1. MVC개요
   1)Model-View-Controller 패턴의 개념
      -소프트웨어 공학에서 사용되는 아키텍쳐 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

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

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

7. DispatcherServlet Class

1)Front Controller pattern 적용

2)web.xml에 설정

3)Client로부터의 모든 요청을 전달 받음

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

8. Spring MVC의 주요 구성 요소

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

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

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

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

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

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

9. Spring MVC의 주요 구성 요소의 요청 처리 과정

-Spring MVC Process.png 그림 참조

1)Client의 요청이 DispatcherServlet에게 전달된다.

2)DispatcherServlet은 HandlerMapping을 사용하여 Client의 요청을 처리할 Controller를 획득한다.

3)DispatcherServlet은 Controller 객체를 이용하여 Client의 요청을 처리한다.

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

5)DispatcherServlet은 ViewResolver로부터 응답 결과를 생성할 View 객체를 구한다.

6)View는 Client에게 전송할 응답을 생성한다.

10. Spring MVC 기반 Web Application 작성 절차

1)Client의 요청을 받는 DispatcherServlet를 web.xml에 설정

2)Client의 요청을 처리할 Controller를 작성

3)Spring Bean으로 Controller를 등록

4)JSP를 이용한 View 영역의 코드를 작성

5)Browser 상에서 JSP를 실행

11. Lab

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

2)Select Spring MVC Project

3)Project name : 1207 > Next

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

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

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

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

Hello world!

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

8)한글 깨짐을 수정하는 것은 src/main/webapp/WEB-INF/views/home.jsp에서
<%@ page session="false" pageEncoding="UTF-8" contentType="text/html; charset=UTF-8"%>로 수정

9)처리 순서

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

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

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

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

-com.javasoft.springmvcdemo에 @Controller를 찾는다.

-@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.

-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.

-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.

-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       }
182
183       -src/main/webapp/WEB-INF/views/view.jsp 생성
184
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 ${currentDate} 입니다.
195             </body>
196          </html>
197
198
199    3)View에 Data 전달
200
201       package com.javasoft.biz;
202
203       import org.springframework.stereotype.Controller;
204       import org.springframework.ui.Model;
205       import org.springframework.web.bind.annotation.RequestMapping;
206       import org.springframework.web.bind.annotation.RequestMethod;
207       import org.springframework.web.servlet.ModelAndView;
208
209       @Controller
210       public class UserController {
211
212          @RequestMapping("/view")
213          public String view(Model model){
214             model.addAttribute("currentDate", new java.util.Date());
215             return "view";    // /WEB-INF/views/view  + .jsp
216          }
217
218          @RequestMapping("/bbs/view")
219          public String bbs_view(Model model){
220             String [] array = {"Apple", "Mango", "Lemon", "Grape"};
221
222             model.addAttribute("fruits", array);
223
224             return "/bbs/view";    // /WEB-INF/views/bbs/view  + .jsp
225          }
226       }
227
228       -src/main/webapp/WEB-INF/views/bbs/view.jsp 생성
229
230          <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
231          <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
232          <!DOCTYPE html>
233          <html>
234          <head>
235          <meta charset="UTF-8">
236          <title>Insert title here</title>
237          </head>
238          <body>
239             <h2>/bbs/view.jsp</h2>
240             <ul>과일 종류
241             <c:forEach items="${fruits}" var="fruit">
242                <li>${fruit}</li>
```

```
            </c:forEach>
            </ul>
        </body>
        </html>

    -http://localhost:8080/biz/view  --> /view.jsp
    -http://localhost:8080/biz/bbs/view  --> /bbs/view.jsp

4)View에 ModelAndView 객제로 Data 전달

    @RequestMapping(value = "/demo", method = RequestMethod.GET)
    public ModelAndView demo() {

        ModelAndView mav = new ModelAndView();
        mav.addObject("userid", "javasoft");
        mav.addObject("passwd", "12345678");
        mav.setViewName("/demo");
        return mav;
    }

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

        <%@ 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 />
                패스워드 : ${passwd}
            </body>
        </html>

5)Controller class에 @RequestMapping 적용
    -src/main/java/com.javasoft.biz.StudentController.java 생성

        package com.javasoft.biz;

        import org.springframework.stereotype.Controller;
        import org.springframework.web.bind.annotation.RequestMapping;
        import org.springframework.web.bind.annotation.RequestMethod;
        import org.springframework.web.servlet.ModelAndView;

        @Controller
        @RequestMapping("/bbs")
        public class StudentController {

            @RequestMapping(value="/get", method = RequestMethod.GET)
            public ModelAndView getStudent() {

                ModelAndView mav = new ModelAndView();
                mav.setViewName("/bbs/get");   //  /WEB-INF/views/bbs/get.jsp
                mav.addObject("name", "한지민");
                mav.addObject("age", 25);
                return mav;
            }
        }

    -src/main/webapp/WEB-INF/views/bbs/get.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>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
        </head>
        <body>
           학생 이름 : ${name} <br />
           학생 나이 : ${age}
        </body>
        </html>
```

14. Lab : Form Data 처리하기

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

```
package com.javasoft.springwebdemo;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class RequestController {
```

6)HttpServletRequest class 이용하기

-RequestController.java

```
   @RequestMapping(value="/confirm", method=RequestMethod.GET)
   public String confirm(HttpServletRequest request, Model model) {
      String userid = request.getParameter("userid");
      String passwd = request.getParameter("passwd");
      String name = request.getParameter("name");
      int age = Integer.parseInt(request.getParameter("age"));
      String gender = request.getParameter("gender");

      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 />
         패스워드 : ${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

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 />
      패스워드 : ${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;
```

```
    }
    public String getName() {
       return name;
    }
    public void setName(String name) {
       this.name = name;
    }
    public int getAge() {
       return age;
    }
    public void setAge(int age) {
       this.age = age;
    }
    public String getGender() {
       return gender;
    }
    public void setGender(String gender) {
       this.gender = gender;
    }
    @Override
    public String toString() {
       return "UserVO [userid=" + userid + ", passwd=" + passwd + ", name=" + name + ",
       age=" + age + ", gender="
             + gender + "]";
    }
 }

-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) {

      UserVO userVO = new UserVO();
      userVO.setUserid(userid);
      userVO.setPasswd(passwd);
      userVO.setName(name);
      userVO.setAge(age);
      userVO.setGender(gender);

      model.addAttribute("userVO", userVO);

      return "confirm1";   //  /WEB-INF/views/confirm1.jsp
   }

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

   <%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
   <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
   <c:set var="user" value="${userVO}"/>
   <!DOCTYPE html>
   <html>
   <head>
   <meta charset="UTF-8">
   <title>Insert title here</title>
   </head>
   <body>
      <h1>confirm1.jsp</h1>
      <h2>사용자 정보</h2>
      아이디 : ${user.userid} <br />
      패스워드 : ${user.passwd} <br />
      이름 : ${user.name} <br />
      나이 : ${user.age} <br />
```

```
        성별 : ${user.gender}
    </body>
    </html>
```

9)Data Commander 객체 이용하기2

-RequestController.java

```
    @RequestMapping(value="/confirm", method=RequestMethod.GET)
    public String confirm(UserVO userVO) {

        return "confirm2";   //  /WEB-INF/views/confirm1.jsp
    }
```

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

```
    <%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    <c:set var="user" value="${userVO}"/>
    <!DOCTYPE html>
    <html>
    <head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
    </head>
    <body>
        <h1>confirm1.jsp</h1>
        <h2>사용자 정보</h2>
        아이디 : ${user.userid} <br />
        패스워드 : ${user.passwd} <br />
        이름 : ${user.name} <br />
        나이 : ${user.age} <br />
        성별 : ${user.gender}
    </body>
    </html>
```

10)@PathVariable 이용하기

-RequestController.java

```
@RequestMapping(value="/confirm/{userid}", method=RequestMethod.GET)
public String confirm(@PathVariable String userid, Model model) {
    model.addAttribute("userid", userid);
    return "confirm3";
}
```

-src/main/webapp/WEB-INF/views/confirm3.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>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Insert title here</title>
    </head>
    <body>
        유저 아이디 : ${userid}
    </body>
    </html>
```

http://localhost:8080/springwebdemo/confirm?userid=javasoft ==>
http://localhost:8080/springwebdemo/confirm/javasoft

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>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
   아이디 : ${userid} <br />
   패스워드 : ${passwd}
</body>
</html>
```

2)@ModelAttribute annotation 이용하기

-@ModelAttribute annotation을 이용하면 Data Commander 객체의 이름을 변경할 수 있다.

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

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원가입 폼</title>
</head>
<body>
   <form method="GET" action="/springwebdemo/confirm">
      아이디 : <input type="text" name="userid" /><br />
      패스워드 : <input type="password" name="passwd" /><br />
      이름 : <input type="text" name="name" /><br />
      나이 : <input type="number" name="age" /><br />
      성별 : <input type="radio" name="gender" value="남성" />남성   
          <input type="radio" name="gender" value="여성" />여성<br />
      <input type="submit" value="가입하기" />
   </form>
</body>
</html>
```

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

```
@RequestMapping(value="/register", method=RequestMethod.POST)
public String register(@ModelAttribute("u") UserVO userVO) {    //userVO가 아니라 u로 변경

   return "register";
}
```

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

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="user" value="${u}"/>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
   <h1>사용자 정보</h1>
   <ul>
      <li>아이디 : ${user.userid}</li>
      <li>패스워드 : ${user.passwd}</li>
      <li>이름 : ${user.name}</li>
      <li>나이 : ${user.age}</li>
      <li>성별 : ${user.gender}</li>
   </ul>
</body>
</html>
```

-Spring에서 POST 방식으로 Data를 보낼 때 한글깨짐 현상 발생

-해결점

-web.xml

```
693
694         <filter>
695       <filter-name>encodingFilter</filter-name>
696       <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
697             <init-param>
698                <param-name>encoding</param-name>
699                <param-value>UTF-8</param-value>
700             </init-param>
701          </filter>
702          <filter-mapping>
703                      <filter-name>encodingFilter</filter-name>
704                      <url-pattern>/*</url-pattern>
705          </filter-mapping>
706
707    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          <%@ page language="java" contentType="text/html; charset=UTF-8"
           pageEncoding="UTF-8"%>
734          <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
           "http://www.w3.org/TR/html4/loose.dtd">
735          <html>
736          <head>
737          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
738          <title>Insert title here</title>
739          </head>
740          <body>
741             <h1>관리자 페이지</h1>
742             권한 : ${authority}
743          </body>
744          </html>
745
746       -/src/main/webapp/WEB-INF/views/user.jsp
747
748          <%@ page language="java" contentType="text/html; charset=UTF-8"
           pageEncoding="UTF-8"%>
749          <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
           "http://www.w3.org/TR/html4/loose.dtd">
750          <html>
751          <head>
752          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
753          <title>Insert title here</title>
754          </head>
755          <body>
```

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


16. Lab : Form Data Validation
  1)Package Explorer > right-click > New > Spring Legacy Project
  2)Select Spring MVC Project
  3)Project name : 1208 > Next
  4)Enter a topLevelPackage : com.javasoft.biz > Finish
  5)UserVO 객체 생성
    -src/main/java/com.javasoft.vo package 생성
    -src/main/java/com.javasoft.vo.UserVO class

        package com.javasoft.vo;

        public class UserVO {
            private String name;
            private int age;
            private String userid;
            public String getName() {
                return name;
            }
            public void setName(String name) {
                this.name = name;
            }
            public int getAge() {
                return age;
            }
            public void setAge(int age) {
                this.age = age;
            }
            public String getUserid() {
                return userid;
            }
            public void setUserid(String userid) {
                this.userid = userid;
            }
            @Override
            public String toString() {
                return "UserVO [name=" + name + ", age=" + age + ", userid=" + userid + "]";
            }
        }

  6)Validator를 이용한 검증
    -Data Command 객체에서 유효성 검사를 할 수 있다.
    -UserValidator 객체 생성
    -src/main/java/com.javasoft.biz.UserValidator class

        package com.javasoft.biz;

        import org.springframework.validation.Errors;
        import org.springframework.validation.Validator;

        import com.javasoft.vo.UserVO;

        public class UserValidator implements Validator {

            @Override
            public boolean supports(Class<?> arg0) {
                //검증할 객체의 클래스 타입 정보를 반환
                return UserVO.class.isAssignableFrom(arg0);
            }

            @Override
            public void validate(Object obj, Errors errors) {
```

```
            System.out.println("검증시작");
            UserVO userVO = (UserVO)obj;

            String username = userVO.getName();
            if(username == null || username.trim().isEmpty()) {
               System.out.println("이름의 값이 빠졌습니다.");
               errors.rejectValue("name", "No Value");
            }

            int userage = userVO.getAge();
            if(userage == 0) {
               System.out.println("나이의 값이 빠졌습니다.");
               errors.rejectValue("age", "No Value");
            }

            String userid = userVO.getUserid();
            if(userid == null || userid.trim().isEmpty()) {
               System.out.println("아이디의 값이 빠졌습니다.");
               errors.rejectValue("userid", "No Value");
            }
         }
      }
```

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

```
      @RequestMapping(value = "/register", method=RequestMethod.GET)
      public String register() {
         return "register";
      }

      @RequestMapping(value = "/register", method=RequestMethod.POST)
      public String register(@ModelAttribute("userVO") UserVO userVO, BindingResult result) {
         String page = "register_ok";
         UserValidator validator = new UserValidator();
         validator.validate(userVO, result);
         if(result.hasErrors()) {
            page = "register";
         }
         return page;
      }
```

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

```
      <%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
      <!DOCTYPE html>
      <html>
      <head>
      <meta charset="UTF-8">
      <title>회원 가입 폼</title>
      </head>
      <body>
         <form action="/biz/register" method="post">
            Name : <input type="text" name="name" /><br />
            Age : <input type="number" name="age" /><br />
            ID : <input type="text" name="userid" /><br />
            <input type="submit" value="ê°PADìPMNELíMWSOSê¸°" />
         </form>
      </body>
      </html>
```

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

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

```
<meta charset="UTF-8">
<title>회원 가입 결과 창</title>
</head>
<body>
   <ul>
      <li>이름 : ${user.name}</li>
      <li>나이 : ${user.age}</li>
      <li>아이디 : ${user.userid}</li>
   </ul>
</body>
</html>
```

7)ValidataionUtils class를 이용한 검증
-ValidatationUtils class는 validate() 메소드를 좀 더 편리하게 사용할 수 있게 해줌.
-UserValidator.java 수정

```
/*String username = userVO.getName();
if(username == null || username.trim().isEmpty()) {
   System.out.println("이름의 값이 빠졌습니다.");
   errors.rejectValue("name", "No Value");
}*/

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "No Value");
```

8)@Valid와 @InitBinder 이용하기
-Spring Framework이 대신 검증해 줌
-mvnrepository에서 'hibernate validator'로 검색

```
<dependency>
      <groupId>org.hibernate.validator</groupId>
      <artifactId>hibernate-validator</artifactId>
      <version>6.0.5.Final</version>
</dependency>
```

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

```
@RequestMapping(value = "/register", method=RequestMethod.POST)
public String register(@ModelAttribute("userVO") @Valid UserVO userVO, BindingResult
result) {
   String page = "register_ok";
   //UserValidator validator = new UserValidator();
   //validator.validate(userVO, result);
   if(result.hasErrors()) {
      page = "register";
   }

   return page;
}

@InitBinder
protected void initBinder(WebDataBinder binder) {
   binder.setValidator(new UserValidator());
}
```

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 생선된 pom.xml파일에 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 mvn'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 foler 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 생성시(톰켓으로 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:/config/beans.xml</param-value>
        </context-param>

  11)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>
           <servlet-name>springDispatcherServlet</servlet-name>
           <url-pattern>url</url-pattern>
        </servlet-mapping>

     -아래의 코드로 변환
        <init-param>
           <param-name>contextConfigLocation</param-name>
           <param-value>classpath:/config/beans*.xml</param-value>
        </init-param>

        <servlet-mapping>
           <servlet-name>springDispatcherServlet</servlet-name>
           <url-pattern>*.do</url-pattern>
        </servlet-mapping>

  12)mvnrepository에서 'jstl'로 검색 후 설치
     -목록에서 2번째 : 1.2버전

        <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
        <dependency>
              <groupId>javax.servlet</groupId>
              <artifactId>jstl</artifactId>
              <version>1.2</version>
        </dependency>

     -pom.xml에 붙여넣고 Maven Clean > Maven Install

  13)Controller와 JSP 호출순서
     -Controller와 JSP 호출순서.png 그림 참조

  14)Hello Controller 작성
     -Client의 요청을 처리할 POJO 형태의 HelloController class를 작성
```

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

-src/com.javasoft.vo package 생성
-src/com.javasoft.HelloVO class 생성

```
package com.javasoft.vo;

public class HelloVO {
   private String name;

   public void setName(String name) {
      this.name = name;
   }

   public String sayHello() {
      return "Hello " + name;
   }
}
```

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

```
package com.javasoft.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import com.javasoft.vo.HelloVO;

@Controller
public class HelloController {
   @Autowired
   private HelloVO helloBean;

   @RequestMapping("/hello.do")
   public String hello(Model model) {
      String msg = helloBean.sayHello();
      model.addAttribute("greet", msg);
      return "hello.jsp";
   }
}
```

15)View에 data를 전달하는 Model class
-Controller에서 Service를 호출한 결과를 받아서 View에게 전달하기 위해, 전달받은 결과를 Model 객체에 저장
-Model addAttribute(String name, Object value)
--value객체를 name의 이름으로 저장하고, view code에서는 name으로 지정한 이름을 통해서 value를 사용

16)beans.xml 수정

```
<context:component-scan base-package="com.javasoft" />

<bean id="helloVO" class="com.javasoft.vo.HelloVO">
   <property name="name" value="한지민" />
</bean>
```

17)WebContent/hello.jsp 생성

```
<%@ 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>
    ${greet}
  </body>
</html>
```

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

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

Hello 한지민

18. 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 생성된 pom.xml파일에 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>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>
  <dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.1</version>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>4.3.13.RELEASE</version>
  </dependency>
</dependencies>
```

7)Build path에 config foler 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

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 생성시(톰켓으로 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>
```

```
            <servlet-name>springDispatcherServlet</servlet-name>
            <url-pattern>url</url-pattern>
        </servlet-mapping>

    -아래의 코드로 변환
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:beans*.xml</param-value>
        </init-param>

        <servlet-mapping>
            <servlet-name>springDispatcherServlet</servlet-name>
            <url-pattern>*.do</url-pattern>
        </servlet-mapping>

11)UserVO class 생성
    -src/com.javasoft.vo package 생성
    -src/com.javasoft.vo.UserVO class 생성

        package com.javasoft.vo;

        public class UserVO {
            private String userId;
            private String name;
            private String gender;
            private String city;
            public UserVO() {}
            public UserVO(String userId, String name, String gender, String city) {
                this.userId = userId;
                this.name = name;
                this.gender = gender;
                this.city = city;
            }
            public String getUserId() {
                return userId;
            }
            public void setUserId(String userId) {
                this.userId = userId;
            }
            public String getName() {
                return name;
            }
            public void setName(String name) {
                this.name = name;
            }
            public String getGender() {
                return gender;
            }
            public void setGender(String gender) {
                this.gender = gender;
            }
            public String getCity() {
                return city;
            }
            public void setCity(String city) {
                this.city = city;
            }
            @Override
            public String toString() {
                return "UserVO [userId=" + userId + ", name=" + name + ", gender=" + gender + ",
                city=" + city + "]";
            }
        }

12)UserDao 객체 생성
    -src/com.javasoft.dao package 생성
    -src/com.javasoft.dao.UserDao interface
```

```
package com.javasoft.dao;

import java.util.List;

import com.javasoft.vo.UserVO;

public interface UserDao {
   void insert(UserVO user);

   List<UserVO> readAll();

   void update(UserVO user);

   void delete(String id);

   UserVO read(String id);
}
```

-src/com.javasoft.dao.UserDaoImplJDBC.java 생성

```
package com.javasoft.dao;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;

import com.javasoft.vo.UserVO;

@Repository("userDao")
public class UserDaoImplJDBC implements UserDao {
   private JdbcTemplate jdbcTemplate;

   @Autowired
   public void setDataSource(DataSource dataSource) {
      this.jdbcTemplate = new JdbcTemplate(dataSource);
   }

   class UserMapper implements RowMapper<UserVO> {
      public UserVO mapRow(ResultSet rs, int rowNum) throws SQLException {
         UserVO user = new UserVO();
         user.setUserId(rs.getString("userid"));
         user.setName(rs.getString("name"));
         user.setGender(rs.getString("gender"));
         user.setCity(rs.getString("city"));
         return user;
      }
   }

   @Override
   public void insert(UserVO user) {
      String SQL = "INSERT INTO users (userid, name, gender,city) VALUES (?, ?, ?, ?)";
      jdbcTemplate.update(SQL, user.getUserId(), user.getName(), user.getGender(),
      user.getCity());

      System.out.println("등록된 Record UserId=" + user.getUserId() + " Name=" +
      user.getName());
   }
```

```
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(),
                  user.getCity(),user.getUserId());
1408               System.out.println("갱신된 Record with ID = " + user.getUserId() );
1409            }
1410
1411            @Override
1412            public void delete(String id) {
1413               String SQL = "delete from users where userid = ?";
1414               jdbcTemplate.update(SQL, id);
1415               System.out.println("삭제된 Record with ID = " + id );
1416            }
1417
1418            @Override
1419            public UserVO read(String id) {
1420               String SQL = "SELECT * FROM users WHERE userid = ?";
1421               try {
1422                  UserVO user = jdbcTemplate.queryForObject(SQL,
1423                        new Object[] { id }, new UserMapper());
1424                  return user;
1425               }catch(EmptyResultDataAccessException e){
1426                  return null;
1427               }
1428            }
1429         }
1430
1431      13)UserService 객체 생성
1432         -src/com.javasoft.service package 생성
1433         -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            }
1452
1453         -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;
```

```
import com.javasoft.vo.UserVO;

@Service("userService")
public class UserServiceImpl implements UserService {

   @Autowired
   UserDao userDao;

   @Override
   public void insertUser(UserVO user) {
      this.userDao.insert(user);
   }

   @Override
   public List<UserVO> getUserList() {
      return this.userDao.readAll();
   }

   @Override
   public void deleteUser(String id) {
      this.userDao.delete(id);
   }

   @Override
   public UserVO getUser(String id) {
      return this.userDao.read(id);
   }

   @Override
   public void updateUser(UserVO user) {
      this.userDao.update(user);
   }
}
```

14)UserController 객체 생성

-src/com.javasoft.controller package 생성

-com.javasoft.controller.UserController class 생성

```
package com.javasoft.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.javasoft.service.UserService;
import com.javasoft.vo.UserVO;

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

   @RequestMapping("/getUser.do")
   public String getUserList(@RequestParam("userId") String userId, Model model) {
      UserVO user = userService.getUser(userId);
      //System.out.println(user);
      model.addAttribute("user", user);
      return "userInfo.jsp";
   }
}
```

15)config/dbinfo.properties 파일 생성

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

```
db.username=scott
db.password=tiger
```

16)beans.xml 수정

```
<context:component-scan base-package="com.javasoft" />

<context:property-placeholder location="classpath:dbinfo.properties" />
<bean id="dataSource" class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
   <property name="driverClass" value="${db.driverClass}" />
   <property name="url" value="${db.url}" />
   <property name="username" value="${db.username}" />
   <property name="password" value="${db.password}" />
</bean>
```

17)WebContent/index.jsp 생성

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

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 먼너 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.mkyong.com/java/jackson-2-convert-java-object-to-from-json/ 참조

6)pom.xml에 다음과 같이 dependency를 추가한다.

```
<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
   <version>2.7.2</version>
</dependency>
```

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

```
-jackson-databind-2.7.2.jar
-jackson-annotations-2.7.0.jar
-jackson-core-2.7.2.jar
```

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.

```
<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.

```
package com.javasoft.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

import com.javasoft.service.UserService;
```

```
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

```
package com.javasoft.vo;

import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

import org.springframework.stereotype.Component;

@XmlRootElement(name="userList")
@XmlAccessorType(XmlAccessType.FIELD)
@Component
public class UserListVO {
   @XmlElement(name="user")
   private List<UserVO> userList;

   public List<UserVO> getUserList() {
      return userList;
   }

   public void setUserList(List<UserVO> userList) {
      this.userList = userList;
   }
}
```

-XML 문서는 반드시 단 하나의 root element를 가져야 한다.
-여러 UserVO를 표현하려면 root element로 사용할 또 다른 Java class가 필요하다.
-새로 생성한 UserListVO객체는 이 객체가 root element에 해당하는 객체이며, root element 이름을 userList로 설정하겠다는 의미로 @XmlRootElement(name="userList") 설정을 추가했다.
-그리고 userList 변수 위에도 @XmlElement(name="user")를 추가했는데, UserVO 마다 element 이름을 user로 변경할 것이다.

7)src/com.javasoft.vo.UserVO.java 수정

```
package com.javasoft.vo;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlRootElement;

import org.springframework.stereotype.Component;

@XmlRootElement(name="user")
@XmlAccessorType(XmlAccessType.FIELD)
@Component
public class UserVO {
   @XmlAttribute
   private String userId;
```

-VO class에 선언된 @XmlAccessorType은 UserVO 객체를 XML로 변환할 수 있다는 의미이다.
-그리고 XmlAccessType.FIELD 때문에 이 객체가 가진 필드, 즉 변수들은 자동으로 자식 element로 표현된다.
-하지만, 이 중에서 userId에만 @XmlAttribute가 붙었는데, 이는 userId를 속성으로 표현하라는 의미이다.
-만일 JSON 변환시 @JsonIgnore가 변환시 제외하는 것처럼, XML변환시에도 제외할 변수는 @XmlTransient를 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 MappingJackson2HttpMessageConverter를 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">
```

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

9)UserController.java 코드 추가

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

10실행결과

```
        <userList>
            <user userId="jimin">
                <name>한지민</name>
                <gender>여</gender>
                <city>서울</city>
            </user>
        </userList>
```