**@Configuration 에서 @Bean Data 와 @Bean(”b1”) Data2를 선언하고 사용하기**

@Autowired 를 통해 자동주입되도록함

@Resource(name="b1") //@Autowired,@Qualifier도 사용가능 Data2 d2;

@GetMapping("/t2") public String t2() { d1.setD1("in"); d1.setD2("out");

d2.setD3("up");

d2.setD4("down");

return "forward:/final2";

}

@GetMapping("final2")

public String f2(Model m) {

m.addAttribute("d1",d1);

m.addAttribute("d2",d2);

return "final2";

}

// @Component 를 사용해서 Bean을 생성할 경우 ComponentScan으로 패키지를 지정해줘야한다

자바 파일이 아닌 xml 파일을 설정파일로 할 경우

root-context.xml 파일에서 사용할 bean을 선언해야하며

scope=”request” 는 실행과 동시에 bean 객체가 생성되기때문에 data 객체의 autowired가 작동되기 위해 요청마다주입되는 @Lazy 를 부여

root-context.xml 파일

<bean class="aa.bb.cc.beans.Data" scope="request"/> <bean id="b1" class="aa.bb.cc.beans.Data2" scope="request"/>

SpController 파일

@Autowired @Lazy Data d1;

servlet-context.xml 파일 ( @Component 파일 스캔위한 설정)

<context:component-scan base-package="aa.bb.cc.beans"/>

**redirect 에서 session과 request의 차이**

@GetMapping("/t2") public String t2(HttpServletRequest request, HttpSession session) { session.setAttribute("d1", "in"); request.setAttribute("d1", "in"); return "redirect:/final1"; } @GetMapping("final1") public String f1(HttpServletRequest request,HttpSession session) { String d1 = (String) request.getAttribute("d1"); System.out.println("request : " +d1);

String d2 = (String) session.getAttribute("d1");

System.out.println("session : " +d2);

return "final1";

}

// redirect 에서 request는 요청처리 후 새로운 요청을 만드는 방식이므로 request : null 로 출력되고

session은 유지가 되므로 session : in 으로 출력된다

forward 로 할경우에는 request 도 유지된다

새로운 요청이 발생하면 @SessionAttributes({"session1","session2"}) 이름으로 등록되어져 있는 메소드가 자동으로 호출됨 (public Data session1())

Data1이 반환하는 객체의 주소값을 session1이라는 이름으로 세션영역에 저장을 하게됨

public String a4(@ModelAttribute("session1") Data bean) 로 주입되게 된다. => 원래는 request영역인데 session영역이라고 선언함

**@Bean,@SessionScope,@Autowired,@Resource 사용하기**

@Bean @SessionScope public Data d1() { return new Data(); }

@Bean("b1")

@SessionScope

public Data2 d2() {

return new Data2();

}

@Autowired

Data d1;

@Resource(name="b1")

Data2 d2;

맵핑을 2개로 나눠서 t1에서 data,data2를 set 해주고

final1에서 Model를 사용해서 값을 담으면

final1.jsp 에서는 requestScope 으로 불러올 수 있다

@GetMapping("/t1") public String t1() { d1.setD1("in"); d1.setD2("out"); d2.setD1("down"); d2.setD2("up");

return "t1";

}

@GetMapping("/final1")

public String f1(Model m) {

m.addAttribute("data1",d1);

m.addAttribute("data2",d2);

return "final1";

}

final1.jsp에서

${requestScope.data1.d1 }

${requestScope.data1.d2 }

${data2.d1 }

${data2.d2 } //이렇게 값을 가져올 수 있다

Eclipse 에 Properties Editor 설치

<http://propedit.sourceforge.jp/eclipse/updates>

WEB-INF 폴더 밑에 propertise 폴더 생성후 File 로 생성하며 확장자는 .properties이다

사용할 컨트롤러에 어노테이션으로 지정해줘야한다

@PropertySources({@PropertySource("/WEB-INF/properties/d1.properties"),

@PropertySource("/WEB-INF/properties/d2.properties")})

프로퍼티 파일을 Autowired 로 사용 하기위해서는 빈으로 등록할 때

ReloadableResourceBundleMessageSource 로 반환 받으며

setBasenames()로 프로퍼티파일을 지정해줘야 한다

//프로퍼티파일을 메세지로 등록

@Bean

**public** ReloadableResourceBundleMessageSource messageSource() {

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

res.setBasenames("/WEB-INF/properties/d1","/WEB-INF/properties/d2");

**return** res;

}

@Validation 유효성 검사

기준을 설정해서 기준에서 벗어난 값들을 잡는다

@Size(min=3, max=10),@Max(100) 등이 있음

jsp 페이지에서 taglib으로 spring태그를 사용하며

<spring:hasBindErrors> 로 입력된 객체의 값을 검사한다

Interface 인 Errors 를 사용해서 오류메세지를 출력하도록 한다

@Size(min=3, max=10)

**private** String d1;

@Max(100)

**private** **int** d2;

<form action=*"po"* method=*"post"*>

d1:<input type=*"text"* name=*"d1"*><br>

<spring:hasBindErrors name=*"data1"*>

<c:if test=*"*${errors.hasFieldErrors('d1') }*"*>

${errors.getFieldError('d1').defaultMessage }<br>

</c:if>

</spring:hasBindErrors>

d2:<input type=*"text"* name=*"d2"*><br>

<spring:hasBindErrors name=*"data1"*>

<%

**if**(errors.hasFieldErrors("d2")){

errors.getFieldError("d2").getDefaultMessage();

}

%>

</spring:hasBindErrors>

<button type=*"submit"*>확인</button>

</form>

결과 출력 페이지에서 아래와 같이 값을 불러올 수 있다

${requestScope.data.d1 }

<br>

${data.d2 }