

전자정부 표준프레임워크 실행환경(화면처리)



Contents

1. _ 개요
2. _ MVC
3. _ Internationalization
4. _ Validation
5. _ Ajax Support
6. _ UI Adaptor

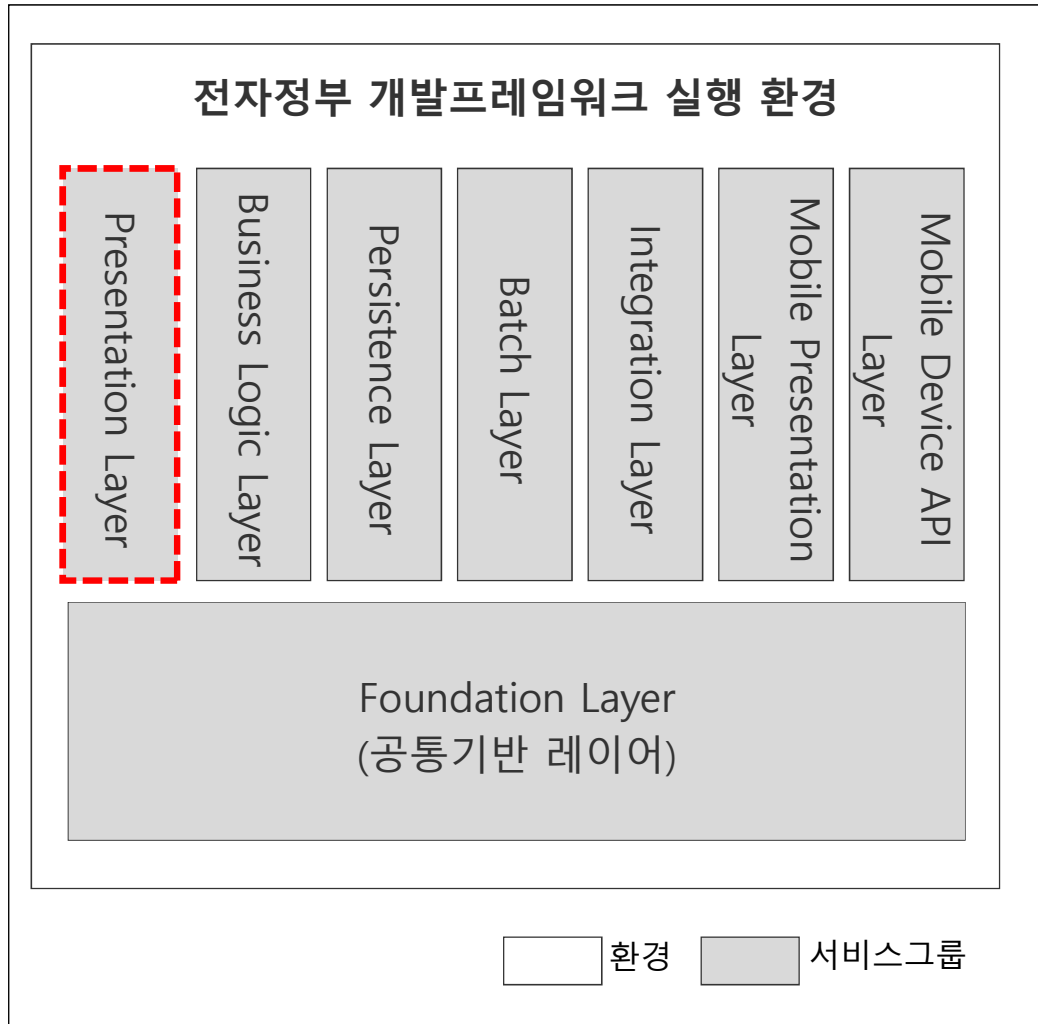


실행환경(화면처리)

개요

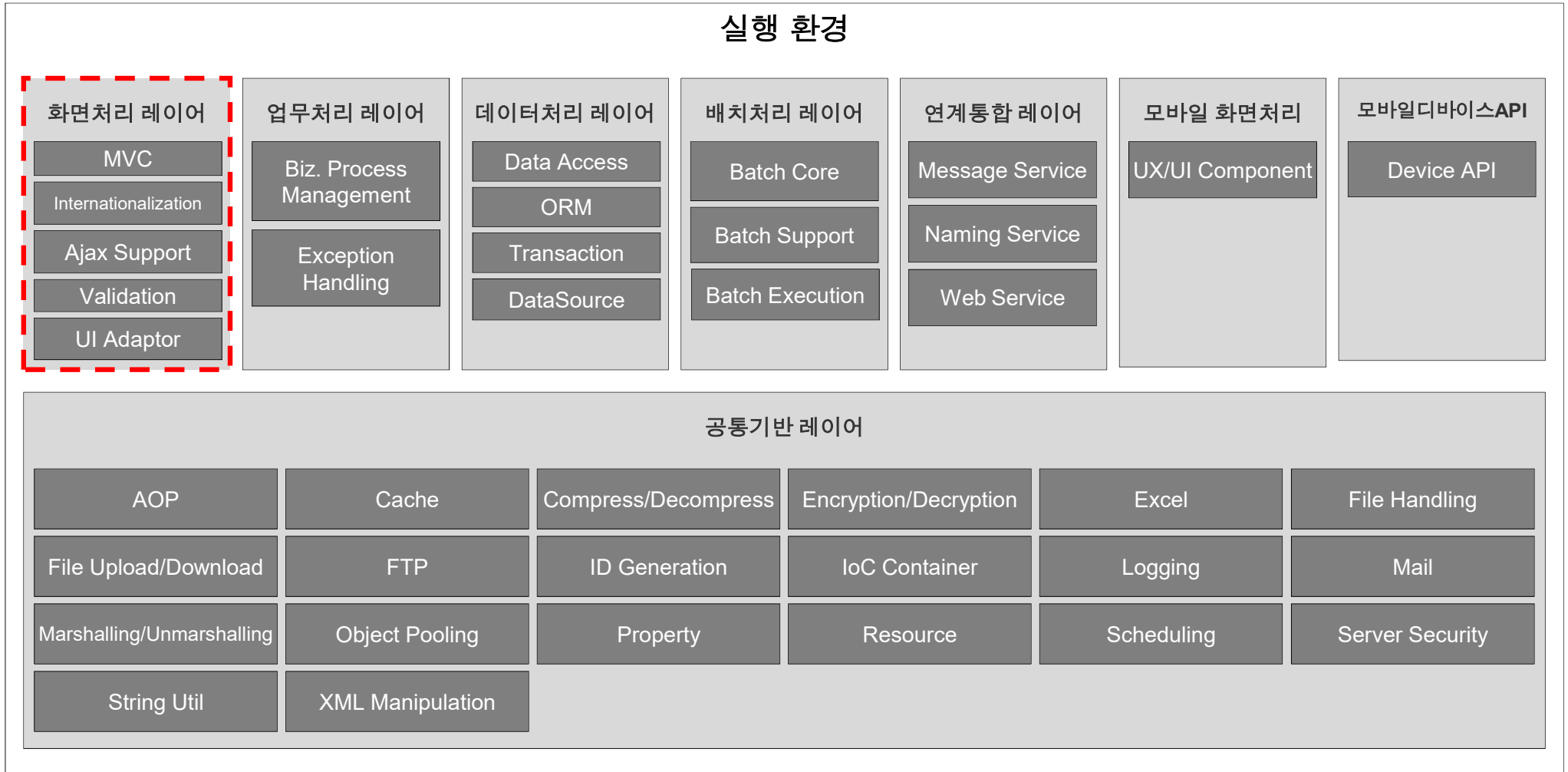
1. 실행환경 화면처리 레이어

- 화면처리 레이어는 **업무 프로그램과 사용자 간의 Interface**를 담당하는 **Layer**로서, 사용자 화면 구성, 사용자 입력 정보 검증 등의 기능 제공



서비스 그룹	설명
Presentation Layer	<ul style="list-style-type: none"> 업무 프로그램과 사용자 간의 Interface를 담당하는 Layer로서, 사용자 화면 구성, 사용자 입력 정보 검증 등의 기능을 제공함
Business Logic Layer	<ul style="list-style-type: none"> 업무 프로그램의 업무 로직을 담당하는 Layer로서, 업무 흐름 제어, 에러 처리 등의 기능을 제공함
Persistence Layer	<ul style="list-style-type: none"> 데이터베이스에 대한 연결 및 영속성 처리, 선언적인 트랜잭션 관리를 제공하는 Layer임
Batch Layer	<ul style="list-style-type: none"> 대용량 데이터 처리를 위한 기반 환경을 제공하는 Layer임
Integration Layer	<ul style="list-style-type: none"> 타 시스템과의 연동 기능을 제공하는 Layer임
Foundation Layer	<ul style="list-style-type: none"> 실행 환경의 각 Layer에서 공통적으로 사용하는 공통 기능을 제공함

- 화면처리 레이어는 MVC, Ajax Support 등 총 5개의 서비스를 제공함



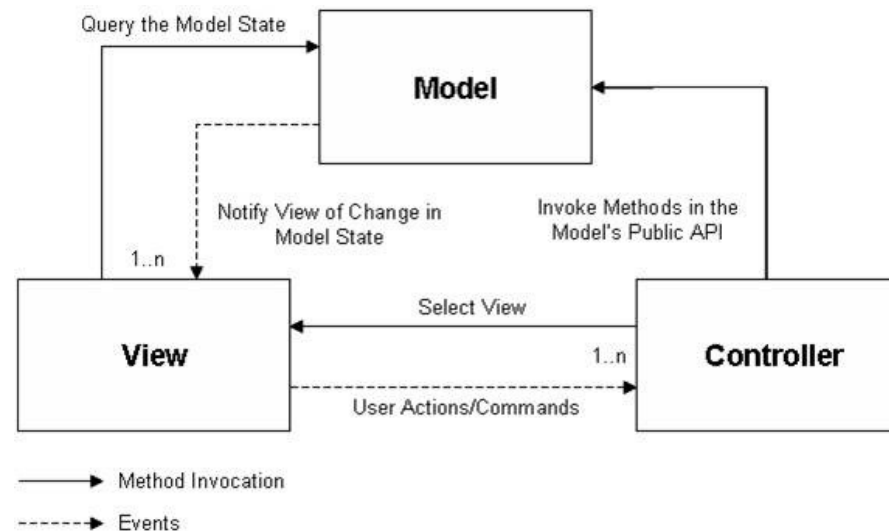
실행환경(화면처리)

MVC

1. 개요
2. Spring MVC Architecture
3. DispatcherServlet
4. @MVC
5. HandlerMapping
6. @Controller 관련 어노테이션
7. @Controller 메소드 시그니처
8. @MVC 예제

□ 서비스 개요

- MVC(Model-View-Controller) 패턴은 코드를 기능에 따라 Model, View, Controller 3가지 요소로 분리한다.
 - **Model** : 어플리케이션의 데이터와 비즈니스 로직을 담는 객체이다.
 - **View** : Model의 정보를 사용자에게 표시한다. 하나의 Model을 다양한 View에서 사용할 수 있다.
 - **Controller** : Model과 View의 중계역할. 사용자의 요청을 받아 Model에 변경된 상태를 반영하고, 응답을 위한 View를 선택한다.
- MVC 패턴은 UI 코드와 비즈니스 코드를 분리함으로써 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 MVC**를 채택하였다.
 - Framework내의 특정 클래스를 상속하거나, 참조, 구현해야 하는 등의 제약사항이 비교적 적다.
 - IOC Container가 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 비즈니스 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.

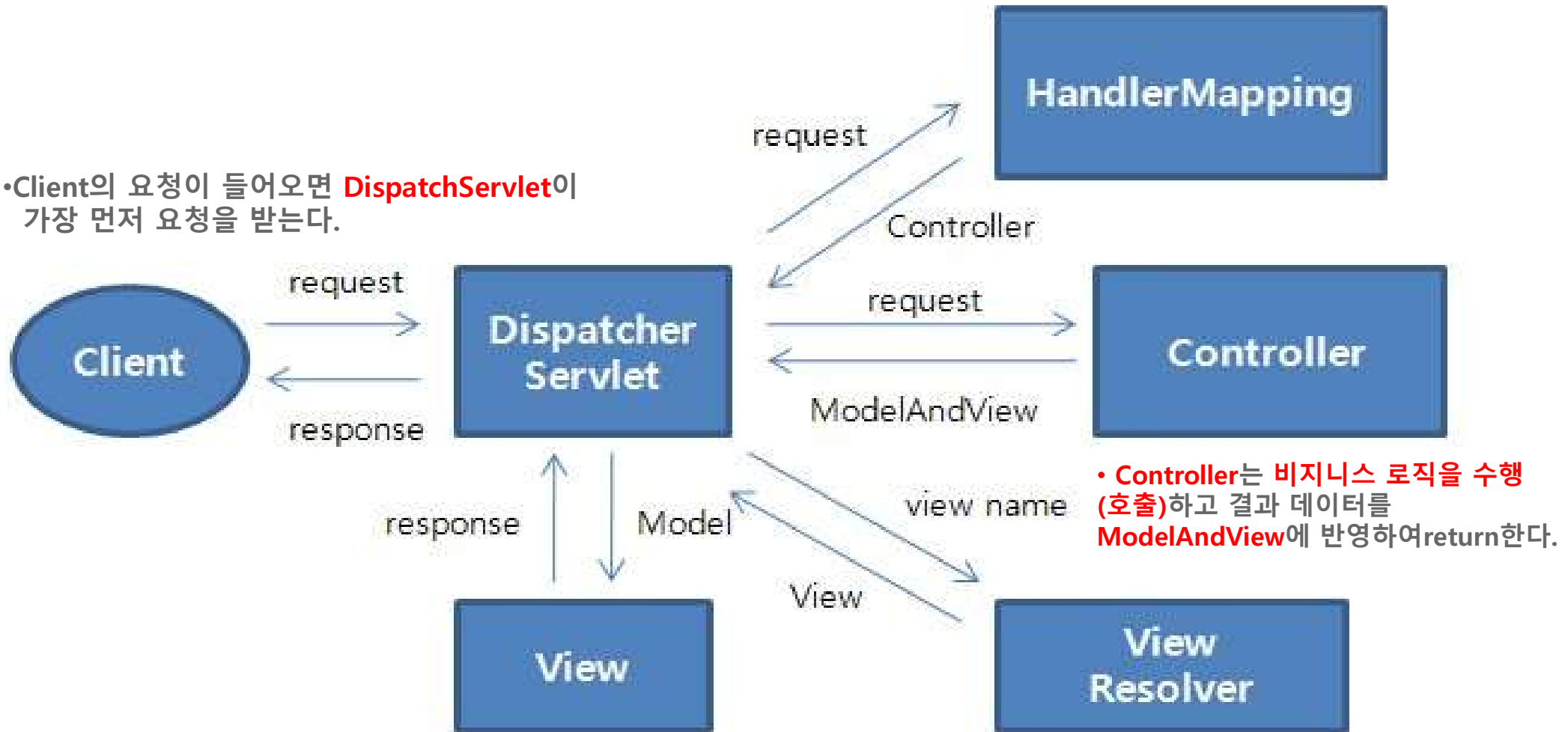
□ Spring MVC의 핵심 Component

- **DispatcherServlet**
 - Spring MVC Framework의 Front Controller, 웹요청과 응답의 Life Cycle을 주관한다.
- **HandlerMapping**
 - 웹요청시 해당 URL을 어떤 Controller가 처리할지 결정한다.
- **Controller**
 - 비즈니스 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

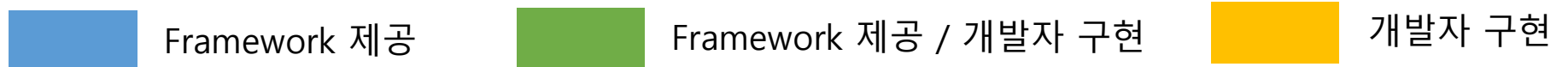
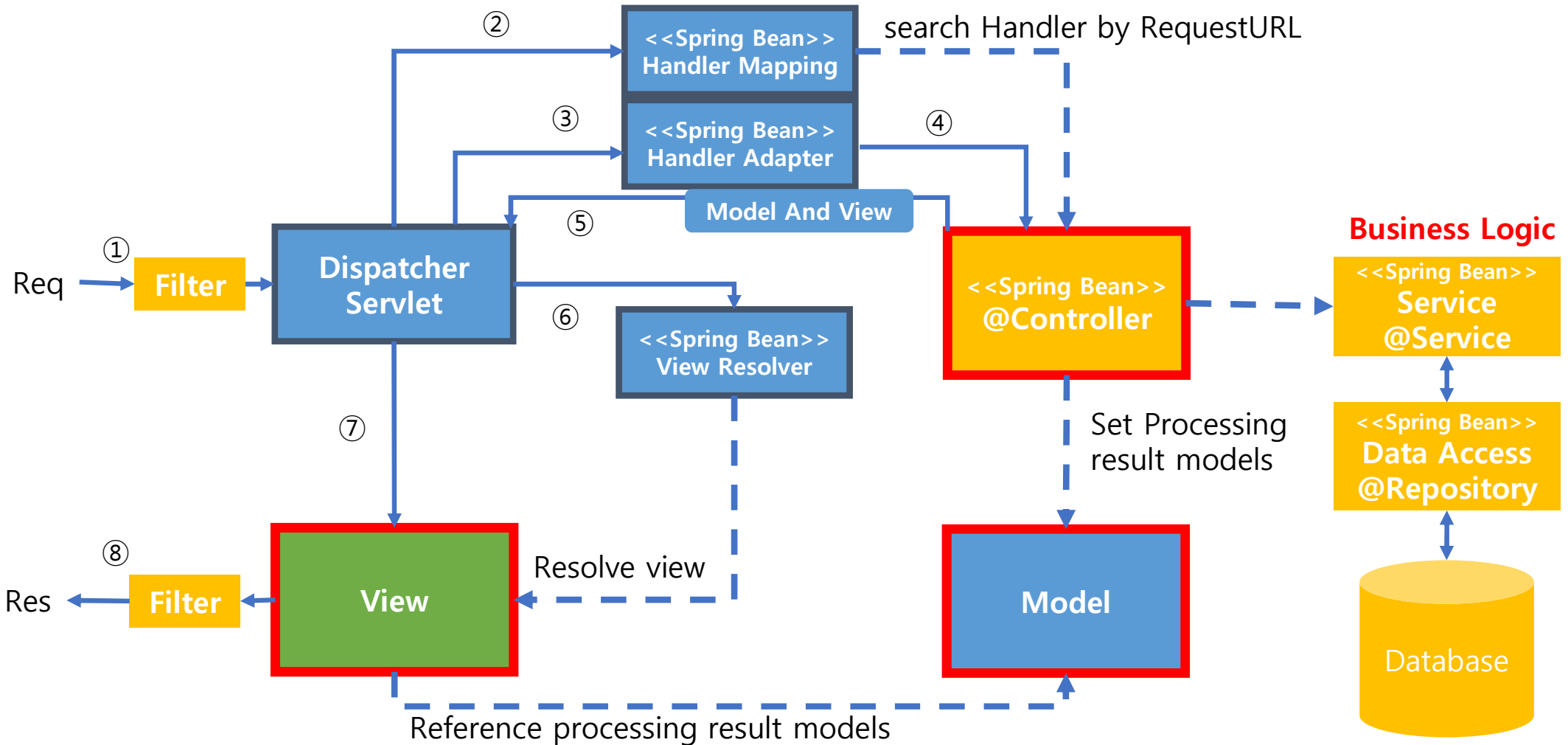
• **HandlerMapping**이 요청에 해당하는 **Controller**를 return한다.

• Client의 요청이 들어오면 **DispatcherServlet**이 가장 먼저 요청을 받는다.



• **View**는 Model 객체를 받아 rendering한다.

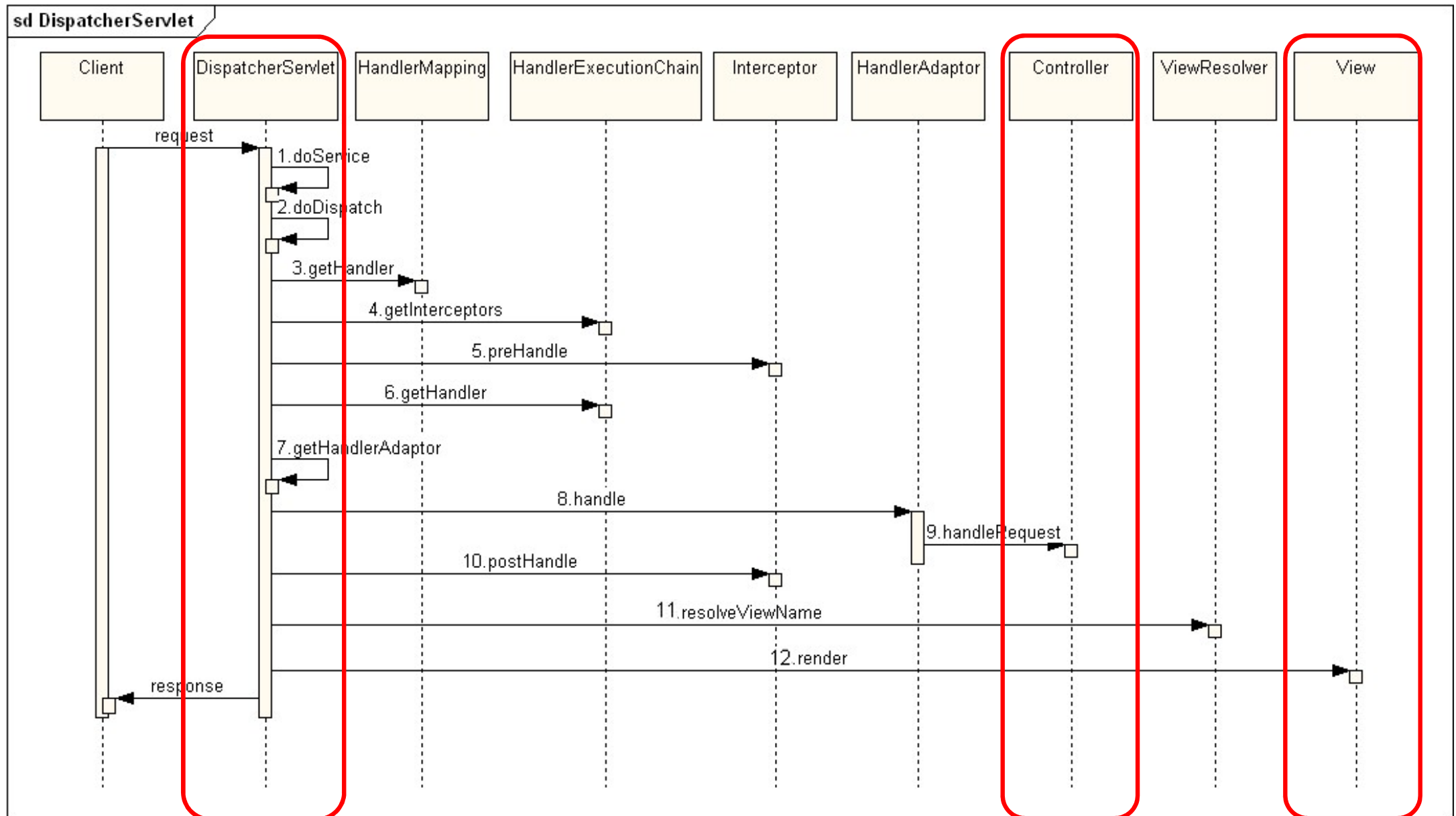
• **ViewResolver**는 view name을 받아 해당하는 **View** 객체를 return한다.



2. MVC - DispatcherServlet(1/5)

4. 화면처리 레이어

❑ Spring MVC의 웹요청 Life Cycle 을 주관하는 DispatcherServlet



□ Spring MVC의 웹요청 Life Cycle 을 주관하는 DispatcherServlet

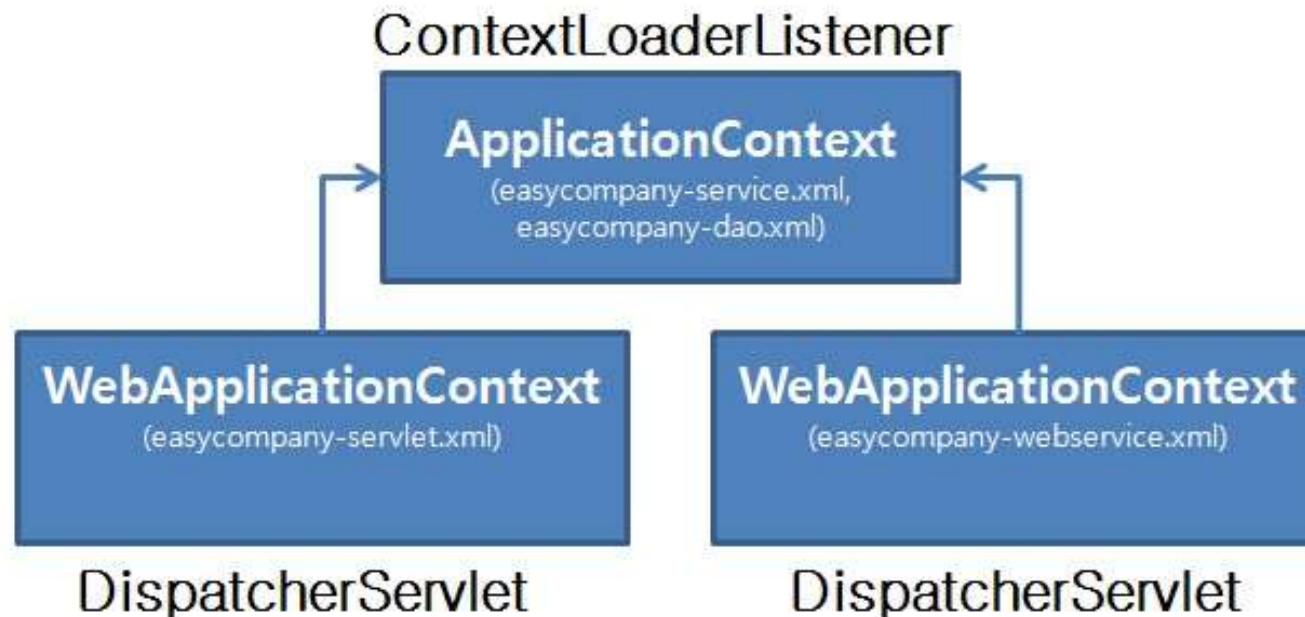
- 1. **doService** 메소드에서부터 웹요청의 처리가 시작된다. DispatcherServlet에서 사용되는 몇몇 정보를 request 객체에 담는 작업을 한 후 doDispatch 메소드를 호출한다.
- 2. 아래 3번~12번 작업이 **doDispatch** 메소드안에 있다. Controller, View 등의 컴포넌트들을 이용한 실제적인 웹요청처리가 이루어 진다.
- 3. **getHandler** 메소드는 RequestMapping 객체를 이용해서 요청에 해당하는 Controller를 얻게 된다.
- 4. 요청에 해당하는 Handler를 찾았다면 Handler를 HandlerExecutionChain 객체에 담아 리턴하는데, 이때 HandlerExecutionChain은 요청에 해당하는 interceptor들이 있다면 함께 담아 리턴한다.
- 5. 실행될 interceptor들이 있다면 interceptor의 preHandle 메소드를 차례로 실행한다.
- 6. Controller의 인스턴스는 HandlerExecutionChain의 **getHandler** 메소드를 이용해서 얻는다.
- 7. HandlerMapping과 마찬가지로 여러개의 HandlerAdaptor를 설정할 수 있는데, **getHandlerAdaptor** 메소드는 Controller에 적절한 HandlerAdaptor 하나를 리턴한다.
- 8. 선택된 HandlerAdaptor의 handle 메소드가 실행되는데, 실제 실행은 파라미터로 넘겨 받은 Controller를 실행한다.

□ Spring MVC의 웹요청 Life Cycle 을 주관하는 DispatcherServlet

- 9. 계층형 Controller인 경우는 handleRequest 메소드가 실행된다. @Controller인 경우는 HanlderAdaptor(AnnotationMethodHandlerAdapter)가 HandlerMethodInvoker를 이용해 실행할 Controller의 메소드를 invoke()한다.
- 10. interceptor의 postHandle 메소드가 실행된다.
- 11. resolveViewName 메소드는 논리적 뷰 이름을 가지고 해당 View 객체를 반환한다.
- 12. Model 객체의 데이터를 보여주기 위해 해당 View 객체의 render 메소드가 수행된다.

□ DispatcherServlet, ApplicationContext, WebApplicationContext

- 하나의 빈 설정파일에 모든 빈을 등록할 수도 있지만, 아래와 같이 **Layer 별로 빈파일을 나누어 등록하고 ApplicationContext, WebApplicationContext 사용하는것을 권장.**
- **ApplicationContext** : **ContextLoaderListener**에 의해 생성. **persistance, service layer의 빈**
- **WebApplicationContext** : **DispatcherServlet**에 의해 생성. **presentation layer의 빈**
- ContextLoaderListener는 웹 어플리케이션이 시작되는 시점에 ApplicationContext를 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)

Hello world 예제

□ @MVC

- **어노테이션을 이용한 설정** : XML 기반으로 설정하던 정보들을 어노테이션을 사용해서 정의한다.
- **유연해진 메소드 시그니처** : 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 읽어들이어 ApplicationContext, WebApplicationContext에 빈정보를 저장, 관리한다.
- @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-4.0.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.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>
```

❑ RequestMappingHandlerMapping (DefaultAnnotationHandlerMapping는 deprecated)

- @MVC 개발을 위한 HandlerMapping. 표준프레임워크 3.0(Spring 3.2.9)이상 에서 사용가능.
- 기존 DefaultAnnotationHandlerMapping이 deprecated되면서 대체됨.
- **@RequestMapping에 지정된 url과 해당 Controller의 메소드 매핑**
- RequestMappingHandlerMapping은 기본 HandlerMapping이며, 선언하기 위해서는 다음과 같이 세가지 방법이 있다.
- 선언하지 않는 방법 : 기본 HandlerMapping이므로 지정하지 않아도 사용가능하다.
- <mvc:annotation-driven/>을 선언하는 방법 : @MVC사용 시 필요한 빈들을 등록해주는 <mvc:annotation-driven/>을 설정하면 내부에서 RequestMappingHandlerMapping, RequestMappingHandlerAdapter 이 구성된다.
- RequestMappingHandlerMapping을 직접 선언하는 방법 : 다른 HandlerMapping과 함께 사용할 때 선언한다.

```
<?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-4.0.xsd http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.0.xsd" >

  <context:component-scan base-package="org.mycode.controller" />
  <!--명시적 선언-->
  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>
</beans>
```

❑ SimpleUrlAnnotationHandlerMapping(표준프레임워크3.0부터deprecated됨 mvc 태그로 변경)

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

표준프레임워크 2.7 이하

```
<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"/>
```

표준프레임워크 3.0 이상

```
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/admin/*.do"/>
    <mvc:mapping path="/user/userInfo.do"/>
    <mvc:mapping path="/development/**/code*.do"/>
    <mvc:mapping path="/egov/**"/>
    <mvc:exclude-mapping path="/egov/admin/**"/>
    <bean class="egov.interceptors.AuthenticInterceptor"/>
  </mvc:interceptor>
</mvc:interceptors>
```

□ 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
@CommandMap	Controller메소드의 파라미터를 Map형태로 받을 때 웹요청 파라미터와 맵핑하기 위한 어노테이션(egov 3.0부터 추가)

□ @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	<ul style="list-style-type: none"> - @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	<ul style="list-style-type: none"> - @RequestMapping(method = RequestMethod.POST) - 사용 가능한 메소드 : GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE
params	String[]	HTTP Request 파라미터	<ul style="list-style-type: none"> - params="myParam=myValue" : HTTP Request URL중에 myParam이라는 파라미터가 있어야 하고 값은 myValue이어야 매핑 - params="myParam" : 파라미터 이름만으로 조건을 부여 - "!myParam" : myParam이라는 파라미터가 없는 요청 만을 매핑 - @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 매핑하기 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

❑ @CommandMap(실행환경 3.2부터 deprecated됨 @RequestParam으로 대체)

- 실행환경 3.0부터 추가되었으며 Controller에서 Map형태로 웹요청 값을 받았을 때 다른 Map형태의 argument와 구분해 주기 위한 어노테이션이다.
- @CommandMap은 파라미터 레벨(type level)에서 선언할 수 있다.
- 사용 방법은 다음과 같다.

```
@RequestMapping("/test.do")
public void test(@CommandMap Map<String, String> commandMap, HttpServletRequest request){
    //생략
}
```

- CommandMap을 이용하기 위해서는 반드시 EgovRequestMappingHandlerAdapter와 함께 AnnotationCommandMapArgumentResolver를 등록해 주어야 한다.

```
<bean class="egovframework.rte.ptl.mvc.bind.annotation.EgovRequestMappingHandlerAdapter">
    <property name="customArgumentResolvers">
        <list>
            <bean class="egovframework.rte.ptl.mvc.bind.AnnotationCommandMapArgumentResolver" />
        </list>
    </property>
</bean>
```


□ @Controller 메소드 시그니처

- 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

부서번호	1100
부서이름	<input type="text" value="회식메뉴혁신팀"/>
상위부서	<input type="text" value="경영기획실"/>
설명	<div>매번 삼겹살 지겹지 않으세요? 저희 회식메뉴혁신팀에서는 지속적인 즐거운 회사문화 조성을 위해...</div>
<div>저장 리스트페이지</div>	

1. 파라미터 부서번호의 해당 부서정보 데이터를 불러와 입력폼을 채운다.
2. 상위부서(selectbox)는 부서정보 데이터와는 별도로, 상위부서에 해당하는 부서리스트 데이터를 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 데이터를 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)

로그인 예제

❑ Spring Framework API

- <http://docs.spring.io/spring/docs/3.2.x/javadoc-api/>
- 이전 버전 참조
 - <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://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/spring-web.html>
- 이전 버전 참조
 - <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>

실행환경(화면처리)

Internationalization(국제화)

1. 개요
2. 설명

□ 서비스 개요

- ApplicationContext의 **MessageSource** 인터페이스를 통해 다국어 지원을 도와준다.
- 전자정부 표준 프레임워크에서는 Spring MVC 에서 제공하는 **LocaleResolver**를 이용한다.
- Spring MVC 는 다국어를 지원하기 위하여 아래와 같은 종류의 LocaleResolver 를 제공하고 있다.
 - **CookieLocaleResolver** : 쿠키를 이용한 locale정보 사용
 - **SessionLocaleResolver** : 세션을 이용한 locale정보 사용
 - **AcceptHeaderLocaleResolver** : 클라이언트의 브라우저에 설정된 locale정보 사용

* Bean 설정 파일에 정의하지 않을 경우 **AcceptHeaderLocaleResolver** 가 default 로 적용된다.

❑ CookieLocaleResolver

- CookieLocaleResolver 를 설정하는 경우 사용자의 쿠키에 설정된 Locale 을 읽어들인다.
- Bean 설정정보(xx-servlet.xml)

```
<bean id= "localeResolver"
  class= "org.springframework.web.servlet.i18n.CookieLocaleResolver" >
  <property name= "cookieName" value= "clientlanguage"/>
  <property name= "cookieMaxAge" value= "100000"/>
  <property name= "cookiePath" value= "web/cookie"/>
</bean>
```

- Property 속성

속성	기본값	설명
cookieName	classname + locale	쿠키명
cookieMaxAge	integer.MAX_INT	-1 로 해두면 브라우저를 닫을 때 없어짐
cookiepath	/	Path 를 지정하면 해당하는 Path와 그 하위 Path 에서만 참조

❑ SessionLocaleResolver

- request가 가지고 있는 **Session**으로 부터 locale 정보를 가져온다.
- Bean 설정정보(xx-servlet.xml)

```
<bean id="localeResolver"  
      class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />
```

❑ AcceptHeaderLocaleResolver

- 사용자의 브라우저에서 보내진 request 의 헤더에 accept-language 부분에서 Locale 을 읽어들인다. 사용자 브라우저의 Locale 을 나타낸다.
- Bean 설정정보(xx-servlet.xml)

```
<bean id="localeResolver"  
      class="org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver" />
```

□ 샘플예제 (SessionLocaleResolver)

– Web Configuration –Filter 설정

- Web 을 통해 들어오는 요청을 Charset UTF-8 적용

```
<filter>
  <filter-name>encoding-filter</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>
</filter>

<filter-mapping>
  <filter-name>encoding-filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

– Spring Configuration (xx-servlet.xml)

- 아래와 같이 localeResolver 와 localeChangeInterceptor 를 등록하고 Annotation 기반에서 동작할 수 있도록 **RequestMappingHandlerMapping** 에 interceptor 로 등록을 해준다.

(표준프레임워크 3.0에서 사용가능 - 기존 DefaultAnnotationHandlerMapping이 deprecated됨)

- SessionLocaleResolver 를 이용하여 위와 같이 하였을 경우 Locale 결정은 **language** Request Parameter 로 넘기게 된다.

```
<bean id="localeResolver"
class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />

<!--
쿠키를 이용한 Locale 이용시 <bean id="localeResolver"
class="org.springframework.web.servlet.i18n.CookieLocaleResolver"/>
-->
<bean id="localeChangeInterceptor"
class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
<property name="paramName" value="language" />
</bean>

<bean id="annotationMapper"
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">
<property name="interceptors">
<list>
<ref bean="localeChangeInterceptor" />
</list>
</property>
</bean>
```

– Message Source 설정

- 먼저 다국어를 지원해야 하므로 메시지를 MessageSource 로 추출하여 구현해야 한다.
- ReloadableResourceBundleMessageSource를 사용하면 어플리케이션 실행 중 메세지 리소스 변경을 reload 한다.
자세한 MessageSource 내용은 Resource 를 참고하길 바란다.
messageSource는 아래와 같이 설정하였다.

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

– Message Source 설정

- message properties 파일은 아래와 같다.
locale에 따라 ko, en 으로 구분하였다.
message_ko.properties 파일 : view.category=카테고리
message_en.properties 파일 : view.category=category

```
#label#  
login.form.title=로그인 입력 폼  
login.form.type=로그인 타입  
login.form.id=로그인 ID  
login.form.name= 로그인 명  
login.form.password=로그인 암호  
login.form.submit=로그인  
view.category=카테고리  
  
#label#  
login.form.title=Login Form  
login.form.type=Login Type  
login.form.id=ID  
login.form.name=Name  
login.form.password=Password  
login.form.submit=Login  
view.category=category
```


– JSP

- Spring Message Tag 이용 : `<spring:message code="view.category" />`

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

<form:form commandName="message" >
....
    <table border="1" cellspacing="0" class="boardList" summary="List of Category">
        <thead>
            <tr>
                <th scope="col">No.</th>
                <th scope="col">
                    <input name="checkAll" type="checkbox" class="inputCheck" title="Check All"
onclick="javascript:fncCheckAll();" />
                </th>
                <th scope="col"><spring:message code="view.category" /> ID</th>
                <th scope="col"><spring:message code="view.category" /> 명</th>
                <th scope="col">사용여부</th>
                <th scope="col">Description</th>
                <th scope="col">등록자</th>
            </tr>
        </thead>
    </table>
....
```

– 결과

화면상으로 해당 페이지를 실행해보면 아래와 같다.

한글인 경우 :

<http://localhost:8080/sample-web/sale/listCategory.do?language=ko>

No.	<input type="checkbox"/>	카테고리 ID	카테고리 명	사용여부	
1	<input type="checkbox"/>	0000000001	Sample Test	Y	This is initial test data,
2	<input type="checkbox"/>	0000000002	test Name	Y	test Desc

<< End
 < Prev
 11
 12

영어인 경우 :

<http://localhost:8080/sample-web/sale/listCategory.do?language=en>

No.	<input type="checkbox"/>	category ID	category 명	사용여부	
1	<input type="checkbox"/>	0000000001	Sample Test	Y	This is
2	<input type="checkbox"/>	0000000002	test Name	Y	test De

<< En

LAB 301-mvc 실습 (3)

세션 및 국제화 예제

□ The Spring Framework - Reference Documentation

- <http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/>
- 이전 버전 참조
 - <http://static.springsourceorg/spring/docs/2.5.x/reference/>
 - <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/>

실행환경(화면처리)

Validation

1. 개요
2. Jakarta Commons Validator
3. Spring+Commons Validator
4. DefaultValidatorFactory, DefaultBeanValidator
5. validator-rules.xml
6. validator.xml
7. Server-Side Validation
8. Client-Side Validation

□ Validation 서비스

- 화면 처리 레이어의 Validation 서비스는 **입력 값 유효성 검증 기능을 제공**한다.
- 유효성 검증 후, 잘못된 유효성이 있을 경우 오류 메시지를 표시한다.
- 유효성 검사 규칙(validation rules)을 템플릿 형태로 제공하여 쉽게 적용이 가능하다.
- **입력 값 유효성 검증(validation)**을 위한 기능은 Valang, Jakarta Commons, Spring 등에서 제공한다.
- 기반 오픈소스로 **Jakarta Commons Validator**를 선택.
- MVC 프레임워크인 Spring MVC와 Jakarta Commons Validator의 연계와 활용방안을 제공한다.

□ Jakarta Commons Validator

- Jakarta Commons Validator는 필수 값, 각종 primitive type (int,long,float...), 최대-최소길이, 이메일, 신용카드번호 등의 값 체크 등을 할 수 있도록 Template이 제공된다.
- Template은 Java 뿐 아니라 JavaScript로도 제공되어 **client-side, server-side의 검증을 함께** 할 수 있으며, Configuration과 에러메시지를 client-side, server-side 별로 따로 하지 않고 한곳에 같이 쓰는 관리상의 장점이 있다.

□ Spring Framework + Jakarta Commons Validator

- Struts에서는 Commons Validator를 사용하기 위한 `org.apache.struts.validator.ValidatorPlugIn` 같은 플러그인 클래스를 제공하는데, Spring에서는 **Spring Modules**(<https://springmodules.dev.java.net/>) 프로젝트에서 연계 모듈을 제공한다.
- 필요라이브러리
 - commons-validator : Ver 1.4.0 아래 4개의 파일에 대한 dependency가 있다.
 1. commons-beanutils : Ver 1.8.3
 2. commons-digester : Ver 1.8
 3. commons-logging : Ver 1.1.1
 4. junit : Ver 3.8.2
 - spring modules : Ver 0.9 다운 받고 압축을 풀어 보면 여러 파일들이 있으나 여기서 필요한 건 `spring-modules-validation.jar` 뿐이다. 예제를 보려면 `samples\src\spring-modules-validation-commons-samples-src.zip`도 필요하다.
 - 위에서 언급한 라이브러리들을 설치한다.

□ DefaultValidatorFactory, DefaultBeanValidator 설정

- DefaultValidatorFactory
 - 프로퍼티 'validationConfigLocationsApache'에 정의된 Validation rule을 기반으로 Commons Validator들의 인스턴스를 얻는다.
- DefaultBeanValidator
 - DefaultBeanValidator는 org.springframework.validation.Validator를 implements하고 있지만, DefaultValidatorFactory가 가져온 Commons Validator의 인스턴스를 이용해 validation을 수행한다. Controller의 validation을 수행할 때 이 DefaultBeanValidator를 참조하면 된다.
- 빈 정의 파일에 아래와 같이 DefaultValidatorFactory, DefaultBeanValidator를 선언한다.

```
<!-- Integrated Apache Commons Validator by Spring Modules -->
<bean id="beanValidator" class="org.springframework.validation.commons.DefaultBeanValidator">
  <property name="validatorFactory" ref="validatorFactory"/>
</bean>

<bean id="validatorFactory" class="org.springframework.validation.commons.DefaultValidatorFactory">
  <property name="validationConfigLocations">
    <list>
      <value>/WEB-INF/config/validator-rules.xml</value>
      <value>/WEB-INF/config/validator.xml</value>
    </list>
  </property>
</bean>
```

□ validator-rules.xml 설정

- validator-rules.xml은 application에서 사용하는 모든 validation rule에 대해 정의하는 파일이다.
- <validator> 태그를 구성하는 주요 요소는 아래와 같다.
 - name : validation rule(required,mask,integer,email...)
 - classname : validation check를 수행하는 클래스명(org.springframework.validation.commons.FieldChecks)
 - method : validation check를 수행하는 클래스의 메소드명(validateRequired,validateMask...)
 - methodParams : validation check를 수행하는 클래스의 메소드의 파라미터
 - msg : 에러 메시지 key . (message 프로퍼티에서 설정한다.)
 - javascript : client-side validation을 위한 자바스크립트 코드

```
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.0//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_0.dtd">
<form-validation>
  <global>
    <validator name="required"
      classname="org.springframework.validation.commons.FieldChecks"
      method="validateRequired"
      methodParams="java.lang.Object,
        org.apache.commons.validator.ValidatorAction,
        org.apache.commons.validator.Field,
        org.springframework.validation.Errors"
      msg="errors.required">
      <javascript><![CDATA[
        ...
      ]]>
    </javascript>
  </validator>
</form-validation>
```

□ Spring Modules Validation Rule

name(validation rule)	FieldCheck 클래스	FieldCheck 메소드	기능
required	org.springframework.validation.commons.FieldChecks	validateRequired	필수값 체크
minlength	org.springframework.validation.commons.FieldChecks	validateMinLength	최소 길이 체크
maxlength	org.springframework.validation.commons.FieldChecks	validateMaxLength	최대 길이 체크
mask	org.springframework.validation.commons.FieldChecks	validateMask	정규식 체크
byte	org.springframework.validation.commons.FieldChecks	validateByte	Byte형 체크
short	org.springframework.validation.commons.FieldChecks	validateShort	Short형 체크
integer	org.springframework.validation.commons.FieldChecks	validateInteger	Integer형 체크
long	org.springframework.validation.commons.FieldChecks	validateLong	Long형 체크
float	org.springframework.validation.commons.FieldChecks	validateFloat	Float형 체크
double	org.springframework.validation.commons.FieldChecks	validateDouble	Double형 체크
date	org.springframework.validation.commons.FieldChecks	validateDate	Date형 체크
range	org.springframework.validation.commons.FieldChecks	validateIntRange	범위 체크
intRange	org.springframework.validation.commons.FieldChecks	validateIntRange	int형 범위 체크
floatRange	org.springframework.validation.commons.FieldChecks	validateFloatRange	Float형 범위체크
creditCard	org.springframework.validation.commons.FieldChecks	validateCreditCard	신용카드번호체크
email	org.springframework.validation.commons.FieldChecks	validateEmail	이메일체크

❑ eGovframework Validation Rule

Name (validation rule)	FieldCheck 클래스	FieldCheck 메소드	기능
ihidnum	egovframework.rte.ptl.mvc.validation.RteFieldChecks	validateIhIdNum	주민등록번호체크
korean	egovframework.rte.ptl.mvc.validation.RteFieldChecks	validateKorean	한글체크
password1	egovframework.rte.ptl.mvc.validation.RteFieldChecks	validatePassword1	비밀번호 자리 수 체크
pwdCheckComb3	egovframework.rte.ptl.mvc.validation.RteFieldChecks	validatePwdCheckComb3	비밀번호 최소 3가지 조합 체크 (영문,숫자,특수문자 ¹⁾)
pwdCheckComb4	egovframework.rte.ptl.mvc.validation.RteFieldChecks	validatePwdCheckComb4	비밀번호 최소 4가지 조합 체크 (영대문자,영소문자,숫자,특수문자 ¹⁾)
pwdCheckSeries	egovframework.rte.ptl.mvc.validation.RteFieldChecks	pwdCheckSeries	비밀번호 3개 이상 문자 연속 체크 (ex : 123, abc)
pwdCheckRepeat	egovframework.rte.ptl.mvc.validation.RteFieldChecks	pwdCheckRepeat	비밀번호 3개 이상 문자 반복 체크 (ex : 111, aaa)

특수문자¹⁾ : ~!@#\$%^&*?

□ validator.xml 설정

- validator.xml은 validation rule과 validation할 Form을 매핑한다.
form name과 field property의 name-rule은 Server-side와 Client-side인 경우에 따라 다르다.
- Server-side validation의 경우
 - form name과 field property는 validation할 폼 클래스의 이름, 필드가 각각 매핑된다.(camel case)
폼 클래스가 Employee면 employee, DepartmentForm 이면 departmentForm을 form name으로 지정한다.
- Client-side의 경우
 - form name은 JSP에서 설정한 <validator:javascript formName="employee" .../> 태그의 formName과 매핑되고, field property는 각각의 폼 필드의 이름과 일치하면 된다.
- 따라서, Server-side, Client-side 둘 다 수행하려면,
 - JSP의 <validator:javascript formName="employee" .../> 태그의 formName은 폼 클래스의 이름이 되어야 하고, JSP의 폼 필드들은 폼 클래스의 필드와 일치해야 한다.

- depends는 해당 필드에 적용할 (validator-rules.xml에 정의된 rule name) validator를 의미한다.
<arg key...>는 메시지 출력시 파라미터를 지정하는 값이다.
- 아래와 같이 작성했다면, Employee 클래스의 name 필드에 대해서 필수값 체크를, age 필드에 대해서 integer 체크를, email 필드에 대해서 필수값과 email 유효값 체크를 하겠다는 의미이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1//EN"
    http://jakarta.apache.org/commons/dtds/validator_1_1.dtd>
<form-validation>
  <formset>
    <form name="employee">
      <field property="name" depends="required,korean">
        <arg0 key="employee.name" />
      </field>
      <field property="age" depends="integer">
        <arg0 key="employee.age" />
      </field>
      <field property="email" depends="email,htmltag">
        <arg0 key="employee.email" />
      </field>
      <field property="ihidnum" depends="ihidnum"/>
    </form>
  </formset>
</form-validation>
```

❑ Server-Side Validation

– Controller

- Server-Side Validation을 위해 Controller에 validation 로직을 추가.

```
package com.easycompany.controller.annotation;
...
import org.springframework.validation.commons.DefaultBeanValidator;
...
@Controller
public class UpdateEmployeeController {

    @Autowired
    private DefaultBeanValidator beanValidator;

    ...
    @RequestMapping(value = "/updateEmployee.do", method = RequestMethod.POST)
    public String updateEmployee(@ModelAttribute("employee") Employee employee,
        BindingResult bindingResult, Model model) {

        beanValidator.validate(employee, bindingResult); //validation 수행

        if (bindingResult.hasErrors()) { //만일 validation 에러가 있으면...
            ....
            return "modifyemployee";
        }

        employeeManageService.updateEmployee(employee);
        return "changenotify";
    }
}
```

– JSP

- form submit을 하면 이름, 나이, 이메일등의 입력값이 Employee 클래스에 바인딩이 되어서 Controller에 전달이 되고, Controller에 validation 수행 로직이 있으면 validator.xml 내용에 따라 validation이 이루어 진다.
만일 에러가 발생하면 <form:error.../>에 해당 에러메시지를 출력한다.

```
<form:form commandName="employee">
  <table>
    <tr>
      <th>이름</th><td><form:input path="name" size="20"/><form:errors path="name" /></td>
    </tr>
    <tr>
      <th>비밀번호</th><td><form:input path="password" size="10"/></td>
    </tr>
    <tr>
      <th>나이</th><td><form:input path="age" size="10"/><form:errors path="age" /></td>
    </tr>
    <tr>
      <th>이메일</th><td><form:input path="email" size="50"/><form:errors path="email" /></td>
    </tr>
  </table>
  <table width="80%" border="1">
    <tr>
      <td><input type="submit"/>
      <input type="button" value="LIST" onclick="location.href='/easycompany/employeeList.do'"/>
    </td>
    </tr>
  </table>
</form:form>
```


– 에러 메시지 등록

- 메시지 파일(message.properties)에 에러 메시지를 등록한다.
- validation 에러가 발생하면 validator-rules.xml에서 정의했던 msg값으로 에러메시지의 key값을 찾아 해당하는 메시지를 출력해준다. 예를 들어, email validation에서 에러가 발생하면 msg값이 errors.email 이므로, "유효하지 않은 이메일 주소입니다."라는 에러 메시지를 JSP에 있는 <form:errors path="email" /> 부분에 출력하게 된다.

```
employee.name=이름  
employee.email=이메일  
employee.age=나이  
employee.password=비밀번호  
  
# -- validator errors --  
errors.required={0}은 필수 입력값입니다.  
errors.minlength={0}은 {1}자 이상 입력해야 합니다.  
errors.maxlength={0}은 {1}자 이상 입력할수 없습니다.  
errors.invalid={0}은 유효하지 않은 값입니다.  
  
errors.byte={0}은 byte타입이어야 합니다.  
errors.short={0}은 short타입이어야 합니다.  
errors.integer={0}은 integer 타입이어야 합니다.  
  
....  
  
errors.email=유효하지 않은 이메일 주소입니다.
```

– TEST

- 이름 필드에 값을 비우고 submit하면, name에 필수값(required) validation rule이 설정되어 있으므로 아래와 같이 이름 필드 옆에 에러 메시지가 출력된다.

사원정보 사원정보추가 부서정보 실적집계

사원번호	2
부서번호	<div>경영기획실</div> <div>회식메뉴혁신팀</div>
이름	<input type="text"/> 이름은 필수 입력값입니다.
비밀번호	<input type="password"/>
나이	<input type="text"/>
이메일	<input type="text"/>

□ Client-Side Validation

– validator.jsp 추가

```
<%@ page language="java" contentType="javascript/x-javascript" %>
<%@ taglib prefix="validator" uri="http://www.springmodules.org/tags/commons-validator" %>
<validator:javascript dynamicJavascript="false" staticJavascript="true"/>
```

- /validator.do로 호출하도록 Controller에서 메소드를 추가하고 requestmapping 한다.
validator.jsp를 <http://localhost:8080/easycompany/validator.do> 브라우저에서 호출해보면, validator-rules.xml에서 정의한 javascript 함수들이 다운로드 되거나 화면에 print 되는 걸 확인할 수 있다.
validator.jsp는 client-validation을 위해 validator-rules.xml에서 정의한 javascript 함수들을 로딩한다.
따라서 client-validation을 할 페이지에서는 이 validator.jsp를 **<script type="text/javascript" src="<c:url value="/validator.do"/>"></script>** 같이 호출한다.

– JSP 설정(taglib,javascript) 추가

- client-validation을 위해서는 해당 JSP에 아래와 같은 작업이 추가 되어야 한다.
- commons-validator taglib를 선언한다.
`<%@ taglib prefix="validator" uri="http://www.springmodules.org/tags/commons-validator" %>`
- 필요한 자바 스크립트 함수를 generate 하기 위한 코드를 추가 한다. validation-rules.xml에서 선언한 함수를 불러 오기 위해, 위에서 작성한 validator.jsp를 아래와 같이 호출한다.
`<script type="text/javascript" src="<c:url value="/validator.do"/>"></script>`

- 위의 자바 스크립트 함수를 이용해 필요한 validation과 메시지 처리를 위한 자바 스크립트를 generate 하기 위한 코드를 추가 한다. formName에는 validator.xml에서 정의한 form의 이름을 써준다.
<validator:javascript formName="employee" staticJavascript="false" xhtml="true" cdata="false"/>
- form submit시에 validateVO클래스명() 함수를 호출한다.
... onsubmit="return validateEmployee(this)" ">
- 따라서 앞의 server-side validation에서 작성한 modifyemployee.jsp은 아래와 같이 변경된다.

```
<!-- commons-validator tag lib 선언-->
<%@ taglib prefix="validator" uri="http://www.springmodules.org/tags/commons-validator" %>
<!--for including generated Javascript Code(in validation-rules.xml)-->
<script type="text/javascript" src="<c:url value="/validator.do"/>"></script>
<!--for including generated JavaScriptCode(validateEmployee(),formName:validator.xml에서 정의한 form의 이름)-->
<validator:javascript formName="employee" staticJavascript="false" xhtml="true" cdata="false"/>
<script type="text/javascript">
function save(form){
    if(!validateEmployee(form)){
        return;
    }else{
        form.submit();
    }
}
</script>
<form:form commandName="employee">
<!--<input type="submit"/>-->
<input type="button" value="SAVE" onclick="save(this.form)"/> <!-- client-validation을 위해 바로 submit하지 않고 먼저
validateEmployee 함수를 호출-->
<input type="button" value="LIST" onclick="location.href='/easycompany/employeeList.do'"/>
</td></tr>
</table>
</form:form>
```

- TEST

- 이름 필드의 값을 지우고 저장 버튼을 누르면 아래와 같은 alert 메시지가 보인다.

사원정보 사원정보추가 부서정보 실적집계

사원번호	2
부서번호	경영기획실 회식메뉴혁신팀
이름	
비밀번호	2
나이	39
이메일	kk2@ejb.com

SAVE LIST

Windows Internet Explorer

!

이름은 필수 입력값입니다.

확인

- ❑ Spring Modules Reference Documentation v 0.9

실행환경(화면처리)

Ajax Support

1. 개요 및 설정
2. 기본 기능
3. 기본 예제
4. ajax함수의 jqXHR기능
5. callback함수 예제
6. jQuery.get()
7. jQuery.post()
8. 응용

□ jQuery Ajax

- jQuery는 브라우저 호환성을 제공하는 자바스크립트 라이브러리.
- jQuery는 자바스크립트 프레임워크로서 간결한 문장의 표현으로 front-end(화면) 개발시 생산성 향상.
- 다양한 오픈소스 라이브러리들을 통해 java script로 ajax, json parser, css selector, dom selector, event 등... 다양한 ui 기능 등을 제공하고 front-end(화면)을 동적으로 제어가능

□ jQuery 설정 - jQuery java script 추가

- jQuery url을 직접 명시

```
<script src="https://code.jquery.com/jquery-1.11.0.min.js"></script>
```

- jQuery script를 직접 추가하여 참조

```
<script type="text/javascript" src="jQuery파일 경로"></script>
```

* jquery-버전.js를 다운받아 프로젝트 하위 경로에 추가한 후, 저장한 경로를 설정.

* jquery는 MIT 라이선스를 사용함.

□ jQuery.ajax(url[,settings]) 함수

- jQuery Ajax기능을 위해서는 기본적으로 jQuery.ajax(url[,settings]) 함수를 이용

설정	설명	default	type
type	http 요청 방식 설정(POST, GET, PUT,DELETE...)	GET	type string
url	request를 전달할 url명	N/A	url string
data	request에 담아 전달할 data명과 data값	N/A	String/Plain Object/Array
contentType	server로 데이터를 전달할 때 contentType	'application/x-www-form-urlencoded; charset=UTF-8'	contentType String
dataType	서버로부터 전달받을 데이터 타입	xml, json, script, or html	xml/html/script/json/jsonp/multiple, space-separated values
statusCode	HTTP 상태코드에 따라 분기처리되는 함수	N/A	상태 코드로 분리되는 함수
beforeSend	request가 서버로 전달되기 전에 호출되는 콜백함수	N/A	Function(jqXHR jqXHR, PlainObject settings)
error	요청을 실패할 경우 호출되는 함수	N/A	Function(jqXHR jqXHR, String textStatus, String errorThrown)
success	요청에 성공할 경우 호출되는 함수	N/A	Function(PlainObject data, String textStatus, jqXHR jqXHR)
crossDomain	crossDomain request(jsonP와 같은)를 강제할 때 설정(cross-domain request설정 필요)	same-domain request에서 false, cross-domain request에서는 true	Boolean

□ 기본 예제

- 하나의 파라미터를 Ajax request로 전달하는 예제.

```
$.ajax({
  type : "POST",
  url : "<c:url value='/example01.do'/>",
  contentType: "application/x-www-form-urlencoded; charset=UTF-8",
  dataType:'json',
  data : {
    sampleInput : "sampleData"
  },
  success : function(data, status, jqXHR) {
    // 통신이 정상적일때 해당 함수 실행
  },
  error : function(request, status, error){
    // 통신이 비정상적일때 해당 함수 실행
  },
  complete : function(jqXHR, status) {
    //통신의 성공과 실패시 해당 함수 실행
  }
});
```

- Http 요청 동기 Ajax Post Method 방식으로 example01.do로 호출
- contentType을 application/x-www-form-urlencoded 방식으로 charset을 UTF-8 으로 설정.
- sampleInput이란 데이터명으로 “sampleData” String을 전달
- 통신의 요청이 성공할 경우 success 함수 호출
- 통신이 비정상적일때 error 함수 호출
- 통신의 성공과 실패시 complete 함수 호출

LAB 302-Ajax 실습(1)

자동 완성 예제

□ ajax함수의 jqXHR(XMLHttpRequest)

- jQuery 1.5부터 jQuery의 모든 ajax함수는 XMLHttpRequest객체의 상위 집합을 리턴받을 수 있다. 이 객체를 jQuery에서는 jqXHR(응답값, 통신 status, readyStatus)이라 부르며, jqXHR의 함수로 콜백함수를 정의한다.
- jQuery 1.8부터는 deprecated 되어서, success/error/complete callback option만 사용해야 한다.

□ jqXHR(XMLHttpRequest)의 callback 함수

- 사용자 정의에 의해 순차적으로 실행

함수명	설명
jqXHR.done(function(data, textStatus, jqXHR) {});	성공시 호출되는 콜백 함수
jqXHR.fail(function(jqXHR, textStatus, errorThrown) {});	실패시 호출되는 콜백 함수
jqXHR.always(function(data jqXHR, textStatus, jqXHR errorThrown) { });	항상 호출되는 콜백 함수

□ Ajax 내부 callback과 jqXHR(XMLHttpRequest) callback의 실행순서

- 성공 : success(내부) > complete (내부) > done(jqXHR) > always(jqXHR)
- 실패 : error (내부) > complete (내부) > fail(jqXHR) > always(jqXHR)

□ jqXHR(XMLHttpRequest) callback함수 예제1

- 여러 개의 데이터를 전달하며 호출 후 콜백함수로 서버에서 값을 받는 예제.

```
$.ajax({  
  url : "<c:url value='/example02.do'/>",  
  data : {  
    name : "gil-dong",  
    location : "seoul"  
  },  
})  
.done(function( data ) {  
  if ( console && console.log ) {  
    console.log( "Sample of data:", data.slice( 0, 100 ) );  
  }  
});
```

- example02.do를 호출하며
- name, location을 요청데이터로 전달
- 성공 시에 done 콜백함수를 호출

□ jqXHR(XMLHttpRequest) callback함수 예제2

- example03.do를 호출하며
성공 시 done 콜백함수, 실패 시 fail 콜백함수 호출.

```
var jqxhr = $.ajax( "<c:url value='/example03.do'/>", )  
  .done(function() {  
    alert( "success" );  
  })  
  .fail(function() {  
    alert( "error" );  
  })  
  .always(function() {  
    alert( "complete" );  
  });  
  
jqxhr.always(function() {  
  alert( "second complete" );  
});
```

- 성공, 실패여부에 상관없이 always 콜백함수는 항상 호출
- done, fail, always콜백함수는 ajax함수를 통해 리턴되어 request로 호출가능

□ get함수

- jQuery 1.5부터 success 콜백함수는 jqXHR (XMLHttpRequest의 상위 집합 객체)를 받을 수 있지만, JSONP와 같은 cross-domain request의 GET요청 시에는 jqXHR을 사용하여도 XHR인자는 success함수에서 undefined로 인식된다.
- jQuery.get()은 ajax를 GET요청하는 함수이며 jqXHR을 반환받는다.
따라서 \$.ajax()와 동일하게 done, fail, always 콜백함수를 쓸 수 있다.
- get함수는 ajax함수로 나타내면 다음과 같다.

```
$.ajax({
  url: url,
  data: data,
  success: success,
  dataType: dataType
});
```

- get함수 설정

설정	설명	default	type
url	request를 전달할 url명	N/A	url String
data	request에 담아 전달할 data명과 data값	N/A	String/Plain Object
dataType	서버로부터 전달받을 데이터 타입	xml, json, script, or html	String
success	요청에 성공할 경우 호출되는 함수	N/A	Function(PlainObject data, String textStatus, jqXHR jqXHR)

□ url만 호출하고 결과값은 무시하는 경우

```
$.get( "example.do" );
```

□ url로 데이터만 보내고 결과는 무시하는 경우

```
$.get( "example.do", { name: "gil-dong", location: "seoul" } );
```

□ url을 호출하고 결과값을 Alert창으로 띄우는 경우

```
$.get( " example.do", function( data ) {  
    alert( "Data Loaded: " + data );  
});
```

□ url로 데이터를 보내고 결과값을 Alert창으로 띄우는 경우

```
$.get( "example.do", { name: "gil-dong", location: "seoul" } )  
    .done(function( data ) {  
        alert( "Data Loaded: " + data );  
    });
```

□ post함수

- jQuery.post()은 ajax를 POST로 요청하는 함수로서 jqXHR을 반환받는다.
따라서 ajax(), get()와 동일하게 done, fail, always콜백함수를 쓸 수 있다.
- jQuery.post 함수 설정은 get함수와 동일하다.(jQuery.post(url [, data] [, success] [, dataType]))
- jQuery.post함수를 ajax함수로 쓰면 다음과 같다.

```
$.ajax({  
  type: "POST",  
  url: url,  
  data: data,  
  success: success,  
  dataType: dataType  
});
```


□ url만 호출하고 결과값은 무시하는 경우

```
$.post( "example.do" );
```

□ url로 데이터만 보내고 결과는 무시하는 경우

```
$.post( "example.do", { name: "gil-dong", location: "seoul" } );
```

□ url을 호출하고 결과값을 console log를 남기는 경우

```
$.post( "example.do", function( data ) {  
    console.log( data.name );  
    console.log( data.location );  
});
```

□ url로 데이터를 보내고 결과값을 Alert창으로 띄우는 경우

```
$.post( "example.do", { name: "gil-dong", location: "seoul" } )  
    .done(function( data ) {  
        alert( "Data Loaded: " + data );  
    });
```

□ Auto complete

- jQuery에서는 input창에서 예상되는 텍스트값을 보여주는 자동완성기능을 쉽게 구현할 수 있도록 autoComplete()을 제공
- autoComplete의 설정

구분	설정	설명	Type
Options	source	하단에 뜨는 자동완성리스트(필수값)	Array, String, function
Options	minLength	자동완성이 동작하는 최소 문자열 수	Integer
Options	disabled	disable 여부	Boolean
Events	change(event, ui)	값 변경시 발생하는 이벤트 함수	autocompletechange
Events	focus(event, ui)	값이 포커스될 때 발생하는 이벤트 함수	autocompletefocus
Events	select(event, ui)	값이 선택될 때 발생하는 이벤트 함수	autocompleteselect

* jQuery UI script를 직접 추가하여 참조하는 경우는 jQuery-ui.js와 jQuery-ui.css를 다운받아 프로젝트 하위 경로에 추가한 후, 저장한 경로를 지정

□ autoComplete 기본 예제

```
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery UI Autocomplete - Default functionality</title>
  <link rel="stylesheet"
    href="//code.jquery.com/ui/1.11.0/themes/smoothness/jquery-ui.css">
  <script src="//code.jquery.com/jquery-1.10.2.js"></script>
  <script src="//code.jquery.com/ui/1.11.0/jquery-ui.js"></script>
  <link rel="stylesheet" href="/resources/demos/style.css">
  <script>
    $(function() {
      var availableTags = [ "ActionScript", "AppleScript", "Asp", "BASIC",
                            "C", "C++", "Clojure", "COBOL", "ColdFusion", "Erlang",
                            "Fortran", "Groovy", "Haskell", "Java", "JavaScript", "Lisp",
                            "Perl", "PHP", "Python", "Ruby", "Scala", "Scheme" ];

      $( "#tags" ).autocomplete({
        source: availableTags
      });
    });
  </script>
</head>
<body>

<div class="ui-widget">
  <label for="tags">Tags: </label>
  <input id="tags">
</div>
</body>
</html>
```

□ autoComplete 예제 결과

Tags:

- ActionScript
- AppleScript
- Asp
- BASIC
- Erlang
- Fortran
- Haskell
- Java
- JavaScript
- Scala

* minLength는 default값이 1
- input에 1개이상의 문자를 입력했을 때 source의 String배열들이 자동문자 리스트로 표시됨.

□ autoComplete와 ajax를 이용한 응용예제

- 목표 : ajax를 통해 리스트를 받아와 autoComplete의 source로 뿌려주는 예제
- 제한 : ajax를 통해 source를 가져오기 위해서는 서버호출 결과값의 타입이 제한된다.
 - * String array타입
 - * Object(id, label,value 값을 갖는) array타입

* request.term은 input값으로 사용자가 입력한 값으로서 사용자가 "je"를 입력하면 input = je 로 값이 전달됨.
- 이 때 Controller에서는 request.getParameter("input") 또는 @RequestParam("input") String input으로 값을 사용할 수 있음

□ 1. String array로 가져오는 경우(1/2)

- example.do라는 url로 ajax호출을 통해 source를 가져오고 선택 시 값이 alert되도록 하는 예제

```
...  
<script type="text/javascript">  
$(function() {  
    $('#autoValue').autocomplete(  
        {  
            source : function(request, response) {  
                $.ajax({  
                    url : "<c:url value='/example.do'/>",  
                    data : { input : request.term }, //사용자의 입력값  
                    success : function(data) {  
                        response( data.locations );  
                    }  
                });  
            },  
            minLength : 1,  
            select : function(event, ui) {  
                alert( ui.item ? "Selected : " + ui.item.label  
                    : "Nothing select, input was " + this.value);  
            }  
        });  
    });  
</script>  
//... 생략  
<input type="text" id="autoValue" />  
//... 생략
```

□ 1. String array로 가져오는 경우(2/2)

- 목표 : MappingJacksonJsonView의 빈을 jsonView로 등록했을 때 Controller에서 data를 꺼내고 결과값을 client로 넘겨주는 예제
- 가정 : 비즈니스상 처리되는 query와 결과값은 아래와 같이 동작한다.
(query 예제 - mybatis)

```
@RequestMapping(value="/autoList.do")
public String autoList(HttpServletRequest request,
                      ModelMap model) {

    String input = request.getParameter("input");
    List<String> resultList = new ArrayList<String>();
    //...생략...
    //서비스클래스를 통해 결과값을 resultList에 담음

    model.addAttribute("locations", resultList );

    return "jsonView";
}
```

```
<select id="selectLocationList" parameterType="string" resultType="string">
SELECT LOCATION_NM
FROM LOCATION
WHERE upper(LOCATION_NM) LIKE '%' || upper("#{input}) || '%'
</select>
```

(결과값)

```
{"locations":["Daejeon","Jeju-do","Jeolabuk-do","Jeolanam-do"]}
```

- ajax의 성공시 콜백함수인 success에서는 data.locations로 값을 꺼내 autocomplete의 source를 설정한다.

입력값 :

- Daejeon
- Jeju-do
- Jeolabuk-do
- Jeolanam-do

□ 2. Object array로 가져오는 경우

- 목표 : 입력값에 따라 데이터를 검색하고 Object array로 서버에서 결과값을 가져오는 예제
- 가정 : Controller에서 List<Object>의 값을 ModelMap에 “locations”라는 이름으로 클라이언트로 넘겨주고 서버에서 다음과 같이 json data가 넘어왔을 때

```
{
  "locations": [
    {
      "locationId": "0006",
      "locationNm": "Daejeon",
      "localNb": "042"
    },
    {
      "locationId": "0010",
      "locationNm": "Jeju-do",
      "localNb": "064"
    },
    {
      "locationId": "0011",
      "locationNm": "Jeolabuk-do",
      "localNb": "063"
    },
    {
      "locationId": "0012",
      "locationNm": "Jeolanam-do",
      "localNb": "061"
    }
  ]
}
```

- autocomplete의 source에 넘어온 값이 나타나도록 하는 Object는 label, id, value값을 가질 수 있다. 그렇기 때문에 넘어온 request의 값을 success콜백 함수에서 source에 나타나도록 하는 Object형태로 변환해야 한다. 나머지 jQuery구현은 동일하다.

```
success : function(data) {
    response($.map(data.locations, function(item) {
        return{
            id: item.locationId,
            label: item.locationNm,
            //value: item.localNb
        });
    }));
}
```

이 때, 자동완성 리스트로 나타나는 값은 label이며, value를 설정해주었을 때는 자동완성 리스트에서 값 선택 시 input값에 label값이 아닌 value값이 대입된다.

□ selectbox(combobox) 제어

- jQuery에서는 별도로 select box ui함수를 제공하지 않는다.
- 목표 : jQuery를 통해 selectbox를 제어하는 방법과 selectbox에 나타나는 리스트를 ajax로 구현하는 방법 확인
- 다루고자 하는 selectbox가 다음과 같다고 가정하자.

```
<select id="combobox">
  <option value="">===locations===</option>
  <option value="01">Seoul</option>
  <option value="02">Busan</option>
  <option value="03">Jeu-do</option>
  <option value="04">Incheon</option>
</select>
```

□ selectbox 값 가져오기

- selectbox에서 선택된 value를 가져오는 방법

```
var selectedVal = $("#combobox option:selected").val();
```

□ selectbox 내용 가져오기

- selectbox에서 선택된 text(ex:Seoul)를 가져오는 방법

```
var selectedText= $("#combobox option:selected").text();
```

□ selectbox에서 선택된 Index값 가져오기

- selectbox의 리스트에서 선택된 Index를 구하는 방법

```
var selectedIndex = $("#combobox option").index(  
    $("#combobox option:selected");
```

□ selectbox 값 교체하기

- selectbox의 리스트의 값들을 교체하는 방법

```
$("#combobox").html(  
    "<option value=''>==locations==</option>  
    <option value='01'>Jeju-do</option>  
    <option value='02'>Seoul</option>  
    <option value='03'>Incheon</option>  
    <option value='04'>Daejeon</option> ")
```

□ selectbox에서 값이 선택될 때 콜백함수

- selectbox에서 값이 선택되었을 때 호출되는 콜백 함수

```
$("#combobox).change(function() {  
    //기능구현  
});
```

□ selectbox에 리스트 마지막에 추가하기

- selectbox의 리스트에 값을 마지막에 추가하는 방법

```
$("#combobox").append(  
    "<option value='05'> Daejeon </option>");
```

맨 앞에 추가하는 경우는 .prepend() 사용

□ selectbox에서 값 삭제하기

- selectbox의 리스트에서 선택된 값을 삭제하는 방법

```
$("#combobox option:selected").remove();
```


□ selectbox 만들기

- selectbox 제어기능과 jQuery ajax함수를 이용하여 간단한 selectbox 구성

부서번호 :

✓ 근무부서를 선택하세요.

회식메뉴혁신팀

점심메뉴기획팀

야근금지역량팀

사랑의짝대기팀

```

<script type="text/javascript">

    $('#superdeptid').change(function() {
    $.ajax({
    url:"<c:url value='/autoSelectDept.do'/>",
    contentType: "application/x-www-form-urlencoded; charset=UTF-8",
    data: {depth:2, superdeptid:encodeURIComponent($('#superdeptid option:selected').val())},
    dataType:'json',
    success: function(returnData, status){
        $('#departmentid').loadSelectDept(returnData,"근무부서를 선택하세요.");
    }
    });
    });
</script>
  
```

```

//loadSelectDept... 생략(jQuery plugin)
$.fn.loadSelectDept = function(optionsDataArray,defaultText) {
    return this.emptySelect().each(function(){
        ....
    });
  }
  
```

LAB 302-Ajax 실습(2)

Selectbox jQuery Ajax Call

실행환경(화면처리)



UI Adaptor

1. 개요
2. 설명

□ 서비스 개요

- 전자정부 표준프레임워크와 UI 솔루션(Rich Internet Application) 연동 .
- 보통 Web Framework 과 UI 솔루션과의 연동을 하는 방법 중 가장 많이 사용하는 방식은 Controller 역할을 수행하는 Servlet 객체에서 업무 로직을 호출 전 데이터를 DTO 형태로 변환하여 업무 로직으로 넘기는 방식.
- 전자정부 표준프레임워크에서는 Spring MVC Annotation 기반으로 개발 시 메소드의 파라미터로 넘어오는 객체가 request 객체가 아닌 업무용 DTO 클래스로 넘어올 수 있도록 가이드 하는 방식을 선택.

□ 전자정부 표준 프레임워크의 UI Adaptor

– 구현 시나리오

- UI솔루션데이터에서 DTO로 변환
 1. AnnotationMethodHandlerAdapter 의 CustomRiaArgumentResolver 등록 ⇒ **CustomRiaArgumentResolver**
 2. UIAdaptor 구현/등록 ⇒ **UIAdaptorImpl**
- **Controller** 메소드 구현
- **BeanNameViewResolver** 설정
- **RiaView** 구현
 - ✓ RiaView 구현/등록 ⇒ **RiaView**

□ UI솔루션데이터에서 DTO로 변환

- AnnotationMethodHandlerAdapter 의 CustomRiaArgumentResolver 등록 ⇒ **CustomRiaArgumentResolver**
 - WebArgumentResolver의 구현체인 CustomRiaArgumentResolver 는 uiAdaptor 에 세팅된 Adaptor 를 실행해준다.

```
<bean
  class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
  <property name="webBindingInitializer">
    <bean class="egovframework.rte.fdl.web.common.EgovBindingInitializer" />
  </property>
  <property name="customArgumentResolvers">
    <list>
      <bean class="egovframework.rte.fdl.sale.web.CustomRiaArgumentResolver">
        <property name="uiAdaptor">
          <list>
            <ref bean="riaAdaptor" />
          </list>
        </property>
      </bean>
    </list>
  </property>
</bean>
```

- **CustomRiaArgumentResolver**

UiAdaptor의 구현체는 아래와 같다. 여기서는 Miplatform의 예를 들어 코드를 작성하였다.
UI 솔루션의 객체에서 DTO로 데이터를 옮기는 역할은 convert 메소드에서 수행한다.

```
public class CustomRiaArgumentResolver implements WebArgumentResolver {

    private UiAdaptor[] uiAs;
    public void setUiAdaptors(UiAdaptor[] uiAs) {
        this.uiAs = uiAs;
    }

    public Object resolveArgument(MethodParameter methodParameter, NativeWebRequest webRequest)
        throws Exception {

        Class<?> type = methodParameter.getParameterType();
        Object uiObject = null;
        if (uiAs == null)
            return UNRESOLVED;

        // 여기서 Adapter를 찾는 방법을 필히 물어볼 것....
        for (UiAdaptor uiA : uiAs) {
            if (type.equals(uiA.getModelName())) {
                HttpServletRequest request = (HttpServletRequest) webRequest.getNativeRequest();
                // uiObject = (UdDTO) uiA.convert(request);
                uiObject = (Object) uiA.convert(request);
                return uiObject;
            }
        }
        return UNRESOLVED;
    }
}
```

- UIAdaptor 구현/등록 ⇒ **UIAdaptorImpl (riaAdaptor)** → convert(HttpServletRequest request) 메소드 구현

```
...
Public Object convert(HttpServletRequest request) throws Exception {

    PlatformRequest platformRequest = null;

    try {
        platformRequest = new PlatformRequest(request, PlatformRequest.CHARSET_UTF8);
        platformRequest.receiveData();
    } catch (IOException ex) {
        ex.printStackTrace();
        // throw new IOException("PlatformRequest error");
    }

    /*
     * TODO platformRequest 에서 VariableList 와 DatasetList 를 뽑아 결정된 Map
     * 형태(example : ModelMap) 의 객체에 담아 Request 에 다시 담고 return true 로 넘긴다.
     */

    //CategoryVO dto = converte4In(platformRequest);

    //Class dto = Class.forName("egovframework.rte.fdl.sale.service.CategoryVO");
    Object dto = converte4In(platformRequest, request);

    //request.setAttribute("CategoryVO", dto);

    return dto;
}
...
```


□ Controller 메소드 구현

- UdDTO 클래스는 CustomRiaArgumentResolver 에서 만들어져 Controller 메소드의 parameter 형태로 가져온다.

```
@Controller
public class MiCategoryController {
    ...
    @RequestMapping("/sale/miplatform.do")
    public ModelAndView selectCategoryList4Mi(UdDTO miDto, Model model)
    throws Exception {

        ModelAndView mav = new ModelAndView("miplatformView");

        // 조회조건으로 넣는다.
        Map<String, String> smp = new HashMap<String, String>();
        try {

            List resultList = categoryService.selectCategoryList(smp);
            log.debug(" CategoryController START =====");
            mav.addObject("MiDTO", resultList);
            log.debug(" CategoryController End =====");

        } catch (Exception ex) {
            mav.addObject("MiResultCode", "-1");
            mav.addObject("MiResultMsg", ex.toString());
            log.info(ex.getStackTrace(), ex);
        }
        return mav;
    }
    ...
}
```

□ BeanNameViewResolver 설정

- 모델 객체의 이름이 miplatformView 이다. 이것은 Bean Name을 직접 명시 한것으로 아래와 같은 설정 (BeanNameViewResolver)이 필요하다.

```
<bean class="org.springframework.web.servlet.view.BeanNameViewResolver"
p:order="0" />

<bean class="org.springframework.web.servlet.view.UrlBasedViewResolver"
p:order="1" p:viewClass="org.springframework.web.servlet.view.JstlView"
p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />

<bean id="miplatformView" class="egovframework.rte.fdl.sale.web.MiPlatformMapView" />
```

□ RiaView 구현

- RiaView 구현/등록 ⇒ **miplatformView**

```
public class MiPlatformMapView extends AbstractView {  
  
    ...  
    @SuppressWarnings("unchecked")  
    @Override  
    protected void renderMergedOutputModel(Map model, HttpServletRequest request, HttpServletResponse response)  
        throws Exception {  
  
        PlatformData platformData = new PlatformData(miVariableList, miDatasetList);  
  
        if ((String) model.get("MiResultCode") != null) {  
            this.setMiResultMessage((String) model.get("MiResultCode"), (String) model.get("MiResultMsg"));  
        } else {  
            this.setMiResultMessage("0", "sucess~~");  
        }  
        //데이터 변환...  
        .....  
  
        try {  
  
            new PlatformResponse(response, PlatformConstants.CHARSET_UTF8).sendData(platformData);  
  
        } catch (IOException ex) {  
            if (log.isDebugEnabled()) {  
                log.error("Exception occurred while writing xml to MiPlatform Stream.", ex);  
            }  
  
            throw new Exception();  
        }  
    }  
}
```

– 결과

The screenshot displays a web application interface. In the background, a table is visible with the following columns: ID, USEYN, REGUSER, NAME, and DESCRIPTION. The first row of data shows '0000000001', '사용', 'test', 'Sample Test', and 'This is initial test 123444'. The second row shows '0000000002', '사용', 'test', 'test Name', and 'test Desc'. Above the table are two buttons: '조회' (Search) and '신규' (New). Overlaid on the right side of the table is a 'New Form' dialog box. This dialog box contains several input fields: 'id' with the value '0000000001', 'name' with 'Sample Test', 'useyn' as a dropdown menu currently set to '사용', 'description' with a large text area containing 'This is initial test 123444', and 'reguser' with 'test'. At the bottom of the dialog box are three buttons: 'OK', '수정' (Modify), and '삭제' (Delete).

□ The Spring Framework - Reference Documentation

- <http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/>
- 이전 버전 참조
 - <http://static.springframework.org/spring/docs/2.5.x/reference/>
 - <http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/>

Q & A

감사합니다.