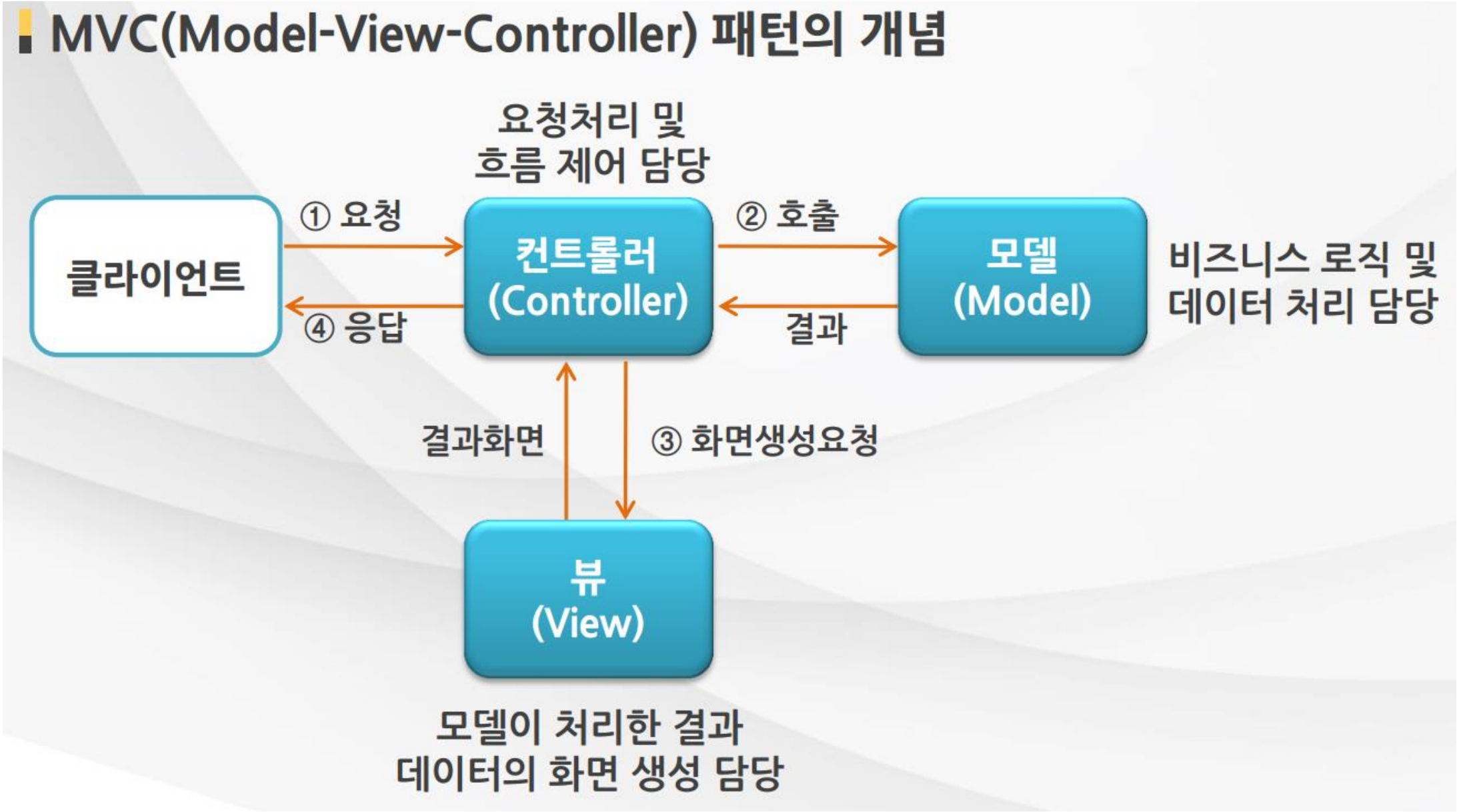
Spring MVC

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Spring5

MVC

- ***Model-View-Controller Pattern***의 개념
  - Software 공학에서 사용되는 Architecture Pattern
  - 주 목적은 Business Logic과 Presentation Logic을 분리하기 위함
  - 이 Pattern을 통해, User Interface로부터 Business Logic을 분리하여 Application의 시각적 요소나 그 이면에서 실행되는 Business Logic을 서로 영향없이 쉽게 고칠 수 있는 Application을 만들 수 있음.
- Model : Application의 정보(Data, Business Logic 포함)
- View : User에게 제공할 화면(Presentation Logic)
- Controller : Model과 View사이의 상호 작용을 관리

MVC (Cont.)


MVC(Model-View-Controller) 패턴의 개념
요청처리 및
흐름 제어 담당
클라이언트
① 요청
④ 응답
컨트롤러
(Controller)
② 호출
결과
모델
(Model)
비즈니스 로직 및
데이터 처리 담당
결과화면
③ 화면생성요청
뷰
(View)
모델이 처리한 결과
데이터의 화면 생성 담당

각각의 Component의 역할

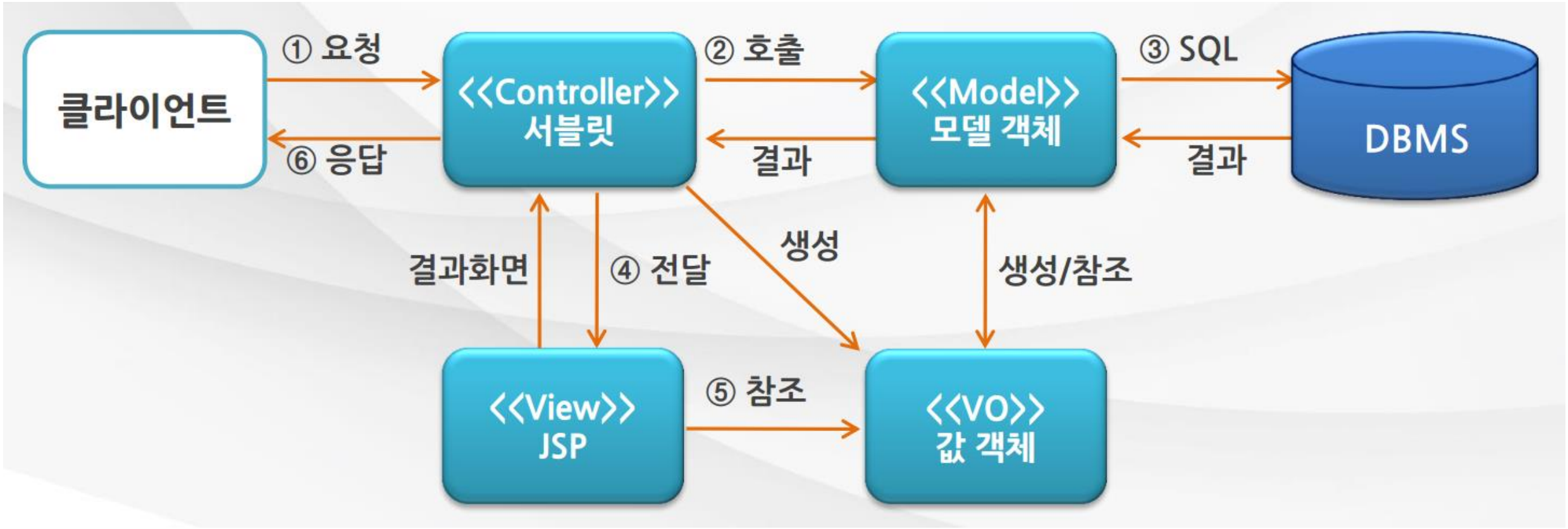
- Model Component
  - Data 저장소(ex : DB)와 연동하여 사용자가 입력한 Data나 사용자에게 출력할 Data를 다루는 역할
  - 여러 개의 Data 변경 작업(추가, 변경, 삭제)을 하나의 작업으로 묶는 Transaction을 다루는 역할
  - DAO class, Service class에 해당
- View Component
  - Model이 처리한 Data나 그 작업 결과를 가지고 User에게 출력할 화면을 만드는 역할
  - 생성된 화면은 Web Browser가 출력하고, View Component는 HTML과 CSS, JavaScript를 사용하여 Web Browser가 출력할 UI 생성
  - HTML과 JSP를 사용하여 작성

각각의 Component의 역할 (Cont.)

- Controller Component
  - Client의 요청을 받았을 때 그 요청에 대해 실제 업무를 수행하는 Model Component를 호출하는 역할
  - Client가 보낸 Data가 있다면, Model을 호출할 때 전달하기 쉽게 Data를 적절히 가공하는 역할
  - Model이 업무 수행을 완료하면, 그 결과를 가지고 화면을 생성하도록 View에게 전달(Client 요청에 대해 Model과 View를 결정하여 전달)
  - Servlet과 JSP를 사용하여 작성

Model2 Architecture

- Model1 : Controller의 역할을 ***JSP***가 담당
- Model2 : Controller의 역할을 ***Servlet***이 담당



Model2 Architecture 호출 순서

①Web Browser가 Web Application 실행을 요청하면, Web Server가 그 요청을 받아서 Servlet Container(ex:Tomcat Server)에게 넘겨준다.

- Servlet Container는 URL을 확인하여 그 요청을 처리할 Servlet을 찾아서 실행한다.

②Servlet은 실제 업무를 처리하는 Model Java 객체의 Method를 호출한다.

- 만약 Web Browser가 보낸 Data를 저장하거나 변경해야 한다면, 그 Data를 가공하여 VO객체를 생성하고, Model 객체의 Method를 호출할 때 인자 값으로 넘긴다.
- Model 객체는 일반적으로 POJO로 된 Service, DAO 일 수 있다.

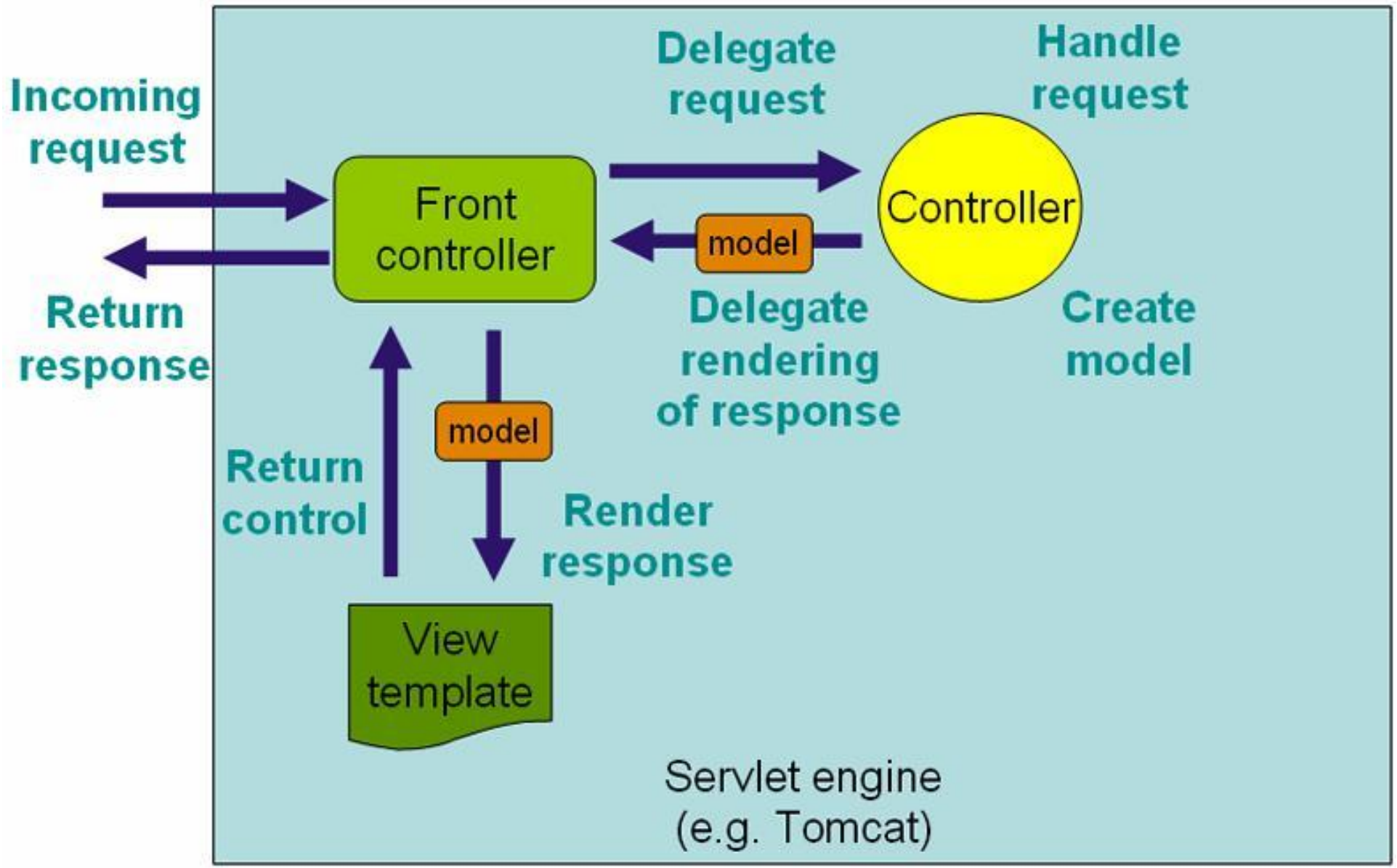
③Model객체는 JDBC를 사용하여 매개변수로 넘어온 값 객체를 Database에 저장하거나, Database로부터 질의 결과를 가져와서 VO 객체로 만들어 반환한다.

④Servlet은 Model 객체로부터 반환 받은 값을 JSP에 전달한다.

⑤JSP는 Servlet으로부터 전달받은 값 객체를 참조하여 Web Browser가 출력할 결과 화면을 만들고, Web Browser에 출력함으로써 요청 처리를 완료한다.

⑥Web Browser는 Server로부터 받은 응답 내용을 화면에 출력한다.

Front Controller Pattern Architecture



Front Controller Pattern Architecture (Cont.)

- Front Controller는 Client가 보낸 요청을 받아서 공통적인 작업을 먼저 수행
- Front Controller는 적절한 세부 Controller에게 작업을 위임
- 각각의 Application Controller는 Client에게 보낼 View를 선택해서 최종 결과를 생성하는 작업
- ***Front Controller Pattern***은 인증이나 권한 Check처럼 모든 요청에 대하여 공통적으로 처리해야 하는 Logic이 있을 경우 전체적으로 Client의 요청을 중앙 집중적으로 관리하고자 할 경우에 사용

Spring MVC 개념

- 특징
  - Spring은 DI나 AOP 같은 기능 뿐만 아니라 Servlet 기반의 Web 개발을 위한 MVC Framework를 제공
  - Spring MVC나 ***Model2 Architecture***와 ***Front Controller pattern***을 Framework 차원에서 제공
  - Spring MVC Framework는 Spring을 기반으로 하고 있기 때문에 Spring이 제공하는 Transaction 처리나 DI 및 AOP등을 손쉽게 사용
- Spring MVC와 Front Controller Pattern
  - 대부분의 MVC Framework들은 ***Front Controller pattern***을 적용해서 구현
  - Spring MVC도 Front Controller 역할을 하는 `DispatcherServlet`이라는 Class를 계층의 맨 앞단에 놓고, Server로 들어오는 모든 요청을 받아서 처리하도록 구성
- 예외가 발생했을 때 일관된 방식으로 처리하는 것도 Front Controller의 역할

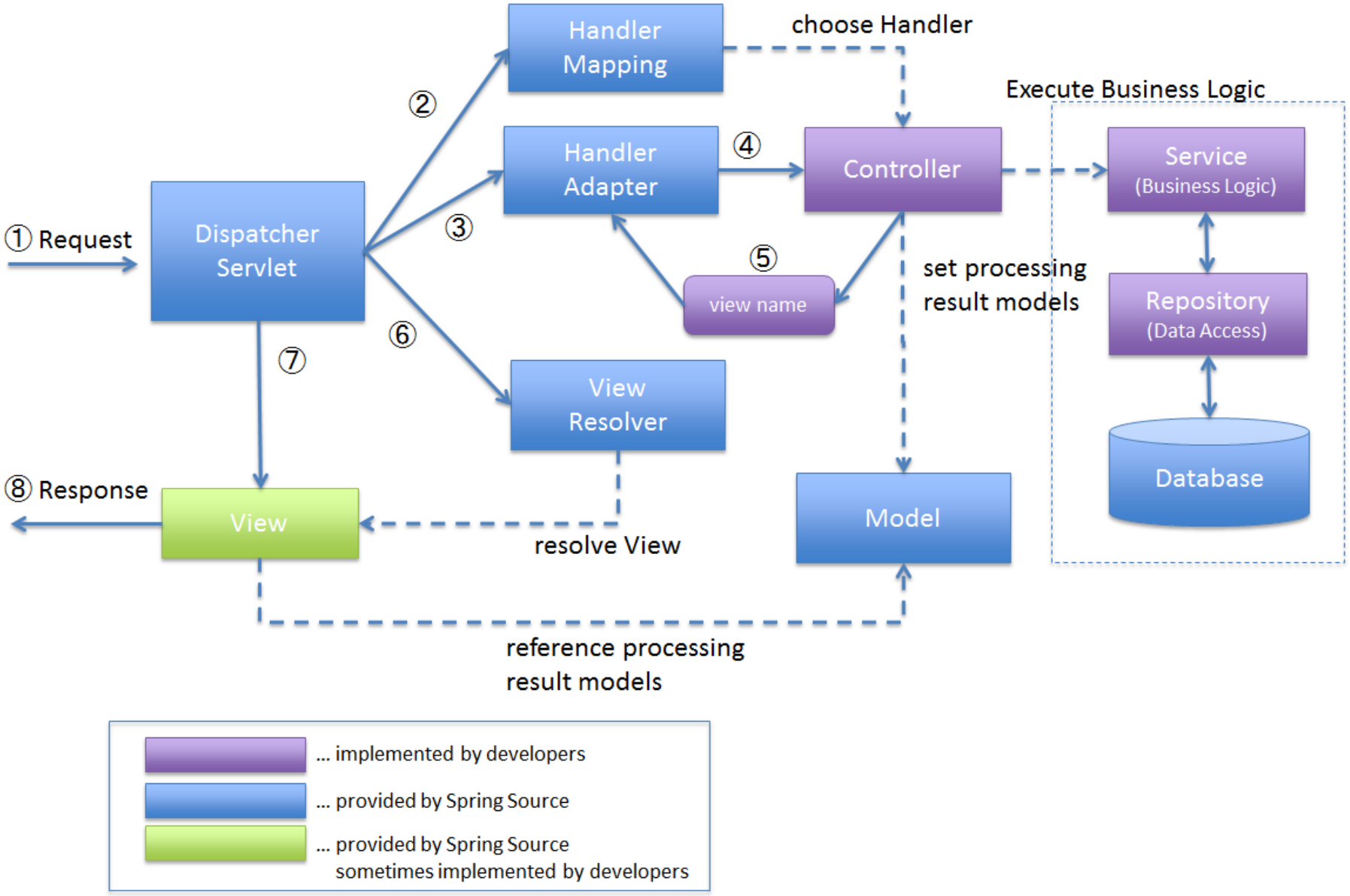
DispatcherServlet Class

- `org.springframework.web.servlet.DispatcherServlet`
- ***Front Controller pattern*** 적용
- web.xml에 설정
- Client로부터의 모든 요청을 전달 받음
- Controller나 View와 같은 Spring MVC의 구성요소를 이용하여 Client에게 Service를 제공

Spring MVC의 주요 구성 요소

- **DispatcherServlet**
  - Client의 요청을 받아서 Controller에게 Client의 요청을 전달하고, Return한 결과값을 View에게 전달하여 알맞은 응답을 생성
- HandlerMapping
  - URL과 요청 정보를 기준으로 어떤 Handler 객체를 사용할지 결정하는 객체이며, DispatcherServlet은 하나 이상의 Handler Mapping을 가질 수 있음.
- Controller
  - Client의 요청을 처리한 뒤, Model를 호출하고 그 결과를 DispatcherServlet에게 알려 줌.
- **ModelAndView**
  - Controller가 처리한 data 및 화면에 대한 정보를 보유한 객체
- View
  - Controller의 처리 결과 화면에 대한 정보를 보유한 객체
- **ViewResolver**
  - Controller가 return한 View 이름을 기반으로 Controller 처리 결과를 생성할 View를 결정

Spring MVC의 주요 구성 요소의 요청 처리 과정

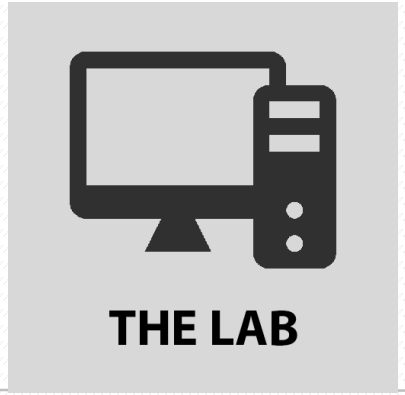


Spring MVC의 주요 구성 요소의 요청 처리 과정 (Cont.)

- Client의 요청이 **DispatcherServlet**에게 전달된다.
- **DispatcherServlet**은 HandlerMapping을 사용하여 Client의 요청을 처리할 Controller를 획득한다.
- **DispatcherServlet**은 Controller 객체를 이용하여 Client의 요청을 처리한다.
- Controller는 Client 요청 처리 결과와 View 페이지 정보를 담은 **ModelAndView** 객체를 반환한다.
- **DispatcherServlet**은 **ViewResolver**로부터 응답 결과를 생성할 View 객체를 구한다.
- View는 Client에게 전송할 응답을 생성한다.

Spring MVC 기반 Web Application 작성 절차

- Client의 요청을 받는 **DispatcherServlet**를 web.xml에 설정
- Client의 요청을 처리할 Controller를 작성
- Spring Bean으로 Controller를 등록
- JSP를 이용한 View 영역의 코드를 작성
- Browser 상에서 JSP를 실행



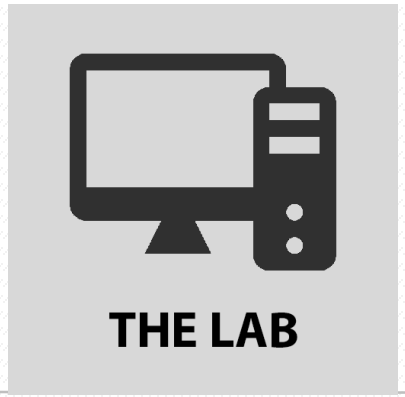
Task 1. Spring MVC Demo

Spring MVC Demo 처리 순서

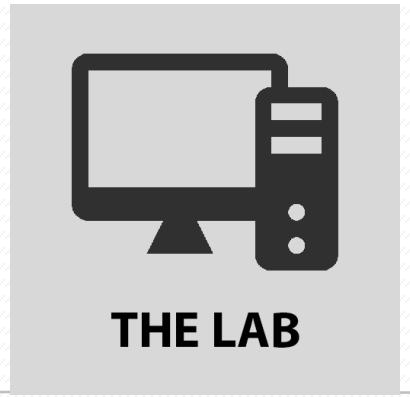
- Web Browser의 요청을 web.xml의 `<url-pattern></url-pattern>`를 통해 `/`의 요청을 받는다.
- `servlet-name`이 `appServlet`인 `servlet-class`는 `org.springframework.web.servlet.DispatcherServlet`이다.
- 이 `DispatcherServlet`는 Loading하면서 `/WEB-INF/spring/appServlet/servlet-context.xml`를 Parameter로 초기화한다.
- `servlet-context.xml`에서 `<context:component-scan base-package="com.example.biz" />`를 통해 Scan을 `base-package`에서 한다.
- `com.example.biz`에 `@Controller`를 찾는다.
- `@Controller`가 있는 `HomeController.java`에서 `@RequestMapping(value = "/", method = RequestMethod.GET)`가 설정되어 있는 method인 `public String home(Locale locale, Model model)`를 찾는다.
- 왜냐하면 지금 Browser가 요청한 Method는 ***GET***이고, 요청 경로는 `/`이기 때문이다.

Spring MVC Demo 처리 순서 (Cont.)

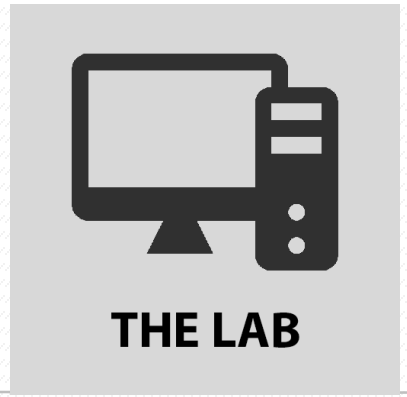
- **serverTime**을 설정하고 model의 **addAttribute()** method를 통해 View에게 사용할 값을 저장한다.
- 그리고 **return "home"**을 통해 jsp File이름을 반환한다.
- 다시 **servlet-context.xml**에서 **ViewResolver**는 ***prefix***가 **/WEB-INF/views/**이고, ***suffix***가 **.jsp**이며 방금 반환된 jsp File 이름인 home은 prefix + file 이름 + suffix를 하면 **/WEB-INF/views/home.jsp**가 된다.
- 그래서 **home.jsp**를 Browser에게 전송한다.
- 이때 JSP는 Model에 저장된 **servetTime**을 함께 View에 출력하게 된다.



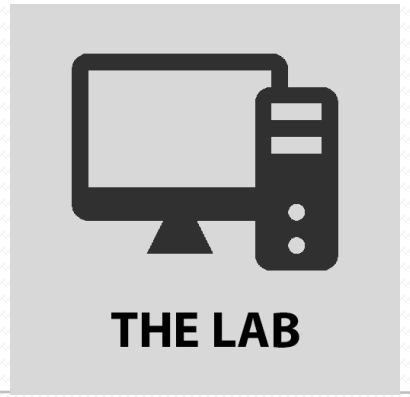
Task 2. resources Folder 이용하기



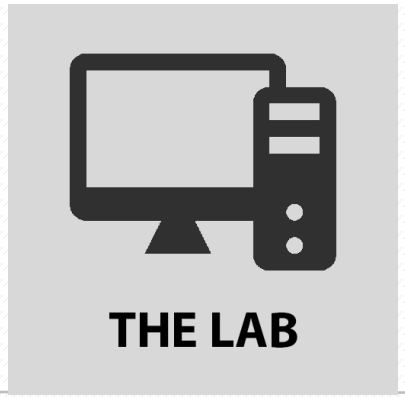
Task 3. Controller Class 제작하기



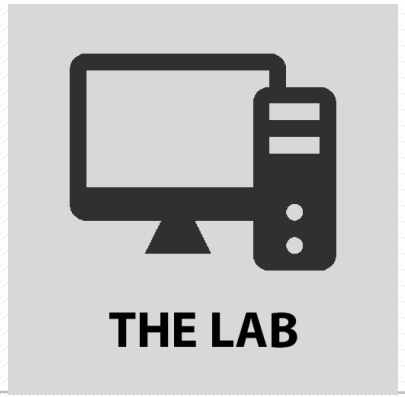
Task 4. 다양한 GET Request 처리하기



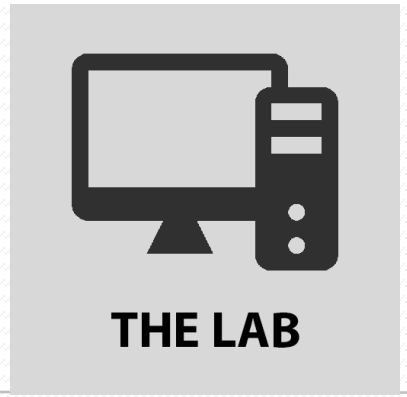
Task 5. @RequestMapping Parameter 다루기



Task 6. Database와 연동하기



Task 7. Form Data Validataion



Task 8. Convert J2EE to Spring MVC

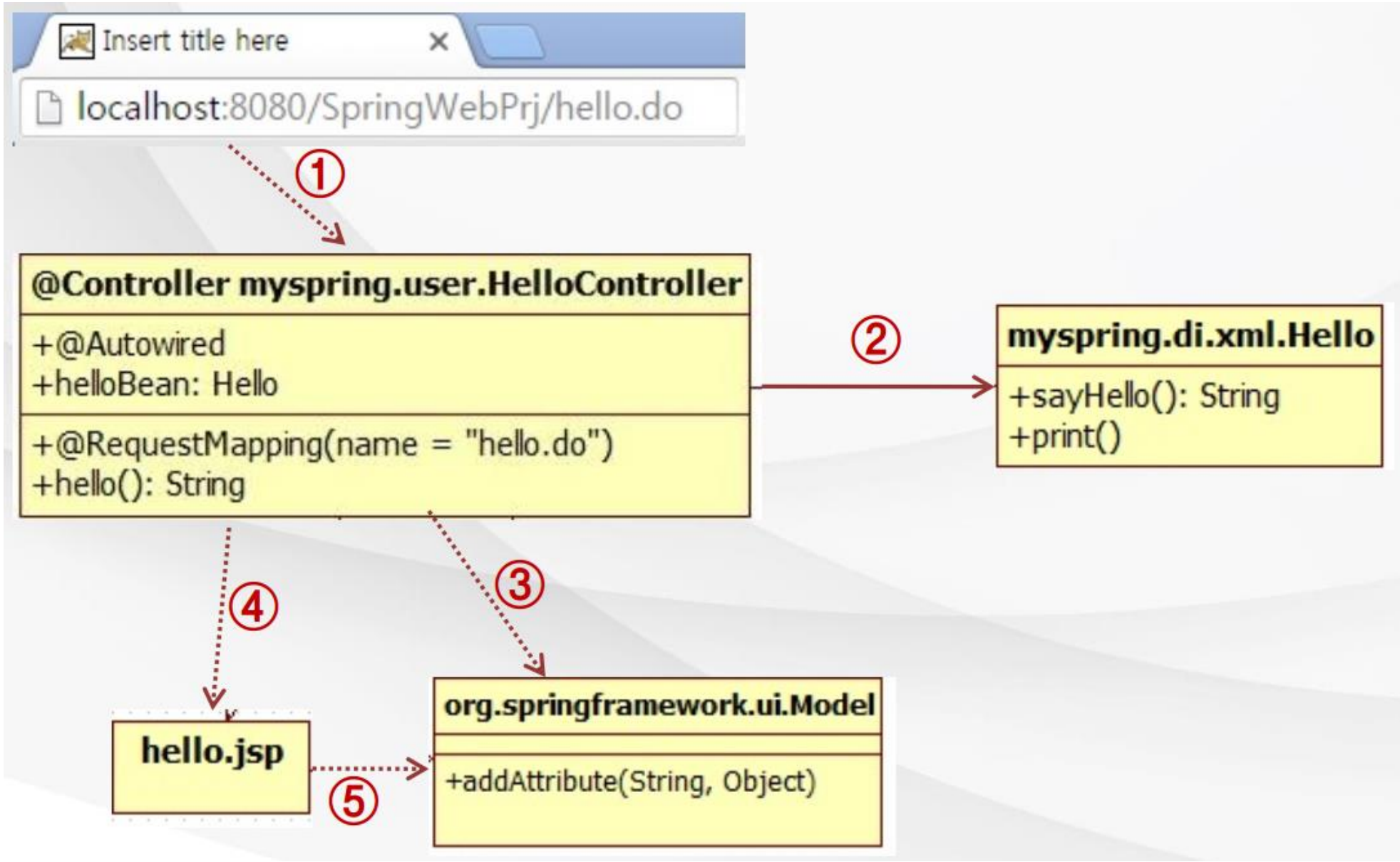
ContextLoaderListener Class 설정

- Business Logic용의 Spring 설정 File (ex:***applicationContext.xml***)을 작성했기 때문에 Listener로 **ContextLoaderListener** Class를 정의해야.
- **DispatcherServlet**이 생성되어 Spring Container를 구동하면 Controller들이 Memory에 Loading된다.
- 하지만 Controller들이 생성되기 전에 누군가가 먼저 config폴더에 있는 ***beans.xml*** File을 읽어 Business Component들을 Memory에 생성해야 한다.
- 이때 사용하는 Class가 **ContextLoaderListener** Class이다.
- **<listener>** 하위에 **<listener-class>** Tag를 이용하여 Spring에서 제공하는 **ContextLoaderListener**를 등록하면 된다.

ContextLoaderListener Class 설정 (Cont.)

- **ContextLoaderListener** Class는 Spring 설정 File(Default에서 File명 ***applicationContext.xml***)을 Load하면 **ServletContextListener** interface를 구현하고 있기 때문에 **ServletContext** instance 생성 시(Tomcat으로 Application이 Load된 때)에 호출된다.
- 즉, **ContextLoaderListener** Class는 **DispatcherServlet** Class의 Loading보다 먼저 동작하여 Business Logic층을 정의한 Spring 설정 File을 Load한다.
- 중요한 것은 **ContextLoaderListener** Class는 Servlet Container가 ***web.xml*** File을 읽어서 구동할 때, 자동으로 Memory에 생성된다.
- **ContextLoaderListener**는 Client의 요청이 없어도 Container가 구동될 때 Pre-Loading 되는 객체이다.

Controller와 JSP 호출순서



Hello Controller 작성

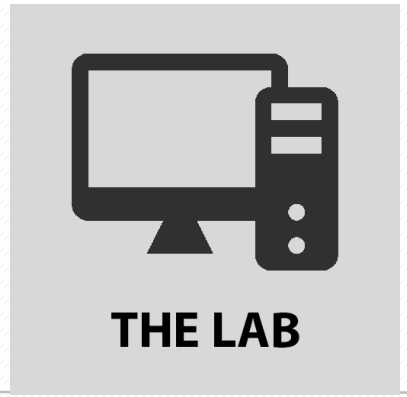
- Client의 요청을 처리할 POJO 형태의 **HelloController** Class를 작성
- Controller Class에 **@Controller** Annotation 선언
- 요청을 처리할 Method를 작성하고 **@RequestMapping** Annotation을 선언
- **@RequestMapping**
  - HTTP 요청 URL을 처리할 Controller Method 정의

View에 data를 전달하는 Model Class

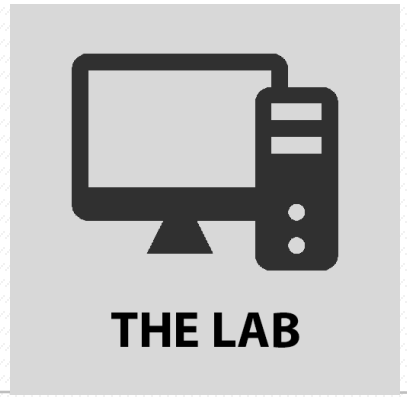
- Controller에서 Service를 호출한 결과를 받아서 View에게 전달하기 위해, 전달받은 결과를 Model 객체에 저장

```
Model addAttribute(String name, Object value)
```

- **value**객체를 **name**의 이름으로 저장하고, View Code에서는 **name**으로 지정한 이름을 통해서 **value**를 사용



Task 9. Convert J2EE to Spring MVC



Task 10. File Upload with Spring MVC

Server에 File을 저장할 때 고려해야 할 사항들

- File Upload 방식 결정하기.
  - Post 방식으로 전송할지 아니면 Ajax 방식으로 전송할지 결정.
  - 요즘은 주로 Ajax방식을 선호.
- File이름 중복문제.
  - DB에 File을 저장할 수도 있지만, 일반적으로 ***FileSystem***에 File을 저장하게 된다.
  - 따라서 Upload 되는 File의 이름의 중복을 해결할 방법이 필요하다.
  - UUID로 해결가능

Server에 File을 저장할 때 고려해야 할 사항들 (Cont.)

- File 저장경로에 대한문제.
  - Windows나 Linux등 운영체제에서 Folder내의 File 개수가 너무 많아지게 되면, 속도저하 문제가 발생하게 된다.
  - 특히 Windows의 ***FileSystem***의 경우 Folder내 최대 File 개수의 제한이 있다. (100만단위가 넘어가긴 하지만...)
  - 위 문제를 해결하기 위해서 보통 File이 Upload 되는 시점별로 Folder를 관리 한다.
  - 예를 들어 2018년 9월 6일 File이 Upload 되면, 그 File은 특정 Folder의 경로의 /2018/09/06/ 경로에 저장하면 위 문제를 해결 할 수있다.
  - 즉 Upload 할 때 File을 저장할 Folder의 유무에 따라 Folder 생성 Logic이 필요하다.

Server에 File을 저장할 때 고려해야 할 사항들 (Cont.)

- Image File의 경우 Thumbnail 생성.
  - Image File인 경우 저장된 File을 다시 화면에 보여줄 때, 보통 그 Image File의 Thumbnail File을 보여주게 된다.
  - 따라서 Image File이 Server에 저장될 때는 추가적으로 그 Image File의 Thumbnail File을 생성해 주어야 한다.
  - imgscalr-lib Library가 Image의 Thumbnail 생성을 해준다.

Spring MulipartResolver

- UI에서 ***multipart/form-data***방식으로 Server에 전송되는 Data를 Spring MVC의 **MulipartResolver**로 처리할 수 있다.
- File Upload library를 추가했다면, **CommonMultipartResolver**를 bean에 등록해야 한다.
- **MultipartResolver**는 **Muiltpart** 객체를 **Controller**에 전달하는 역할을 한다.
- 이 설정에서 아주 중요한 점은, **CommonMultipartResolver** Class의 **id**나 **이름**값이다.

Spring MulipartResolver (Cont.)

- 이 class는 이름이 정해져 있다.
- 정확하게는 `DispatcherServlet`이 특정 이름으로 등록된 `CommonsMultipartResolver` 객체만 인식하도록 되어 있다.
- 따라서 반드시 `CommonsMultipartResolver`의 `id`값은 ***multipartResolver***를 사용해야 한다.

Spring MulipartResolver (Cont.)

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver"
>
      <!-- 최대 upload 가능한 byte크기 -->
      <property name="maxUploadSize" value="10240000" />
      <!-- Disk에 임시 File을 생성하기 전에 Memory에 보관할 수있는 최대
Byte 크기 -->
      <!-- property name="maxInMemorySize" value="52428800" / -->
      <!-- defaultEncoding -->
      <property name="defaultEncoding" value="utf-8" />
</bean>
```

Spring MulipartResolver (Cont.)

- **maxUploadSize**에 대한 Setter Injection은 Upload할 수 있는 File의 크기를 제한하기 위한 설정이다.
- 지정하지 않으면 기본으로 **-1**이 설정되는데, 이는 File의 크기가 ***무제한***이라는 의미이다.