

1. MVC개요
 - 1)Model-View-Controller 패턴의 개념
 - 소프트웨어 공학에서 사용되는 아키텍처 패턴
 - 주 목적은 Business logic과 Presentation logic을 분리하기 위함
 - 이 패턴을 통해, User Interface로부터 Business logic을 분리하여 applicaton의 시각적 요소나 그 이면에서 실행되는 Business logic을 서로 영향없이 쉽게 고칠 수 있는 application을 만들 수 있음.
 - Model : Application의 정보(Data, Business logic 포함)
 - View : User에게 제공할 화면(Presentation logic)
 - Controller : Model과 View사이의 상호 작용을 관리
 - 2)MVCPattern.png 참조
2. 각각의 Component의 역할
 - 1)Model Component
 - Data 저장소(ex:DB)와 연동하여 사용자가 입력한 데이터나 사용자에게 출력할 data를 다루는 역할
 - 여러 개의 data 변경 작업(추가, 변경, 삭제)을 하나의 작업으로 묶는 Tansaction을 다루는 역할
 - DAO 클래스, Service 클래스에 해당
 - 2)View Component
 - Model이 처리한 data나 그 작업 결과를 가지고 User에게 출력할 화면을 만드는 역할
 - 생성된 화면은 Web Browser가 출력하고, View Component는 HTML과 CSS, JavaScript를 사용하여 Web Browser가 출력할 UI 생성
 - HTML과 JSP를 사용하여 작성
 - 3)Controller Component
 - Client의 요청을 받았을 때 그 요청에 대해 실제 업무를 수행하는 Model Component를 호출하는 역할
 - Client가 보낸 data가 있다면, Model을 호출할 때 전달하기 쉽게 data를 적절히 가공하는 역할
 - Model이 업무 수행을 완료하면, 그 결과를 가지고 화면을 생성하도록 View에게 전달(Client 요청에 대해 Model과 View를 결정하여 전달)
 - Servlet과 JSP를 사용하여 작성
3. Model2 Architecture
 - 1)Model1 : Controller의 역할을 JSP가 담당
 - 2)Model2 : Controller의 역할을 Servlet이 담당
 - Model2 Architecture.png 참조
4. Model2 Architecture 호출 순서
 - 1)Web Browser가 Web Application 실행을 요청하면, Web Server가 그 요청을 받아서 Servlet Container(ex:Tomcat Server)에게 넘겨준다.
 - Servlet Container는 URL을 확인하여 그 요청을 처리할 Servlet을 찾아서 실행한다.
 - 2)Servlet은 실제 업무를 처리하는 Model Java 객체의 Method를 호출한다.
 - 만약 Web Browser가 보낸 Data를 저장하거나 변경해야 한다면, 그 Data를 가공하여 VO객체를 생성하고, Model 객체의 Method를 호출할 때 인자 값으로 넘긴다.
 - Model 객체는 일반적으로 POJO로 된 Service, DAO 일 수 있다.
 - 3)Model객체는 JDBC를 사용하여 매개변수로 넘어온 값 객체를 Database에 저장하거나, Database로부터 질의 결과를 가져와서 VO 객체로 만들어 반환한다.
 - 4)Servlet은 Model 객체로부터 반환 받은 값을 JSP에 전달한다.
 - 5)JSP는 Servlet으로부터 전달받은 값 객체를 참조하여 Web Browser가 출력할 결과 화면을 만들고, Web Browser에 출력함으로써 요청 처리를 완료한다.
 - 6)Web Browser는 Server로부터 받은 응답 내용을 화면에 출력한다.
5. Front Controller Pattern Architecture
 - 1)Front Controller Pattern Architecture.jpg 참조
 - 2)Front Controller는 Client가 보낸 요청을 받아서 공통적인 작업을 먼저 수행
 - 3)Front Controller는 적절한 세부 Controller에게 작업을 위임
 - 4)각각의 Application Controller는 Client에게 보낼 View를 선택해서 최종 결과를 생성하는 작업
 - 5)Front Controller pattern은 인증이나 권한 체크처럼 모든 요청에 대하여 공통적으로 처리해야 하는 logic이 있을 경우 전체적으로 Client의 요청을 중앙 집중적으로 관리하고자 할 경우에 사용
6. Spring MVC 개념
 - 1)특징
 - Spring은 DI나 AOP 같은 기능뿐만 아니라 Servlet 기반의 Web 개발을 위한 MVC Framework를 제공
 - Spring MVC나 Model2 Architecture와 Front Controller pattern을 Framework 차원에서 제공
 - Spring MVC Framework는 Spring을 기반으로 하고 있기 때문에 Spring이 제공하는 Transaction 처리나 DI 및 AOP등을 손쉽게 사용
 - 2)Spring MVC와 Front Controller Pattern
 - 대부분의 MVC Framework들은 Front Controller pattern을 적용해서 구현

59 -Spring MVC도 Front Controller 역할을 하는 DispatcherServlet이라는 class를 계층의 맨 앞단에 놓고,
 Server로 들어오는 모든 요청을 받아서 처리하도록 구성

60 3)예외가 발생했을 때 일관된 방식으로 처리하는 것도 Front Controller의 역할

61

62 7. DispatcherServlet Class

63 1)Front Controller pattern 적용

64 2)web.xml에 설정

65 3)Client로부터의 모든 요청을 전달 받음

66 4)Controller나 View와 같은 Spring MVC의 구성요소를 이용하여 Client에게 service를 제공

67

68 8. Spring MVC의 주요 구성 요소

69 1)DispatcherServlet : Client의 요청을 받아서 Controller에게 Client의 요청을 전달하고, Return한
 결과값을 View에게 전달하여 알맞은 응답을 생성

70 2)HandlerMapping : URL과 요청 정보를 기준으로 어떤 Handler 객체를 사용할지 결정하는 객체이며,
 DispatcherServlet은 하나 이상의 Handler Mapping을 가질 수 있음.

71 3)Controller : Client의 요청을 처리한 뒤, Model를 호출하고 그 결과를 DispatcherServlet에게 알려 줌.

72 4)ModelAndView : Controller가 처리한 data 및 화면에 대한 정보를 보유한 객체

73 5)View : Controller의 처리 결과 화면에 대한 정보를 보유한 객체

74 6)ViewResolver : Controller가 return한 View 이름을 기반으로 Controller 처리 결과를 생성할 View를 결정

75

76 9. Spring MVC의 주요 구성 요소의 요청 처리 과정

77 -Spring MVC Process.png 그림 참조

78 1)Client의 요청이 DispatcherServlet에게 전달된다.

79 2)DispatcherServlet은 HandlerMapping을 사용하여 Client의 요청을 처리할 Controller를 획득한다.

80 3)DispatcherServlet은 Controller 객체를 이용하여 Client의 요청을 처리한다.

81 4)Controller는 Client 요청 처리 결과와 View 페이지 정보를 담은 ModelAndView 객체를 반환한다.

82 5)DispatcherServlet은 ViewResolver로부터 응답 결과를 생성할 View 객체를 구한다.

83 6)View는 Client에게 전송할 응답을 생성한다.

84

85 10. Spring MVC 기반 Web Application 작성 절차

86 1)Client의 요청을 받는 DispatcherServlet을 web.xml에 설정

87 2)Client의 요청을 처리할 Controller를 작성

88 3)Spring Bean으로 Controller를 등록

89 4)JSP를 이용한 View 영역의 코드를 작성

90 5)Browser 상에서 JSP를 실행

91

92

93 11. Lab

94 1)Package Explorer > right-click > New > Spring Legacy Project

95 2)Select Spring MVC Project

96 3)Project name : 1207 > Next

97 4)Enter a topLevelPackage : com.javasoft.biz > Finish

98 5)Open src/main/java/com.javasoft.biz/HomeController.java

99 6)project right-click > Run As > Run on Server > Finish

100 7)http://localhost:8080/biz/

101

102 Hello world!

103

104 The time on the server is 2017년 12월 5일 (화) 오후 11시 40분 58초.<--원래 한글 깨짐

105

106 8)한글 깨짐을 수정하는 것은 src/main/webapp/WEB-INF/views/home.jsp에서

107 <%@ page session="false" pageEncoding="UTF-8" contentType="text/html";
 charset=UTF-8"%>로 수정

108

109 9)처리 순서

110 -Web Browser의 요청을 web.xml의 <url-pattern>/</url-pattern>를 통해 /의 요청을 받는다.

111 -servlet-name이 appServlet인 servlet-class는
 org.springframework.web.servlet.DispatcherServlet이다.

112 -이 DispatcherServlet는 로딩하면서 /WEB-INF/spring/appServlet/servlet-context.xml를 파라미터로
 초기화한다.

113 -servlet-context.xml에서 <context:component-scan
 base-package="com.javasoft.springmvcdemo" />를 통해 scan을 base-package에서 한다.

114 -com.javasoft.springmvcdemo에 @Controller를 찾는다.

115 -@Controller가 있는 HomeController.java에서 @RequestMapping(value = "/", method =
 RequestMethod.GET)가 설정되어 있는 메소드인 public String home(Locale locale, Model model)를
 찾는다. 왜냐하면 지금 Browser가 요청한 메소드는 GET이고, 요청 경로는 /이기 때문이다.

116 -serverTime을 설정하고 model의 addAttribute메소드를 통해 View에게 사용할 값을 저장한다. 그리고

return "home"을 통해 jsp 파일이름을 반환한다.

-다시 servlet-context.xml에서 ViewResolver는 prefix가 /WEB-INF/views/이고, suffix가 .jsp이며
방금 반환된 jsp파일 이름인 home은 prefix + 파일 이름 + suffix를 하면 /WEB-INF/views/home.jsp가
된다.

-그래서 home.jsp를 Browser에게 전송한다. 이때 JSP는 Model에 저장된 servetTime을 함께 View에
출력하게 된다.

10)Context name 변경하기

-server.xml에서 다음과 같이 수정한다.

```
<Context docBase="1207" path="/demo" reloadable="true"
source="org.eclipse.jst.jee.server:1207"/>
```

-수정 후 restart 하면 http://localhost:8080/biz --> http://localhost:8080/demo 로 변경됨

12. Lab : resources folder 이용하기

1)그림 경로 알아내기

-다운로드받은 이미지를 src/main/webapp/resources/images/에 넣는다.

-home.jsp에 아래 코드를 추가한다.

```
<p></p>
```

-이미지가 잘 나온다.

2)이미지 경로 변경

-이미지 경로를 src/main/webapp/images/로 이동.

-하지만 이렇게 하면 이미지가 보이지 않는다.

-왜냐하면, servlet-context.xml에서 resource의 경로는 <resources mapping="/resources/**"
location="/resources/" />이기 때문.

-즉, 기본적으로 resources 폴더 아래에서 resource를 찾는다.

3)<resources />추가

-resources폴더처럼 하위에 images폴더를 생성하고 이미지를 넣고 home.jsp에 아래의 코드를 추가한다.

```
<p></p>
```

```
<p></p>
```

-하지만 아래의 이미지는 보이지 않는다.

-왜냐하면 새로 추가한 images 폴더는 servlet-context.xml에서 설정하지 않았기 때문.

-이미지를 보이게 하기 위해 servlet-context.xml에 아래의 코드를 추가한다.

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

```
<resources mapping="/images/**" location="/resources/" />
```

13. Lab : Controller Class 제작하기

1)제작순서

-@Controller를 이용한 class 생성

-@RequestMapping을 이용한 요청 경로 지정

-요청 처리 메소드 구현

-View 이름 return

-src/main/java/com.javasoft.biz.UserController class 생성

```
@Controller
```

```
public class UserController {
```

2)요청 처리 메소드 생성

```
package com.javasoft.biz;
```

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

```
import org.springframework.ui.Model;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RequestMethod;
```

```
import org.springframework.web.servlet.ModelAndView;
```

```
@Controller
```

```
public class UserController {
```

```
    @RequestMapping("/view")
```

```
    public String view(Model model){
```

```

178         model.addAttribute("currentDate", new java.util.Date());
179         return "view";    // /WEB-INF/views/view + .jsp
180     }
181 }

```

-src/main/webapp/WEB-INF/views/view.jsp 생성

```

185 <%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
186 <!DOCTYPE html>
187 <html>
188     <head>
189         <meta charset="UTF-8">
190         <title>Insert title here</title>
191     </head>
192     <body>
193         <h1>view.jsp 입니다.</h1>
194         현재 날짜와 시간은 ${currentDate} 입니다.
195     </body>
196 </html>

```

3)View에 Data 전달

```

201 package com.javasoft.biz;
202
203 import org.springframework.stereotype.Controller;
204 import org.springframework.ui.Model;
205 import org.springframework.web.bind.annotation.RequestMapping;
206 import org.springframework.web.bind.annotation.RequestMethod;
207 import org.springframework.web.servlet.ModelAndView;
208
209 @Controller
210 public class UserController {
211
212     @RequestMapping("/view")
213     public String view(Model model){
214         model.addAttribute("currentDate", new java.util.Date());
215         return "view";    // /WEB-INF/views/view + .jsp
216     }
217
218     @RequestMapping("/bbs/view")
219     public String bbs_view(Model model){
220         String [] array = {"Apple", "Mango", "Lemon", "Grape"};
221
222         model.addAttribute("fruits", array);
223
224         return "/bbs/view";    // /WEB-INF/views/bbs/view + .jsp
225     }
226 }

```

-src/main/webapp/WEB-INF/views/bbs/view.jsp 생성

```

230 <%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
231 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
232 <!DOCTYPE html>
233 <html>
234 <head>
235     <meta charset="UTF-8">
236     <title>Insert title here</title>
237 </head>
238 <body>
239     <h2>/bbs/view.jsp</h2>
240     <ul>과일 종류
241     <c:forEach items="${fruits}" var="fruit">
242         <li>${fruit}</li>

```

```

243         </c:forEach>
244     </ul>
245 </body>
246 </html>
247
248 -http://localhost:8080/biz/view --> /view.jsp
249 -http://localhost:8080/biz/bbs/view --> /bbs/view.jsp
250

```

251 4)View에 ModelAndView 객체로 Data 전달

```

252
253 @RequestMapping(value = "/demo", method = RequestMethod.GET)
254 public ModelAndView demo() {
255
256     ModelAndView mav = new ModelAndView();
257     mav.addObject("userid", "javasoft");
258     mav.addObject("passwd", "12345678");
259     mav.setViewName("/demo");
260     return mav;
261 }
262
263 -src/main/webapp/WEB-INF/views/demo.jsp 생성
264
265 <%@ page language="java" contentType="text/html; charset=UTF-8"
266     pageEncoding="UTF-8"%>
267 <!DOCTYPE html>
268 <html>
269     <head>
270         <meta charset="UTF-8">
271         <title>Insert title here</title>
272     </head>
273     <body>
274         아이디 : ${userid} <br />
275         패스워드 : ${passwd}
276     </body>
277 </html>

```

278 5)Controller class에 @RequestMapping 적용

```

279 -src/main/java/com.javasoft.biz.StudentController.java 생성
280
281 package com.javasoft.biz;
282
283 import org.springframework.stereotype.Controller;
284 import org.springframework.web.bind.annotation.RequestMapping;
285 import org.springframework.web.bind.annotation.RequestMethod;
286 import org.springframework.web.servlet.ModelAndView;
287
288 @Controller
289 @RequestMapping("/bbs")
290 public class StudentController {
291
292     @RequestMapping(value="/get", method = RequestMethod.GET)
293     public ModelAndView getStudent() {
294
295         ModelAndView mav = new ModelAndView();
296         mav.setViewName("/bbs/get"); // /WEB-INF/views/bbs/get.jsp
297         mav.addObject("name", "한지민");
298         mav.addObject("age", 25);
299         return mav;
300     }
301 }
302
303 -src/main/webapp/WEB-INF/views/bbs/get.jsp
304 <%@ page language="java" contentType="text/html; charset=UTF-8"
305     pageEncoding="UTF-8"%>
306 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
307     "http://www.w3.org/TR/html4/loose.dtd">
308 <html>

```

```

307 <head>
308 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
309 <title>Insert title here</title>
310 </head>
311 <body>
312     학생 이름 : ${name} <br />
313     학생 나이 : ${age}
314 </body>
315 </html>
316
317

```

318 14. Lab : Form Data 처리하기

- 319 1)Package Explorer > right-click > New > Spring Legacy Project
- 320 2)Select Spring MVC Project
- 321 3)Project name : 1207-1 > Next
- 322 4)Enter a topLevelPackage : com.javasoft.springwebdemo > Finish
- 323 5)src/main/java/com.javasoft.springwebdemo/RequestController.java 생성

```

324
325     package com.javasoft.springwebdemo;
326
327     import org.springframework.stereotype.Controller;
328     import org.springframework.ui.Model;
329     import org.springframework.web.bind.annotation.PathVariable;
330     import org.springframework.web.bind.annotation.RequestMapping;
331     import org.springframework.web.bind.annotation.RequestMethod;
332

```

```

333     @Controller
334     public class RequestController {
335

```

- 336 6)HttpServletRequest class 이용하기
- 337 -RequestController.java

```

338
339     @RequestMapping(value="/confirm", method=RequestMethod.GET)
340     public String confirm(HttpServletRequest request, Model model) {
341         String userid = request.getParameter("userid");
342         String passwd = request.getParameter("passwd");
343         String name = request.getParameter("name");
344         int age = Integer.parseInt(request.getParameter("age"));
345         String gender = request.getParameter("gender");
346
347         model.addAttribute("userid", userid);
348         model.addAttribute("passwd", passwd);
349         model.addAttribute("name", name);
350         model.addAttribute("age", age);
351         model.addAttribute("gender", gender);
352         return "confirm"; // /WEB-INF/views/confirm.jsp
353     }
354

```

355 -src/main/webapp/WEB-INF/views/confirm.jsp

```

356
357     <%@ page language="java" contentType="text/html; charset=UTF-8"
358     pageEncoding="UTF-8"%>
359     <!DOCTYPE html>
360     <html>
361     <head>
362         <meta charset="UTF-8">
363         <title>Insert title here</title>
364     </head>
365     <body>
366         아이디 : ${userid} <br />
367         비밀번호 : ${passwd} <br />
368         사용자 이름 : ${name} <br />
369         나이 : ${age} <br />
370         성별 : ${gender} <br />
371     </body>
372 </html>

```

```
-localhost:8080/springwebdemo/confirm?name=한지민&gender=여성&age=25&userid=javasoft&passwd=1234
```

7)@RequestParam annotation 이용하기

```
-RequestController.java
```

```
@RequestMapping(value="/confirm", method=RequestMethod.GET)
public String confirm(@RequestParam("userid") String userid,
                     @RequestParam("passwd") String passwd,
                     @RequestParam("name") String name,
                     @RequestParam("age") int age,
                     @RequestParam("gender") String gender ,Model model) {

    model.addAttribute("userid", userid);
    model.addAttribute("passwd", passwd);
    model.addAttribute("name", name);
    model.addAttribute("age", age);
    model.addAttribute("gender", gender);
    return "confirm"; // /WEB-INF/views/confirm.jsp
}
```

```
-src/main/webapp/WEB-INF/views/confirm.jsp
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Insert title here</title>
    </head>
    <body>
        아이디 : ${userid} <br />
        비밀번호 : ${passwd} <br />
        사용자 이름 : ${name} <br />
        나이 : ${age} <br />
        성별 : ${gender} <br />
    </body>
</html>
```

```
-localhost:8080/springwebdemo/confirm?name=한지민&gender=여성&age=25&userid=javasoft&passwd=1234
```

8)Data Commander 객체 이용하기1

```
-src/main/java/com.javasoft.vo.UserVO.java 생성
```

```
package com.javasoft.vo;

public class UserVO {
    private String userid;
    private String passwd;
    private String name;
    private int age;
    private String gender;
    public String getUserid() {
        return userid;
    }
    public void setUserid(String userid) {
        this.userid = userid;
    }
    public String getPasswd() {
        return passwd;
    }
    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }
}
```

```

435     }
436     public String getName() {
437         return name;
438     }
439     public void setName(String name) {
440         this.name = name;
441     }
442     public int getAge() {
443         return age;
444     }
445     public void setAge(int age) {
446         this.age = age;
447     }
448     public String getGender() {
449         return gender;
450     }
451     public void setGender(String gender) {
452         this.gender = gender;
453     }
454     @Override
455     public String toString() {
456         return "UserVO [userid=" + userid + ", passwd=" + passwd + ", name=" + name + ",
            age=" + age + ", gender="
457             + gender + "]\n";
458     }
459 }

```

460 -RequestController.java

```

461
462
463     @RequestMapping(value="/confirm", method=RequestMethod.GET)
464     public String confirm(@RequestParam("userid") String userid,
465         @RequestParam("passwd") String passwd,
466         @RequestParam("name") String name,
467         @RequestParam("age") int age,
468         @RequestParam("gender") String gender ,Model model) {
469
470         UserVO userVO = new UserVO();
471         userVO.setUserid(userid);
472         userVO.setPasswd(passwd);
473         userVO.setName(name);
474         userVO.setAge(age);
475         userVO.setGender(gender);
476
477         model.addAttribute("userVO", userVO);
478
479         return "confirm1"; // /WEB-INF/views/confirm1.jsp
480     }
481

```

482 -src/main/webapp/WEB-INF/views/confirm1.jsp

```

483
484     <%@ page language="java" contentType="text/html; charset=UTF-8"
485         pageEncoding="UTF-8"%>
486     <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
487     <c:set var="user" value="${userVO}"/>
488     <!DOCTYPE html>
489     <html>
490     <head>
491     <meta charset="UTF-8">
492     <title>Insert title here</title>
493     </head>
494     <body>
495         <h1>confirm1.jsp</h1>
496         <h2>사용자 정보</h2>
497         아이디 : ${user.userid} <br />
498         비밀번호 : ${user.passwd} <br />
499         이름 : ${user.name} <br />
500         나이 : ${user.age} <br />

```



```

500         성별 : ${user.gender}
501     </body>
502 </html>
503
504

```

9)Data Commander 객체 이용하기2

-RequestController.java

```

509     @RequestMapping(value="/confirm", method=RequestMethod.GET)
510     public String confirm(UserVO userVO) {
511
512         return "confirm2"; // /WEB-INF/views/confirm1.jsp
513     }
514

```

-src/main/webapp/WEB-INF/views/confirm2.jsp

```

517     <%@ page language="java" contentType="text/html; charset=UTF-8"
518     pageEncoding="UTF-8"%>
519     <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
520     <c:set var="user" value="${userVO}"/>
521     <!DOCTYPE html>
522     <html>
523     <head>
524     <meta charset="UTF-8">
525     <title>Insert title here</title>
526     </head>
527     <body>
528         <h1>confirm1.jsp</h1>
529         <h2>사용자 정보</h2>
530         아이디 : ${user.userid} <br />
531         비밀번호 : ${user.passwd} <br />
532         이름 : ${user.name} <br />
533         나이 : ${user.age} <br />
534         성별 : ${user.gender}
535     </body>
536 </html>

```

10)@PathVariable 이용하기

-RequestController.java

```

541     @RequestMapping(value="/confirm/{userid}", method=RequestMethod.GET)
542     public String confirm(@PathVariable String userid, Model model) {
543         model.addAttribute("userid", userid);
544         return "confirm3";
545     }
546

```

-src/main/webapp/WEB-INF/views/confirm3.jsp

```

549     <%@ page language="java" contentType="text/html; charset=UTF-8"
550     pageEncoding="UTF-8"%>
551     <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
552     "http://www.w3.org/TR/html4/loose.dtd">
553     <html>
554     <head>
555     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
556     <title>Insert title here</title>
557     </head>
558     <body>
559         유저 아이디 : ${userid}
560     </body>
561 </html>

```

<http://localhost:8080/springwebdemo/confirm?userid=jasvasoft> ==>
<http://localhost:8080/springwebdemo/confirm/jasvasoft>

563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626

15. Lab : @RequestMapping Parameter

1)GET 방식과 POST 방식

-src/main/java/com.javasoft.springwebdemo/HomeController.java

```
@RequestMapping(value="/login", method=RequestMethod.POST)
public String login(@RequestParam("userid") String userid,
    @RequestParam("passwd") String passwd,
    Model model) {

    model.addAttribute("userid", userid);
    model.addAttribute("passwd", passwd);
    return "login";
}
```

-src/main/webapp/resources/login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>로그인 폼</title>
</head>
<body>
    <form method="GET" action="/springwebdemo/login">
        아이디 : <input type="text" name="userid" /><br />
        패스워드 : <input type="password" name="passwd" /><br />
        <input type="submit" value="로그인하기" />
    </form>
</body>
</html>
```

-http://localhost:8080/springwebdemo/resources/login.html에서 submit 하면 405 error 발생

-왜냐하면 서로의 method가 불일치하기 때문

-해결방법

-src/main/java/com.javasoft.springwebdemo/HomeController.java 수정

-즉 login 메소드(요청 처리 메소드)의 이름은 같지만 파라미터의 타입과 리턴타입이 틀리기 때문에 Method Overloading 됨.

```
@RequestMapping(value="/login", method=RequestMethod.POST)
public String login(@RequestParam("userid") String userid,
    @RequestParam("passwd") String passwd,
    Model model) {

    model.addAttribute("userid", userid);
    model.addAttribute("passwd", passwd);
    return "login";
}

@RequestMapping(value="/login", method=RequestMethod.GET)
public ModelAndView login(@RequestParam("userid") String userid,
    @RequestParam("passwd") String passwd) {

    ModelAndView mav = new ModelAndView();
    mav.addObject("userid", userid);
    mav.addObject("passwd", passwd);
    mav.setViewName("login");
    return mav;
}
```

-src/main/webapp/WEB-INF/views/login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```

627 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
628 <title>Insert title here</title>
629 </head>
630 <body>
631     아이디 : ${userid} <br />
632     비밀번호 : ${passwd}
633 </body>
634 </html>
635

```

2) @ModelAttribute annotation 이용하기

- @ModelAttribute annotation을 이용하면 Data Commander 객체의 이름을 변경할 수 있다.
- src/main/webapp/resources/register.html

```

640 <!DOCTYPE html>
641 <html>
642 <head>
643 <meta charset="UTF-8">
644 <title>회원가입 폼</title>
645 </head>
646 <body>
647     <form method="GET" action="/springwebdemo/confirm">
648         아이디 : <input type="text" name="userid" /> <br />
649         비밀번호 : <input type="password" name="passwd" /> <br />
650         이름 : <input type="text" name="name" /> <br />
651         나이 : <input type="number" name="age" /> <br />
652         성별 : <input type="radio" name="gender" value="남성" /> 남성 &nbsp;&nbsp;&nbsp;
653             <input type="radio" name="gender" value="여성" /> 여성 <br />
654         <input type="submit" value="가입하기" />
655     </form>
656 </body>
657 </html>
658

```

- src/main/java/com.javasoft.springwebdemo/HomeController.java

```

661 @RequestMapping(value="/register", method=RequestMethod.POST)
662 public String register(@ModelAttribute("u") UserVO userVO) {    //userVO가 아니라 u로 변경
663
664     return "register";
665 }
666

```

- src/main/webapp/WEB-INF/views/register.jsp

```

669 <%@ page language="java" contentType="text/html; charset=UTF-8"
670 pageEncoding="UTF-8"%>
671 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
672 <c:set var="user" value="${u}" />
673 <!DOCTYPE html>
674 <html>
675 <head>
676 <meta charset="UTF-8">
677 <title>Insert title here</title>
678 </head>
679 <body>
680     <h1>사용자 정보</h1>
681     <ul>
682         <li>아이디 : ${user.userid}</li>
683         <li>비밀번호 : ${user.passwd}</li>
684         <li>이름 : ${user.name}</li>
685         <li>나이 : ${user.age}</li>
686         <li>성별 : ${user.gender}</li>
687     </ul>
688 </body>
689 </html>
690

```

- Spring에서 POST 방식으로 Data를 보낼 때 한글깨짐 현상 발생
- 해결점
- web.xml

```

693     <filter>
694     <filter-name>encodingFilter</filter-name>
695     <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
696     <init-param>
697         <param-name>encoding</param-name>
698         <param-value>UTF-8</param-value>
699     </init-param>
700 </filter>
701 <filter-mapping>
702     <filter-name>encodingFilter</filter-name>
703     <url-pattern>/*</url-pattern>
704 </filter-mapping>
705
706

```

3)redirect: 키워드 이용하기

```

708 -src/main/java/com.javasoft.springwebdemo/HomeController.java
709

```

```

710 @RequestMapping("/verify")
711 public String verify(HttpServletRequest request, Model model) {
712     String userid = request.getParameter("userid");
713     if(userid.equals("admin")) {    //만일 userid가 admin 이면 /admin으로 리다이렉트
714         return "redirect:admin";
715     }
716     //return "redirect:user";        //만일 userid가 admin 이 아니면 /user로 리다이렉트
717     return "redirect:http://www.naver.com";    //절대 경로도 가능
718 }
719
720 @RequestMapping("/admin")
721 public String verify1(Model model) {
722     model.addAttribute("authority", "관리자권한");
723     return "admin";
724 }
725
726 @RequestMapping("/user")
727 public String verify2(Model model) {
728     model.addAttribute("authority", "일반사용자");
729     return "user";
730 }
731

```

```

732 -/src/main/webapp/WEB-INF/views/admin.jsp
733

```

```

734 <%@ page language="java" contentType="text/html; charset=UTF-8"
735     pageEncoding="UTF-8"%>
736 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
737     "http://www.w3.org/TR/html4/loose.dtd">
738 <html>
739 <head>
740 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
741 <title>Insert title here</title>
742 </head>
743 <body>
744 <h1>관리자 페이지</h1>
745     권한 : ${authority}
746 </body>
747 </html>
748

```

```

746 -/src/main/webapp/WEB-INF/views/user.jsp
747

```

```

748 <%@ page language="java" contentType="text/html; charset=UTF-8"
749     pageEncoding="UTF-8"%>
750 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
751     "http://www.w3.org/TR/html4/loose.dtd">
752 <html>
753 <head>
754 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
755 <title>Insert title here</title>
756 </head>
757 <body>

```

```
756     <h1>일반 사용자 페이지</h1>
757     권한 : ${authority}
758 </body>
759 </html>
```

762 16. Lab : Form Data Validation

763 1)Package Explorer > right-click > New > Spring Legacy Project

764 2)Select Spring MVC Project

765 3)Project name : 1208 > Next

766 4)Enter a topLevelPackage : com.javasoft.biz > Finish

767 5)UserVO 객체 생성

768 -src/main/java/com.javasoft.vo package 생성

769 -src/main/java/com.javasoft.vo.UserVO class

```
770
771 package com.javasoft.vo;
772
773 public class UserVO {
774     private String name;
775     private int age;
776     private String userid;
777     public String getName() {
778         return name;
779     }
780     public void setName(String name) {
781         this.name = name;
782     }
783     public int getAge() {
784         return age;
785     }
786     public void setAge(int age) {
787         this.age = age;
788     }
789     public String getUserid() {
790         return userid;
791     }
792     public void setUserid(String userid) {
793         this.userid = userid;
794     }
795     @Override
796     public String toString() {
797         return "UserVO [name=" + name + ", age=" + age + ", userid=" + userid + "];"
798     }
799 }
```

801 6)Validator를 이용한 검증

802 -Data Command 객체에서 유효성 검사를 할 수 있다.

803 -UserValidator 객체 생성

804 -src/main/java/com.javasoft.biz.UserValidator class

```
805
806 package com.javasoft.biz;
807
808 import org.springframework.validation.Errors;
809 import org.springframework.validation.Validator;
810
811 import com.javasoft.vo.UserVO;
812
813 public class UserValidator implements Validator {
814
815     @Override
816     public boolean supports(Class<?> arg0) {
817         //검증할 객체의 클래스 타입 정보를 반환
818         return UserVO.class.isAssignableFrom(arg0);
819     }
820
821     @Override
822     public void validate(Object obj, Errors errors) {
```

```

823         System.out.println("검증시작");
824         UserVO userVO = (UserVO)obj;
825
826         String username = userVO.getName();
827         if(username == null || username.trim().isEmpty()) {
828             System.out.println("이름의 값이 빠졌습니다.");
829             errors.rejectValue("name", "No Value");
830         }
831
832         int usage = userVO.getAge();
833         if(usage == 0) {
834             System.out.println("나이의 값이 빠졌습니다.");
835             errors.rejectValue("age", "No Value");
836         }
837
838         String userid = userVO.getUserid();
839         if(userid == null || userid.trim().isEmpty()) {
840             System.out.println("아이디의 값이 빠졌습니다.");
841             errors.rejectValue("userid", "No Value");
842         }
843     }
844 }

```

-src/main/java/com.javasoft.biz/HomeController.java

```

848     @RequestMapping(value = "/register", method=RequestMethod.GET)
849     public String register() {
850         return "register";
851     }
852
853     @RequestMapping(value = "/register", method=RequestMethod.POST)
854     public String register(@ModelAttribute("userVO") UserVO userVO, BindingResult result) {
855         String page = "register_ok";
856         UserValidator validator = new UserValidator();
857         validator.validate(userVO, result);
858         if(result.hasErrors()) {
859             page = "register";
860         }
861         return page;
862     }
863 }

```

-src/main/webapp/WEB-INF/views/register.jsp

```

865     <%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
866     <!DOCTYPE html>
867     <html>
868     <head>
869     <meta charset="UTF-8">
870     <title>회원 가입 폼</title>
871     </head>
872     <body>
873         <form action="/biz/register" method="post">
874             Name : <input type="text" name="name" /><br />
875             Age : <input type="number" name="age" /><br />
876             ID : <input type="text" name="userid" /><br />
877             <input type="submit" value="PADiPMNELiMWSOS, °" />
878         </form>
879     </body>
880 </html>
881

```

-src/main/webapp/WEB-INF/views/register_ok.jsp

```

883     <%@ page language="java" contentType="text/html; charset=UTF-8"
884     pageEncoding="UTF-8"%>
885     <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
886     <c:set var="user" value="${userVO}" />
887     <!DOCTYPE html">
888     <html>
889     <head>

```

```

889 <meta charset="UTF-8">
890 <title>회원 가입 결과 창</title>
891 </head>
892 <body>
893 <ul>
894 <li>이름 : ${user.name}</li>
895 <li>나이 : ${user.age}</li>
896 <li>아이디 : ${user.userid}</li>
897 </ul>
898 </body>
899 </html>

```

7) ValidationUtils class를 이용한 검증

- ValidationUtils class는 validate() 메소드를 좀 더 편리하게 사용할 수 있게 해줌.
- UserValidator.java 수정

```

906 /*String username = userVO.getName();
907 if(username == null || username.trim().isEmpty()) {
908     System.out.println("이름의 값이 빠졌습니다.");
909     errors.rejectValue("name", "No Value");
910 }*/
911
912 ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "No Value");

```

8) @Valid와 @InitBinder 이용하기

- Spring Framework이 대신 검증해 줌
- mvnrepository에서 'hibernate validator'로 검색

```

918 <dependency>
919 <groupId>org.hibernate.validator</groupId>
920 <artifactId>hibernate-validator</artifactId>
921 <version>6.0.5.Final</version>
922 </dependency>

```

- pom.xml에 넣고 Maven Clean > Maven Install
- HomeController.java 수정

```

927 @RequestMapping(value = "/register", method=RequestMethod.POST)
928 public String register(@ModelAttribute("userVO") @Valid UserVO userVO, BindingResult
929 result) {
930     String page = "register_ok";
931     //UserValidator validator = new UserValidator();
932     //validator.validate(userVO, result);
933     if(result.hasErrors()) {
934         page = "register";
935     }
936     return page;
937 }
938
939 @InitBinder
940 protected void initBinder(WebDataBinder binder) {
941     binder.setValidator(new UserValidator());
942 }

```

17. Lab

- 1) In J2EE Perspective
- 2) Project Explorer > right-click > New > Dynamic Web Project
- 3) Project name : SpringWebDemo > Next > Check [Generate web.xml deployment descriptor] > Finish
- 4) Convert to Maven Project
 - project right-click > Configure > Convert to Maven Project > Finish
- 5) Add Spring Project Nature

-project right-click > Spring Tools > Add Spring Project Nature

6) 새로 생긴 pom.xml 파일에 필요한 library 추가 > Maven Clean > Maven Install

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.3.13.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>4.3.13.RELEASE</version>
  </dependency>
</dependencies>
```

7) Spring mvc library 검색 및 설치

-http://mvnrepository.com에서 'spring mvc'로 검색
-pom.xml에 추가

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>4.3.13.RELEASE</version>
</dependency>
```

-Maven Clean > Maven Install

8) Build path에 config folder 추가

-project right-click > Build Path > Configure Build Path > Select [Source] tab
-Click [Add Folder] > Select 현재 project > Click [Create New Folder...]
-Folder name : config > Finish > OK > Apply and Close
-Java Resources > config 폴더 확인

9) config folder에 beans.xml 파일 생성

-Spring Perspective로 전환

-beans.xml

-생성시 beans, context, mvc 체크

```
<?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:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="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-3.2.xsd">

  </beans>
```

10) ContextLoaderListener class 설정

-비즈니스 로직용의 스프링 설정 파일 (ex: applicationContext.xml)을 작성했기 때문에 listener로 ContextLoaderListener 클래스를 정의해야 한다.

-ContextLoaderListener 클래스는 스프링 설정 파일(디폴트에서 파일명 applicationContext.xml)을 로드하면 ServletContextListener 인터페이스를 구현하고 있기 때문에 ServletContext 인스턴스 생성 시(톰캣으로 어플리케이션이 로드된 때)에 호출된다. 즉, ContextLoaderListener 클래스는 DispatcherServlet 클래스의 로드보다 먼저 동작하여 비즈니스 로직층을 정의한 스프링 설정 파일을 로드한다.

-web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener - ContextLoaderListener]를 선택하면 아래의 코드가 자동 삽입


```

1013 <!-- needed for ContextLoaderListener -->
1014 <context-param>
1015     <param-name>contextConfigLocation</param-name>
1016     <param-value>location</param-value>
1017 </context-param>
1018
1019 <!-- Bootstraps the root web application context before servlet initialization -->
1020 <listener>
1021     <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
1022 </listener>
1023
1024 -아래 코드로 변환
1025 <context-param>
1026     <param-name>contextConfigLocation</param-name>
1027     <param-value>classpath:/config/beans.xml</param-value>
1028 </context-param>
1029
1030 11)DispatcherServlet Class 추가
1031 -web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherServlet -
DispatcherServlet declaration] 선택하면 아래의 코드가 자동 추가된다.
1032
1033 <!-- The front controller of this Spring Web application, responsible for handling all
application requests -->
1034 <servlet>
1035     <servlet-name>springDispatcherServlet</servlet-name>
1036     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
1037     <init-param>
1038         <param-name>contextConfigLocation</param-name>
1039         <param-value>location</param-value>
1040     </init-param>
1041     <load-on-startup>1</load-on-startup>
1042 </servlet>
1043
1044 <!-- Map all requests to the DispatcherServlet for handling -->
1045 <servlet-mapping>
1046     <servlet-name>springDispatcherServlet</servlet-name>
1047     <url-pattern>url</url-pattern>
1048 </servlet-mapping>
1049
1050 -아래의 코드로 변환
1051 <init-param>
1052     <param-name>contextConfigLocation</param-name>
1053     <param-value>classpath:/config/beans*.xml</param-value>
1054 </init-param>
1055
1056 <servlet-mapping>
1057     <servlet-name>springDispatcherServlet</servlet-name>
1058     <url-pattern>*.do</url-pattern>
1059 </servlet-mapping>
1060
1061 12)mvnrepository에서 'jstl'로 검색 후 설치
1062 -목록에서 2번째 : 1.2버전
1063
1064 <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
1065 <dependency>
1066     <groupId>javax.servlet</groupId>
1067     <artifactId>jstl</artifactId>
1068     <version>1.2</version>
1069 </dependency>
1070
1071 -pom.xml에 붙여넣고 Maven Clean > Maven Install
1072
1073 13)Controller와 JSP 호출순서
1074 -Controller와 JSP 호출순서.png 그림 참조
1075
1076 14)Hello Controller 작성
1077 -Client의 요청을 처리할 POJO 형태의 HelloController class를 작성

```

1078 -Controller class에 @Controller annotation 선언
1079 -요청을 처리할 method를 작성하고 @RequestMapping annotation을 선언
1080 --@RequestMapping : HTTP 요청 URL을 처리할 Controller 메소드 정의
1081 -JSP를 이용한 View 영역의 코드를 작성
1082 -Browser 상에서 JSP를 실행

1083
1084
1085 -src/com.javasoft.vo package 생성
1086 -src/com.javasoft.HelloVO class 생성

```
1087 package com.javasoft.vo;  
1088  
1089 public class HelloVO {  
1090     private String name;  
1091  
1092     public void setName(String name) {  
1093         this.name = name;  
1094     }  
1095  
1096     public String sayHello() {  
1097         return "Hello " + name;  
1098     }  
1099 }  
1100  
1101  
1102
```

1103 -src/com.javasoft.controller package 생성
1104 -com.javasoft.controller.HelloController class 생성

```
1105 package com.javasoft.controller;  
1106  
1107 import org.springframework.beans.factory.annotation.Autowired;  
1108 import org.springframework.stereotype.Controller;  
1109 import org.springframework.ui.Model;  
1110 import org.springframework.web.bind.annotation.RequestMapping;  
1111  
1112 import com.javasoft.vo.HelloVO;  
1113  
1114 @Controller  
1115 public class HelloController {  
1116     @Autowired  
1117     private HelloVO helloBean;  
1118  
1119     @RequestMapping("/hello.do")  
1120     public String hello(Model model) {  
1121         String msg = helloBean.sayHello();  
1122         model.addAttribute("greet", msg);  
1123         return "hello.jsp";  
1124     }  
1125 }  
1126  
1127
```

15)View에 data를 전달하는 Model class

1128
1129 -Controller에서 Service를 호출한 결과를 받아서 View에게 전달하기 위해, 전달받은 결과를 Model 객체에 저장
1130 -Model addAttribute(String name, Object value)
1131 --value객체를 name의 이름을 저장하고, view code에서는 name으로 지정한 이름을 통해서 value를 사용

16)beans.xml 수정

```
1133 <context:component-scan base-package="com.javasoft" />  
1134  
1135 <bean id="helloVO" class="com.javasoft.vo.HelloVO">  
1136     <property name="name" value="한지민" />  
1137 </bean>
```

17)WebContent/hello.jsp 생성

```
1140  
1141  
1142 <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
```

```
1143 <!DOCTYPE html>
1144 <html>
1145   <head>
1146     <meta charset="UTF-8">
1147     <title>Insert title here</title>
1148   </head>
1149   <body>
1150     ${greet}
1151   </body>
1152 </html>
```

1153 18)project > right-click > Run As > Run on Server > Finish

1154 19)http://localhost:8080/SpringWebDemo/hello.do

1155 Hello 한지민

1156

1157 18. Lab

1158 1)In J2EE Perspective

1159 2)Project Explorer > right-click > New > Dynamic Web Project

1160 3)Project name : SpringWebDemo > Next > Check [Generate web.xml deployment descriptor] > Finish

1161

1162 4)Convert to Maven Project

1163 -project right-click > Configure > Convert to Maven Project > Finish

1164

1165 5)Add Spring Project Nature

1166 -project right-click > Spring Tools > Add Spring Project Nature

1167

1168 6)새로 생성된 pom.xml파일에 필요한 library 추가 > Maven Clean > Maven Install

1169 <dependencies>

1170 <dependency>

1171 <groupId>org.springframework</groupId>

1172 <artifactId>spring-context</artifactId>

1173 <version>4.3.13.RELEASE</version>

1174 </dependency>

1175 <dependency>

1176 <groupId>junit</groupId>

1177 <artifactId>junit</artifactId>

1178 <version>4.12</version>

1179 <scope>test</scope>

1180 </dependency>

1181 <dependency>

1182 <groupId>org.springframework</groupId>

1183 <artifactId>spring-jdbc</artifactId>

1184 <version>4.3.13.RELEASE</version>

1185 </dependency>

1186 <dependency>

1187 <groupId>javax.servlet</groupId>

1188 <artifactId>jstl</artifactId>

1189 <version>1.2</version>

1190 </dependency>

1191 <dependency>

1192 <groupId>com.oracle</groupId>

1193 <artifactId>ojdbc6</artifactId>

1194 <version>11.1</version>

1195 </dependency>

1196 <dependency>

1197 <groupId>org.springframework</groupId>

1198 <artifactId>spring-webmvc</artifactId>

1199 <version>4.3.13.RELEASE</version>

1200 </dependency>

1201 </dependencies>

1202

1203 7)Build path에 config folder 추가

1204 -project right-click > Build Path > Configure Build Path > Select [Source] tab

-Click [Add Folder] > Select 현재 project > Click [Create New Folder...]
-Folder name : config > Finish > OK > Apply and Close
-Java Resources > config 폴더 확인

8)config folder에 beans.xml 파일 생성

-Spring Perspective로 전환
-config right-click > New > Spring Bean Configuration File
-File name : beans.xml
-생성시 beans,context, mvc 체크
 <?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:context="http://www.springframework.org/schema/context"
 xsi:schemaLocation="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-3.2.xsd">

 </beans>

9)ContextLoaderListener class 설정

-비즈니스 로직용의 스프링 설정 파일 (ex:applicationContext.xml)을 작성했기 때문에 listener로 ContextLoaderListener 클래스를 정의해야 한다.
-ContextLoaderListener 클래스는 스프링 설정 파일(디폴트에서 파일명 applicationContext.xml)을 로드하면 ServletContextListener 인터페이스를 구현하고 있기 때문에 ServletContext 인스턴스 생성 시(톰캣으로 어플리케이션이 로드된 때)에 호출된다. 즉, ContextLoaderListener 클래스는 DispatcherServlet 클래스의 로드보다 먼저 동작하여 비즈니스 로직층을 정의한 스프링 설정 파일을 로드한다.
-web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener - ContextLoaderListener] 를 선택하면 아래의 코드가 자동 삽입

```
<!-- needed for ContextLoaderListener -->  
<context-param>  
    <param-name>contextConfigLocation</param-name>  
    <param-value>location</param-value>  
</context-param>  
  
<!-- Bootstraps the root web application context before servlet initialization -->  
<listener>  
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
</listener>
```

-아래 코드로 변환

```
<context-param>  
    <param-name>contextConfigLocation</param-name>  
    <param-value>classpath:beans.xml</param-value>  
</context-param>
```

10)DispatcherServlet Class 추가

-web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet - DispatcherServlet declaration] 선택하면 아래의 코드가 자동 추가된다.

```
<!-- The front controller of this Spring Web application, responsible for handling all  
application requests -->  
<servlet>  
    <servlet-name>springDispatcherServlet</servlet-name>  
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
    <init-param>  
        <param-name>contextConfigLocation</param-name>  
        <param-value>location</param-value>  
    </init-param>  
    <load-on-startup>1</load-on-startup>  
</servlet>  
  
<!-- Map all requests to the DispatcherServlet for handling -->  
<servlet-mapping>
```

```

1266         <servlet-name>springDispatcherServlet</servlet-name>
1267         <url-pattern>url</url-pattern>
1268     </servlet-mapping>
1269
1270     -아래의 코드로 변환
1271     <init-param>
1272         <param-name>contextConfigLocation</param-name>
1273         <param-value>classpath:beans*.xml</param-value>
1274     </init-param>
1275
1276     <servlet-mapping>
1277         <servlet-name>springDispatcherServlet</servlet-name>
1278         <url-pattern>*.do</url-pattern>
1279     </servlet-mapping>
1280

```

11) UserVO class 생성

```

1282 -src/com.javasoft.vo package 생성
1283 -src/com.javasoft.vo.UserVO class 생성
1284

```

```

1285     package com.javasoft.vo;
1286
1287     public class UserVO {
1288         private String userId;
1289         private String name;
1290         private String gender;
1291         private String city;
1292         public UserVO() {}
1293         public UserVO(String userId, String name, String gender, String city) {
1294             this.userId = userId;
1295             this.name = name;
1296             this.gender = gender;
1297             this.city = city;
1298         }
1299         public String getUserId() {
1300             return userId;
1301         }
1302         public void setUserId(String userId) {
1303             this.userId = userId;
1304         }
1305         public String getName() {
1306             return name;
1307         }
1308         public void setName(String name) {
1309             this.name = name;
1310         }
1311         public String getGender() {
1312             return gender;
1313         }
1314         public void setGender(String gender) {
1315             this.gender = gender;
1316         }
1317         public String getCity() {
1318             return city;
1319         }
1320         public void setCity(String city) {
1321             this.city = city;
1322         }
1323         @Override
1324         public String toString() {
1325             return "UserVO [userId=" + userId + ", name=" + name + ", gender=" + gender + ",
1326                 city=" + city + "]";
1327         }
1328     }

```

12) UserDao 객체 생성

```

1330 -src/com.javasoft.dao package 생성
1331 -src/com.javasoft.dao.UserDao interface

```

```

1332
1333 package com.javasoft.dao;
1334
1335 import java.util.List;
1336
1337 import com.javasoft.vo.UserVO;
1338
1339 public interface UserDao {
1340     void insert(UserVO user);
1341
1342     List<UserVO> readAll();
1343
1344     void update(UserVO user);
1345
1346     void delete(String id);
1347
1348     UserVO read(String id);
1349 }
1350
1351 -src/com.javasoft.dao.UserDaoImplJDBC.java 생성
1352
1353 package com.javasoft.dao;
1354
1355 import java.sql.ResultSet;
1356 import java.sql.SQLException;
1357 import java.util.List;
1358
1359 import javax.sql.DataSource;
1360
1361 import org.springframework.beans.factory.annotation.Autowired;
1362 import org.springframework.dao.EmptyResultDataAccessException;
1363 import org.springframework.jdbc.core.JdbcTemplate;
1364 import org.springframework.jdbc.core.RowMapper;
1365 import org.springframework.stereotype.Repository;
1366
1367 import com.javasoft.vo.UserVO;
1368
1369 @Repository("userDao")
1370 public class UserDaoImplJDBC implements UserDao {
1371     private JdbcTemplate jdbcTemplate;
1372
1373     @Autowired
1374     public void setDataSource(DataSource dataSource) {
1375         this.jdbcTemplate = new JdbcTemplate(dataSource);
1376     }
1377
1378     class UserMapper implements RowMapper<UserVO> {
1379         public UserVO mapRow(ResultSet rs, int rowNum) throws SQLException {
1380             UserVO user = new UserVO();
1381             user.setUserId(rs.getString("userid"));
1382             user.setName(rs.getString("name"));
1383             user.setGender(rs.getString("gender"));
1384             user.setCity(rs.getString("city"));
1385             return user;
1386         }
1387     }
1388
1389     @Override
1390     public void insert(UserVO user) {
1391         String SQL = "INSERT INTO users (userid, name, gender,city) VALUES (?, ?, ?, ?)";
1392         jdbcTemplate.update(SQL, user.getUserId(), user.getName(), user.getGender(),
            user.getCity());
1393
1394         System.out.println("등록된 Record UserId=" + user.getUserId() + " Name=" +
            user.getName());
1395     }
1396

```

```

1397    @Override
1398    public List<UserVO> readAll() {
1399        String SQL = "SELECT * FROM users";
1400        List<UserVO> userList = jdbcTemplate.query(SQL, new UserMapper());
1401        return userList;
1402    }
1403
1404    @Override
1405    public void update(UserVO user) {
1406        String SQL = "UPDATE users SET name = ?, gender = ?, city = ? WHERE userid = ?";
1407        jdbcTemplate.update(SQL, user.getName(), user.getGender(),
1408            user.getCity(), user.getId());
1409        System.out.println("갱신된 Record with ID = " + user.getId() );
1410    }
1411
1412    @Override
1413    public void delete(String id) {
1414        String SQL = "delete from users where userid = ?";
1415        jdbcTemplate.update(SQL, id);
1416        System.out.println("삭제된 Record with ID = " + id );
1417    }
1418
1419    @Override
1420    public UserVO read(String id) {
1421        String SQL = "SELECT * FROM users WHERE userid = ?";
1422        try {
1423            UserVO user = jdbcTemplate.queryForObject(SQL,
1424                new Object[] { id }, new UserMapper());
1425            return user;
1426        } catch (EmptyResultDataAccessException e) {
1427            return null;
1428        }
1429    }

```

13) UserService 객체 생성

- src/com.javasoft.service package 생성
- src/com.javasoft.service.UserService interface

```

1434
1435    package com.javasoft.service;
1436
1437    import java.util.List;
1438
1439    import com.javasoft.vo.UserVO;
1440
1441    public interface UserService {
1442        void insertUser(UserVO user);
1443
1444        List<UserVO> getUserList();
1445
1446        void deleteUser(String id);
1447
1448        UserVO getUser(String id);
1449
1450        void updateUser(UserVO user);
1451    }

```

- src/com.javasoft.service.UserServiceImpl.java

```

1454
1455    package com.javasoft.service;
1456
1457    import java.util.List;
1458
1459    import org.springframework.beans.factory.annotation.Autowired;
1460    import org.springframework.stereotype.Service;
1461
1462    import com.javasoft.dao.UserDao;

```

```

1463 import com.javasoft.vo.UserVO;
1464
1465 @Service("userService")
1466 public class UserServiceImpl implements UserService {
1467
1468     @Autowired
1469     UserDao userDao;
1470
1471     @Override
1472     public void insertUser(UserVO user) {
1473         this.userDao.insert(user);
1474     }
1475
1476     @Override
1477     public List<UserVO> getUserList() {
1478         return this.userDao.readAll();
1479     }
1480
1481     @Override
1482     public void deleteUser(String id) {
1483         this.userDao.delete(id);
1484     }
1485
1486     @Override
1487     public UserVO getUser(String id) {
1488         return this.userDao.read(id);
1489     }
1490
1491     @Override
1492     public void updateUser(UserVO user) {
1493         this.userDao.update(user);
1494     }
1495 }
1496

```

14)UserController 객체 생성

- src/com.javasoft.controller package 생성
- com.javasoft.controller.UserController class 생성

```

1501 package com.javasoft.controller;
1502
1503 import org.springframework.beans.factory.annotation.Autowired;
1504 import org.springframework.stereotype.Controller;
1505 import org.springframework.ui.Model;
1506 import org.springframework.web.bind.annotation.RequestMapping;
1507 import org.springframework.web.bind.annotation.RequestParam;
1508
1509 import com.javasoft.service.UserService;
1510 import com.javasoft.vo.UserVO;
1511
1512 @Controller
1513 public class UserController {
1514     @Autowired
1515     private UserService userService;
1516
1517     @RequestMapping("/getUser.do")
1518     public String getUserList(@RequestParam("userId") String userId, Model model) {
1519         UserVO user = userService.getUser(userId);
1520         //System.out.println(user);
1521         model.addAttribute("user", user);
1522         return "userInfo.jsp";
1523     }
1524 }
1525

```

15)config/dbinfo.properties 파일 생성

```

1528 db.driverClass=oracle.jdbc.driver.OracleDriver
1529 db.url=jdbc:oracle:thin:@localhost:1521:XE

```



```

1530 db.username=scott
1531 db.password=tiger
1532
1533 16)beans.xml 수정
1534
1535 <context:component-scan base-package="com.javasoft" />
1536
1537 <context:property-placeholder location="classpath:dbinfo.properties" />
1538 <bean id="dataSource" class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
1539     <property name="driverClass" value="${db.driverClass}" />
1540     <property name="url" value="${db.url}" />
1541     <property name="username" value="${db.username}" />
1542     <property name="password" value="${db.password}" />
1543 </bean>
1544

```

17)WebContent/index.jsp 생성

```

1545
1546
1547 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
1548 <c:redirect url="userinfo.do" />
1549

```

18)project > right-click > Run As > Run on Server > Finish

19)http://localhost:8080/SpringWebDemo/userinfo.do?userId=jimin

19. 데이터 변환 - JSON으로 변환

- 1)시스템이 복잡해지면서 다른 시스템과 정보를 주고받을 일이 발생하는데, 이 때 데이터 교환 포맷으로 JSON을 사용할 수 있다.
- 2)검색결과를 JSON 데이터로 변환하려면 가장 먼저 jackson2 라이브러리를 다운받아야 한다.
- 3)Jackson2는 자바 객체를 JSON으로 변환하거나 JSON을 자바 객체로 변환해주는 라이브러리다.
- 4)<https://www.concretepage.com/spring-4/spring-4-rest-xml-response-example-with-jackson-2> 참조
- 5)<https://www.mkkyong.com/java/jackson-2-convert-java-object-to-from-json/> 참조
- 6)pom.xml에 다음과 같이 dependency를 추가한다.

```

1562 <dependency>
1563     <groupId>com.fasterxml.jackson.core</groupId>
1564     <artifactId>jackson-databind</artifactId>
1565     <version>2.7.2</version>
1566 </dependency>

```

7)Maven clean > Maven Install하면 Maven Dependencies에 아래와 같은 jar파일이 추가된다.

```

1567 -jackson-databind-2.7.2.jar
1568 -jackson-annotations-2.7.0.jar
1569 -jackson-core-2.7.2.jar
1570

```

8)보통 User가 Servlet이나 JSP를 요청하면 서버는 요청한 파일을 찾아서 실행한다.

9)그 실행결과는 HTTP Response package의 body에 저장하여 Browser에 전송한다.

10)그런데, 이 응답결과를 HTML이 아니라 JSON이나 XML로 변환하여 body에 저장하려면 Spring에서 제공하는 변환기(Converter)를 사용해야 한다.

11)Spring은 HttpMessageConverter를 구현한 다양한 변환기를 제공한다.

12)이 변환기를 이용하면 Java 객체를 다양한 타입으로 변환하여 HTTP Response body에 설정할 수 있다.

13)HttpMessageConverter를 구현한 클래스는 여러가지가 있으며, 이 중에서 Java 객체를 JSON response body로 변환할 때는 MappingJackson2HttpMessageConverter를 사용한다.

14)따라서 MappingJackson2HttpMessageConverter를 Spring 설정 파일에 등록하면 되는데, 혹시 이후에 XML 변환도 처리할 예정이라면 다음처럼 설정한다.

```

1579 <mvc:annotation-driven />

```

15)Spring Bean Configuration File에 위와 같이 설정하면 HttpMessageConverter를 구현한 모든 변환기가 생성된다.

16)src/com.javasoft.controller.UserController.java에 다음과 같이 수정한다.

```

1582 package com.javasoft.controller;
1583
1584 import org.springframework.beans.factory.annotation.Autowired;
1585 import org.springframework.stereotype.Controller;
1586 import org.springframework.web.bind.annotation.RequestMapping;
1587 import org.springframework.web.bind.annotation.RequestParam;
1588 import org.springframework.web.bind.annotation.ResponseBody;
1589
1590 import com.javasoft.service.UserService;
1591

```

```

import com.javasoft.vo.UserVO;

@Controller
public class UserController {
    @Autowired
    private UserService userService;

    /*@RequestMapping("/userinfo.do")
    public String getUserList(@RequestParam("userId") String userId, Model model) {
        UserVO user = userService.getUser(userId);
        model.addAttribute("user", user);
        return "userinfo.jsp";
    }*/

    @RequestMapping("/userinfo.do")
    @ResponseBody
    public UserVO userinfo(@RequestParam("userId") String userId) {
        return userService.getUser(userId);
    }
}

```

17)이전 메소드와 달리 @ResponseBody라는 annotation을 추가했는데, Java 객체를 Http Response 프로토콜의 body로 변환하기 위해 사용된다.

18)이미 Spring Configuration File에 <mvc:annotation-driven>을 추가했기 때문에 @ResponseBody가 적용된 메소드의 실행 결과는 JSON으로 변환되어 HTTP Response Body에 다음과 같이 설정된다.

```
{ "userId": "jimin", "name": "한지민", "gender": "여", "city": "서울" }
```

19)만일 이때, Java 객체를 JSON으로 변환할 때, 특정 변수를 제외시키려면 @JsonIgnore annotation을 해당 변수의 getter에 설정하면 된다.

```

package com.javasoft.vo;
import com.fasterxml.jackson.annotation.JsonIgnore;
public class UserVO {
    ...
    @JsonIgnore
    public String getGender() {
        return gender;
    }
}

```

20)이렇게 하면 아래와 같이 성별이 포함되지 않는다는 것을 알 수 있다.

```
{ "userId": "jimin", "name": "한지민", "city": "서울" }
```

20. 데이터 변환 - XML로 변환

- 1)Maven Repository에서 'spring xml'로 검색
- 2)Spring Object/XML Marshalling에서 4.3.13.RELEASE 선택
- 3)pom.xml에 아래 dependency 추가 > Maven Clean > Mavan Install

```

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-oxm</artifactId>
    <version>4.3.13.RELEASE</version>
</dependency>

```

4)Maven Repository에서 'jaxb'로 검색, Jaxb Api에서 2.3.0

5)아래의 dependency를 pom.xml에 추가 > Maven Clean > Mavan Install

```

<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>

```

6)src/com.javasoft.vo/UserListVO.java 생성

```

1656 package com.javasoft.vo;
1657
1658 import java.util.List;
1659
1660 import javax.xml.bind.annotation.XmlAccessType;
1661 import javax.xml.bind.annotation.XmlAccessorType;
1662 import javax.xml.bind.annotation.XmlElement;
1663 import javax.xml.bind.annotation.XmlRootElement;
1664
1665 import org.springframework.stereotype.Component;
1666
1667 @XmlRootElement(name="userList")
1668 @XmlAccessorType(XmlAccessType.FIELD)
1669 @Component
1670 public class UserListVO {
1671     @XmlElement(name="user")
1672     private List<UserVO> userList;
1673
1674     public List<UserVO> getUserList() {
1675         return userList;
1676     }
1677
1678     public void setUserList(List<UserVO> userList) {
1679         this.userList = userList;
1680     }
1681 }

```

-XML 문서는 반드시 단 하나의 **root element**를 가져야 한다.
 -여러 **UserVO**를 표현하려면 **root element**로 사용할 또 다른 **Java class**가 필요하다.
 -새로 생성한 **UserListVO**객체는 이 객체가 **root element**에 해당하는 객체이며, **root element** 이름을 **userList**로 설정하겠다는 의미로 **@XmlRootElement(name="userList")** 설정을 추가했다.
 -그리고 **userList** 변수 위에도 **@XmlElement(name="user")**를 추가했는데, **UserVO** 마다 **element** 이름을 **user**로 변경할 것이다.

7)src/com.javasoft.vo.UserVO.java 수정

```

1690 package com.javasoft.vo;
1691
1692 import javax.xml.bind.annotation.XmlAccessType;
1693 import javax.xml.bind.annotation.XmlAccessorType;
1694 import javax.xml.bind.annotation.XmlAttribute;
1695 import javax.xml.bind.annotation.XmlRootElement;
1696
1697 import org.springframework.stereotype.Component;
1698
1699 @XmlRootElement(name="user")
1700 @XmlAccessorType(XmlAccessType.FIELD)
1701 @Component
1702 public class UserVO {
1703     @XmlAttribute
1704     private String userId;
1705 }

```

-VO class에 선언된 **@XmlAccessorType**은 **UserVO** 객체를 XML로 변환할 수 있다는 의미이다.
 -그리고 **XmlAccessType.FIELD** 때문에 이 객체가 가진 필드, 즉 변수들은 자동으로 자식 **element**로 표현된다.
 -하지만, 이 중에서 **userId**에만 **@XmlAttribute**가 붙었는데, 이는 **userId**를 속성으로 표현하라는 의미이다.
 -만일 JSON 변환시 **@JsonIgnore**가 변환시 제외하는 것처럼, XML변환시에도 제외할 변수는 **@XmlTransient**를 붙이면 된다.
 -마지막으로 변환시 변수가 참조형이면 반드시 기본 생성자가 있어야만 한다.

8)Spring 설정 파일에서 p와 oxm 체크후, 아래 코드 추가

-JSON 변환시 Java 객체를 JSON response body로 변환해주는 **MappingJackson2HttpMessageConverter**를 Spring 설정파일에 추가해야 하는데, 설정 파일에 **<mvc:annotation-driven />**으로 대체했었다.
 -마찬가지로 Java 객체를 XML response body로 변환할 때는 아래의 코드를 추가한다.

```

<bean id="xmlViewer" class="org.springframework.web.servlet.view.xml.MarshallingView">

```

```
1717     <constructor-arg>
1718         <bean class="org.springframework.oxm.jaxb.Jaxb2Marshaller"
1719             p:classesToBeBound="com.javasoft.vo.UserListVO"/>
1720     </constructor-arg>
1721 </bean>
```

9)UserController.java 코드 추가

```
1725     @RequestMapping(value="/userlist.do", produces="application/xml")
1726     @ResponseBody
1727     public UserListVO userlist(){
1728         UserListVO listVO = new UserListVO();
1729         listVO.setUserList(this.userService.getUserList());
1730         return listVO;
1731     }
```

10실행결과

```
1734     <userList>
1735         <user userId="jimin">
1736             <name>한지민</name>
1737             <gender>여</gender>
1738             <city>서울</city>
1739         </user>
1740     </userList>
```