장희정강사
8253jang@daum.net

# Spring Web MVC

# MVC란 ?

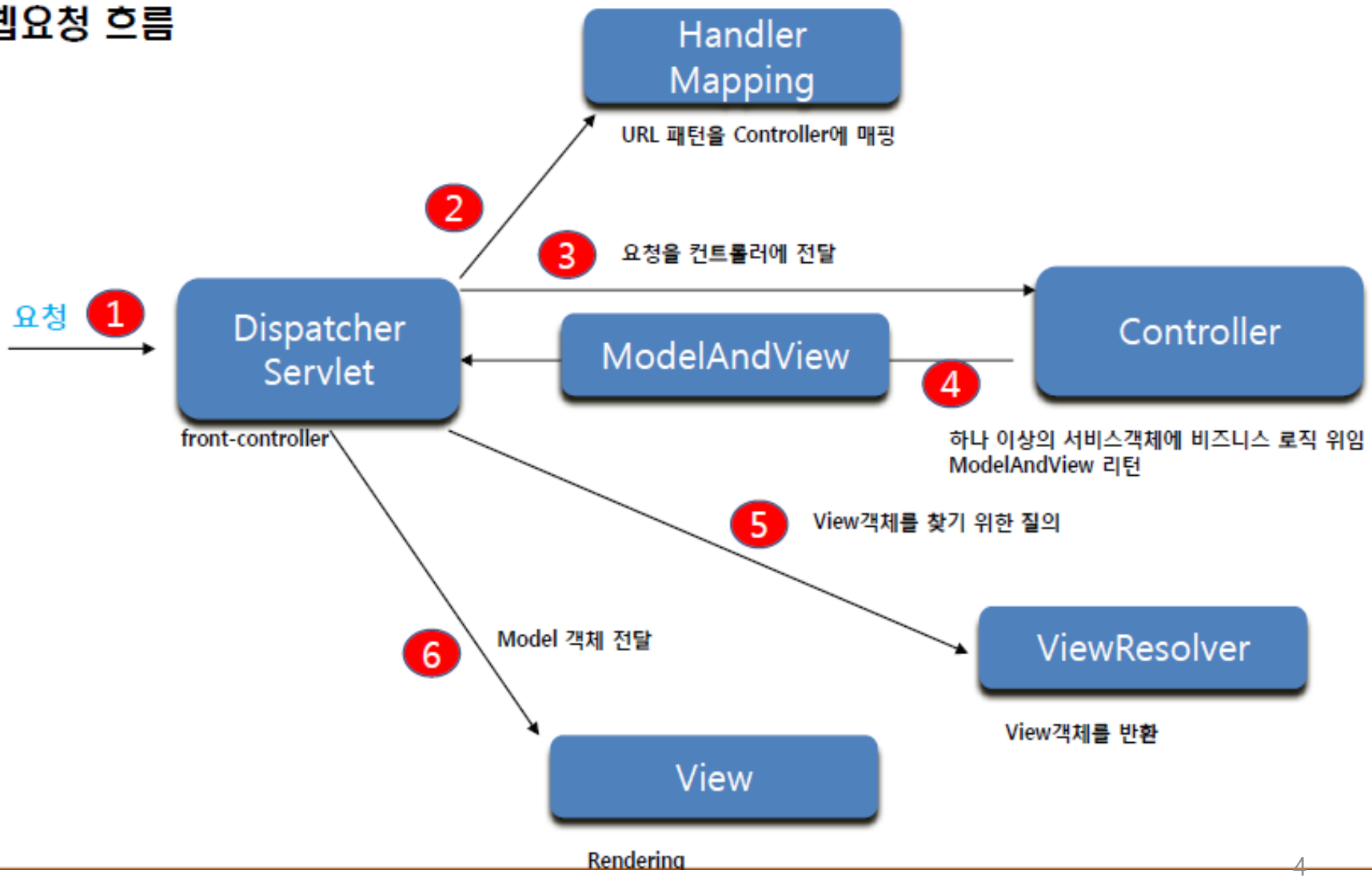
**MVC(Model – View – Contrlloer) 패턴은 코드를 기능에 다라 Model, View, Controller 3가지 요소로 분리한다.**
**MVC패턴은 UI코드와 비즈니스 코드를 분리 함으로써 종속성을 줄이고 재사용성을 높이고 보다 쉬운 변경이 가능하도록 한다.**

| | |
|---|---|
| Model | 어플리케이션의 데이터와 비즈니스 로직을 담는 객체이다. |
| View | Model의 정보를 사용자에게 표시한다. |
| Controller | Model과 view의 중계역할을 한다.<br>사용자의 요청을 받아 Model에 변경된 상태를 반영하고 응답을 위한 View를 선택한다. |

# Spring MVC의 핵심 Componet

- 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



# 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 준비

① Dynamic WebProject를 생성한다.

② WebContent/WEB-INF/lib 폴더에 관련 라이브러 추가한다.
(5.x 라이브러리, jstl.jar , standard.jar , servlet-api.jar)

③ web.xml 문서에 DispatcherServlet 등록한다.

④ web.xml문서에 등록된 서블릿이름-servlet.xml 문서 만든다.

# DispatcherServlet 등록

```
<servlet>
   <servlet-name>springWeb</servlet-name>
   <servlet-class>
     org.springframework.web.servlet.DispatcherServlet
   </servlet-class>
 </servlet>

  <servlet-mapping>
   <servlet-name>springWeb</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

WEB-INF 폴더에
springWeb-servlet.xml 만들기

# DispatcherServlet 등록

```
<servlet>
   <servlet-name>springWeb</servlet-name>
   <servlet-class>
     org.springframework.web.servlet.DispatcherServlet
   </servlet-class>
 <init-param>
     <param-name>contextConfigLocation</param-
name>
     <param-value>
       /WEB-INF/springConfig/springWeb-servlet.xml
   </param-value>
   </init-param>
 </servlet>
</web-app>
```

Servlet 설정 파일 원하는 폴더에 설정하기

# Controller관련 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

작성한 클래스에 @Controller를 붙여준다. 특정 클래스를 구현하거나 상속 할 필요없다.

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

@Controller
public class HelloController {

...
}
```

# @RequestMapping

**요청에대해 어떤 Controller, 어떤 메소드가 처리 할지를 mapping하기 위한 어노테이션이다.**

관련속성

| 이름 | 타입 | 설명 |
|---|---|---|
| value | String[] | URL 값으로 맵핑 조건을 부여한다.<br>@RequestMapping(value="/hello.do") 또는<br>@RequestMapping(value={"/hello.do", "/world.do" })와 같이 표기하며,<br>기본값이기 때문에 @RequestMapping("/hello.do")으로 표기할 수도 있다.<br>"/myPath/*.do"와 같이 Ant-Style의 패턴매칭을 이용할 수도 있다. |
| method | RequestMethod[] | HTTP Request 메소드값을 맵핑 조건으로 부여한다.<br>HTTP 요청 메소드값이 일치해야 맵핑이 이루어 지게 한다.<br>@RequestMapping(method = RequestMethod.POST)같은 형식으로 표기한다.<br>사용 가능한 메소드는 GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE이다 |

# @RequestMapping

관련속성

| | | |
|---|---|---|
| params | String[] | HTTP Request 파라미터를 맵핑 조건으로 부여한다.<br>params="myParam=myValue"이면 HTTP Request URL중에 myParam이라는 파라미터가 있어야 하고 값은 myValue이어야 맵핑한다.<br>params="myParam"와 같이 파라미터 이름만으로 조건을 부여할 수도 있고, "!myParam" 하면 myParam이라는 파라미터가 없는 요청 만을 맵핑한다.<br>@RequestMapping(params={"myParam1=myValue", "myParam2", "!myParam3"})와 같이 조건을 주었다면,<br>HTTP Request에는 파라미터 myParam1이 myValue값을 가지고 있고, myParam2 파라미터가 있어야 하고, myParam3라는 파라미터는 없어야 한다. |

# @RequestMapping

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

/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(){
        ...
    }
}
```

# @RequestMapping

/hello.do 요청이 오면 hello 메소드,

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

```
@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(){
		...
	}
}
```

# @RequestMapping

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. |

# @RequestParam

- 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. - SimpleFormContriller의 referenceData 메소드와 유사한 기능.

- 관련 속성

| 이름 | 타입 | 설명 |
|---|---|---|
| value | String | 바인드하려는 Model 속성 이름. |

# Spring 한글 인코딩 설정

Web.xml에 추가

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

# @SessionAttribute

- @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

기존의 계층형 Controller에 비해 유현한 메소드 파리미터,리턴값을 갖는다.

- 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 줄 수 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.

# @Controller 메소드

메소드 리턴타입

- **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.

# ViewResolver 등록

```
<bean class=“
 org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/jsp/"/> //결과페이지가 있을 위치
  <property name="suffix" value=".jsp"/> <!-- 확장자 지정하기 -->
</bean>
```

```
ModelAndView mv =new ModelAndView();
mv.setViewName("hello");
```

=> /WEB-INF/jsp/hello.jsp 뷰가 보여진다.

# 웹 어플리케이션을 위한 ApplicationContext 설정

Web.xml에 추가

```
<listener>
   <listener-class>
     org.springframework.web.context.ContextLoaderListener
  </listener-class>
 </listener>

 <context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>
     /WEB-INF/springConfig/commonServlet.xml
   </param-value>
 </context-param>
```

# 웹 어플리케이션을 위한 ApplicationContext 설정

실제로 ContextLoaderListener와 DispatcherServlet은 각각 WebApplicationContext객체를 생성한다.

ContextLoaderListener가 생성하는 WebApplicationContext는 웹 어플리케이션에서 루트 컨텍스트가 되며 DispatcherServlet이 생성하는 WebApplicationContext는 자식 컨텍스트가 된다.

이때 자식은 root가 제공하는 빈을 사용할수 있기 때문에 각각의 DispatcherServlet이 공통으로 필요로 하는 빈을 ContextLoaderListener를 이용하여 설정하는 좋다.

# 컨트롤러 클래스 자동 스캔

Servlet문서에 추가

```
<context:component-scan base-package="sist.test.exam"/>
```

만약, 컨트롤러 클래스 자동 스캔시 property를 통해 DI를 적용해야 한다면 메소드에 @Autowired를 선언한다.

# View이름을 Redirect View로 지정하는 방법

Controller를 실행 후 ViewResolver가 실행되는 것이 아니라 다른 Controller로 이동해야 하는 경우
ex) 게시물의 글을 등록 후 select하는 Controller로 이동

```
@RequestMapping(value="multiInsert.do")
public ModelAndView multi(UserList user, Model map){
     //B/L 실행
    return new ModelAndView("redirect:redirectTest.do");
}
```

```
@RequestMapping(value="multiInsert.do")
public String multi(UserList user, Model map){
     //B/L 실행
    return "redirect:redirectTest.do";
}
```

# View이름을 Redirect View로 지정하는 방법

Controller에서 Controller로 이동할 때 인수 넘기기
ex) get방식의 형태로 넘어간다.

```
@RequestMapping(value="multiInsert.do")
public ModelAndView multi(UserList user, Model map){

    map.addAttribute("no", 10);
    map.addAttribute("message", "안녕");

    return new ModelAndView("redirect:redirectTest.do");
}

@RequestMapping(value="/redirectTest.do")
public String redirectTest(UserList user, int no,String message){
    System.out.println(" no = " + no+" , message = " +message);

    return "insert_ok";
}
```

# List type의 property

```
<tr >
   <td align="center"><input type="checkbox" name="list[0].state"></td>
   <td><input type="text" name="list[0].id"></td>
   <td><input type="text" name="list[0].name"></td>
   <td><input type="text" name="list[0].age"></td>
</tr>
<tr>
   <td align="center"><input type="checkbox" name="list[1].state"></td>
   <td><input type="text" name="list[1].id"></td>
   <td><input type="text" name="list[1].name"></td>
   <td><input type="text" name="list[1].age"></td>
</tr>

<tr>
   <td align="center"><input type="checkbox" name="list[2].state"></td>
   <td><input type="text" name="list[2].id"></td>
   <td><input type="text" name="list[2].name"></td>
   <td><input type="text" name="list[2].age"></td>
</tr>
```

List type의 property

| 선택 | 아이디 | 이름 | 나이 |
|---|---|---|---|
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| 전송 | | | |

# List type의 property

```
@RequestMapping(value="multiInsert.do")
public ModelAndView multi(UserList user){
    System.out.println("multi 실행됨.");
```

```
public class UserList {
   ArrayList<UserListModel> list;

    public ArrayList<UserListModel> getList() {
        return list;
    }

    public void setList(ArrayList<UserListModel> list) {
        this.list = list;
    }

}
```

# List type의 property

```
@RequestMapping(value="multiInsert.do")
public ModelAndView multi(UserList user){
    System.out.println("multi 실행됨.");
```

```
public class UserListModel {
    private String id;
    private String name;
    private int age;
    private boolean state;
```

```
public class UserList {
   ArrayList<UserListModel> list;

    public ArrayList<UserListModel> getList() {
        return list;
    }

    public void setList(ArrayList<UserListModel> list) {
        this.list = list;
    }

}
```

# PathVariable어노테이션을 이용한 URI템플릿 설정

- 일반적으로 DB테이블에 type= notice/ faq / qna 로 보통 만듦

- 요청은 /boardDetail.jsp?type=notice&pk=1023 =>http방식 단순화 /boardDetail/notice/1023

# PathVariable어노테이션을 이용한 URI템플릿 설정

- 보통 http://hostName/contextPath/hello.do?id=dev.won 와 같은 URI 요청 패턴을 구성한다.

  만약 http://hostName/contextPath/hello/dev.won 과 같이 URI를 구성해야 한다면 과연 어떻게 Controller와 매핑할것인지 알아보자.

  위의 URI와 같이 구성하는 경우는 다음과 같은 경우가 있을것으로 보인다.

  1. REST 서비스를 위하여 구성하는 경우
  2. SEO 최적화(permanent link, canonical url등) 를 위한 URL 단순 구성

- 이런경우에 PathVariable어노테이션을 이용하면 아주 간편하게 URI 템플릿을 구성하여 Controller와 매핑 할 수 있다.

# PathVariable어노테이션을 이용한 URI템플릿 설정

이때 다음의 두가지만 추가로 작업해주면 된다.<br>

1. @RequestMapping 어노테이션의 값으로 {템플릿변수} 를 사용한다.

2. @PathVariable 어노테이션을 이용해서 {템블릿변수} 와 동일한 이름을 갖는 파라미터를 요청처리 메소드에 추가한다.

요청 : http://localhost:8000/springMVC3Exam/board/notice/list/3.do

```
@RequestMapping(value="/{board}/{boardType}/{action}/{idx}.do")
public String action(@PathVariable  int idx ,
                     @PathVariable  String boardType,
                     @PathVariable String action,
                     @PathVariable String board,
                     Model model){

    model.addAttribute("boardType", boardType);
    model.addAttribute("idx" , idx);
    model.addAttribute("action" , action);
    model.addAttribute("board" , board);

    System.out.println("action [1] !");
    return "pathResult";//뷰이름
}
```

# PathVariable어노테이션을 이용한 URI템플릿 설정

```java
/*
 * 요청 : http://localhost:8000/springMVC3Exam/test/action.do
 * */
@RequestMapping("/{data1}/action.do")
public String action2(@PathVariable("data1")
                      String data ,
                      Model model){
    model.addAttribute("param1", data);

    System.out.println("action2=>"+ data);
    return "pathResult";
}
```

# PathVariable어노테이션을 이용한 URI템플릿 설정

```
/**
 * PathVariable 어노테이션을 이용한 URI 템플릿 설정.
 * 만약 {템플릿변수} 의 이름과 요청처리 메소드의 파라미터 변수 이름이 다르다면, PathVariable("템플릿변수명")으로 지정하여 매핑 할수 있다.
 * @param type
 * @param action
 * @param no
 * @param model
 * @return
 *
 * 요청 : http://localhost:8000/springMVC3Exam/rest/notice/list/3
 *   */
@RequestMapping(value="/{boardType}/{action}/{idx}")
public String action2(@PathVariable("boardType")
                        String type,
                      @PathVariable
                        String action,
                      @PathVariable("idx")
                        int no ,
                     Model model){
    model.addAttribute("boardType", type);
    model.addAttribute("action" , action);
    model.addAttribute("idx" , no);

    System.out.println("action [2] !");
    return "pathResult";
}
```

# @ResponseBody 어노테이션 사용

웹서비스 또는 REST 요청의 응답 내용은 JSP 에 의해 렌더링 되는 HTML이 아닌 XML 문자열 자체이다.

@RequestBody 어노테이션과 @ResponseBody 어노테이션은 각각 HTTP 요청 몸체를 자바 객체로 변환하고 자바 객체를 HTTP 응답 몸체로 변환해주는데 사용한다.

1. @RequestBody 어노테이션은 HTTP 요청 몸체를 자바 객체로 전달 받을 수 있다.
2. @ResponseBody 어노테이션은 자바객체를 HTTP응답 몸체로 전송할 수 있다.

# @ResponseBody 어노테이션 사용

```jsp
<form action="<%=request.getContextPath()%>/bodyAnnotation/action.do" method="post">
    <input type="text" name="id" value="dev.won" size="3"><br>
    <input type="text" name="pw" value="1234" size="3"><br>
    <input type="submit">
</form>
```

```java
@RequestMapping("/bodyAnnotation/action.do")
@ResponseBody //=> 뷰를 거치지 말고 요기에서 응답을 바로 클라이언트에게 전달.
//http://localhost:8888/SpringMvc3.xSample2/bodyAnnotation/action.do
public String action(){
    //리턴타입이 String이면 리턴.jsp였으나 @ResponseBody 이기에 문자열을 바로 클라이언트에게 전송
    return "응답데이터";
}
```

# @ResponseBody 어노테이션 사용

```
<!-- @ResponseBody 사용시 변환값에 한글이 있으면 깨지는 현상을 해결하기 위한 설정 조치 -->
<bean
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" >
   <property name="messageConverters">
      <list>
         <bean class =
"org.springframework.http.converter.StringHttpMessageConverter">
            <property name = "supportedMediaTypes">
               <list>
                  <value>text/plain;charset=UTF-8</value>
               </list>
            </property>
         </bean>
      </list>
   </property>
</bean>
```

# 예외처리

@RequestMapping 메서드는 모든 타입의 예외를 발생 시킬 수있다. 예외를 발생시킬 경우 웹 브라우저는 500 응답코드와 함께 서블릿 컨테이너가 출력한 에러 페이지가 출력된다.
예외타입에 따라 스프링 MVC와 연동된 뷰를 이용해서 에러 페이지를 출력 할 수 있다. 예외발생시 사용자에게 보여줄 특정 페이지를 만들어 출력한다. (공통의 예외를 한 페이지에서 처리 가능.)

- 처리 방법
  - @ExceptionHandler 어노테이션을 이용한 예외처리
  - *SimpleMappingExceptionResolver 클래스를 이용한 예외처리*

# 예외처리

- @ExceptionHandler 어노테이션을 이용한 예외처리
  => @ExceptionHandler 메소드를 만든 Controller영역에서만 유효함.

```
@ExceptionHandler(ArrayIndexOutOfBoundsException.class)
public String arithmetic(ArrayIndexOutOfBoundsException e){
    System.out.println("ArrayIndexOutOfBoundsException 실행됨 : " + e);
    return "array";
}
```

# 예외처리

- *SimpleMappingExceptionResolver 클래스를 이용한 예외처리*
  => xml문서에서 설정함.(오류 종류에 따라 다른 페이지 이동)

```
<!-- Exception 등록 -->
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
   <property name="exceptionMappings">
      <props>
        <prop key="java.lang.ArithmeticException">
           exception
        </prop>
        <prop key="java.lang.NullPointerException">
           exception2
        </prop>
      </props>
   </property>
</bean>
```

# 예외처리

- *SimpleMappingExceptionResolver 클래스를 이용한 예외처리 와* @ExceptionHandler 어노테이션을 함께 사용하고자 할 때 xml문서에 아래와 같이 bean을 선언한다.

```
<!-- SimpleMappingExceptionResolver를 등록하였을 경우 기본으로 되어있던 어노테이션을 사용하려면 bean등록해줘야 함.  -->
 <bean
 class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerExceptionResolver"/>
```

# 파일 업로드

-파일 업로드를 위해서 *MultipartResolver 를 설정한다.*
 => multipartResolver의 경우에는 반드시 bean의 id를 multipartResolver로 지정해야 한다.

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver" />
```

# 파일 업로드

```
<h2> 파일 업로드 기능</h2>
<form action="upload.do"
      method="post"
      enctype="multipart/form-data">
  이름 : <input type="text" name="name"/><p>
  파일 첨부 : <input type="file" name="file"/>
<input type="submit" value="전송"/>
</form>
```

# 파일 업로드

```
@RequestMapping(value="/upload.do")
public ModelAndView upload(String name ,
        @RequestParam(value="file") MultipartFile file){

    String fName = file.getOriginalFilename();//첨부된파일이름
    long size = file.getSize();//첨부된파일용량

    //파일저장
    try {
        file.transferTo(new File("D:/uploadSave/"+fName));
    } catch (Exception e) {
        System.out.println(e+"=> 파일저장실패");
    }

    ModelAndView m = new ModelAndView();
    m.addObject("fName", fName);
    m.addObject("size", size);
    m.addObject("name", name);
    m.setViewName("uplod");
    return m;
}
```

# 파일 다운로드

Controller가 리턴하는 ViewName과 동일한 이름을 갖는 Bean을(Bean id="[BeanName]" 뷰 객체로 사용한다.
주로 CustomView 클래스를 뷰로 사용해야 하는 경우에 사용.

```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.BeanNameViewResolver" >
        <property name="order"  value="1" />
</bean>
```

```
  <!-- ViewResolver를 등록 -->
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
     <property name="prefix" value="/WEB-INF/view/" />
     <property name="suffix" value=".jsp" />
     <property name="order" value="2"/>
</bean>
```

# 파일 다운로드

```java
@RequestMapping(value="/down.do")
public ModelAndView down(String pathName){
    System.out.println("pathName =" + pathName);

    File f = new File("D:/upLoadSave/"+pathName);

    ModelAndView m =new ModelAndView();
    m.addObject("downPath", f);
    m.setViewName("customView");//customView를 bean의 이름인식

    return m;
}
```

```xml
<!-- 다운로드 기능을 담당하는 클래스 선언 -->
<bean id="customView" class="soa.spring3.down.DownLoadCustomView"/>
```

# 파일 다운로드

```
public class DownLoadCustomView extends AbstractView{
    public DownLoadCustomView(){
        this.setContentType("application/download;chatset-UTF-8");

    }
    @Override
    protected void renderMergedOutputModel(Map<String, Object> model,
            HttpServletRequest request, HttpServletResponse response) throws Exception {
        File file = (File) model.get("downPath");//downPath

        response.setContentType(this.getContentType());
        response.setContentLength((int)file.length());
```