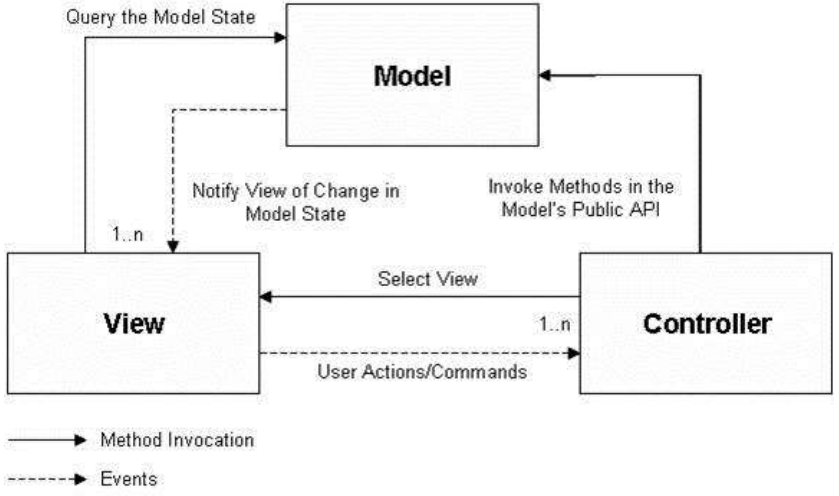
❑ 서비스 개요

- MVC(Model-View-Controller) 패턴은 코드를 기능에 따라 Model, View, Controller 3가지 요소로 분리한다.
  - **Model** : 어플리케이션의 **데이터와 비지니스 로직을 담는 객체**이다.
  - **View** : **Model의 정보를 사용자에게 표시**한다. 하나의 Model을 다양한 View에서 사용할 수 있다.
  - **Controller** : **Model과 View의 중계역활**로 view를 선택한다. 사용자의 요청을 받아 Model에 변경된 상태를 반영하고, 응답을 위한 V
- MVC 패턴은 **UI 코드와 비지니스 코드를 분리**함으로써 **종속성을 줄이고, 재사용성을 높이고, 보다 쉬운 변경이 가능하도록 한다.**
- 전자정부프레임워크에서 "MVC 서비스"란 MVC 패턴을 활용한 Web MVC Framework를 의미한다.



❑ 오픈소스 Web MVC Framework

- Spring MVC, Struts, Webwork 등이 있다.
- **전자정부프레임워크에서는 Spring MVC를 채택**하였다.
  - Framework내의 특정 클래스를 상속하거나, 참조, 구현해야 하는 등의 제약사항이 비교적 적다.
  - IOC Contatiner가 Spring 이라면 연계를 위한 추가 설정 없이 Spring MVC를 사용할 수 있다.
  - 오픈소스 프로젝트가 활성화(꾸준한 기능 추가, 빠른 bug fix와 Q&A) 되어 있으며 로드맵이 신뢰할 만 하다.
  - 국내 커뮤니티 활성화 정도, 관련 참고문서나 도서를 쉽게 구할 수 있다.

❑ Spring MVC

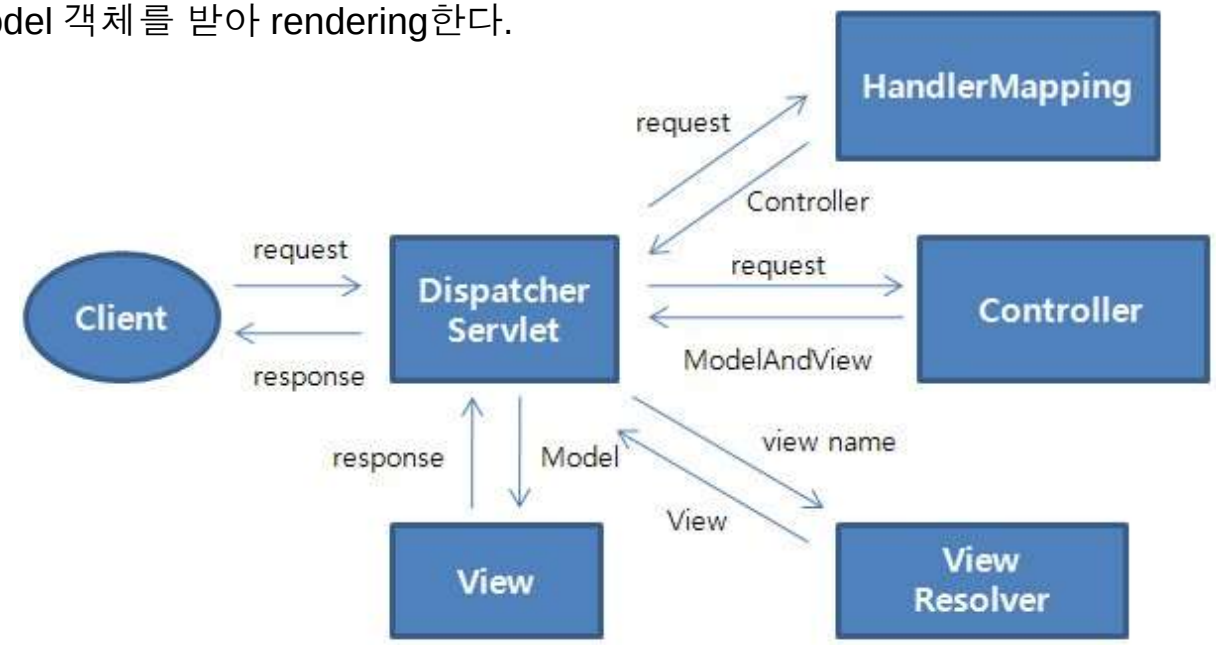
- DispatcherServlet, HandlerMapping, Controller, Interceptor, ViewResolver, View등 각 **컴포넌트들의 역할이 명확하게 분리**된다.
- HandlerMapping, Controller, View등 컴포넌트들에 **다양한 인터페이스 및 구현 클래스를 제공**한다.
- Controller(@MVC)나 폼 클래스(커맨드 클래스) 작성시에 특정 클래스를 상속받거나 참조할 필요 없이 **POJO 나 POJO-style의 클래스를 작성함으로써 비지니스 로직에 집중한 코드를 작성**할 수 있다.
- 웹요청 파라미터와 커맨드 클래스간에 데이터 매핑 기능을 제공한다.
- 데이터 검증을 할 수 있는, Validator와 Error 처리 기능을 제공한다.
- JSP Form을 쉽게 구성하도록 Tag를 제공한다.

❑ Spring MVC의 핵심 Component

- **DispatcherServlet**
  - Spring MVC Framework의 Front Controller, 웹요청과 응답의 Life Cycle을 주관한다.
- HandlerMapping
  - 웹요청시 해당 URL을 어떤 Controller가 처리할지 결정한다.
- **Controller**
  - 비지니스 로직을 수행하고 결과 데이터를 ModelAndView에 반영한다.
- ModelAndView
  - Controller가 수행 결과를 반영하는 Model 데이터 객체와 이동할 페이지 정보(또는 View객체)로 이루어져 있다.
- ViewResolver
  - 어떤 View를 선택할지 결정한다.
- **View**
  - 결과 데이터인 Model 객체를 display한다.

❑ Spring MVC 컴포넌트간의 관계와 흐름

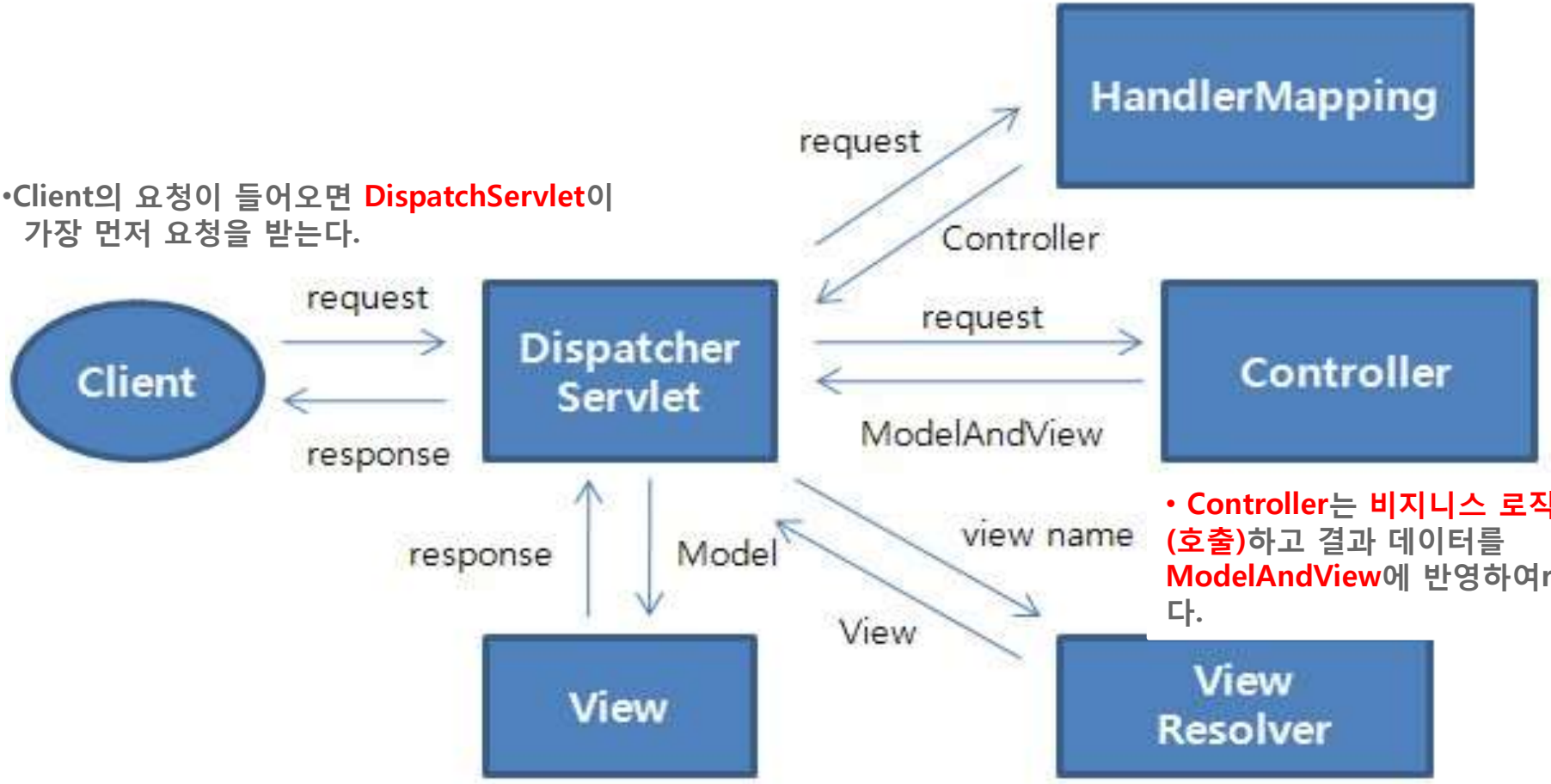
- Client의 요청이 들어오면 DispatchServlet이 가장 먼저 요청을 받는다.
- HandlerMapping이 요청에 해당하는 Controller를 return한다.
- Controller는 비지니스 로직을 수행(호출)하고 결과 데이터를 ModelAndView에 반영하여 return한다.
- ViewResolver는 view name을 받아 해당하는 View 객체를 return한다.
- View는 Model 객체를 받아 rendering한다.



❑ Spring MVC 컴포넌트간의 관계와 흐름

•HandlerMapping이 요청에 해당하는 Controller를 return한다.

•Client의 요청이 들어오면 DispatchServlet이 가장 먼저 요청을 받는다.



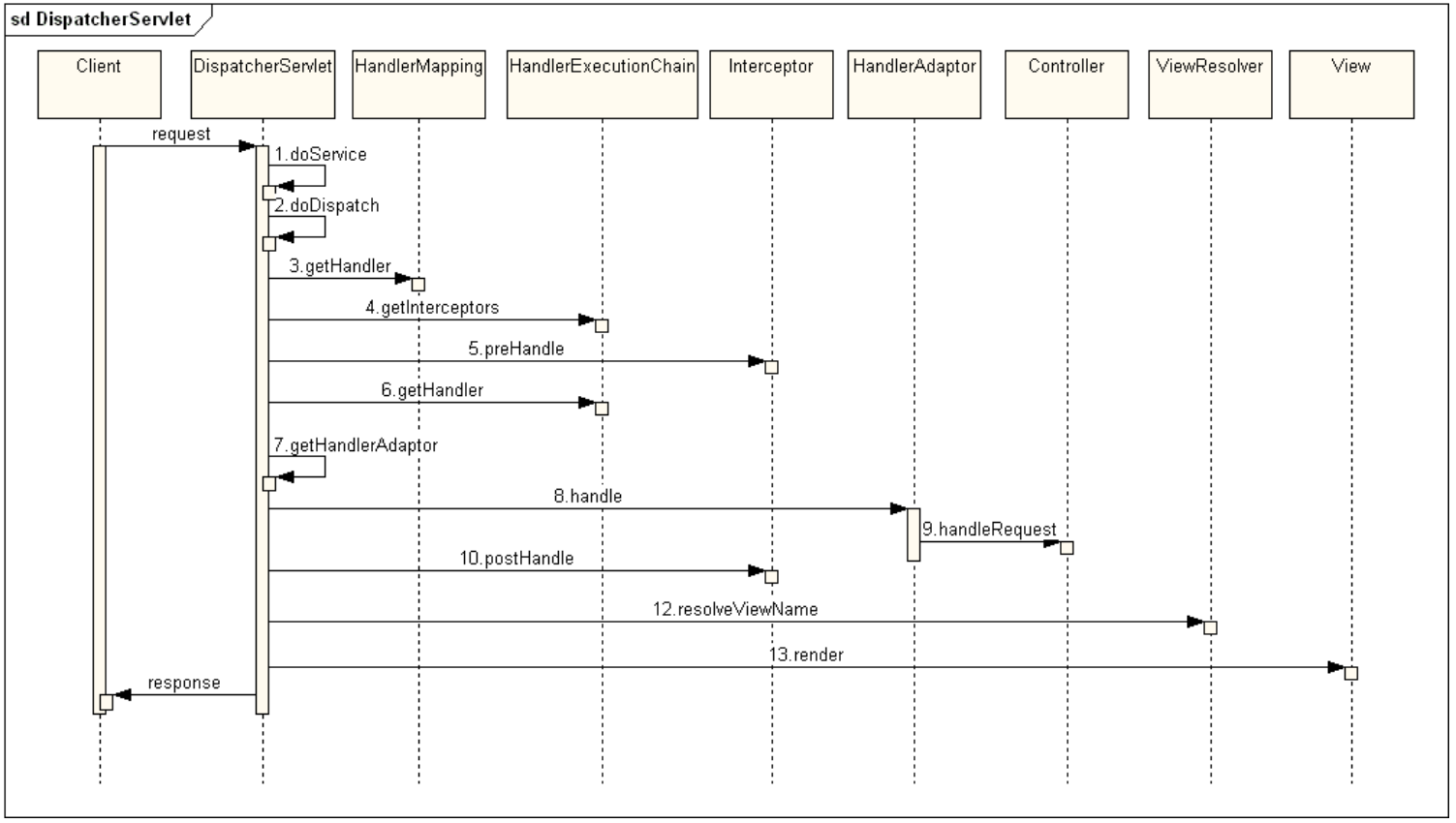
• Controller는 비지니스 로직을 수행(호출)하고 결과 데이터를 ModelAndView에 반영하여return한다.

•View는 Model 객체를 받아 rendering한다.

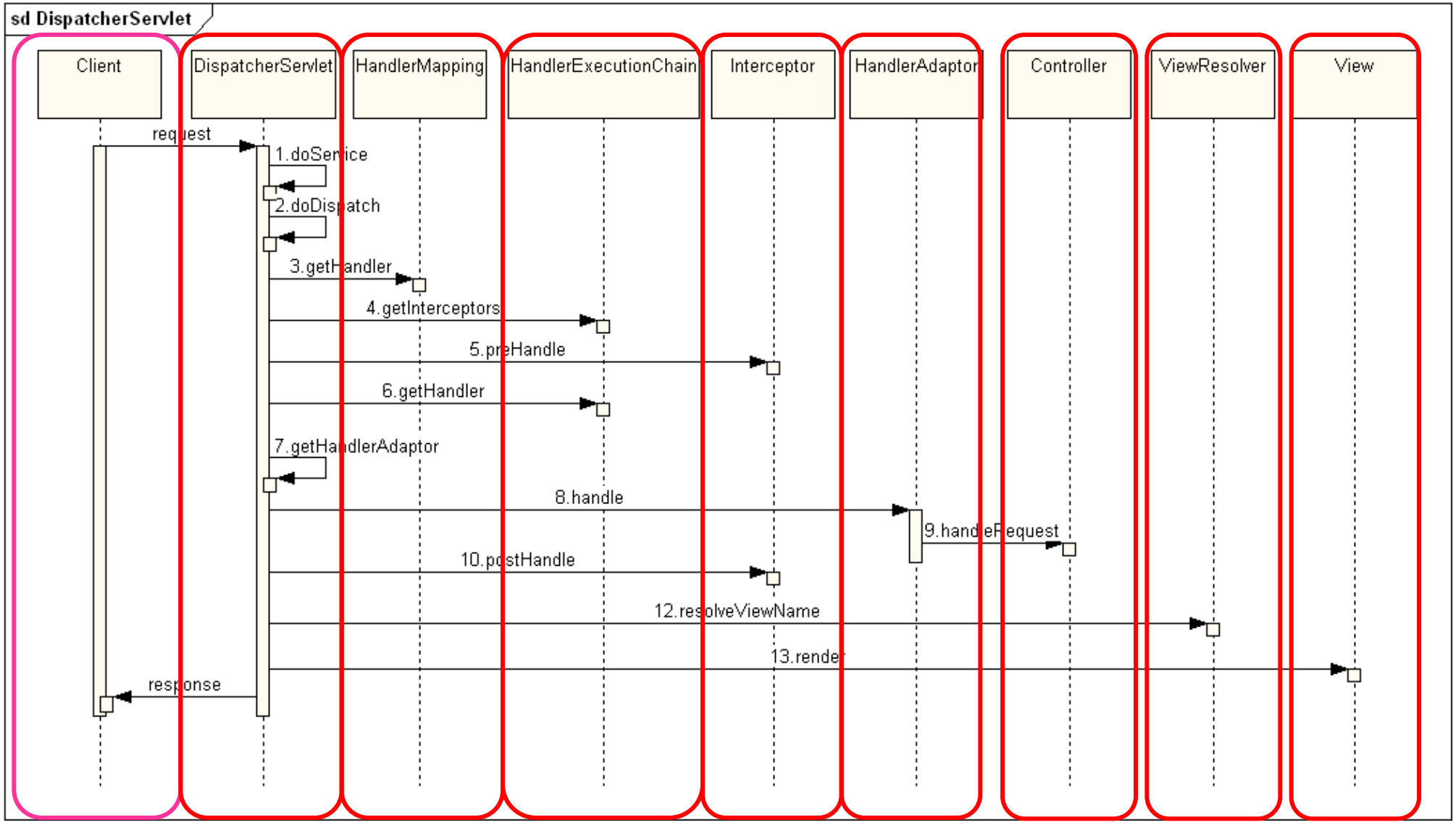
•ViewResolver는 view name을 받아 해당하는 View 객체를 return한다.

❑ DispatcherServlet

- **Controller로 향하는 모든 웹요청의 진입점**이며, 웹요청을 처리하며, 결과 데이터를 Client에게 응답 한다.
- **Spring MVC의 웹요청 Life Cycle을 주관**

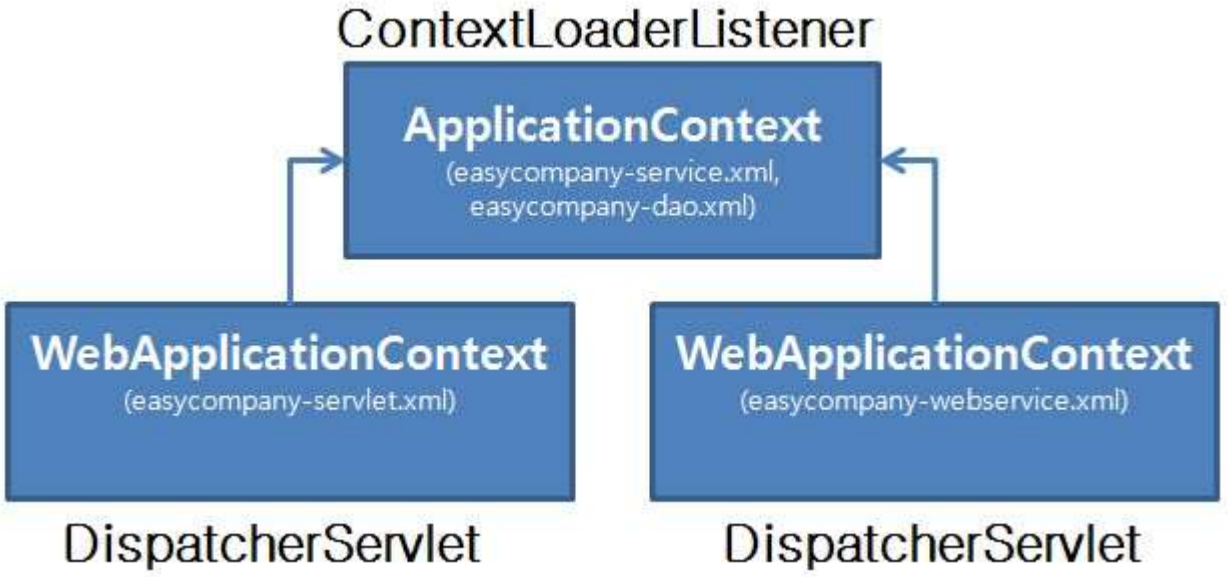


❑ Spring MVC의 웹요청 Life Cycle 을 주관하는 DispatcherServlet



❑ DispatcherServlet, ApplicationContext, WebApplicationContext

- 하나의 빈 설정파일에 모든 빈을 등록할 수도 있지만, 아래와 같이 **Layer 별로 빈파일을 나누어 등록하고 ApplicationContext, WebApplicationContext 사용하는것을 권장.**
- **ApplicationContext** : **ContextLoaderListener에 의해 생성. persistance, service layer의 빈**
- **WebApplicationContext** : **DispatcherServlet에 의해 생성. presentation layer의 빈**
- ContextLoaderListener는 웹 어플리케이션이 시작되는 시점에 ApplicationContext을 만들고, 이 ApplicationContext의 빈 정보는 모든 WebApplicationContext들이 참조할 수 있다.



❑ web.xml에 DispatcherServlet 설정하기

```
<!-- ApplicationContext 빈 설정 파일-->
  <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/config/service/easycompany-service.xml  <!--서비스 빈 정의-->
            /WEB-INF/config/service/easycompany-dao.xml  <!--Dao 빈 정의-->
        </param-value>
  </context-param>
  <listener>
     <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <!-- WebApplicationContext 빈 설정 파일-->
  <servlet>
     <servlet-name>servlet</servlet-name>
     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
     <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
           /WEB-INF/config/easycompany-servlet.xml  <!--web layer 관련 빈 선언-->
        </param-value>
     </init-param>
  </servlet>
  <!-- WebApplicationContext 빈 설정 파일-->
  <servlet>
     <servlet-name>webservice</servlet-name>
     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
     <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
           /WEB-INF/config/easycompany-webservice.xml
        </param-value>
     </init-param>
  </servlet>
```

LAB 301-mvc 실습 (1)

Exercise 1-1-1. "/hello.do" 에 동작하는 Controller 메소드 만들기

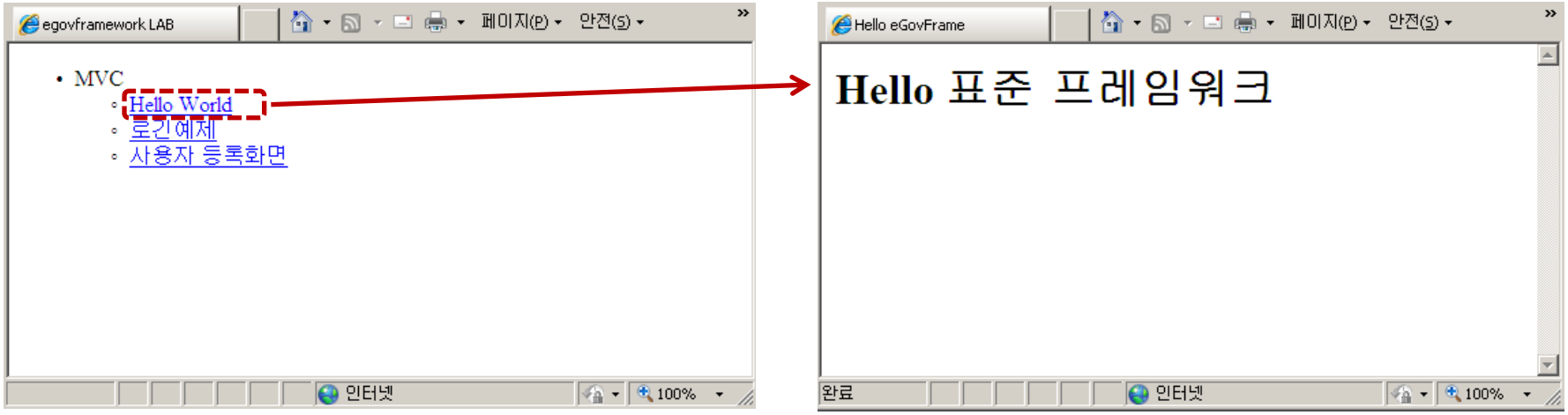
```
@RequestMapping(value = "/hello.do")
   public String helloworld() {
       return getViewName();
   }
```

Exercise 1-1-2. helloworld.jsp 만들기 (위치 : src₩main₩webapp₩WEB-INF₩jsp₩hello)

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Hello eGovFrame</title>
</head>
<body>
<h1>Hello 표준 프레임워크  </h1>
</body>
</html>
```

Exercise 1-1-3. 'Hello World' 예제 실행결과 확인

- 프로젝트 선택 마우스 우클릭 > Run As > Run On Server 실행
- 예제 실행 결과 확인 (http://127.0.0.1:8080/lab301-mvc/)



❑ @MVC

- **어노테이션을 이용한 설정** : XML 기반으로 설정하던 정보들을 어노테이션을 사용해서 정의한다.
- **유연해진 메소드 시그니쳐** : Controller 메소드의 파라미터와 리턴 타입을 좀 더 다양하게 필요에 따라 선택할 수 있다.
- **POJO-Style의 Controller** : Controller 개발시에 특정 인터페이스를 구현 하거나 특정 클래스를 상속해야할 필요가 없다. 하지만, 폼 처리, 다중 액션등 기존의 계층형 Controller가 제공하던 기능들을 여전히 쉽게 구현할 수 있다.
- **Bean 설정파일 작성** : @Controller만 스캔하도록 설정한다.

❑ <context:component-scan/> 설정

- **@Component, @Service, @Repository, @Controller 가 붙은 클래스들을 읽어들여 ApplicationContext, WebApplicationContext에 빈정보를 저장, 관리**한다.
- @Controller만 스캔하려면 <context:include-filter>나 <context:exclude-filter>를 사용해야 한다.

```
<?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"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation=http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd>

        <context:component-scan base-package="com.easycompany.controller.annotation">
            <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>
            <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Service"/>
            <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Repository"/>
        </context:component-scan>
</beans>
```

❑ DefaultAnnotationHandlerMapping

- @MVC 개발을 위한 HandlerMapping. 단 jdk1.5이상의 환경에서 사용가능.
- **@RequestMapping에 지정된 url과 해당 Controller의 메소드 매핑**
- 기본 HandlerMapping이므로 빈 설정 파일에 별도로 선언할 필요 없으나, 다른 HandlerMapping과 함께 사용한다면 선언해야 한다.

❑ SimpleUrlAnnotationHandlerMapping

- DefaultAnnotationHandlerMapping은 특정 url에 대해 interceptor를 적용할수 없음. -> 확장 HandlerMapping
- DefaultAnnotationHandlerMapping과 함께 사용. (order 프로퍼티를 SimpleUrlAnnotationHandlerMapping에 준다.)

```
<bean id="selectAnnotaionMapper"
   class="egovframework.rte.ptl.mvc.handler.SimpleUrlAnnotationHandlerMapping" p:order="1">
    <property name="interceptors">
       <list>
          <ref local="authenticInterceptor"/>
       </list>
    </property>
    <property name="urls">
        <list>
           <value>/admin/*.do</value>
           <value>/user/userInfo.do</value>
           <value>/development/**/code*.do</value>
        </list>
    </property>
</bean>
<bean id="annotationMapper"
class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" p:order="2"/>
```

❑ 관련 어노테이션

@Controller	해당 클래스가 Controller임을 나타내기 위한 어노테이션
@RequestMapping	요청에 대해 어떤 Controller, 어떤 메소드가 처리할지를 맵핑하기 위한 어노테이션
@RequestParam	Controller 메소드의 파라미터와 웹요청 파라미터와 맵핑하기 위한 어노테이션
@ModelAttribute	Controller 메소드의 파라미터나 리턴값을 Model 객체와 바인딩하기 위한 어노테이션
@SessionAttributes	Model 객체를 세션에 저장하고 사용하기 위한 어노테이션

❑ @Controller

- @MVC에서 Controller를 만들기 위해서는 작성한 클래스에 @Controller를 붙여주면 된다. 특정 클래스를 구현하거나 상속할 필요가 없다.

```
import org.springframework.stereotype.Controller;

@Controller
public class HelloController {
...
}
```

❑ @RequestMapping

- 요청에 대해 어떤 Controller, 어떤 메소드가 처리할지를 매핑하기 위한 어노테이션이다
- 관련속성

이름	타입	매핑 조건	설명
value	String[]	URL 값	- @RequestMapping(value="/hello.do") - @RequestMapping(value={"/hello.do", "/world.do" }) - @RequestMapping("/hello.do") - Ant-Style 패턴매칭 이용 : "/myPath/*.do"
method	Request Method[]	HTTP Request 메소드값	- @RequestMapping(method = RequestMethod.POST) - 사용 가능한 메소드 : GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE
params	String[]	HTTP Request 파라미터	- params="myParam=myValue" : HTTP Request URL중에 myParam이라는 파라미터가 있어야 하고 값은 myValue이어야 맵핑 - params="myParam" : 파라미터 이름만으로 조건을 부여 - "!myParam" : myParam이라는 파라미터가 없는 요청 만을 맵핑 - @RequestMapping(params={"myParam1=myValue", "myParam2", "!myParam3"})와 같이 조건을 주었다면, HTTP Request에는 파라미터 myParam1이 myValue값을 가지고 있고, myParam2 파라미터가 있어야 하고, myParam3라는 파라미터는 없어야함.

❑ @RequestMapping 설정

- @RequestMapping은 클래스 단위(type level)나 메소드 단위(method level)로 설정할 수 있다.

type level

/hello.do 요청이 오면 HelloController의 hello 메소드가 수행된다.

type level에서 URL을 정의하고 Controller에 메소드가 하나만 있어도 요청 처리를 담당할 메소드 위에 @RequestMapping 표기를 해야 제대로 맵핑이 된다.

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/hello.do")
public class HelloController {

    @RequestMapping
    public String hello(){
        ...
    }
}
```

method level

/hello.do 요청이 오면 hello 메소드,

/helloForm.do 요청은 GET 방식이면 helloGet 메소드, POST 방식이면 helloPost 메소드가 수행된다.

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HelloController {

	@RequestMapping(value="/hello.do")
	public String hello(){
	    ...
	}

	@RequestMapping(value="/helloForm.do", method = RequestMethod.GET)
	public String helloGet(){
	    ...
	}

	@RequestMapping(value="/helloForm.do", method = RequestMethod.POST)
	public String helloPost(){
	    ...
	}
}
```

type + method level

type level, method level 둘 다 설정할 수도 있는데,

이 경우엔 type level에 설정한 @RequestMapping의 value(URL)를 method level에서 재정의 할수 없다.

/hello.do 요청시에 GET 방식이면 helloGet 메소드, POST 방식이면 helloPost 메소드가 수행된다.

```
@Controller
@RequestMapping("/hello.do")
public class HelloController {

	@RequestMapping(method = RequestMethod.GET)
	public String helloGet(){
		...
	}

	@RequestMapping(method = RequestMethod.POST)
	public String helloPost(){
		...
	}
}
```

❑ @RequestParam

- @RequestParam은 Controller 메소드의 파라미터와 웹요청 파라미터와 맵핑하기 위한 어노테이션이다.
- 관련 속성

이름	타입	설명
value	String	파라미터 이름
required	boolean	해당 파라미터가 반드시 필수 인지 여부. 기본값은 true이다.

- 해당 파라미터가 Request 객체 안에 없을때 그냥 null값을 바인드 하고 싶다면, 아래 예제의 pageNo 파라미터 처럼 required=false로 명시해야 한다.
- name 파라미터는 required가 true이므로, 만일 name 파라미터가 null이면 org.springframework.web.bind.MissingServletRequestParameterException이 발생한다.

```
@Controller
public class HelloController {

    @RequestMapping("/hello.do")
    public String hello(@RequestParam("name") String name,
                        @RequestParam(value="pageNo", required=false) String pageNo){
        ...
    }
}
```

❑ @ModelAttribute

- @ModelAttribute은 Controller에서 2가지 방법으로 사용된다.

  1. Model 속성(attribute)과 메소드 파라미터의 바인딩.
  2. 입력 폼에 필요한 참조 데이터(reference data) 작성. - SimpleFormContrller의 referenceData 메소드와 유사한 기능.

- 관련 속성

이름	타입	설명
value	String	바인드하려는 Model 속성 이름.

❑ @SessionAttributes

- @SessionAttributes는 model attribute를 session에 저장, 유지할 때 사용하는 어노테이션이다.
- @SessionAttributes는 클래스 레벨(type level)에서 선언할 수 있다.
- 관련 속성

이름	타입	설명
value	String[]	session에 저장하려는 model attribute의 이름
required	Class[]	session에 저장하려는 model attribute의 타입

❑ @Controller 메소드 시그니쳐

- 기존의 계층형 Controller(SimpleFormController, MultiAction..)에 비해 유연한 메소드 파라미터, 리턴값을 갖는다.

❑ 메소드 파라미터

- Servlet API - ServletRequest, HttpServletRequest, HttpServletResponse, HttpSession 같은 요청,응답,세션관련 Servlet API.
- org.springframework.web.context.request.WebRequest, org.springframework.web.context.request.NativeWebRequest
- java.util.Locale
- java.io.InputStream / java.io.Reader
- java.io.OutputStream / java.io.Writer
- **@RequestParam** - HTTP Request의 파라미터와 메소드의 argument를 바인딩하기 위해 사용하는 어노테이션.
- java.util.**Map** / org.springframework.ui.**Model** / org.springframework.ui.**ModelMap** - 뷰에 전달할 모델데이터.
- **Command/form 객체** - HTTP Request로 전달된 parameter를 바인딩한 커맨드 객체, @ModelAttribute을 사용하면 alias를 줄수 있다.
- org.springframework.validation.Errors / org.springframework.validation.**BindingResult** - 유효성 검사 후 결과 데이터를 저장한 객체.
- org.springframework.web.bind.support.**SessionStatus** - 세션폼 처리시에 해당 세션을 제거하기 위해 사용된다.

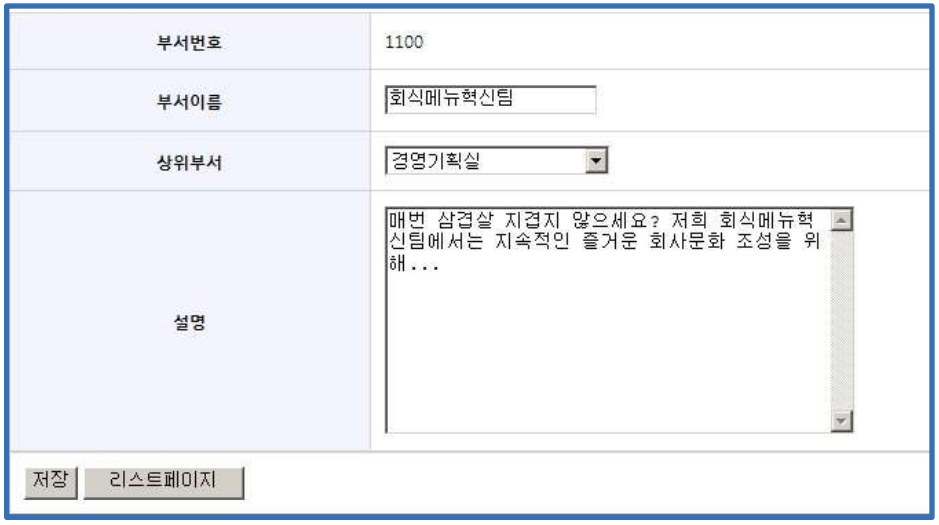
❑ 메소드 리턴 타입

- **ModelAndView** - 커맨드 객체, @ModelAttribute 적용된 메소드의 리턴 데이터가 담긴 Model 객체와 View 정보가 담겨 있다.
- **Model(또는 ModelMap)** - 커맨드 객체, @ModelAttribute 적용된 메소드의 리턴 데이터가 Model 객체에 담겨 있다. View 이름은 RequestToViewNameTranslator가 URL을 이용하여 결정한다.
- **Map** - 커맨드 객체, @ModelAttribute 적용된 메소드의 리턴 데이터가 Map 객체에 담겨 있으며, View 이름은 역시 RequestToViewNameTranslator가 결정한다
- **String** - 리턴하는 String 값이 곧 View 이름이 된다. 커맨드 객체, @ModelAttribute 적용된 메소드의 리턴 데이터가 Model(또는 ModelMap)에 담겨 있다. 리턴할 Model(또는 ModelMap)객체가 해당 메소드의 argument에 선언되어 있어야 한다
- **void** - 메소드가 ServletResponse / HttpServletResponse등을 사용하여 직접 응답을 처리하는 경우이다. View 이름은 RequestToViewNameTranslator가 결정한다.

❑ @Controller 로 폼처리 구현하기

- 부서정보를 수정하고 저장하는 폼페이지를 @Controller로 구현해 보자. 메소드의 이름은 폼처리를 담당하는 기존의 Form Controller인 SimpleFormController와의 비교를 위해 기능별로 동일하게 지었다.

❑ 화면 & 시나리오



1. 파라미터 부서번호의 해당 부서정보 데이터를 불러와 입력폼을 채운다.
2. 상위부서(selectbox)는 부서정보 데이터와는 별도로, 상위부서에 해당하는 부서리스트 데이터을 구해서 참조데이터로 구성한다.
3. 사용자가 데이터 수정을 끝내고 저장 버튼을 누르면 수정 데이터로 저장을 담당하는 서비스(DB)를 호출한다.
4. 저장이 성공하면 부서리스트 페이지로 이동하고 에러가 있으면 다시 입력폼페이지로 이동한다.

❑ Controller 작성하기

```
package com.easycompany.controller.annotation;
...
@Controller
public class UpdateDepartmentController {

	@Autowired
	private DepartmentService departmentService;

	//상위부서(selectbox)는 부서정보 데이터와는 별도로, 상위부서에 해당하는 부서리스트 데이터을 구해서 참조데이터로 구성한다.
	@ModelAttribute("deptInfoOneDepthCategory")
	public Map<String, String> referenceData() {
		Map<String, String> param = new HashMap<String, String>();
		param.put("depth", "1");
		return departmentService.getDepartmentIdNameList(param);
	}

	// 해당 부서번호의 부서정보 데이터를 불러와 입력폼을 채운다
	@RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
	public String formBackingObject(@RequestParam("deptid") String deptid, Model model) {
		Department department = departmentService.getDepartmentInfoById(deptid);
		model.addAttribute("department", department);
		return "modifydepartment";
	}

	//사용자가 데이터 수정을 끝내고 저장 버튼을 누르면 수정 데이터로 저장을 담당하는 서비스(DB)를 호출한다.
	//저장이 성공하면 부서리스트 페이지로 이동하고 에러가 있으면 다시 입력폼페이지로 이동한다.
	@RequestMapping(value = "/updateDepartment.do", method = RequestMethod.POST)
	public String onSubmit(@ModelAttribute("department") Department department) {
		try {
			departmentService.updateDepartment(department);
			return "redirect:/departmentList.do?depth=1";
		} catch (Exception e) {
			return "modifydepartment";
		}
	}
}
```

❑ JSP

- 폼 필드와 모델 데이터의 편리한 데이터 바인딩을 위해 스프링 폼 태그를 사용한다.
- commandName에는 model attribute를 적어주면 된다. "department"

```
...
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<form:form commandName="department">
    <table>
      <tr>
        <th>부서번호</th><td><c:out value="${department.deptid}"/></td>
      </tr>
      <tr>
        <th>부서이름</th><td><form:input path="deptname" size="20"/></td>
      </tr>
      <tr>
        <th>상위부서</th>
        <td>
              <form:select path="superdeptid">
                  <option value="">상위부서를 선택하세요.</option>
                  <form:options items="${deptInfoOneDepthCategory}" />
              </form:select>
        </td>
      </tr>
      <tr>
        <th>설명</th><td><form:textarea path="description" rows="10" cols="40"/></td>
      </tr>
    </table>
...
    <input type="submit" value="저장"/>
    <input type="button" value="리스트페이지" onclick="location.href='/easycompany/departmentList.do?depth=1'"/>
...
</form:form>
```

LAB 301-mvc 실습 (2)

Exercise 1-2-1. context-servlet.xml 설정 변경하기 : messageSource 활성화

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

Exercise 1-2-2. LoginController.java 메소드 추가하기

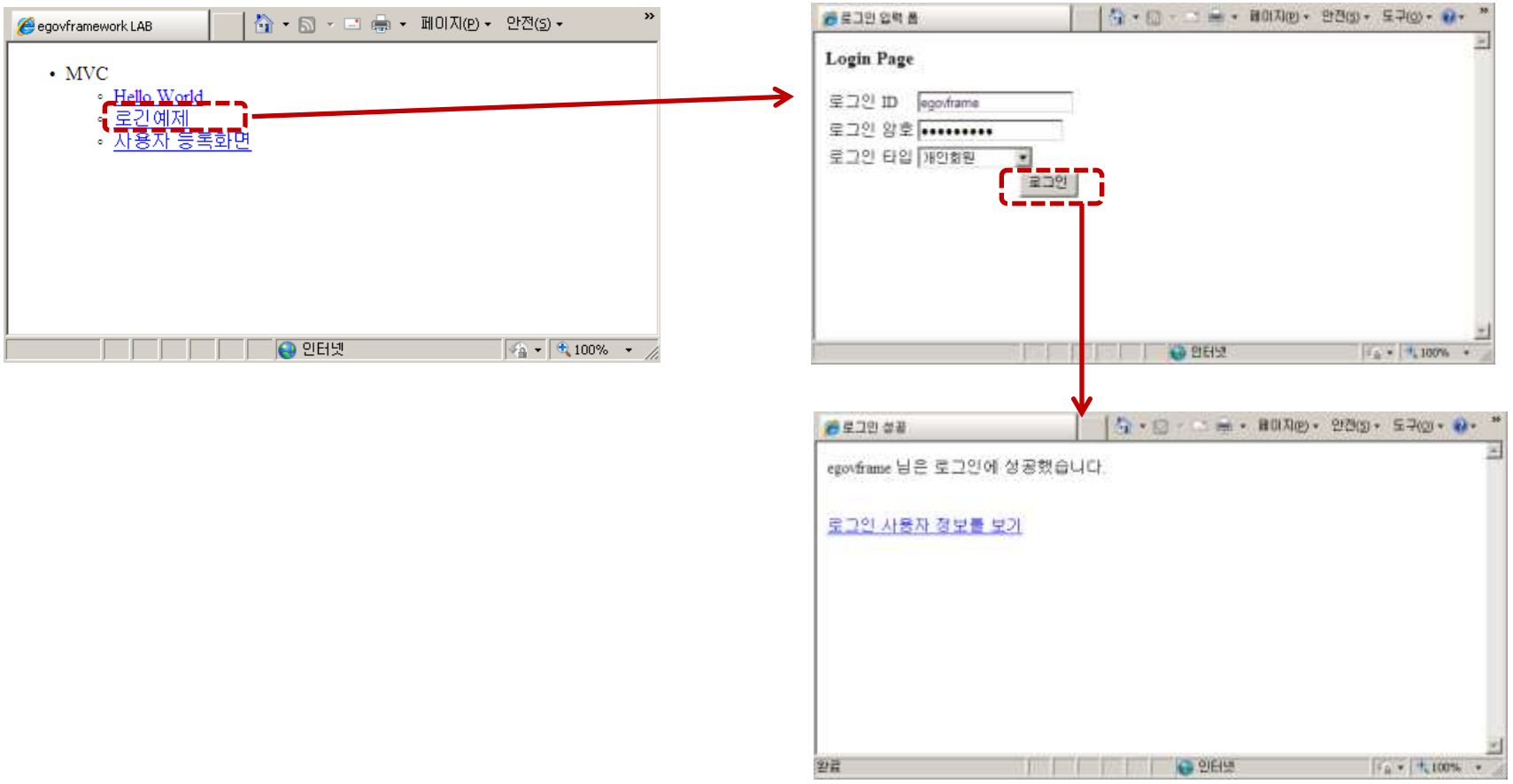
```
@RequestMapping(value = "/loginProcess1.do", method = RequestMethod.GET)
public String loginFormSetUp() {
    return getFormView();
}

@RequestMapping(value = "/loginProcess1.do", method = RequestMethod.POST)
public String loginProcess(@ModelAttribute("login") LoginCommand loginCommand) {

    return getSuccessView();
}

@ModelAttribute("loginTypes")
protected List<LoginType> referenceData() throws Exception {
    List<LoginType> loginTypes = new ArrayList<LoginType>();
    loginTypes.add(new LoginType("A", "개인회원"));
    loginTypes.add(new LoginType("B", "기업회원"));
    loginTypes.add(new LoginType("C", "관리자"));
    return loginTypes;
}

@ModelAttribute("login")
protected Object referenceData4login() throws Exception {
    return new LoginCommand();
}
```

Exercise 1-2-3. LoginCommand.java 완성하기

```
private String id;
private String password;
private String loginType;

public String getId() {
return id;
}

public void setId(String id) {
this.id = id;
}

public String getPassword() {
return password;
}

public void setPassword(String password) {
this.password = password;
}

public String getLoginType() {
return loginType;
}

public void setLoginType(String loginType) {
this.loginType = loginType;
}
```

Exercise 1-2-4. . 'Hello World' 예제 실행결과 확인

- 프로젝트 선택 마우스 우클릭 > Run As > Run On Server 실행
- 예제 실행 결과 확인 (http://127.0.0.1:8080/lab301-mvc/)



❑ Spring Framework API

- http://static.springsource.org/spring/docs/2.5.x/api/index.html
- http://static.springsource.org/spring/docs/3.0.x/javadoc-api/

❑ The Spring Framework - Reference Documentation

- http://static.springsource.org/spring/docs/2.5.x/reference/spring-web.html
- http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/spring-web.html