다이나믹 웹 프로젝트 생성 후 - 컨피겨 - 컨버트 투 메이븐 프로젝트

콘솔창처럼 서버창에서 톰캣서버 더블클릭후 모듈탭에서

해당 프로젝트의 path 를 / 로 만들어서 모든 경로가능으로 변경

클라이언트에서 처음 넘어오는 구간 : front controller

리퀘스트 처리담당하는 컨트롤러를 찾기위한 존재 : HandlerMapping

실제 리퀘스트를 처리하는 로직 : Controller

뷰에 전달해야하는 데이터는 model이라는 객체에 담아서 전달

뷰리졸버는 컨트롤러가 반환한 결과를 어떤 뷰를 통해서 처리하는게 좋을지 해석하는 역할

jsp 를 이용해 생성 , 만들어진 응답은 디스패처서블릿을 통해 전송

@EnableWebMvc // 스프링 mvc 프로젝트의 관련된 설정을 하는 구조

"/WEB-INF/views/",".jsp”

→ /WEB-INF/views/ 에 있는 파일을 jsp 파일로 읽도록 한다

registy.addResourceHandler("/\*\*").addResourceLocations("/resources/");

→ 정적파일을 관리한다

Filter 는 java/servlet을 import 한다

SpringConfigClass 는 web.xml 에서 하는 일을 자바에서 하도록 설정하기위해 만든파일이다

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

→ 모든 경로에서 get 방식으로 접근한다

@GetMapping, @PostMapping : get,post 방식의 URI 주소 지정

httpservletRequest를 변수로 사용

배열값을 request 로 받을때 는 getParameterValues로 받는다

@PostMapping("/t2") public String a(HttpServletRequest request) {

String s1=request.getParameter("d1");

String s3[] = request.getParametervalues("d3");

System.out.println(s1);

for(String s:s3){ System.out.println(s1); }

return "final"; // .jsp가 생략된 상태로 final.jsp 페이지로 이동된다

}

@GetMapping("/t3/{d1}/{d2}/{d3}")

**public** String a3(@PathVariable **int** d1,@PathVariable **int** d2,

@PathVariable **int** d3) {

System.***out***.println(d1);

System.***out***.println(d2);

System.***out***.println(d3);

**return** "final";

}

@PathVariable : url 경로에 변수를 넣게 할 수 있고 사용 할 수있다

============ 8월 3일 =======================================

servlet-context.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xsi:schemaLocation="<http://www.springframework.org/schema/mvc><http://www.springframework.org/schema/mvc/spring-mvc.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context.xsd>" xmlns:context="<http://www.springframework.org/schema/context>" xmlns:beans="<http://www.springframework.org/schema/beans>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xmlns="<http://www.springframework.org/schema/mvc>">

<annotation-driven />

<!-- 패키지 내부 클래스 중 controller가지고있는 클래스 로딩작업 -->

<context:component-scan

base-package="aa.bb.cc.controller" />

<!-- controller 메소드에서 반환하는 문자열 앞 뒤에 붙힐 경로정보 세팅 -->

<beans:bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<beans:property name="prefix" value="/WEB-INF/views/" />

<beans:property name="suffix" value=".jsp" /><!-- jsp확장자

생략가능하게해준다 -->

</beans:bean>

<resources location="/resources/" mapping="/\*\*" />

</beans:beans>

<!-- 스프링 구동 관련 설정 -->

roo-context.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xsi:schemaLocation="<http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans.xsd>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xmlns="<http://www.springframework.org/schema/beans>">

<!-- 이 파일에 정의된 객체들을 빈이라고 하며 스프링영역에 생성된다 객체간의 의존성처리가 됨 -->

</beans>

<!-- 스프링 구동 관련 설정 -->

web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app xsi:schemaLocation="<http://xmlns.jcp.org/xml/ns/javaee><http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xmlns="<http://xmlns.jcp.org/xml/ns/javaee>" version="4.0">

<!-- tomcat 구동관련 설정 -->

<servlet-mapping>

<servlet-name>appServlet</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<servlet><!-- 기본서블릿 지정 (요청정보분석 후 컨트롤러선택) -->

<servlet-name>appServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/config/servlet-context.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup><!-- 로딩순서 -->

</servlet>

<context-param><!-- bean 정의할 xml파일 지정 -->

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/config/root-context.xml</param-value>

</context-param>

<listener>

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

</listener>

<!-- 파라미터 인코딩 필터 설정 -->

<filter>

<filter-name>encodingFilter</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>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

</web-app>

xml 에서 주석이 있으면 에러가 발생 할 수있음

ModelAttribute를 사용한 파라미터는 무조건 model에 담아서 전달

파라미터로 전달된 데이터를 다시화면에서 사용해야할때 유용하게 쓰인다

@ModelAttribute

//파라미터로 넘긴 값 d1,d2,d3 가 Data 클래스의 필드값에 자동 주입된다(이름이 같아야한다)

자동 주입된값을 el 언어로 사용할때는 클래스에서 접근할 경우 소문자로 시작한다

requestScope.data.d2 또는 data.d2 와 같이 사용

@ModelAttribute("spring") 로 변수에 이름을 지정할 수 있고 el언어 사용시

spring.클래스로 값을 사용할 수 있다

@RequestScope

스프링mvc 에서의 빈관리

브라우저가 보낸 요청을 httpservletrequest가 저장하며 응답결과를 브라우저로 전송할때까지 유지되면 사용가능하고 이 범위가 RequestScope 이다

${d1}, ${requestScope.d1} 같은 결과를 가진다

Model 클래스

addAttribute 로 값을 설정하면 model에 저장되는게 아니라 httpservletrequest에 저장된다

ModelAndView 클래스

addObject 로 값을 저장하며 setViewName으로 값을 넘길 페이지를 지정

model 클래스와 마찬가지로 httpservletrequest 를 이용한다

스프링에서 제공하는 form 커스텀 태그 활용법

@GetMapping("/t3") public String t3(@ModelAttribute("aa") Data dd) { dd.setName("namgoungminsu"); dd.setId("ngms"); dd.setPw("1234"); dd.setAdr("seoul"); dd.setEmail("[ngms@aa.com](mailto:ngms@aa.com)");

@GetMapping("/t3")

public String t3(@ModelAttribute("aa") Data dd) {

dd.setName("namgoungminsu");

dd.setId("ngms");

dd.setPw("1234");

dd.setAdr("seoul");

dd.setEmail("[ngms@aa.com](<mailto:ngms@aa.com>)");

return "t3";

}

<form:form modelAttribute="data"> 이름 : <form:input path="name"/><br> 아이디 : <form:input path="id"/><br> </form:form>

// public String t3(Model m){ Data dd=new Data(); dd.setName("namgoungminsu"); dd.setId("ngms"); dd.setPw("1234"); dd.setAdr("seoul"); dd.setEmail("[ngms@aa.com](mailto:ngms@aa.com)"); m.addAttribute("spring",dd);

return "t4";

} //위와 같은 방식의 form:form 을 이용할 수 있다

form:button => submit 속성을 가지고 있는 버튼

HTML버튼 기본선택값(checked같은기능)을

다르게 구현하는법(java web으로?)

@GetMapping("/t1")

public String a1(Data dd,Model m) {

dd.setS1("in");

dd.setS2("in");

dd.setS3("in");

dd.setS4("in");

String[] arr = {"out","up"};

dd.setS5(arr);

dd.setS6(arr);

dd.setS7(arr);

dd.setS8(arr);

dd.setS9("in");

dd.setS11("in");

dd.setS10("in");

dd.setS12("in");

String[] ary = {"down","center","in"};

m.addAttribute("ary",ary);

ArrayList<String> ary3 = new ArrayList<>();

ary3.add("in");

ary3.add("out");

ary3.add("down");

m.addAttribute("ary3",ary3);

return "t1";

}

<form:form modelAttribute="data"> <form:select path="s1"> <form:option value="down">down</form:option> <form:option value="center">center</form:option> <form:option value="in">in</form:option> </form:select> <br> <br> <form:select path="s2"> <form:options items="${ary}"/> </form:select>

<form:select path="s3">

<form:options items="${ary3}"/>

</form:select>

<br>

<br>

<form:checkbox path="s5" value="in"/>in

<form:checkbox path="s5" value="out"/>out

<form:checkbox path="s5" value="up"/>up

<br>

<br>

<form:checkboxes items="${requestScope.ary }" path="s9"/>

<br>

<br>

<form:radiobutton path="s10" value="in"/>in

<form:radiobutton path="s10" value="out"/>out

<form:radiobutton path="s10" value="center"/>center

<br>

<br>

<form:radiobuttons path="s11" items="${ary }"/>

</form:form>

Redirect 또는 forward 로 작업시 이동할 페이지가 마찬가지로 맵핑이 되어있어야한다

@GetMapping("/t2")

**public** String t2() {

**return** "forward:/s2";

}

@GetMapping("/s2")

**public** String s2() {

**return** "s2";

}

아래와 같이 맵핑되있을 때 final1 에서 d1을 el언어로출력하기

${requestScope.d1 }

@GetMapping("/t1")

**public** String t1(HttpServletRequest request) {

request.setAttribute("d1", "in");

**return** "forward:/final1";

}

@GetMapping("/final1")

**public** String f1(HttpServletRequest request) {

String str = (String) request.getAttribute("d1");

System.***out***.println(str);

**return** "final1";

}