

삼성 청년 SW 아카데미

Spring Framework

<알림>

본 강의는 삼성 청년 SW아카데미의 콘텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

SpringFramework - Spring Web MVC

목차

1. Spring Web MVC
2. Controller
3. View
4. Model
5. Spring Web MVC 동작 정리

Spring Web MVC

삼성 청년 SW 아카데미

✓ MVC(Model-View-Controller) Pattern.

▪ Model

- 어플리케이션 상태의 캡슐화.
- 상태 쿼리에 대한 응답.
- 어플리케이션의 기능 표현.
- 변경을 view에 통지.

▪ View

- 모델을 화면에 시각적으로 표현.
- 모델에게 업데이트 요청.
- 사용자의 입력을 컨트롤러에 전달.
- 컨트롤러가 view를 선택하도록 허용.

▪ Controller

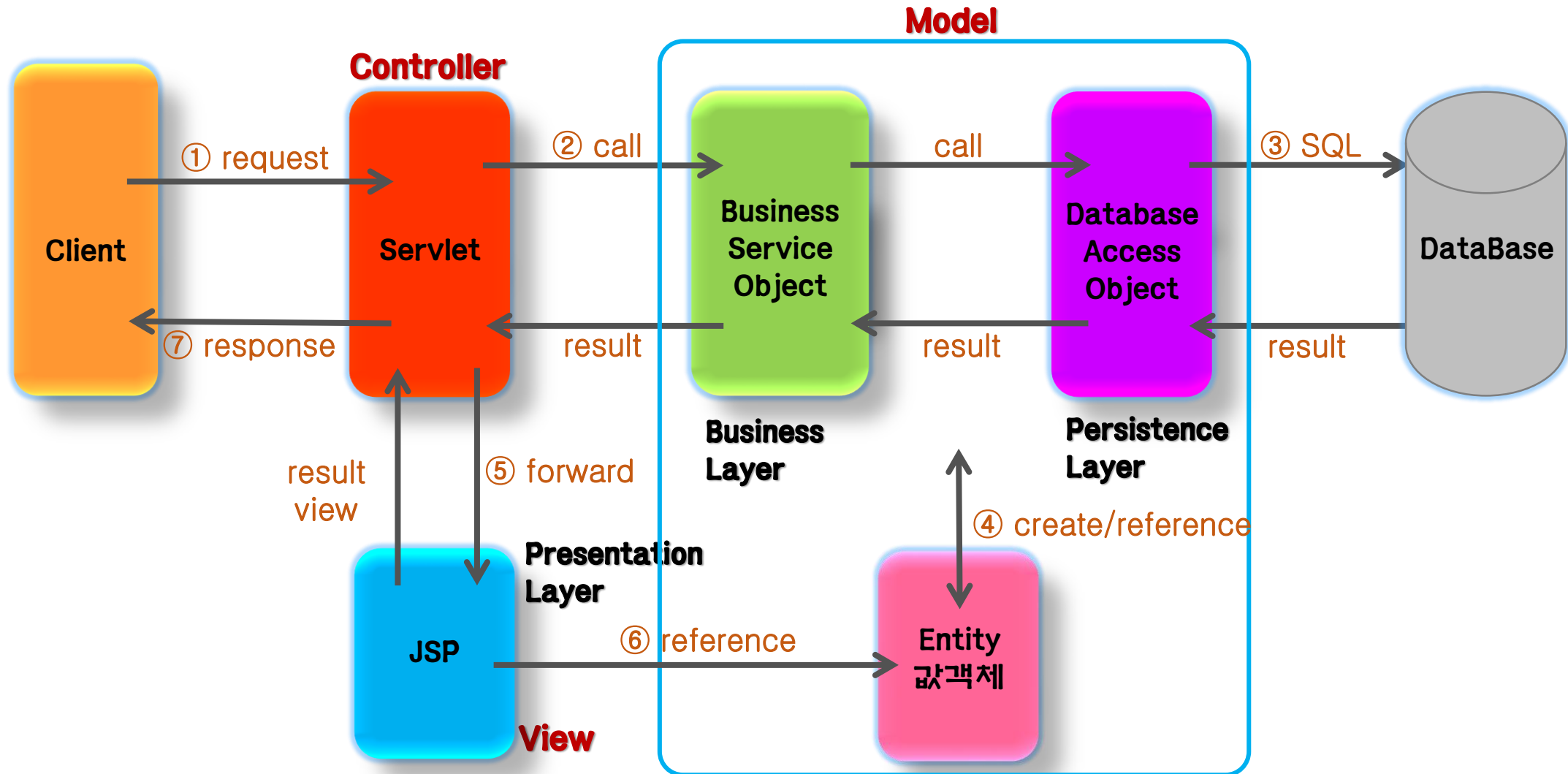
- 어플리케이션의 행위 정의.
- 사용자 액션을 모델 업데이트와 mapping.
- 응답에 대한 view 선택 .

✓ MVC(Model-View-Controller) Pattern.

- 어플리케이션의 확장을 위해 Model, View, Controller 세가지 영역으로 분리.
- 컴포넌트의 변경이 다른 영역 컴포넌트에 영향을 미치지 않음(유지보수 용이).
- 컴포넌트 간의 결합성이 낮아 프로그램 수정이 용이(확장성이 뛰어남).
- 장점
 - 화면과 비즈니스 로직을 분리해서 작업 가능.
 - 영역별 개발로 인하여 확장성이 뛰어남.
 - 표준화된 코드를 사용하므로 공동작업이 용이하고 유지보수성이 좋음.
- 단점
 - 개발과정이 복잡해 초기 개발속도가 늦음.
 - 초보자가 이해하고 개발하기에 다소 어려움.

Spring Web MVC

✓ Model 2 (Web MVC) 요청 흐름.



✓ Spring MVC.

▪ Spring MVC 특징.

- Spring은 DI나 AOP같은 기능 뿐만 아니라, Servlet 기반의 WEB 개발을 위한 MVC Framework를 제공.
- Spring MVC는 Model2 Architecture와 Front Controller Pattern을 Framework 차원에서 제공.
- Spring MVC Framework는 Spring을 기반으로 하고 있기 때문에 Spring이 제공하는 Transaction 처리나 DI 및 AOP등을 손쉽게 사용.

✓ Spring MVC.

▪ Spring MVC 구성요소. (1/2)

• DispatcherServlet (Front Controller)

- 모든 클라이언트의 요청을 전달받음.
- Controller에게 클라이언트의 요청을 전달하고, Controller가 리턴 한 결과값을 View에게 전달하여 알맞은 응답을 생성.

• HandlerMapping

- 클라이언트의 요청 URL을 어떤 Controller가 처리할지를 결정.
- URL과 요청 정보를 기준으로 어떤 핸들러 객체를 사용할지 결정하는 객체이며, DispatcherServlet은 하나 이상의 핸들러 매핑을 가질 수 있음.

• Controller

- 클라이언트의 요청을 처리한 뒤, Model을 호출하고 그 결과를 DispatcherServlet에 알려준다.

✓ Spring MVC.

▪ Spring MVC 구성요소. (2/2)

- ModelAndView

- Controller가 처리한 데이터 및 화면에 대한 정보를 보유한 객체.

- ViewResolver

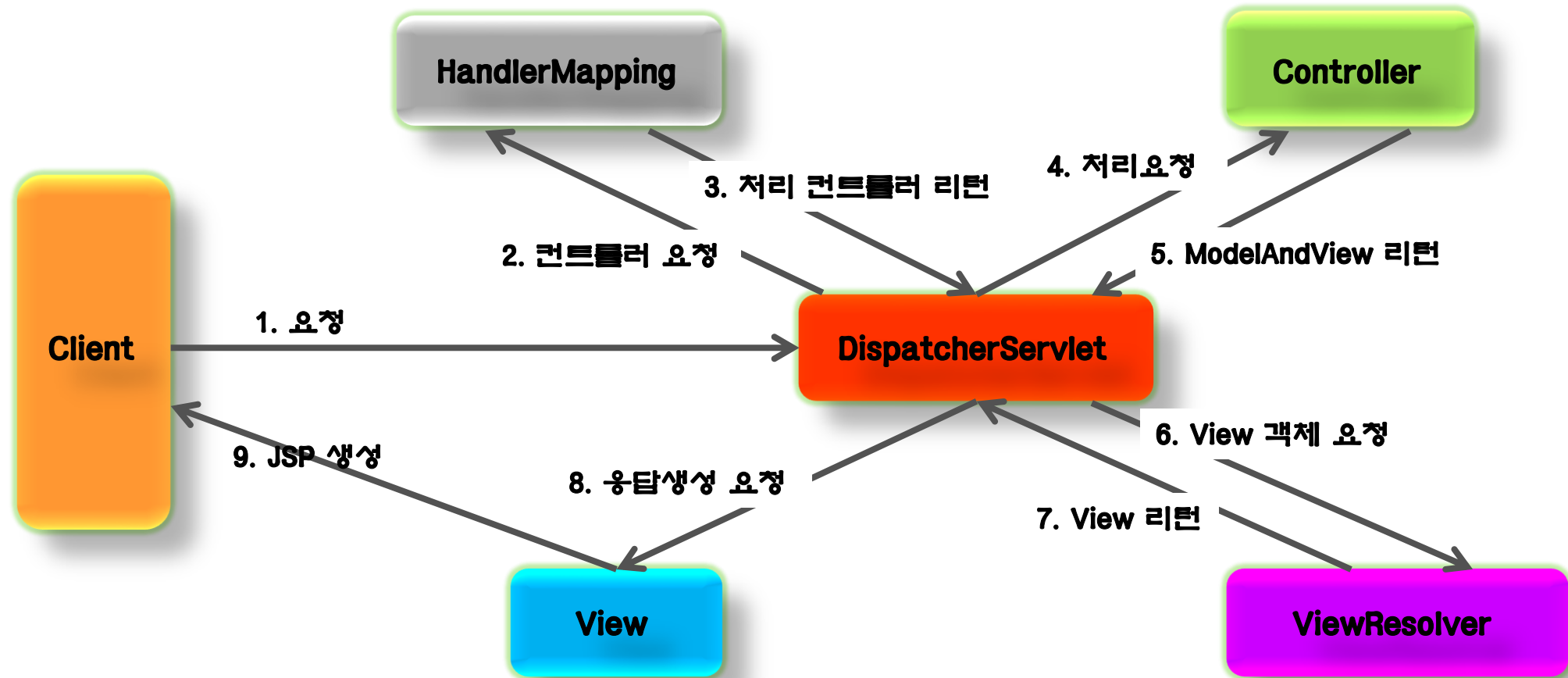
- Controller가 리턴 한 뷰 이름을 기반으로 Controller의 처리 결과를 보여줄 View를 결정.

- View

- Controller의 처리결과를 보여줄 응답화면을 생성.

Spring Web MVC

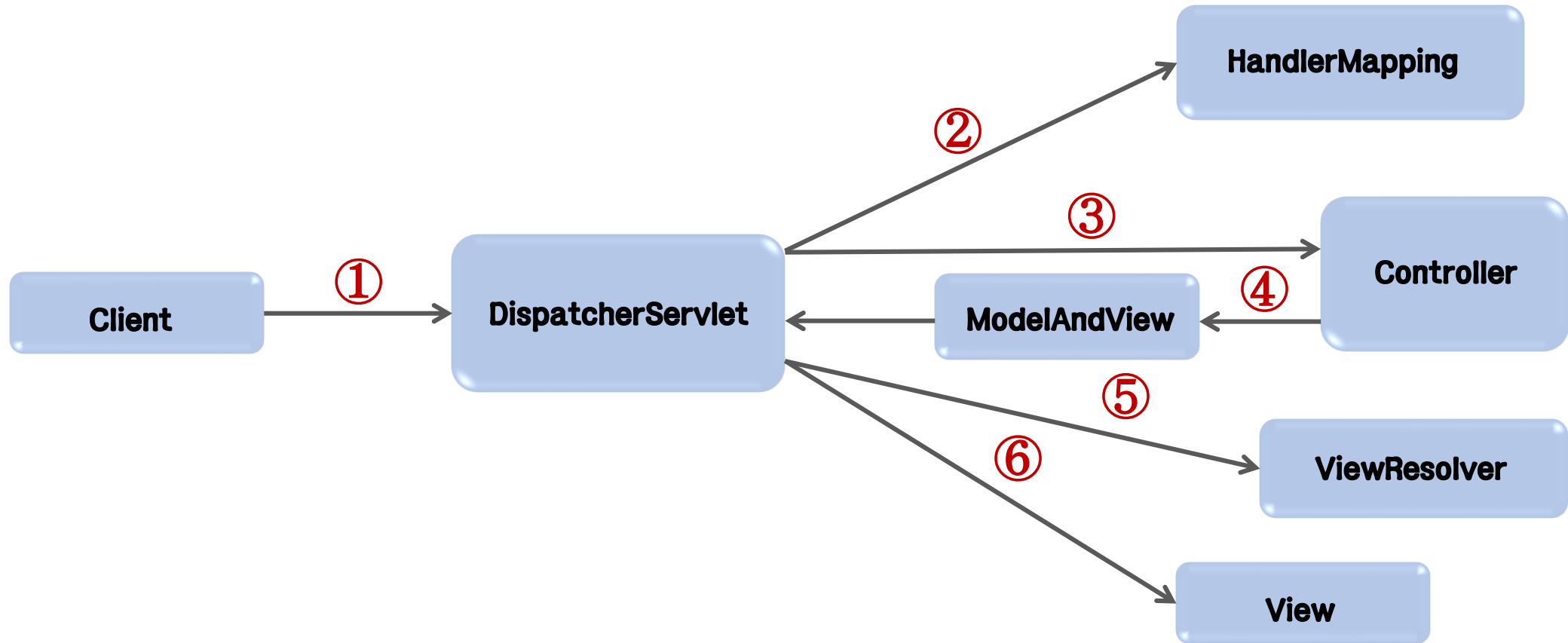
✓ Spring MVC 요청 흐름.



Spring Web MVC

✓ Spring MVC.

- Spring MVC 실행 순서 (1/2).



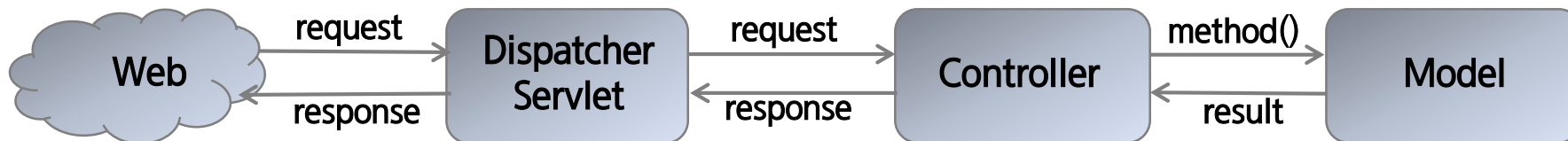
✓ Spring MVC.

▪ Spring MVC 실행 순서 (2/2).

- ① DispatcherServlet이 요청을 수신.
 - 단일 Front Controller Servlet.
 - 요청을 수신하여 처리를 다른 컴포넌트에 위임.
 - 어느 Controller에 요청을 전송할지 결정 .
- ② DispatcherServlet은 Handler Mapping에 어느 Controller를 사용할 것인지 문의.
 - URL과 Mapping.
- ③ DispatcherServlet은 요청을 Controller에게 전송하고 Controller는 요청을 처리한 후 결과 리턴.
 - Business Logic 수행 후 결과 정보(Model)가 생성되어 JSP와 같은 view에서 사용됨.
- ④ ModelAndView Object에 수행결과가 포함되어 DispatcherServlet에 리턴.
- ⑤ ModelAndView는 실제 JSP정보를 갖고 있지 않으며, ViewResolver가 논리적 이름을 실제 JSP이름으로 변환.
- ⑥ View는 결과정보를 사용하여 화면을 표현함 .

✓ Spring MVC 구현.

- Spring MVC를 이용한 Application 구현 Step
 - web.xml에 DispatcherServlet 등록 및 Spring 설정파일 등록.
 - 설정 파일에 HandlerMapping 설정.
 - Controller 구현 및 Context 설정 파일(servlet-context.xml)에 등록.
 - Controller와 JSP의 연결을 위해 View Resolver 설정.
 - JSP 코드 작성.
- Controller 작성
 - 좋은 디자인은 Controller가 많은 일을 하지 않고 Service에 처리를 위임.



✓ Spring MVC 구현.

▪ **web.xml** - **DispatcherServlet** 설정.

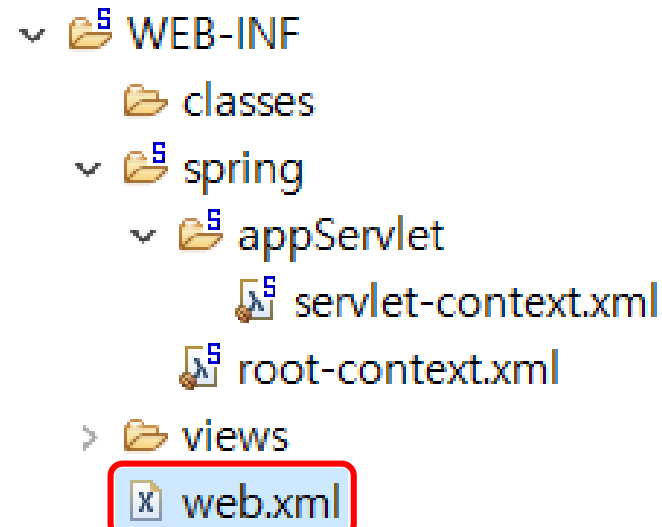
- <init-param>을 설정 하지 않으면 “<servlet-name>-servlet.xml” file에서 applicationContext의 정보를 load.
- Spring Container는 설정파일의 내용을 읽고 ApplicationContext 객체를 생성.
- <url-pattern>은 DispatcherServlet이 처리하는 URL Mapping pattern을 정의.
- Servlet이므로 1개 이상의 DispatcherServlet 설정 가능.
- <load-on-startup>1</load-on-startup>설정 시 WAS startup시 초기화 작업진행.

✓ Spring MVC 구현.

- **web.xml** - **DispatcherServlet** 설정.

```
<!-- 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>
```



✓ Spring MVC 구현.

- **web.xml** - **DispatcherServlet** 설정.
 - DispatcherServlet을 여러 개 설정가능.
 - 각 DispatcherServlet마다
각각의 ApplicationContext 생성.

```
<!-- 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>
  <servlet-name>appServlet2</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-context2.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>appServlet2</servlet-name>
  <url-pattern>*.action</url-pattern>
</servlet-mapping>
```

✓ Spring MVC 구현.

- web.xml - 최상위 Root ContextLoader 설정.

- Context 설정 파일들을 로드하기 위해 web.xml 파일에 리스너 설정.(ContextLoaderListener)

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

- 리스너 설정이 되면 /WEB-INF/spring/root-context.xml 파일을 읽어서 공통적으로 사용되는 최상위 Context를 생성.
- 그 외의 다른 컨텍스트 파일들을 최상위 어플리케이션 컨텍스트에 로드하기 위해서는...

```
<!-- 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
    classpath:com/test/web/application.xml
  </param-value>
</context-param>
```

```
<param-value>
  /WEB-INF/spring/root-*.xml
</param-value>
```

클래스 패스에 위치한 설정파일로부터 로드

✓ Spring MVC 구현.

- Application Context 분리.
 - 어플리케이션 레이어에 따라 어플리케이션 컨텍스트 분리.

Layer	설정파일
Security Layer	board-security.xml
Web Layer	board-servlet.xml
Service Layer	board-service.xml
Persistence Layer	board-dao.xml

✓ Spring MVC 구현.

- Controller Class 작성. (HomeController.java)

```
@Controller
public class HomeController {

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

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("welcome home! The client locale is {}.", locale);

        model.addAttribute("message", "안녕하세요 스프링!!!!");

        return "index";
    }
}
```

- Context 설정파일에 Controller 등록. (servlet-context.xml)

```
<beans:bean class="com.test.web.HomeController"/>
```

✓ Spring MVC 구현.

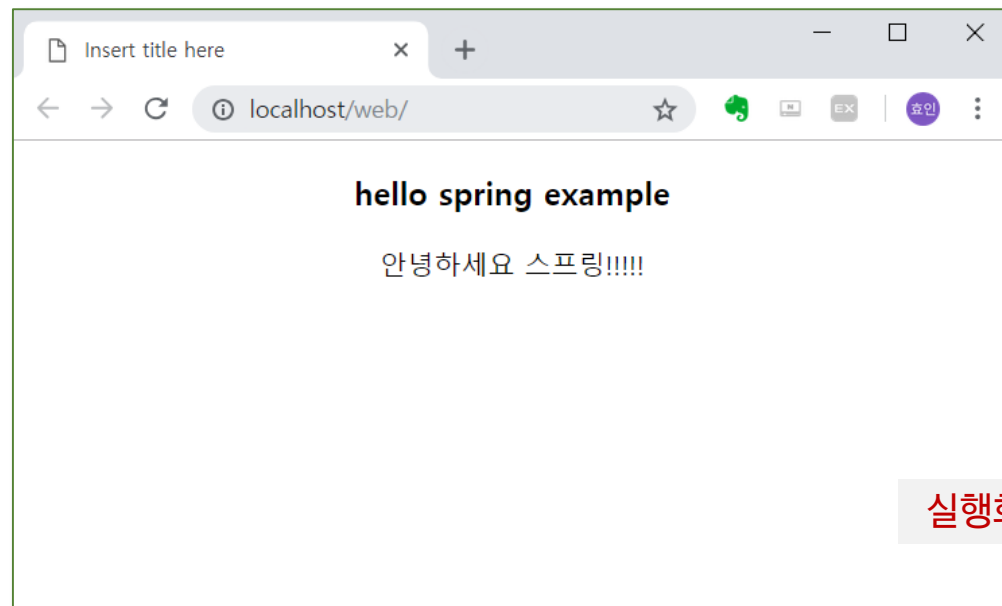
- Controller와 response page 연결을 위한 ViewResolver 설정. (servlet-context.xml)

```
<beans:bean class="com.test.web.HomeController"/>

<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>
```

- JSP (index.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<div align="center">
<h3>hello spring example</h3>
${message}
</div>
</body>
</html>
```



이어서..

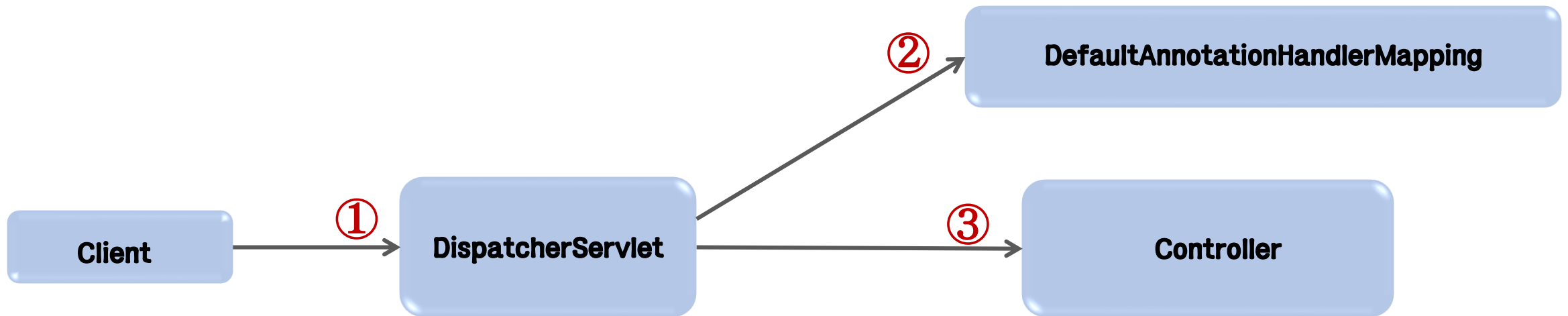
삼성 청년 SW 아카데미

Controller

삼성 청년 SW 아카데미

✓ @Controller.

- @Controller와 @RequestMapping선언.
 - method 단위의 mapping이 가능.
 - DefaultAnnotationHandlerMapping과 AnnotationHandlerAdapter를 사용함.
 - Spring 3.0부터는 기본 설정이므로 별도의 추가 없이 사용 가능.



✓ @Controller.

- Controller Class는 Client의 요청을 처리.
- @Controller 선언.
 - Class 타입에 적용.
 - Spring 3.0부터는 @Controller 사용을 권장.

("/list.ssafy")

AbstractController class, AbstractCommonController class, Controller interface등을 확장해서 Controller 작성을 지양함.

```
@Controller
@RequestMapping("/reboard")
public class ReboardController {

    @Autowired
    private ReboardService reboardService;

    @Autowired
    private CommonService commonService;

    @RequestMapping(value = "/list.ssafy", method = RequestMethod.GET)
    public ModelAndView list(@RequestParam Map<String, String> map) {
        ModelAndView mav = new ModelAndView();

        List<ReboardDto> list = reboardService.listArticle(map);
        PageNavigation navigation = commonService.getPageNavigation(map);
        navigation.setRoot("/board");
        navigation.makeNavigator();
        mav.addObject("list", list);
        mav.addObject("navigator", navigation);
        mav.setViewName("reboard/list");
    }
}
```

✓ @Controller.

- Controller Class를 <bean>에 등록.

servlet-context.xml

```
<beans:bean id="reboardController" class="com.test.board.controller.ReboardController">  
  <beans:property name="reboardService" ref="reboardService"/>  
</beans:bean>
```

- Controller Class 자동 스캔.
 - context:component-scan 선언
 - base-package에 설정된 package내의 class중 @Controller annotation이 적용된 클래스는 자동 스캔대상이 된다.

```
<context:component-scan base-package="com.test.board.controller"/>
```

✓ @Controller.

- @RequestMapping 선언.
 - 요청 URL mapping 정보를 설정.
 - 클래스타입과 메소드에 설정 가능.

```
@Controller
public class ReboardController {

    private ReboardService reboardService;

    public void setReboardService(ReboardService reboardService) {
        this.reboardService = reboardService;
    }

    @RequestMapping("/reboard/write.do")
    public String write() {
        return "reboard/write";
    }

    @RequestMapping("/reboard/list.do")
    public ModelAndView list(@RequestParam Map<String, String> map) {
        List<ReboardDto> article = reboardService.listArticle(map);
    }
}
```

```
@Controller
@RequestMapping("/reboard")
public class ReboardController {

    private ReboardService reboardService;

    public void setReboardService(ReboardService reboardService) {
        this.reboardService = reboardService;
    }

    @RequestMapping("/write.do")
    public String write() {
        return "reboard/write";
    }

    @RequestMapping("/list.do")
    public ModelAndView list(@RequestParam Map<String, String> map) {
        List<ReboardDto> article = reboardService.listArticle(map);
    }
}
```

✓ @Controller.

- Controller method의 HTTP method에 한정.
 - 같은 URL 요청에 대하여 HTTP method(GET, POST..)에 따라 서로 다른 메소드를 mapping 할 수 있음.

```
@Controller
public class HomeController {

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

    @RequestMapping(value = "/index.do", method = RequestMethod.GET)
    public String home(Model model) {
        model.addAttribute("message", "안녕하세요 스프링(GET)!!!!");

        return "index";
    }

    @RequestMapping(value = "/index.do", method = RequestMethod.POST)
    public String home2(Model model) {
        model.addAttribute("message", "안녕하세요 스프링(POST)!!!!");

        return "index";
    }
}
```

✓ @Controller.

- 아래의 Controller에서 @RequestMapping annotation을 설정하지 않으면? HTTP ERROR 404

```
@Controller
@RequestMapping(value = "/index.do")
public class HomeController {

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

    @RequestMapping(method = RequestMethod.GET)
    public String home(Model model) {
        model.addAttribute("message", "안녕하세요 스프링(GET)!!!!");

        return "index";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String home2(Model model) {
        model.addAttribute("message", "안녕하세요 스프링(POST)!!!!");

        return "index";
    }
}
```

✓ @Controller.

- Controller method의 parameter type. (1/2)
 - Controller method의 parameter로 다양한 Object를 받을 수 있음.

Parameter Type	설명
HttpServletRequest HttpServletResponse HttpSession	필요시 Servlet API를 사용할 수 있음
Java.util.Locale	현재 요청에 대한 Locale
InputStream, Reader	요청 콘텐츠에 직접 접근할 때 사용
OutputStream, Writer	응답 콘텐츠를 생성할 때 사용
@PathVariable annotation 적용 파라미터	URI 템플릿 변수에 접근할 때 사용
@RequestParam annotation 적용 파라미터	HTTP 요청 파라미터를 매핑
@RequestHeader annotation 적용 파라미터	HTTP 요청 헤더를 매핑
@CookieValue annotation 적용 파라미터	HTTP 쿠키 매핑

✓ @Controller.

- Controller method의 parameter type. (2/2)
 - Controller method의 parameter로 다양한 Object를 받을 수 있음.

Parameter Type	설명
@RequestBody annotation 적용 파라미터	HTTP 요청의 body 내용에 접근할 때 사용
Map, Model, ModelMap	view에 전달할 model data를 설정할 때 사용
커맨드 객체(DTO)	HTTP 요청 parameter를 저장 한 객체, 기본적으로 클래스 이름을 모델명으로 사용. @ModelAttribute annotation 설정으로 모델명을 설정할 수 있음.
Errors, BindingResult	HTTP 요청 파라미터를 커맨드 객체에 저장한 결과, 커맨드 객체를 위한 파라미터 바로 다음에 위치
SessionStatus	폼 처리를 완료 했음을 처리하기 위해 사용. @SessionAttributes annotation을 명시한 session속성을 제거하도록 이벤트를 발생 시킨다.

✓ @Controller.

- @RequestParam annotation을 이용한 parameter mapping.

URL 호출 : <http://localhost/web/index.do?name=안효인&age=30>

```
@Controller
public class HomeController {

    @RequestMapping(value = "/index.do", method = RequestMethod.GET)
    public String home(@RequestParam("name") String name,
        @RequestParam("age") int age, Model model) {
        model.addAttribute("message", name + "(" + age + ")님 안녕하세요!!!!");
        return "index";
    }
}
```

```
@Controller
public class HomeController {

    @RequestMapping(value = "/index.do", method = RequestMethod.GET)
    public String home(@RequestParam(value="name", required=false) String name,
        @RequestParam(value="age", defaultValue="25") int age, Model model) {
        model.addAttribute("message", name + "(" + age + ")님 안녕하세요!!!!");
        return "index";
    }
}
```

필수여부

기본값

✓ @Controller.

- HTML form과 Command Object(DTO, VO..).
- SpringMVC는 form에 입력한 data를 JavaBean 객체를 이용해서 전송 할 수 있음.

```
<form method="POST" action="${root}/board/write.do">  
제목 : <input type="text" name="subject"><br>  
내용 : <textarea name="content"></textarea><br>  
<input type="submit" value="글쓰기">
```

```
@Controller  
@RequestMapping("/board")  
public class BoardController {  
  
    @RequestMapping(value="/write.do", method=RequestMethod.GET)  
    public String write() {  
        return "board/write";  
    }  
  
    @RequestMapping(value="/write.do", method=RequestMethod.POST)  
    public String write(BoardDto boardDto) {  
  
        return "board/writeok";  
    }  
}
```

```
public class BoardDto {  
  
    private String subject;  
    private String content;  
  
    public void setSubject(String subject) {  
        this.subject = subject;  
    }  
  
    public void setContent(String content) {  
        this.content = content;  
    }  
}
```

Spring Web MVC

✓ @Controller.

- Command 객체를 List로 받기.

```
<form method="POST" action="${root}/shop/bascket.do">
  <input type="text" name="productList[0].pnum">
  <input type="text" name="productList[0].name">
  <input type="text" name="productList[0].price">
  <br>
  <input type="text" name="productList[1].pnum">
  <input type="text" name="productList[1].name">
  <input type="text" name="productList[1].price">
  <input type="submit" value="장바구니저장">
</form>
```

```
public class Bascket {
    private List<Product> productList;

    public List<Product> getProductList() {
        return productList;
    }

    public void setProductList(List<Product> productList) {
        this.productList = productList;
    }
}
```

```
@Controller
@RequestMapping("/shop")
public class ShopController {

    @RequestMapping(value="/bascket.do", method=RequestMethod.POST)
    public String bascket(Bascket bascket, Model model) {
        model.addAttribute("bascketList", bascket);
        return "shop/bascketlist";
    }
}
```

```
public class Product {
    private int pnum;
    private String name;
    private int price;

    public int getPnum() {
        return pnum;
    }

    public void setPnum(int pnum) {
        this.pnum = pnum;
    }
}
```

✓ @Controller.

- View에서 Command 객체에 접근.
 - Command 객체는 자동으로 반환되는 Model에 추가됨.
 - Controller의 @RequestMapping annotation method에서 전달받은 Command 객체에 접근.

```
@Controller
@RequestMapping("/board")
public class BoardController {

    @RequestMapping(value="/write.do", method=RequestMethod.POST)
    public String write(BoardDto boardDto) {
        return "board/writeok";
    }
}
```

제목 : \${boardDto.subject}

내용 : \${boardDto.content}

writeok.jsp

request에 저장되는 command 객체의 이름은 BoardDto의 첫글자를 소문자로 변경한 것이다. (→boardDto)

- @ModelAttribute를 사용하여 View에서 사용할 Command 객체의 이름을 변경.

```
@Controller
@RequestMapping("/board")
public class BoardController {

    @RequestMapping(value="/write.do", method=RequestMethod.POST)
    public String write(@ModelAttribute("article") BoardDto boardDto) {
        return "board/writeok";
    }
}
```

제목 : \${article.subject}

내용 : \${article.content}

writeok.jsp

✓ @Controller.

- @CookieValue annotation을 이용한 Cookie mapping.

```
@Controller
public class HomeController {

    public String hello(@CookieValue("author") String authorValue) {
        return "ok";
    }

    public String hello(@CookieValue(value="author", required=false, defaultValue="user") String authorValue) {
        return "ok";
    }
}
```

- @RequestHeader annotation을 이용한 header mapping.

```
@Controller
public class HomeController {

    public String hello(@RequestHeader("Accept-Language") String headerLanguage) {
        return "ok";
    }
}
```

✓ @Controller.

- @RequestBody parameter type.
 - HTTP 요청 Body가 그대로 객체에 전달됨.
 - AnnotationMethodHandlerAdapter에는 HttpMessageConverter 타입의 메시지 변환기가 기본으로 여러 개 등록되어 있음.
 - @RequestBody가 붙은 parameter가 있으면 해당 미디어 타입을 확인 후 처리 가능한 변환기(Converter)가 자동으로 객체로 변환시켜 줌.
 - 주로 @ResponseBody와 함께 사용됨. ➔ 비동기처리

✓ @Controller.

- Servlet API 사용.
 - HttpSession의 생성을 직접 제어해야 하는 경우.
 - Controller가 Cookie를 생성해야 하는 경우.
 - Servlet API를 선호하는 경우.
 - javax.servlet.ServletRequest / javax.servlet.http.HttpServletRequest
 - javax.servlet.ServletResponse / javax.servlet.http.HttpServletResponse
 - javax.servlet.http.HttpSession

```
@Controller
public class HomeController {

    public String hello(HttpServletRequest request, HttpServletResponse respons) {
        return "ok";
    }
}
```

✓ @Controller.

- Controller Class에서 method의 return type 종류.

Return Type	설명
ModelAndView	model 정보 및 view 정보를 담고있는 ModelAndView 객체.
Model	view에 전달할 객체 정보를 담고있는 Model을 반환한다. 이때 view 이름은 요청 URL로부터 결정된다. (RequestToViewNameTranslator)
Map	view에 전달 할 객체 정보를 담고 있는 Map을 반환한다. 이때 view 이름은 요청 URL로부터 결정된다. (RequestToViewNameTranslator)
String	view의 이름을 반환한다.
View	view 객체를 직접 리턴, 해당 View 객체를 이용해서 view를 생성한다.
void	method가 ServletResponse나 HttpServletResponse 타입의 parameter를 갖는 경우 method가 직접 응답을 처리한다고 가정한다. 그렇지 않을 경우 요청 URL로부터 결정된 View를 보여준다. (RequestToViewNameTranslator)
@ResponseBody Annotation 적용	method에서 @ResponseBody annotation이 적용된 경우, 리턴 객체를 HTTP 응답으로 전송한다. HttpMessageConverter를 이용해서 객체를 HTTP 응답 스트림으로 변환한다.

이어서..

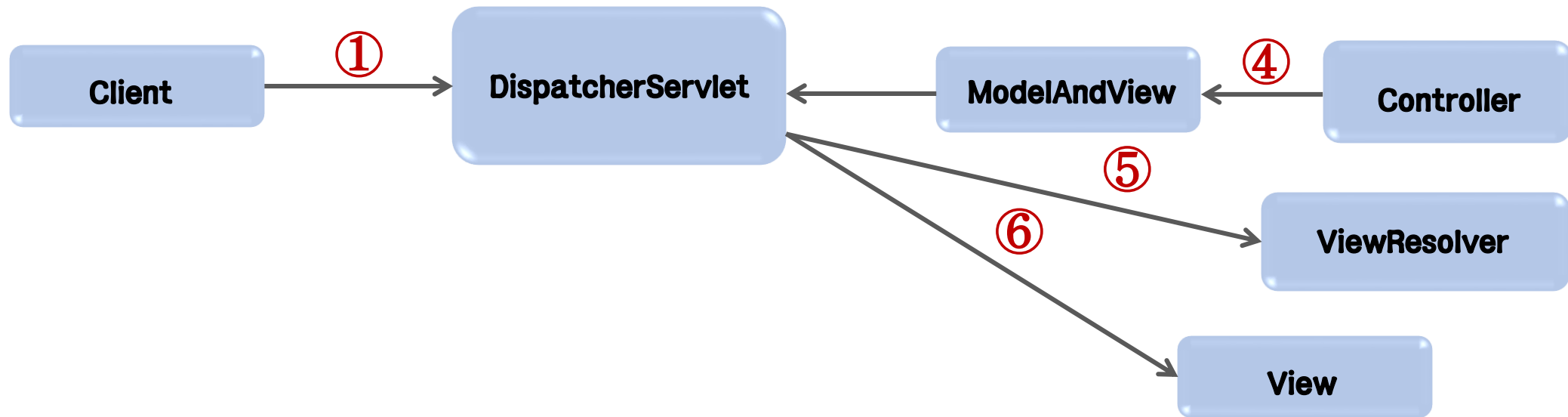
삼성 청년 SW 아카데미

View

삼성 청년 SW 아카데미

✓ View 지정.

- Controller에서는 처리 결과를 보여줄 View 이름이나 객체를 리턴하고, DispatcherServlet은 View이름이나 View 객체를 이용하여 view를 생성.
 - 명시적 지정.
 - 자동 지정.



✓ View 지정.

- ViewResolver : 논리적 view와 실제 JSP파일 mapping.
 - servlet-context.xml

```
<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>
```

InternalResourceViewResolver는
prefix + 논리뷰 + suffix로 설정
ex)/WEB-INF/views/board/list.jsp

✓ View 지정.

- View 이름 명시적 지정.
 - ModelAndView와 String 리턴 타입.

```
@Controller
public class HomeController {

    @RequestMapping("/hello.do")
    public ModelAndView hello() {
        ModelAndView mav = new ModelAndView("hello");
        return mav;
    }
}
```

```
@Controller
public class HomeController {

    @RequestMapping("/hello.do")
    public ModelAndView hello() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("hello");
        return mav;
    }
}
```

```
@Controller
public class HomeController {

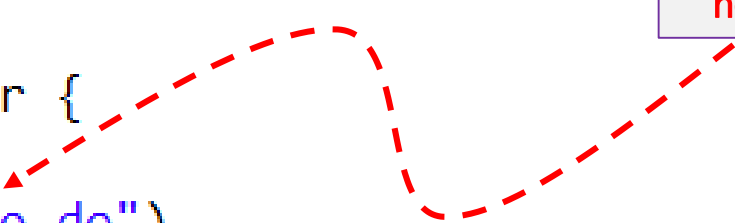
    @RequestMapping("/hello.do")
    public String hello() {
        return "hello";
    }
}
```

✓ View 지정.

- View 자동 지정.
 - RequestToViewNameTranslator를 이용하여 URL로 부터 view 이름을 결정.
 - 자동 지정 유형.
 - return type이 Model이나 Map인 경우.
 - return type이 void이면서 ServletResponse나 HttpServletResponse 타입의 parameter가 없는 경우.

```
@Controller
public class HomeController {

    @RequestMapping("/hello.do")
    public Map<String, Object> hello() {
        Map<String, Object> model = new HashMap<String, Object>();
        return model;
    }
}
```



hello가 view 이름이 됨.

✓ View 지정.

- redirect view.
 - View 이름에 “**redirect:**” 접두어를 붙이면, 지정한 페이지로 redirect 됨.
 - `redirect:/board/list.html?pg=1`
 - `redirect:http://localhost/board/list.html?pg=1`

```
@Controller
public class BoardRegisterController {
    @Autowired
    private BoardService boardService;

    // 글등록후 1페이지 글리스트로 이동.
    @RequestMapping(value="board/register.html", method=RequestMethod.POST)
    public String register(@ModelAttribute("article") BoardDto boardDto) {
        boardService.registerArticle(boardDto);
        return "redirect:board/list.html?pg=1";
    }
}
```

이어서..

삼성 청년 SW 아카데미

Model

삼성 청년 SW 아카데미

✓ Model 생성.

- View에 전달하는 데이터.
 - @RequestMapping annotation이 적용된 method의 Map, Model, ModelMap.
 - @RequestMapping method가 return하는 ModelAndView.
 - @ModelAttribute annotation이 적용된 method가 return 한 객체.

✓ Model 생성.

- Map, Model, ModelMap을 통한 설정.
 - method의 argument로 받는 방식.

```
@Controller
public class HomeController {

    @RequestMapping("/hello.do")
    public String hello(Map model) {
        model.put("message", "안녕하세요");
        return "hello";
    }
}
```

```
@Controller
public class HomeController {

    @RequestMapping("/hello.do")
    public String home2(ModelMap model) {
        model.addAttribute("message", "안녕하세요");
        return "hello";
    }
}
```

```
@Controller
public class HomeController {

    @RequestMapping("/hello.do")
    public String hello(Model model) {
        model.addAttribute("message", "안녕하세요");
        return "hello";
    }
}
```

✓ Model 생성.

- Model Interface 주요 method.
 - Model addAttribute(String name, Object value);
 - Model addAttribute(Object value);
 - Model addAllAttributes(Collection<?> values);
 - Model addAllAttributes(Map<String, ?> attributes);
 - Model mergeAttributes(Map<String, ?> attributes);
 - boolean containsAttribute(String name);

✓ Model 생성.

- ModelAndView를 통한 Model 설정.
 - Controller에서 처리결과를 보여줄 view와 view에 전달할 값(model)을 저장하는 용도로 사용.
 - `setViewName(String viewname);`
 - `addObject(String name, Object value);`

```
@Controller
public class HomeController {

    @RequestMapping("/hello.do")
    public ModelAndView hello() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("hello");
        mav.addObject("message", "안녕하세요");
        return mav;
    }
}
```

✓ Model 생성.

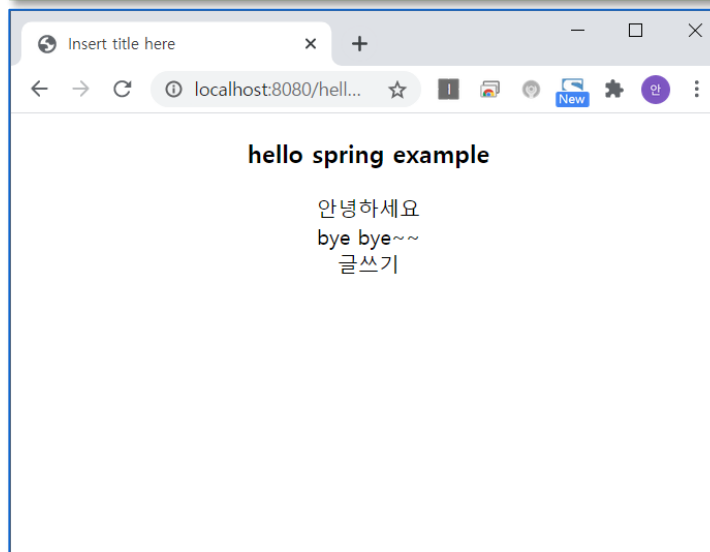
- @ModelAttribute annotation을 이용한 model data 처리.
 - @RequestMapping annotation이 적용되지 않은 별도 method로 모델이 추가될 객체를 생성.

```
@Controller
public class HomeController {

    @ModelAttribute("modelAttrMessage")
    public String maMessage() {
        return "bye bye~~";
    }

    @RequestMapping("/hello.do")
    public String hello(Model model) {
        model.addAttribute("message", "안녕하세요");
        return "index";
    }
}
```

```
<div align="center">
<h3>hello spring example</h3>
${message}<br>
${modelAttrMessage}<br>
<a href="${root}/board/write.do">글쓰기</a>
</div>
```



✓ 요청 URL 매칭.

- 전체 경로와 Servlet기반 경로 매칭.
 - DispatcherServlet은 DefaultAnnotationHandlerMapping Class를 기본으로 HandlerMapping 구현체로 사용.
 - Default로 Context 내의 경로가 아닌 Servlet 경로를 제외한 나머지 경로에 대해 mapping.

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>*.html</url-pattern>
  <url-pattern>/product/*</url-pattern>
</servlet-mapping>
```

@RequestMapping("/board/list.html")

@RequestMapping("/product/list")

✓ 요청 URL 매칭.

- Servlet기반 경로 매칭.
 - Servlet 경로를 포함한 전체 경로를 이용해서 매칭하려는 경우.
 - @RequestMapping("/product/list")

```
<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
    <property name="alwaysUseFullPath" value="true"/>
</bean>

<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
    <property name="alwaysUseFullPath" value="true"/>
</bean>
```


✓ 요청 URL 매칭.

- @PathVariable annotation을 이용한 URI 템플릿.
 - RESTful 방식
 - http://localhost/users/troment
 - http://localhost/product/
 - http://localhost/forum/board1/10
 - @RequestMapping Annotation 값으로 {템플릿변수}를 사용.
 - @PathVariable Annotation을 이용해서 {템플릿변수}와 동일한 이름을 갖는 parameter를 추가.

```
@Controller
public class BoardViewController {
    @Autowired
    private BoardService boardService;

    @RequestMapping("/blog/{userId}/board1/{articleSeq}")
    public String viewArticle(@PathVariable String userId, @PathVariable int articleSeq, Model model) {
        BoardDto boardDto = boardService.getBlogArticle(userId, articleSeq);
        model.addAttribute("article", boardDto);
        return "view";
    }
}
```

✓ 요청 URL 매칭.

- @RequestMapping Annotation의 추가 설정 방법(1/2).
 - @RequestMapping Annotation을 class와 method에 함께 적용하는 경우.

```
@Controller
@RequestMapping("/blog/{userId}")
public class BoardViewController {
    @Autowired
    private BoardService boardService;

    @RequestMapping("/board1/{articleSeq}")
    public String viewArticle(@PathVariable String userId, @PathVariable int articleSeq, Model model) {
        BoardDto boardDto = boardService.getBlogArticle(userId, articleSeq);
        model.addAttribute("article", boardDto);
        return "view";
    }
}
```

✓ 요청 URL 매칭.

- @RequestMapping Annotation의 추가설정 방법(2/2).
 - Ant 스타일의 URI패턴 지원.
 - ? : 하나의 문자열과 대치.
 - * : 하나 이상의 문자열과 대치.
 - ** : 하나 이상의 디렉토리와 대치.

```
@RequestMapping("/members/*.do")
```

```
@RequestMapping("/blog/*/board/{articleSeq}")
```

이어서..

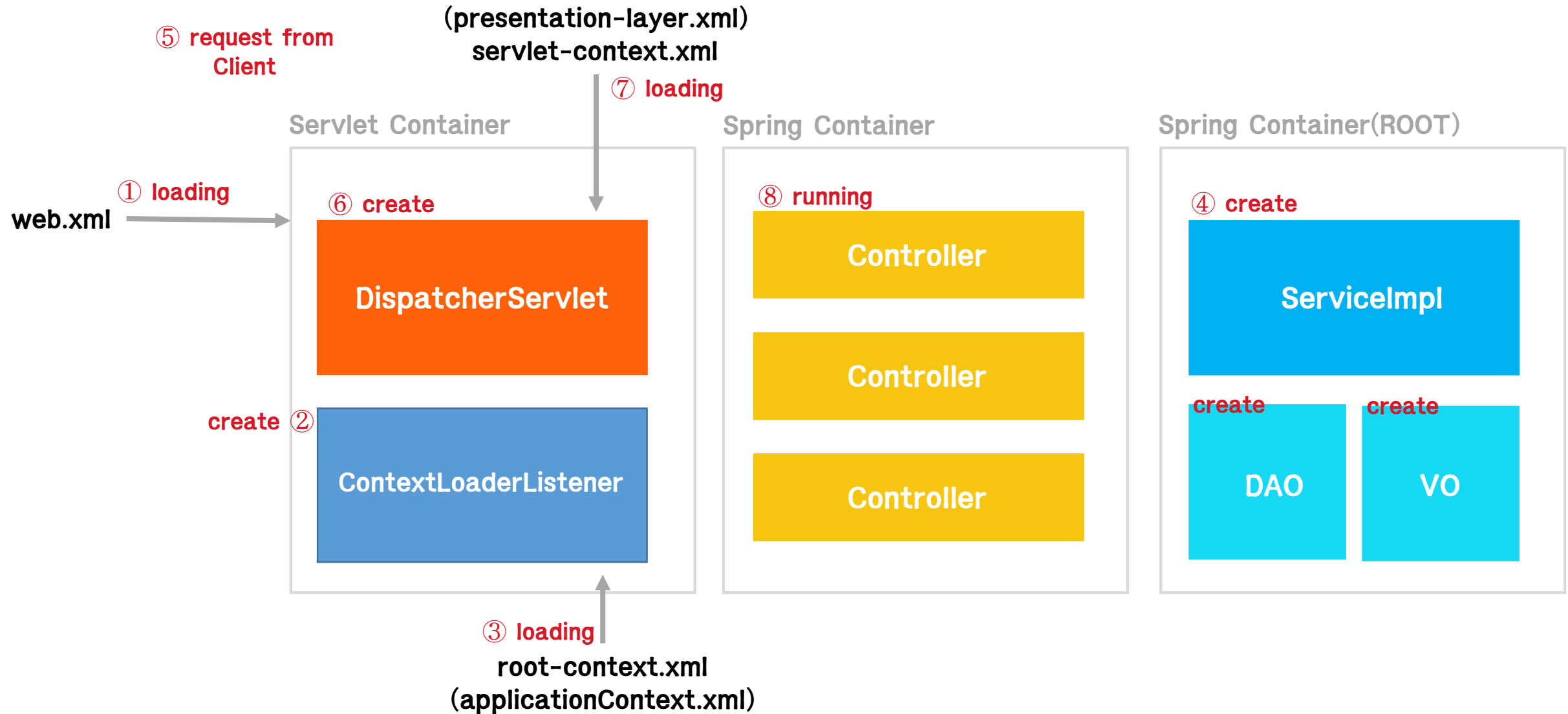
삼성 청년 SW 아카데미

Spring Web MVC 동작 정리

삼성 청년 SW 아카데미

Spring Web MVC

✓ Spring Web Application의 동작 원리.



✓ Spring Web Application의 동작 원리.

▪ 실행 순서

1. 웹 어플리케이션이 실행되면 Tomcat(WAS)에 의해 web.xml이 loading.
2. web.xml에 등록되어 있는 ContextLoaderListener (Java Class)가 생성. ContextLoaderListener class는 ServletContextListener interface를 구현하고 있으며, ApplicationContext를 생성하는 역할을 수행.
3. 생성된 ContextLoaderListener는 root-context.xml을 loading.
4. root-context.xml에 등록되어 있는 Spring Container가 구동. 이 때 개발자가 작성한 Business Logic(Service)에 대한 부분과 Database Logic(DAO), VO 객체들이 생성.
5. Client로 부터 요청(request)가 들어옴.
6. DispatcherServlet(Servlet)이 생성. DispatcherServlet은 FrontController의 역할을 수행.
Client로부터 요청 온 메시지를 분석하여 알맞은 PageController에게 전달하고 응답을 받아 요청에 따른 응답을 어떻게 할 지 결정. 실질적인 작업은 PageController에서 이루어 진다.
이러한 클래스들을 HandlerMapping, ViewResolver Class라고 함.
7. DispatcherServlet은 servlet-context.xml을 loading.
8. 두번째 Spring Container가 구동되며 응답에 맞는 PageController들이 동작. 이 때 첫번째 Spring Container가 구동되면서 생성된 DAO, VO, Service 클래스들과 협업하여 알맞은 작업을 처리.

내일 방송에서 만나요!

삼성 청년 SW 아카데미