**spring MVC 필기**

**Spring 프로젝트 생성.**

1. Dynamic web project를 원하는 이름으로 생성한다.

2. Project를 우클릭해서 configure -> Convert to Maven Project를 통해 Maven 프로젝트로 변경한다. (pom.xml에서 ctrl + A 로 문서 전체를 선택한 후 ctrl + shit + F 를 누른다. (들여쓰기 정리))

3. pom.xml에 다음 라이브러리를 설정한다.

servlet-api

jsp-api

jstl

springWebMVC

라이브러리 설정은 MVNRepository라는 곳에서 각각 검색하며 붙여넣기한다. (pom.xml 참고)

**URL주소**

경로1/경로2/경로3에서 첫번째 경로를 Context Path라고 한다. 한 서버에서 각각의 웹 어플리케이션을 구분하기 위한 이름이며 폴더의 이름 자체가 Context Path가 된다.

예를 들어, SpringTest1 프로젝트에서 작업을 하고 있다면 SpringTest1이 Context Path가 되는 것이다.

**URL mapping**

@RequestMapping(value="/test6",method=RequestMethod.***GET***)

**public** String test6() {

**return** "sub2/test6";

}

requestMapping의 value에 나오는 주소는 실제 주소가 요청을 받는 주소이다.

index.html에서 /test6으로 링크를 걸면 test6()이 실행되고 sub2/test6.jsp를 보여준다.

실제 주소는 return 하는 sub2/test6이므로 sub2폴더안에 test6이 있어야 실행된다.

**요청 방식**

RequestMapping은 요청 주소와 요청 방식을 지정 가능하다.

get 방식과 post 방식이 있는데 동시에 하는 방법은

@RequestMapping(value="/test5",method={RequestMethod.***GET***,RequestMethod.***POST***}) 이렇게 method에 둘 다 써주는 것이다.

**파라미터 추출하기**

클라이언트 요청으로 전달하는 파라미터 데이터를 전달 받는 여러 방식이 있다.

spring MVC는 필요한 데이터나 객체를 주입을 받아 사용한다.

HttpServletRequest : servlet, jsp에서 쓰는 방식 getParameter 등을 이용해서 데이터를 받아온다.

WebRequest : Spring MVC에서 제공하는 HttpServeletRequest를 확장한 것이다.

@PathVariable : 데이터가 요청 주소에 있을시 값을 주입받는 방식

Restful API 서버 프로그래밍에서 사용하는 방식

주소/값1/값2/값3과 같이 값이 온다..

@GetMapping("/test4/{data1}/{data2}")

**public** String test4(@PathVariable String data1,

@PathVariable String data2) {

System.***out***.println(data1 + " " + data2 + " " + data3);

**return** "result";

}

@RequestParam : 파라미터 데이터를 변수로 직접 주입받고 형 변환도 가능하다.

@GetMapping("/test5")

**public** String test5(@RequestParam **int** data1, @RequestParam **int** data2, @RequestParam **int**[] data3)

처럼 변수와 data 이름을 똑같이 받을 수도 있고

@RequestParam(value = "data1") **int** val1, 처럼 변수와 이름이 다르게 할 수도 있다.

**객체로 파라미터 주입받기.**

map이나 list로 Requestparam을 받으면 String으로만 받을 수 있다.

@ModelAttribute

파라미터를 객체로 받을 수 있다.

전달되는 파라미터의 이름과 동일한 프로퍼티에 자동으로 주입된다.

이런 객체를 커맨드 객체라고한다.

**public** String test2(@ModelAttribute DataBean bean)

이런식으로 사용하며 @ModelAttribute를 생략해서 DataBean bean만 써도 실행 가능하다.

DataBean은 입력으로 들어오는 매개변수와 같은 이름으로 멤버변수와 getter, setter가 있어야한다.

**ViewResolver**



View Resolver : Controller에서 전달 받은 jsp의 이름을 토대로 jsp를 찾아 선택하고, 전달받은 데이터를 분석해 응답 결과를 만들어낸다.

view의 이름 지정법과 데이터 전달하는 방법으로 여러가지가 있다.

1.HttpServletRequest 객체를 이용해 전달.

@GetMapping("/test2")

**public** String test2(HttpServletRequest request) {

request.setAttribute("data1", 100);

request.setAttribute("data2", 200);

**return** "test2";

}

2. Model 객체를 이용해서 전달

@GetMapping("/test3")

**public** String test3(Model model) {

model.addAttribute("data1", 100);

model.addAttribute("data2", 200);

**return** "test3";

}

3. ModelAndView 객체를 이용해서 전달

@GetMapping("/test4")

**public** ModelAndView test4(ModelAndView mv) {

mv.addObject("data1",500);

mv.addObject("data2",600);

mv.setViewName("test4");

**return** mv;

}

**커맨드 객체**

클라이언트가 전달해주는 파라미터 데이터를 주입받기 위해 사용하는 객체

@PostMapping("/test1")

**public** String test1(@ModelAttribute DataBean bean) {

System.***out***.println(bean.getData1() + " " + bean.getData2());

**return** "test1";

}

(@ModelAttribute 생략 가능)

커맨드 객체는 자동으로 HttpServletRequest 객체에 저장되어 JSP로 전달된다.

전달한 데이터를 view에서 사용하려면 아래와 같이 사용해야한다.

<h3>data1 : ${requestScope.dataBean.data1 }</h3>

requestScope다음의 영역은 클래스의 첫글자를 소문자로하면 된다. 그 다음 멤버 변수 이름은 그대로 사용 가능하다.

requestScope다음의 영역 이름을 바꾸고 싶으면 아래와 같이 바꾸면 된다.

@PostMapping("/test2")

**public** String test2(@ModelAttribute("testData") DataBean bean) {

**return** "test2";

}

그럼 다음과 같이 view에서 사용할 수 있다.

<h3>data1 : ${requestScope.testData.data1 }</h3>

**Form 커스텀 태그**

Model 객체에 있는 값을 form 요소에 주입할 수 있다.

아래 태그를 jsp 파일 맨 위에 추가해준다.

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

아래 방식 중 맘에드는 방식으로 Controller를 설정한다.

**public** String test2(UserDataBean bean) {

bean.setUser\_name("홍홍");

**public** String test3(@ModelAttribute("testBean") UserDataBean bean) {

bean.setUser\_name("홍홍");

**public** String test4(Model model) {

UserDataBean bean = **new** UserDataBean();

bean.setUser\_name("홍홍");

model.addAttribute("test\_user2",bean);

**return** "test4";

}

그에 따라 jsp의 form 태그도 아래 방식으로 구현한다.

<form:form modelAttribute=*"userDataBean"* action=*'result'*>

이름 : <form:input path=*'user\_name'*/><br/>

<form:form modelAttribute=*"testBean"* action=*'result'*>

이름 : <form:input path=*'user\_name'* />

<form:form modelAttribute=*"test\_user2"* action=*'result'*>

이름 : <form:input path=*'user\_name'* />

**Form 요소**

form 커스텀 태그를 사용해 사용자 입력폼을 만들 수 있다.

주로 정보 수정 페이지를 구성할 떄 사용한다.

**<form:form> 태그**

<form> 태그를 생성한다.

modelAttribute 속성 : form 태그 내의 입력 요소들에 적용될 value값을 가진 객체 이름. (Bean 객체 이름)

action : 요청할 주소를 설정한다.

method : 생략시 post 방식이다. get 방식도 가능하다

<form:button> : submit 버튼을 생성한다.

disabled 속성 : true로 세팅하면 버튼을 누를 수 없게 비활성화한다.

<form:hidden> hidden 태그르 ㄹ생성한다.

<form:input> text 타입의 input 태그 생성.

<form:password> password타입의 input 태그 생성

<form:textarea> textarea 타입의 input 태그 생성

select, option, checkbox, radiobutton 등의 태그들이 있다.

FormElement1과 FormElement2 프로젝트를 참고하자.

**Redirect and Forward**

Redirect : 서버가 클라이언트에게 요청할 주소를 응답결과로 전달하는 것이다.

클라이언트는 응답결과로 받은 요청주소를 직접 요청하게 된다.

주소창의 주소가 변경된다. HttpServletRequest 객체는 소멸 후 새로 생성되고 HttpSession 객체는 그대로 유지된다.

forward : 서버상에서 코드의 흐름이 이동하는 것

브라우저는 다른 곳으로 흐름이 이동한 것을 모르기 때문에 주소창이 바뀌지 않는다.

HttpServletRequest 객체와 HttpSession 객체도 그대로 유지된다.

**RequestScope**

Requset : 브라우저에 의해 요청이 발생하면 브라우저는 서버에 요청과 관련된 정보를 전송한다.

이를 받은 서버는 요청 정보들을 보관하기 위해 HttpServletRequest 객체를 생성해 정보를 저장한다. 이 객체는 응답 결과가 전송될 때까지 유지되며 사용가능하다.

RequestScope : 요청이 발생해 응답결과를 전송할 떄까지 Request 객체를 사용 가능한데, 이 사용범위를 RequestScope 라고한다.

HttpServletRequest 객체에 데이터나 객체를 저장하고 RequestScope 내에서 사용이 가능하다.

클라이언트의 요청을 받을 떈 HttpServletRequest, Model, ModelAndView, @ModelAttribute("bean1") DataBean1 등 여러 방식으로 받을 수 있으나 그 요청을 다른 곳으로 forward 해준다면 forward 된 곳에선 HttpServletRequest 객체만을 이용해서 전달받을 수 있다. 다른 방식으로 받으면 null값이 된다. 참고로 redirect는 request객체가 소멸되니 전달할 수 없다.

아래 메소드가 forward 받은 mapping 예시이다.

@GetMapping("/result1")

**public** String result1(HttpServletRequest request) {

String data1 = (String)request.getAttribute("data1"); **return** "result1";

}

**RequestScope 빈 주입**

Bean을 정의할 때 request scope로 정의하면 요청이 발생할 때마다 Bean 객체가 생성되어 자동으로 주입된다.

하지만 주입된 Bean 객체가 request 영역에 저장되는 것은 아니다.

따라서 Model 객체에 Bean 객체를 주입해야한다.

Xml로 Bean을 설정한 후 byName으로 주입받는 경우에만 request 영역에 자동으로 저장된다.

🡪 <bean class=*'kr.co.beans.DataBean2'* id=*'requestBean2'* scope=*'request'*/> 이렇게 Bean을 설정하면 request 영역에 자동으로 저장된다.

Java 방식은 @RequestScope를 사용,

Xml 방식은 bean을 정의할 떄 scope="request"로 설정한다.

**Session Scope**

Session : 브라우저가 최초로 서버에 요청을 하게 되면 브라우저당 하나씩 메모리 공간을 서버에 할당하게 된다.

이 영역은 브라우저 당 하나씩 지정되며, 새로운 요청이 발생해도 같은 공간을 사용하게 된다. 이런 메모리 공간을 session이라고 한다.

브라우저를 종료할 때까지 서버에서 사용할 수 있다.

SessionScope : 브라우저가 최초의 요청을 발생시키고 브라우저를 닫을 떄 까지를 SessionScope라고 부른다. session 영역에 저장된 데이터와 객체를 자유롭게 사용가능하다.

forward나 redirect를 해도 session은 그대로 사용이 가능하다.

@SessionAttribute : session 영역에 있는 객체를 사용하려 할 때 메소드의 매개변수로 @SessionAttrubute를 설정하면 session 영역에 있는 빈 객체를 주입받을 수 있다.

@SessionAttributes : @ModelAttribute를 통해 주입받은 Bean은 자동으로 request 영역에 저장된다.

@ModelAttribute를 통해 주입받는 Bean을 @SessionAttributes로 지정해 놓이면 request 영역이 아닌 session 영역에 저장되고 session 영역으로 부터 주입 받을 수 있습니다.

주의할 점은 @ModelAttribute를 활용하여 객체를 생성해 반환하는 메서드를 반드시 작성해 줘야 합니다.

@Controller

@SessionAttributes({"sessionBean1", "sessionBean2"}) 🡪 이렇게 객체 여러 개일 땐 배열로

**public** **class** TestController {

@ModelAttribute("sessionBean1")

**public** DataBean1 sessionBean1() {

**return** **new** DataBean1();

}

-> 객체를 반환하는 메소드를 반드시 작성해야한다.

**public** String result5(@ModelAttribute("sessionBean1") DataBean1 sessionBean1) 이렇게 사용가능

**Session scope 빈 객체**

Bean을 정의할 때 session scope로 정의하면 브라우저가 서버에 최초의 요청을 보낼 때 Bean 객체가 주입된다. 그러나 주입만 되는것이지 session 영역에 저장되지는 않는다.

Java 방식 : @SessionScope

XML 방식 : scope="session"을 사용한다.

**Application Scope**

서버가 가동될 때부터 서버가종료되는 시점까지의 범위

처음 요청부터 응답까지의 범위 : requestScope

브라우저의 최초 요청부터 브라우저가 종료할때까지 : sessionScope

서버가 가동될때부터 종료될때까지 : applicationScope

application Scope동안 사용할 수 있는 메모리 영역이 만들어지며 ServletContext라는 클래스 타입의 객체로 관리된다. -> 어플리케이션 영역에 저장되면 브라우저에 관계없이 모든 곳에서 사용가능하다.

ServletContext에 저장된 데이터, 객체는 서버가 종료되기 전까지 동일한 메모리 공간을 사용한다.

ServletContext : HttpServletRequest 객체로부터 추출이 가능하다.

Controller에서 주입받을 수 있다.

서버가 종료될때까지 사용가능하다.

**AppicationScope 빈 주입**

Bean을 정의할 떄 application scope로 정의하면 서버가 가동될 때 자동으로 주입된다.

주입된 bean은 주입만 이루어지므로 application 영역에 저장되지는 않는다.

서버가 가동될 떄 자동 주입되는 것이므로 @Lazy를 설정하지 않아도 된다.

Java : @ApplicationScope를 사용

XML : bean 정의할 때 scope="application"으로 설정한다.

**Cookie**

사용자 웹 브라우저에 저장되는 데이터

요청이 발생했을 때 브라우저는 쿠키에 저장된 정보를 서버에 전달한다.

만일 응답 결과로 쿠키 정보가 전달되면 웹 브라우저가 쿠키에 저장하게 된다.

쿠키는 사용자 브라우저에 저장되는 것이므로 브라우저가 전달해 줄 떄만 쿠키 정보를 사용할 수 있다.

**cookie 저장**

서버측 코드로 쿠키에 데이터를 저장할 수 있는 방법은 없다.

브라우저로 보낼 **응답 결과**에 쿠키 정보를 담아 보내면 **브라우저에 의해** 쿠키가 저장된다.

Spring MVC에서 쿠키 저장은 Servlet/Jsp 에서 사용하는 방법으로 처리할 수 있다.

servlet/jsp 에서는 쿠키 정보를 배열로 받아서 하나하나 일일이 비교하며 사용해야 한다.

Spring에서는 쿠키 정보를 주입받아서 사용이 가능하다. 매개변수에 @CookieValue("cookie1") String cookie1처럼 선언하면 cookie1을 바로 사용이 가능하다.

**Properties**

프로그램 실행 중 절대 변하지 않는 값들을 properties 파일에 작성하고 가져다 사용할 수 있다.

property editor

properties 파일에 한글을 작성하면 unicode 형식의 문자열로 변환된다.

editor를 설치해서 해결할 수 있다.

property를 굉장히 많이 사용하기 때문에 거의 필수로 editor를 설치한다.

xx.properties라는 이름으로 파일을 저장하고 원하는 파일에서 주입한다. 주입은

@PropertySource("/WEB-INF/properties/data1.properties")

@PropertySource("/WEB-INF/properties/data2.properties")

이렇게 각각 써주거나

@PropertySources({

@PropertySource("/WEB-INF/properties/data1.properties"),

@PropertySource("/WEB-INF/properties/data2.properties")

})

이렇게 묶어서 써줄 수 있다.

properties 파일 내의 data를 쓰는 법은

@Value("${aaa.a1}")

**private** **int** a1;

이렇게 사용하면 된다.

**Message**

Properties에서 작성한 값을 jsp에서 사용하려면 properties 파일을 Message로 등록해야한다.

다국어 처리 또한 가능해진다.

**MessageSource**

MessageSource 객체를 이용해 properties 파일을 등록해주면 Message로 등록할 수 있다.

ReloadableResourceBundleMessageSource를 사용하면 일정 시간마다 갱신된다.

Message를 사용하려면 ServletAppContext 파일에서

@Bean

**public** ReloadableResourceBundleMessageSource messageSource() {

ReloadableResourceBundleMessageSource res = **new** ReloadableResourceBundleMessageSource();

//res.setBasename("/WEB-INF/properties/data1");

res.setBasenames("/WEB-INF/properties/data1","/WEB-INF/properties/data2"); //properties를 생략해야 된다.

**return** res;

}

이렇게 메시지를 등록하고

사용하려는 곳에서

@Autowired

ReloadableResourceBundleMessageSource res;

이렇게 주입한 후

String a1 = res.getMessage("aaa.a1", **null**, **null**);

를통해 사용하면 된다.

jsp에서 사용하려면

<h3>aaa.a1 = <spring:message code=*'aaa.a1'*/></h3>

와 같이 사용하면 된다.