1. MVC개요
  1)Model-View-Controller pattern의 개념
    -Software 공학에서 사용되는 architecture pattern
    -주 목적은 business logic과 presentation logic을 분리하기 위함
    -이 pattern을 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 data나 사용자에게 출력할 data를 다루는 역할
    -여러 개의 data 변경 작업(추가, 변경, 삭제)을 하나의 작업으로 묶는 tansaction을 다루는 역할
    -DAO class, Service class에 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 check처럼 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 : HelloWorld > Next
4)Enter a topLevelPackage : com.example.biz > Finish
5)pom.xml 수정하기

```
<properties>
  <java-version>1.8</java-version>
  <org.springframework-version>4.3.24.RELEASE</org.springframework-version>
  <org.aspectj-version>1.9.4</org.aspectj-version>
  <org.slf4j-version>1.7.26</org.slf4j-version>
</properties>
...
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>4.0.1</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>javax.servlet.jsp-api</artifactId>
  <version>2.3.3</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

6)pom.xml > right-click > Run As > Maven install
[INFO] BUILD SUCCESS

7)HelloWorld project > right-click > Properties > Project Facets > Select Java > Change Version 1.8
-Select Runtimes > Check Apache Tomcat v9.0 > Click Apply and Close

8)Open src/main/java/com.example.biz/HomeController.java

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

10)http://localhost:8080/biz/

Hello world!

The time on the server is 2019년 6월 11일 (화) 오후 11시 40분 58초.<--원래 한글 깨짐

11)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

12)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 loading하면서 /WEB-INF/spring/appServlet/servlet-context.xml를 parameter로 초기화한다.
-servlet-context.xml에서 <context:component-scan base-package="com.example.biz" />를 통해 scan을 base-package에서 한다.
-com.example.biz에 @Controller를 찾는다.
-@Controller가 있는 HomeController.java에서 @RequestMapping(value = "/", method = RequestMethod.GET)가 설정되어 있는 method인 public String home(Locale locale, Model model)를 찾는다. 왜냐하면 지금 Browser가 요청한 method는 GET이고, 요청 경로는 /이기 때문이다.
-serverTime을 설정하고 model의 addAttributemethod를 통해 View에게 사용할 값을 저장한다. 그리고 return "home"을 통해 jsp file이름을 반환한다.
-다시 servlet-context.xml에서 ViewResolver는 prefix가 /WEB-INF/views/이고, suffix가 .jsp이며 방금 반환된 jspfile 이름인 home은 prefix + file 이름 + suffix를 하면 /WEB-INF/views/home.jsp가 된다.
-그래서 home.jsp를 Browser에게 전송한다. 이때 JSP는 Model에 저장된 servetTime을 함께 View에 출력하게 된다.

13)Context name 변경하기
-server.xml에서 다음과 같이 수정한다.

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

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

12. Lab : resources folder 이용하기

1)그림 경로 알아내기
-src/main/webapp/resources/에 images folder를 STS Package Explorer에서 생성한다.
-Download받은 image를 src/main/webapp/resources/images/에 넣는다.
-home.jsp에 아래 code를 추가한다.

```
<p><img src="/demo/resources/images/apple.jpg" /></p>
```

-Image가 잘 나온다.

2)Image 경로 변경
-apple.jpg image 경로를 src/main/webapp/images/로 이동.
-하지만 이렇게 하면 image가 보이지 않는다.
-왜냐하면, servlet-context.xml에서 resource의 경로는 <resources mapping="/resources/**" location="/resources/" />이기 때문.
-즉, 기본적으로 resources folder 아래에서 resource를 찾는다.

3)<resources />추가
-resources folder처럼 하위에 images folder를 생성하고 image를 넣고 home.jsp에 아래의 code를 추가한다.

```
<p><img src="/biz/resources/images/apple.jpg" /></p>
<p><img src="/biz/images/apple.jpg" /></p>
```

-하지만 아래의 image는 보이지 않는다.

-왜냐하면 새로 추가한 images folder는 servlet-context.xml에서 설정하지 않았기 때문.
-Image를 보이게 하기 위해 servlet-context.xml에 아래의 code를 추가한다.
<resources mapping="/resources/**" location="/resources/" />
<resources mapping="/images/**" location="/images/" />
-src/main/webapp/images folder 추가
-Project right-click > Run As > Run on Server > Restart >
--Image가 제대로 2개가 나온다.

13. Lab : Controller Class 제작하기
1)제작순서
-@Controller를 이용한 class 생성
-@RequestMapping을 이용한 요청 경로 지정
-요청 처리 method 구현
-View 이름 return

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

```
@Controller
public class UserController {
```

2)요청 처리 method 생성

```
package com.example.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){
    /*
    model.addAttribute("username", "한지민");
    model.addAttribute("userage", 24);
    model.addAttribute("job", "Developer");
    return "view";
    */
    model.addAttribute("currentDate", new java.util.Date());
    return "view";    // /WEB-INF/views/view  + .jsp
  }

  @RequestMapping("/fruits")
  public String fruits(Model model){
    String [] array = {"Apple", "Mango", "Lemon", "Grape"};

    model.addAttribute("fruits", array);

    return "fruits";    // /WEB-INF/views/fruits  + .jsp
  }
}
```

```
246
247   3)View에 Data 전달
248     -src/main/webapp/WEB-INF/views/view.jsp 생성
249
250       <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
251       <!DOCTYPE html>
252       <html>
253         <head>
254           <meta charset="UTF-8">
255           <title>Insert title here</title>
256         </head>
257         <body>
258           <h1>view.jsp 입니다.</h1>
259           현재 날짜와 시간은 ${currentDate} 입니다.
260         </body>
261       </html>
262
263     -src/main/webapp/WEB-INF/views/fruits.jsp 생성
264
265       <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
266       <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
267       <!DOCTYPE html>
268       <html>
269       <head>
270       <meta charset="UTF-8">
271       <title>Insert title here</title>
272       </head>
273       <body>
274         <h2>fruits.jsp</h2>
275         <ul>과일 종류
276         <c:forEach items="${fruits}" var="fruit">
277           <li>${fruit}</li>
278         </c:forEach>
279         </ul>
280       </body>
281       </html>
282
283     -http://localhost:8080/demo/view  --> /view.jsp
284     -http://localhost:8080/demo/fruits  --> /fruits.jsp
285
286   4)View에 ModelAndView 객제로 data 전달
287
288     @RequestMapping(value = "/demo", method = RequestMethod.GET)
289     public ModelAndView demo() {
290       /*
291       ModelAndView mav = new ModelAndView("view2");
292       mav.addObject("username", "한지민");
293       mav.addObject("currentDate", new java.util.Date());
294       return mav;
295       */
296       ModelAndView mav = new ModelAndView();
297       mav.addObject("userid", "example");
```

```
298         mav.addObject("passwd", "12345678");
299         mav.setViewName("/demo");
300         return mav;
301       }
302
303       -src/main/webapp/WEB-INF/views/demo.jsp 생성
304
305         <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
306         <!DOCTYPE html">
307         <html>
308           <head>
309             <meta charset="UTF-8">
310             <title>Insert title here</title>
311           </head>
312           <body>
313             아이디 : ${userid} <br />
314             패스워드 : ${passwd}
315           </body>
316         </html>
317
318       -http://localhost:8080/demo/demo  --> /demo.jsp
319         아이디 : example
320         패스워드 : 12345678
321
322     5)Controller class에 @RequestMapping 적용
323       -src/main/java/com.example.biz.StudentController.java 생성
324
325         package com.example.biz;
326
327         import org.springframework.stereotype.Controller;
328         import org.springframework.web.bind.annotation.RequestMapping;
329         import org.springframework.web.bind.annotation.RequestMethod;
330         import org.springframework.web.servlet.ModelAndView;
331
332         @Controller
333         @RequestMapping("/bbs")
334         public class StudentController {
335
336           @RequestMapping(value="/get", method = RequestMethod.GET)
337           public ModelAndView getStudent() {
338
339             ModelAndView mav = new ModelAndView();
340             mav.setViewName("/bbs/get");   //  /WEB-INF/views/bbs/get.jsp
341             mav.addObject("name", "한지민");
342             mav.addObject("age", 25);
343             return mav;
344           }
345         }
346
347       -src/main/webapp/WEB-INF/views/bbs/get.jsp
348         <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
349         <!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>

      -http://localhost:8080/demo/bbs/get
        학생 이름 : 한지민
        학생 나이 : 25


14. Lab : Form Data 처리하기
  1)Package Explorer > right-click > New > Spring Legacy Project
  2)Select Spring MVC Project
  3)Project name : MVCDemo > Next
  4)Enter a topLevelPackage : com.example.biz > Finish
  5)pom.xml 수정하기
    <properties>
      <java-version>1.8</java-version>
      <org.springframework-version>4.3.24.RELEASE</org.springframework-version>
      <org.aspectj-version>1.9.4</org.aspectj-version>
      <org.slf4j-version>1.7.26</org.slf4j-version>
    </properties>
    ...
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>2.3.3</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>

  6)pom.xml > right-click > Run As > Maven install
    [INFO] BUILD SUCCESS

  7)HelloWorld project > right-click > Properties > Project Facets > Select Java > Change
  Version 1.8
```

-Select Runtimes > Check Apache Tomcat v9.0 > Click Apply and Close

8)src/main/java/com.example.biz/RequestController.java 생성

```
package com.example.biz;

import org.springframework.stereotype.Controller;

@Controller
public class RequestController {
```

9)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>
```

-Project right-click > Run As > Run on Server > restart

-localhost:8080/biz/confirm?name=한지민&gender=여성&age=25&userid=jimin&passwd=1234

```
  아이디 : jimin
  패스워드 : 1234
  사용자 이름 : 한지민
```

```
455       나이 : 25
456       성별 : 여성
457
458   10)@RequestParam annotation 이용하기
459
460     -RequestController.java
461       @RequestMapping(value="/confirm", method=RequestMethod.GET)
462       public String confirm(@RequestParam("userid") String userid,
463                       @RequestParam("passwd") String passwd,
464                       @RequestParam("name") String name,
465                       @RequestParam("age") int age,
466                       @RequestParam("gender") String gender ,Model model) {
467
468         model.addAttribute("userid", userid);
469         model.addAttribute("passwd", passwd);
470         model.addAttribute("name", name);
471         model.addAttribute("age", age);
472         model.addAttribute("gender", gender);
473         return "confirm";   //  /WEB-INF/views/confirm.jsp
474       }
475
476     -src/main/webapp/WEB-INF/views/confirm.jsp
477
478       <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
479       <!DOCTYPE html>
480       <html>
481         <head>
482           <meta charset="UTF-8">
483           <title>Insert title here</title>
484         </head>
485         <body>
486           아이디 : ${userid} <br />
487           패스워드 : ${passwd} <br />
488           사용자 이름 : ${name} <br />
489           나이 : ${age} <br />
490           성별 : ${gender} <br />
491         </body>
492       </html>
493
494     -localhost:8080/biz/confirm?name=한지민&gender=여성&age=25&userid=jimin&passwd=1234
495       아이디 : jimin
496       패스워드 : 1234
497       사용자 이름 : 한지민
498       나이 : 25
499       성별 : 여성
500
501   11)Data Commander 객체 이용하기1
502     -src/main/java/com.example.vo.UserVO.java 생성
503
504       package com.example.vo;
505
506       public class UserVO {
507         private String userid;
```

```
508         private String passwd;
509         private String name;
510         private int age;
511         private String gender;
512 
513         public UserVO(){}
514         public UserVO(String userid, String passwd, String name, int age, String gender){
515           this.userid = userid;
516           this.passwd = passwd;
517           this.name = name;
518           this.age = age;
519           this.gender = gender;
520         }
521         public String getUserid() {
522           return userid;
523         }
524         public void setUserid(String userid) {
525           this.userid = userid;
526         }
527         public String getPasswd() {
528           return passwd;
529         }
530         public void setPasswd(String passwd) {
531           this.passwd = passwd;
532         }
533         public String getName() {
534           return name;
535         }
536         public void setName(String name) {
537           this.name = name;
538         }
539         public int getAge() {
540           return age;
541         }
542         public void setAge(int age) {
543           this.age = age;
544         }
545         public String getGender() {
546           return gender;
547         }
548         public void setGender(String gender) {
549           this.gender = gender;
550         }
551         @Override
552         public String toString() {
553           return "UserVO [userid=" + userid + ", passwd=" + passwd + ", name=" + name + ",
              age=" + age + ", gender="
554                 + gender + "]";
555         }
556       }
557 
558     -RequestController.java
559 
560       @RequestMapping(value="/confirm", method=RequestMethod.GET)
```

```
561         public String confirm(@RequestParam("userid") String userid,
562               @RequestParam("passwd") String passwd,
563               @RequestParam("name") String name,
564               @RequestParam("age") int age,
565               @RequestParam("gender") String gender  ,Model model) {
566
567            UserVO userVO = new UserVO();
568            userVO.setUserid(userid);
569            userVO.setPasswd(passwd);
570            userVO.setName(name);
571            userVO.setAge(age);
572            userVO.setGender(gender);
573
574            model.addAttribute("userVO", userVO);
575
576            return "confirm1";   //  /WEB-INF/views/confirm1.jsp
577         }
578
579       -src/main/webapp/WEB-INF/views/confirm1.jsp
580
581         <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
582         <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
583         <c:set var="user" value="${userVO}"/>
584         <!DOCTYPE html>
585         <html>
586         <head>
587         <meta charset="UTF-8">
588         <title>Insert title here</title>
589         </head>
590         <body>
591            <h1>confirm1.jsp</h1>
592            <h2>사용자 정보</h2>
593            아이디 : ${user.userid} <br />
594            패스워드 : ${user.passwd} <br />
595            이름 : ${user.name} <br />
596            나이 : ${user.age} <br />
597            성별 : ${user.gender}
598         </body>
599         </html>
600
601       -localhost:8080/biz/confirm?name=한지민&gender=여성&age=25&userid=jimin&passwd=1234
602         confirm1.jsp
603
604         사용자 정보
605
606         아이디 : jimin
607         패스워드 : 1234
608         사용자 이름 : 한지민
609         나이 : 25
610         성별 : 여성
611
612    12)Data Commander 객체 이용하기2
613
```

```
-RequestController.java

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

    return "confirm2";   //  /WEB-INF/views/confirm2.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>confirm2.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>

-localhost:8080/biz/confirm?name=한지민&gender=여성&age=25&userid=jimin&passwd=1234
  confirm2.jsp

  사용자 정보

  아이디 : jimin
  패스워드 : 1234
  사용자 이름 : 한지민
  나이 : 25
  성별 : 여성

13)@PathVariable 이용하기

  -RequestController.java

  @RequestMapping(value="/confirm/{userid}/{passwd}/{name}/{age}/{gender}",
  method=RequestMethod.GET)
  public String confirm(@PathVariable String userid, @PathVariable String passwd,
                        @PathVariable String name, @PathVariable int age,
                  @PathVariable String gender,Model model) {
    model.addAttribute("userInfo", new UserVO(userid, passwd, name, age, gender));
    return "confirm3";
  }
```

```
666
667     -src/main/webapp/WEB-INF/views/confirm3.jsp
668
669       <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
670       <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
671       <c:set var="user" value="${userInfo}"/>
672       <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
          "http://www.w3.org/TR/html4/loose.dtd">
673       <html>
674       <head>
675       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
676       <title>Insert title here</title>
677       </head>
678       <body>
679         <h1>confirm3.jsp</h1>
680         <h2>사용자 정보</h2>
681         아이디 : ${user.userid} <br />
682         패스워드 : ${user.passwd} <br />
683         이름 : ${user.name} <br />
684         나이 : ${user.age} <br />
685         성별 : ${user.gender}
686       </body>
687       </html>
688
689     -localhost:8080/biz/confirm/jimin/1234/한지민/25/여성
690       confirm3.jsp
691
692       사용자 정보
693
694       아이디 : jimin
695       패스워드 : 1234
696       사용자 이름 : 한지민
697       나이 : 25
698       성별 : 여성
699
700
701 15. Lab : @RequestMapping Parameter
702   1)GET 방식과 POST 방식
703
704     -src/main/java/com.example.biz/HomeController.java
705
706       @RequestMapping(value="/login", method=RequestMethod.POST)
707       public String login(@RequestParam("userid") String userid,
708                           @RequestParam("passwd") String passwd,
709                           Model model) {
710
711         model.addAttribute("userid", userid);
712         model.addAttribute("passwd", passwd);
713         return "login";
714       }
715
716     -src/main/webapp/resources/login.html
717       <!DOCTYPE html>
```

```
<html>
<head>
<meta charset="UTF-8">
<title>로그인 폼</title>
</head>
<body>
  <form method="GET" action="/biz/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/biz/resources/login.html에서 submit 하면 405 error 발생
-왜냐하면 서로의 method가 불일치하기 때문
-해결방법
-src/main/java/com.example.biz/HomeController.java 수정
-즉 login method(요청 처리 method)의 이름은 같지만 parameter의 type과 return type이 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 />
```

```
769           패스워드 : ${passwd}
770         </body>
771         </html>
772
773       -http://localhost:8080/biz/resources/login.html
774         아이디 : jimin
775         패스워드 : 1234
776
777     2)@ModelAttribute annotation 이용하기
778       -@ModelAttribute annotation을 이용하면 Data Commander 객체의 이름을 변경할 수 있다.
779       -src/main/webapp/resources/register.html
780
781         <!DOCTYPE html>
782         <html>
783         <head>
784         <meta charset="UTF-8">
785         <title>회원가입 폼</title>
786         </head>
787         <body>
788           <form method="POST" action="/biz/register">
789             아이디 : <input type="text" name="userid" /><br />
790             패스워드 : <input type="password" name="passwd" /><br />
791             이름 : <input type="text" name="name" /><br />
792             나이 : <input type="number" name="age" /><br />
793             성별 : <input type="radio" name="gender" value="남성" />남성   
794                  <input type="radio" name="gender" value="여성" />여성<br />
795             <input type="submit" value="가입하기" />
796           </form>
797         </body>
798         </html>
799
800       -src/main/java/com.example.biz/HomeController.java
801
802         @RequestMapping(value="/register", method=RequestMethod.POST)
803         public String register(@ModelAttribute("u") UserVO userVO) {    //userVO가 아니라 u로 변경
804
805           return "register";
806         }
807
808       -src/main/webapp/WEB-INF/views/register.jsp
809
810         <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
811         <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
812         <c:set var="user" value="${u}"/>
813         <!DOCTYPE html>
814         <html>
815         <head>
816         <meta charset="UTF-8">
817         <title>Insert title here</title>
818         </head>
819         <body>
820           <h1>사용자 정보</h1>
821           <ul>
```

```
822             <li>아이디 : ${user.userid}</li>
823             <li>패스워드 : ${user.passwd}</li>
824             <li>이름 : ${user.name}</li>
825             <li>나이 : ${user.age}</li>
826             <li>성별 : ${user.gender}</li>
827           </ul>
828         </body>
829         </html>
830
831       -Spring에서 POST 방식으로 Data를 보낼 때 한글깨짐 현상 발생
832       -해결점
833       -web.xml
834
835         <filter>
836           <filter-name>encodingFilter</filter-name>
837           <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
838           <init-param>
839             <param-name>encoding</param-name>
840             <param-value>UTF-8</param-value>
841           </init-param>
842         </filter>
843         <filter-mapping>
844           <filter-name>encodingFilter</filter-name>
845           <url-pattern>/*</url-pattern>
846         </filter-mapping>
847
848       -http://localhost:8080/biz/resources/register.html -->
849         http://localhost:8080/biz/register
850           사용자 정보
851
852           아이디 : jimin
853           패스워드 : 1234
854           사용자 이름 : 한지민
855           나이 : 25
856           성별 : 여성
857
858     3)redirect: 키워드 이용하기
859       -src/main/java/com.example.biz/HomeController.java
860
861         @RequestMapping("/verify")
862         public String verify(HttpServletRequest request, Model model) {
863           String userid = request.getParameter("userid");
864           if(userid.equals("admin")) {     //만일 userid가 admin 이면 /admin으로 리다이렉트
865             return "redirect:admin";
866           }
867           //return "redirect:user";         //만일 userid가 admin 이 아니면 /user로 리다이렉트
868           return "redirect:http://www.naver.com";    //절대 경로도 가능
869         }
870
871         @RequestMapping("/admin")
872         public String verify1(Model model) {
873           model.addAttribute("authority", "관리자권한");
874           return "admin";
875         }
```

```
876
877         @RequestMapping("/user")
878         public String verify2(Model model) {
879           model.addAttribute("authority", "일반사용자");
880           return "user";
881         }
882
883       -/src/main/webapp/WEB-INF/views/admin.jsp
884         <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
885         <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
          "http://www.w3.org/TR/html4/loose.dtd">
886         <html>
887         <head>
888         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
889         <title>Insert title here</title>
890         </head>
891         <body>
892           <h1>관리자 페이지</h1>
893           권한 : ${authority}
894         </body>
895         </html>
896
897       -/src/main/webapp/WEB-INF/views/user.jsp
898
899         <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
900         <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
          "http://www.w3.org/TR/html4/loose.dtd">
901         <html>
902         <head>
903         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
904         <title>Insert title here</title>
905         </head>
906         <body>
907           <h1>일반 사용자 페이지</h1>
908           권한 : ${authority}
909         </body>
910         </html>
911
912       -http://localhost:8080/biz/verify?userid=admin --> http://localhost:8080/biz/admin
913       -http://localhost:8080/biz/verify?userid=user --> https://www.naver.com
914
915
916   16. Lab : Database와 연동하기
917     1)Package Explorer > right-click > New > Spring Legacy Project
918     2)Select Spring MVC Project
919     3)Project name : MVCDemo1 > Next
920     4)Enter a topLevelPackage : com.example.biz > Finish
921     5)pom.xml 수정하기
922       <properties>
923         <java-version>1.8</java-version>
924         <org.springframework-version>4.3.24.RELEASE</org.springframework-version>
925         <org.aspectj-version>1.9.4</org.aspectj-version>
```

```
926       <org.slf4j-version>1.7.26</org.slf4j-version>
927     </properties>
928     ...
929     <dependency>
930       <groupId>javax.servlet</groupId>
931       <artifactId>javax.servlet-api</artifactId>
932       <version>4.0.1</version>
933       <scope>provided</scope>
934     </dependency>
935     <dependency>
936       <groupId>javax.servlet.jsp</groupId>
937       <artifactId>javax.servlet.jsp-api</artifactId>
938       <version>2.3.3</version>
939       <scope>provided</scope>
940     </dependency>
941     <dependency>
942       <groupId>junit</groupId>
943       <artifactId>junit</artifactId>
944       <version>4.12</version>
945       <scope>test</scope>
946     </dependency>
947
948   6)pom.xml > right-click > Run As > Maven install
949     [INFO] BUILD SUCCESS
950
951   7)HelloWorld project > right-click > Properties > Project Facets > Select Java > Change
      Version 1.8
952     -Select Runtimes > Check Apache Tomcat v9.0 > Click Apply and Close
953
954   8)Create Table in MariaDB
955     CREATE TABLE Member
956     (
957       userid        VARCHAR(20),
958       username        VARCHAR(20)   NOT NULL,
959       userage        TINYINT   NOT NULL,
960       gender      VARCHAR(10)   NOT NULL,
961       city          VARCHAR(50),
962       CONSTRAINT member_userid_pk  PRIMARY KEY(userid)
963     );
964     -반드시 test database의 조합을 utf8_general_ci로 맞출 것
965     -반드시 Member table의 기본조합이 utf8_general_ci 임을 확인할 것
966
967   9)src/main/webapp/static folder 생성
968     -src/main/webapp/static/css folder
969     -src/main/webapp/static/images folder
970     -src/main/webapp/static/js folder
971       --jquery-1.12.4.js
972     -src/main/webapp/static/register.html
973       <!DOCTYPE html>
974       <html lang="en">
975       <head>
976         <meta charset="UTF-8">
977         <title>회원 가입</title>
978       </head>
```

```
      <body>
        <h1>회원 가입 창</h1>
        <form action="/biz/create" method="post">
          <ul>
            <li>ID : <input type="text" name="userid" /></li>
            <li>이름 : <input type="text" name="username" /></li>
            <li>나이 : <input type="number" name="age" /></li>
            <li>성별 : <input type="radio" name="gender" value="남성"/>남성
                       <input type="radio" name="gender" value="여성"/>여성</li>
            <li>거주지 : <input type="text" name="city" /></li>
            <li><input type="submit" value="가입하기" /></li>
          </ul>
        </form>
      </body>
      </html>

  10)src/main/webapp/WEB-INF/spring/appServlet/sevlet-context.xml 수정
    <resources mapping="/static/**" location="/static/" /> 추가

    <context:component-scan base-package="com.example" /> 수정

  11)src/main/resources/mariadb.properties
    db.driverClass=org.mariadb.jdbc.Driver
    db.url=jdbc:mariadb://localhost:3306/test
    db.username=root
    db.password=javamariadb

  12)Spring JDBC 설치
    -JdbcTemplate를 사용하기 위해 pom.xml에 다음 dependency를 추가해야 함.

      <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-jdbc</artifactId>
          <version>4.3.24.RELEASE</version>
      </dependency>

    -pom.xml에 붙여 넣고 Maven Install 하기
      [INFO] BUILD SUCCESS

  13)MariaDB Jdbc Driver library 검색 및 설치
    -Maven Repository 에서 'mariadb'로 검색하여 MariaDB Java Client를 설치한다.

      <dependency>
          <groupId>org.mariadb.jdbc</groupId>
          <artifactId>mariadb-java-client</artifactId>
          <version>2.4.1</version>
      </dependency>

    -pom.xml에 붙여 넣고 Maven Install 하기
      [INFO] BUILD SUCCESS

  14)src/main/webapp/WEB-INF/spring/root-context.xml
    <?xml version="1.0" encoding="UTF-8"?>
```

```
1033      <beans xmlns="http://www.springframework.org/schema/beans"
1034        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1035        xmlns:context="http://www.springframework.org/schema/context"
1036        xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
1037          http://www.springframework.org/schema/context
              http://www.springframework.org/schema/context/spring-context-4.3.xsd">
1038
1039        <!-- Root Context: defines shared resources visible to all other web components -->
1040        <context:property-placeholder location="classpath:mariadb.properties"/>
1041        <bean id="dataSource"
            class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
1042          <property name="driverClass" value="${db.driverClass}" />
1043          <property name="url" value="${db.url}" />
1044          <property name="username" value="${db.username}" />
1045          <property name="password" value="${db.password}" />
1046        </bean>
1047
1048        <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
1049          <property name="dataSource" ref="dataSource" />
1050        </bean>
1051      </beans>
1052
1053    15)src/test/java/com.example.biz/TestApp class 생성(JUnit Test Case)
1054      package com.example.biz;
1055
1056      import org.junit.Before;
1057      import org.junit.Test;
1058      import org.springframework.context.ApplicationContext;
1059      import org.springframework.context.support.GenericXmlApplicationContext;
1060      import org.springframework.jdbc.core.JdbcTemplate;
1061
1062      public class TestApp {
1063        private ApplicationContext ctx;
1064
1065        @Before
1066        public void init() {
1067          this.ctx = new
              GenericXmlApplicationContext("file:src/main/webapp/WEB-INF/spring**/root-context.x
              ml");
1068        }
1069        @Test
1070        public void test() {
1071          JdbcTemplate jdbcTemplate = this.ctx.getBean("jdbcTemplate", JdbcTemplate.class);
1072          System.out.println(jdbcTemplate);
1073        }
1074      }
1075      -Run as > JUnit Test > Green bar
1076
1077    16)package 생성
1078      -src/main/java/com.example.vo
1079      -src/main/java/com.example.dao
1080      -src/main/java/com.example.service
1081
```

17)src/com.example.vo.MemberVO class 생성

```
package com.example.vo;

public class MemberrVO {
  private String userid;
  private String username;
  private int age;
  private String gender;
  private String city;

  public MemberVO(){}
  public MemberVO(String userid, String username, int age, String gender, String city) {
    this.userid = userid;
    this.username = username;
    this.age = age;
    this.gender = gender;
    this.city = city;
  }
  public String getUserid() {
    return userid;
  }
  public void setUserId(String userid) {
    this.userid = userid;
  }
  public String getUsername() {
    return username;
  }
  public void setUsername(String username) {
    this.username = username;
  }
  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;
  }
  public String getCity() {
    return city;
  }
  public void setCity(String city) {
    this.city = city;
  }
  @Override
  public String toString() {
    return "UserVO [userid=" + userid + ", username=" + username + ", age =" + age +
    ", gender=" + gender + ", city=" + city + "]";
  }
```

```
1135        }
1136
1137    18)com/example.dao
1138      -MemberDao interface
1139        package com.example.dao;
1140
1141        import java.util.List;
1142
1143        import com.example.vo.MemberVO;
1144
1145        public interface MemberDao {
1146          int create(MemberVO memberVo);
1147          MemberVO read(String userid);
1148          List<MemberVO> readAll();
1149          int update(MemberVO memberVo);
1150          int delete(String userid);
1151        }
1152
1153      -MemberDaoImpl.java
1154        package com.example.dao;
1155
1156        import java.sql.ResultSet;
1157        import java.sql.SQLException;
1158        import java.util.List;
1159
1160        import org.springframework.beans.factory.annotation.Autowired;
1161        import org.springframework.jdbc.core.JdbcTemplate;
1162        import org.springframework.jdbc.core.RowMapper;
1163        import org.springframework.stereotype.Repository;
1164
1165        import com.example.vo.MemberVO;
1166
1167        @Repository("memberDao")
1168        public class MemberDaoImpl implements MemberDao {
1169          @Autowired
1170          JdbcTemplate jdbcTemplate;
1171
1172          @Override
1173          public int create(MemberVO memberVo) {
1174            String sql = "INSERT INTO Member VALUES(?,?,?,?,?)";
1175            return this.jdbcTemplate.update(sql, memberVo.getUserid(),
1176                memberVo.getUsername(), memberVo.getAge(),
1177                memberVo.getGender(), memberVo.getCity());
1178          }
1179
1180          @Override
1181          public MemberVO read(String userid) {
1182            String sql = "SELECT * FROM Member WHERE userid = ?";
1183            return this.jdbcTemplate.queryForObject(sql, new Object[] {userid},
1184                        new MyRowMapper());
1185          }
1186
1187          class MyRowMapper implements RowMapper<MemberVO>{
1188            @Override
```

```
1189            public MemberVO mapRow(ResultSet rs, int rowNum) throws SQLException {
1190              MemberVO memberVo = new MemberVO(rs.getString("userid"),
1191                   rs.getString("username"), rs.getInt("userage"),
1192                   rs.getString("gender"), rs.getString("city"));
1193              return memberVo;
1194            }
1195          }
1196          @Override
1197          public List<MemberVO> readAll() {
1198            String sql = "SELECT * FROM Member ORDER BY userid DESC";
1199            return this.jdbcTemplate.query(sql, new MyRowMapper());
1200          }
1201
1202          @Override
1203          public int update(MemberVO memberVo) {
1204            String sql = "UPDATE Member SET userage = ?, gender = ?, city = ? " +
1205                      "WHERE userid = ?";
1206            return this.jdbcTemplate.update(sql, memberVo.getAge(),
1207                 memberVo.getGender(), memberVo.getCity(), memberVo.getUserid());
1208          }
1209
1210          @Override
1211          public int delete(String userid) {
1212            String sql = "DELETE FROM Member WHERE userid = ?";
1213            return this.jdbcTemplate.update(sql, userid);
1214          }
1215        }
1216
1217    19)com.example.service
1218      -MemberService interface
1219        package com.example.service;
1220
1221        import java.util.List;
1222
1223        import com.example.vo.MemberVO;
1224
1225        public interface MemberService {
1226          int create(MemberVO memberVo);
1227          MemberVO read(String userid);
1228          List<MemberVO> readAll();
1229          int update(MemberVO memberVo);
1230          int delete(String userid);
1231        }
1232
1233      -MemberServiceImpl.java
1234        package com.example.service;
1235
1236        import java.util.List;
1237
1238        import org.springframework.beans.factory.annotation.Autowired;
1239        import org.springframework.stereotype.Service;
1240
1241        import com.example.dao.MemberDao;
1242        import com.example.vo.MemberVO;
```

```
1243
1244        @Service("memberService")
1245        public class MemberServiceImpl implements MemberService {
1246          @Autowired
1247          MemberDao memberDao;
1248
1249          @Override
1250          public int create(MemberVO memberVo) {
1251            return this.memberDao.create(memberVo);
1252          }
1253
1254          @Override
1255          public MemberVO read(String userid) {
1256            return this.memberDao.read(userid);
1257          }
1258
1259          @Override
1260          public List<MemberVO> readAll() {
1261            return this.memberDao.readAll();
1262          }
1263
1264          @Override
1265          public int update(MemberVO memberVo) {
1266            return this.memberDao.update(memberVo);
1267          }
1268
1269          @Override
1270          public int delete(String userid) {
1271            return this.memberDao.delete(userid);
1272          }
1273
1274        }
1275
1276    20)com.example.biz
1277      -HomeController.java
1278        package com.example.biz;
1279
1280        import java.util.List;
1281
1282        import org.springframework.beans.factory.annotation.Autowired;
1283        import org.springframework.stereotype.Controller;
1284        import org.springframework.ui.Model;
1285        import org.springframework.web.bind.annotation.PathVariable;
1286        import org.springframework.web.bind.annotation.RequestMapping;
1287        import org.springframework.web.bind.annotation.RequestMethod;
1288        import org.springframework.web.bind.annotation.RequestParam;
1289
1290        import com.example.service.MemberService;
1291        import com.example.vo.MemberVO;
1292
1293        /**
1294         * Handles requests for the application home page.
1295         */
1296        @Controller
```

```
1297        public class HomeController {
1298          @Autowired
1299          MemberService memberService;
1300
1301          @RequestMapping(value = "/create", method = RequestMethod.POST)
1302          public String home(MemberVO memberVo, Model model) {
1303            int row = this.memberService.create(memberVo);
1304            if(row == 1) model.addAttribute("status", "Insert Success");
1305            else model.addAttribute("status", "Insert Failure");
1306            return "create";    //   /WEB-INF/views/create.jsp
1307          }
1308
1309          @RequestMapping(value = "/list", method = RequestMethod.GET)
1310          public String list(Model model) {
1311            List<MemberVO> list = this.memberService.readAll();
1312            model.addAttribute("userlist", list);
1313            return "list";    // /WEB-INF/views/list.jsp
1314          }
1315
1316          @RequestMapping(value = "/view/{userid}", method = RequestMethod.GET)
1317          public String view(@PathVariable String userid, Model model) {
1318            MemberVO memberVo = this.memberService.read(userid);
1319            model.addAttribute("member", memberVo);
1320            return "view";
1321          }
1322
1323          @RequestMapping(value = "/delete/{userid}", method = RequestMethod.GET)
1324          public String delete(@PathVariable String userid) {
1325            this.memberService.delete(userid);
1326            return "redirect:/list";
1327          }
1328
1329          @RequestMapping(value = "/update", method = RequestMethod.POST)
1330          public String update(@RequestParam("userid") String userid,
1331              @RequestParam("age") int age,
1332              @RequestParam("gender") String gender,
1333              @RequestParam("city") String city) {
1334            this.memberService.update(
1335                new MemberVO(userid, "", age, gender, city));
1336            return "redirect:/list";
1337          }
1338        }
1339
1340    21)views/create.jsp
1341      <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
1342      <!DOCTYPE html>
1343      <html>
1344      <head>
1345      <meta charset="UTF-8">
1346      <title>Insert title here</title>
1347      </head>
1348      <body>
1349        <h1>${status}</h1>
```

```
1350      </body>
1351      </html>
1352
1353    22)views/list.jsp
1354      <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
1355      <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
1356      <!DOCTYPE html>
1357      <html>
1358      <head>
1359      <meta charset="UTF-8">
1360      <title>Insert title here</title>
1361      </head>
1362      <body>
1363        <h1>Member List</h1>
1364        <table border='1'>
1365          <thead>
1366            <tr>
1367              <th>아이디</th><th>이름</th><th>나이</th><th>성별</th><th>거주지</th>
1368            </tr>
1369          </thead>
1370          <tbody>
1371            <c:forEach items="${userlist}" var="user">
1372              <tr>
1373                <td><a
                  href="/biz/view/${user.userid}">${user.userid}</a></td><td>${user.username
                  }</td>
1374                <td>${user.age}</td><td>${user.gender }</td>
1375                <td>${user.city }</td>
1376              </tr>
1377            </c:forEach>
1378          </tbody>
1379        </table>
1380      </body>
1381      </html>
1382
1383    23)views/view.jsp
1384      <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
1385      <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
1386      <c:set var="user" value="${member}" />
1387      <!DOCTYPE html>
1388      <html>
1389      <head>
1390        <meta charset="UTF-8">
1391        <title>Insert title here</title>
1392        <script src="/biz/static/js/jquery-1.12.4.js"></script>
1393        <script>
1394          $(function(){
1395            $("#btnList").bind("click", function(){
1396              location.href = "/biz/list";
1397            });
1398            $("#btnDelete").bind("click", function(){
1399              location.href = "/biz/delete/${user.userid}";
```

```
          });
        });
      </script>
    </head>
    <body>
      <h1>${user.username}의정보</h1>
      <form action="/biz/update" method="post">
        <input type="hidden" name="userid" value = "${user.userid}" />
        <ul>
          <li>아이디 : ${user.userid }</li>
          <li>나이 : <input type='number' name="age" value='${user.age}' /></li>
          <li>성별 : <c:if test='${user.gender eq "남성"}'>
              <input type="radio" name="gender" value="남성" checked />남성  
              <input type="radio" name="gender" value="여성" />여성
              </c:if>
              <c:if test='${user.gender eq "여성"}'>
              <input type="radio" name="gender" value="남성" />남성  
            <input type="radio" name="gender" value="여성" checked />여성
          </c:if>
          </li>
          <li>거주지 : <input type="text" name="city" value="${user.city }" /></li>
          <li><input type='submit' value='수정하기' /></li>
          <li><input type='button' value='삭제하기'  id="btnDelete"/></li>
          <li><input type='button' value='목록으로'  id="btnList"/></li>
        </ul>
      </form>
    </body>
    </html>
```

24)POST 발송시 한글 깨짐 처리하기
-web.xml

```
      <filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
          <param-name>encoding</param-name>
          <param-value>UTF-8</param-value>
        </init-param>
      </filter>
      <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
      </filter-mapping>
```

25)Test
http://localhost:8080/biz/static/register.html

17. 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.example.biz > Finish

```
1454    5)UserVO 객체 생성
1455      -src/main/java/com.example.vo package 생성
1456      -src/main/java/com.example.vo.UserVO class
1457
1458        package com.example.vo;
1459
1460        public class UserVO {
1461          private String name;
1462          private int age;
1463          private String userid;
1464          public String getName() {
1465            return name;
1466          }
1467          public void setName(String name) {
1468            this.name = name;
1469          }
1470          public int getAge() {
1471            return age;
1472          }
1473          public void setAge(int age) {
1474            this.age = age;
1475          }
1476          public String getUserid() {
1477            return userid;
1478          }
1479          public void setUserid(String userid) {
1480            this.userid = userid;
1481          }
1482          @Override
1483          public String toString() {
1484            return "UserVO [name=" + name + ", age=" + age + ", userid=" + userid + "]";
1485          }
1486        }
1487
1488    6)Validator를 이용한 검증
1489      -Data Command 객체에서 유효성 검사를 할 수 있다.
1490      -UserValidator 객체 생성
1491      -src/main/java/com.example.biz.UserValidator class
1492
1493        package com.example.biz;
1494
1495        import org.springframework.validation.Errors;
1496        import org.springframework.validation.Validator;
1497
1498        import com.example.vo.UserVO;
1499
1500        public class UserValidator implements Validator {
1501
1502          @Override
1503          public boolean supports(Class<?> arg0) {
1504            //검증할 객체의 class 타입 정보를 반환
1505            return UserVO.class.isAssignableFrom(arg0);
1506          }
1507
```

```
1508          @Override
1509          public void validate(Object obj, Errors errors) {
1510            System.out.println("검증시작");
1511            UserVO userVO = (UserVO)obj;
1512
1513            String username = userVO.getName();
1514            if(username == null || username.trim().isEmpty()) {
1515              System.out.println("이름의 값이 빠졌습니다.");
1516              errors.rejectValue("name", "No Value");
1517            }
1518
1519            int userage = userVO.getAge();
1520            if(userage == 0) {
1521              System.out.println("나이의 값이 빠졌습니다.");
1522              errors.rejectValue("age", "No Value");
1523            }
1524
1525            String userid = userVO.getUserid();
1526            if(userid == null || userid.trim().isEmpty()) {
1527              System.out.println("아이디의 값이 빠졌습니다.");
1528              errors.rejectValue("userid", "No Value");
1529            }
1530          }
1531        }
1532
1533      -src/main/java/com.example.biz/HomeController.java
1534
1535        @RequestMapping(value = "/register", method=RequestMethod.GET)
1536        public String register() {
1537          return "register";
1538        }
1539
1540        @RequestMapping(value = "/register", method=RequestMethod.POST)
1541        public String register(@ModelAttribute("userVO") UserVO userVO, BindingResult result) {
1542          String page = "register_ok";
1543          UserValidator validator = new UserValidator();
1544          validator.validate(userVO, result);
1545          if(result.hasErrors()) {
1546            page = "register";
1547          }
1548          return page;
1549        }
1550
1551      -src/main/webapp/WEB-INF/views/register.jsp
1552        <%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
1553        <!DOCTYPE html>
1554        <html>
1555        <head>
1556        <meta charset="UTF-8">
1557        <title>회원 가입 폼</title>
1558        </head>
1559        <body>
1560          <form action="/biz/register" method="post">
1561            Name : <input type="text" name="name" /><br />
```

```
1562            Age : <input type="number" name="age" /><br />
1563            ID : <input type="text" name="userid" /><br />
1564            <input type="submit" value="가입하기" />
1565          </form>
1566        </body>
1567        </html>
1568
1569      -src/main/webapp/WEB-INF/views/register_ok.jsp
1570        <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
1571        <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
1572        <c:set var="user" value="${userVO}" />
1573        <!DOCTYPE html">
1574        <html>
1575        <head>
1576        <meta charset="UTF-8">
1577        <title>회원 가입 결과 창</title>
1578        </head>
1579        <body>
1580          <ul>
1581            <li>이름 : ${user.name}</li>
1582            <li>나이 : ${user.age}</li>
1583            <li>아이디 : ${user.userid}</li>
1584          </ul>
1585        </body>
1586        </html>
1587
1588
1589    7)ValidataionUtils class를 이용한 검증
1590      -ValidatationUtils class는 validate() method를 좀 더 편리하게 사용할 수 있게 해줌.
1591      -UserValidator.java 수정
1592
1593        /*String username = userVO.getName();
1594        if(username == null || username.trim().isEmpty()) {
1595          System.out.println("이름의 값이 빠졌습니다.");
1596          errors.rejectValue("name", "No Value");
1597        }*/
1598
1599        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "No Value");
1600
1601    8)@Valid와 @InitBinder 이용하기
1602      -Spring Framework이 대신 검증해 줌
1603      -mvnrepository에서 'hibernate validator'로 검색
1604
1605        <dependency>
1606            <groupId>org.hibernate.validator</groupId>
1607            <artifactId>hibernate-validator</artifactId>
1608            <version>6.0.5.Final</version>
1609        </dependency>
1610
1611      -pom.xml에 넣고 Maven Clean > Maven Install
1612      -HomeController.java 수정
1613
1614        @RequestMapping(value = "/register", method=RequestMethod.POST)
```

```
1615        public String register(@ModelAttribute("userVO") @Valid UserVO userVO, BindingResult
            result) {
1616          String page = "register_ok";
1617          //UserValidator validator = new UserValidator();
1618          //validator.validate(userVO, result);
1619          if(result.hasErrors()) {
1620            page = "register";
1621          }
1622
1623          return page;
1624        }
1625
1626        @InitBinder
1627        protected void initBinder(WebDataBinder binder) {
1628          binder.setValidator(new UserValidator());
1629        }
1630
1631
1632  18. Lab
1633    1)In J2EE Perspective
1634    2)Project Explorer > right-click > New > Dynamic Web Project
1635    3)Project name : SpringWebDemo > Next > Check [Generate web.xml deployment descriptor]
        > Finish
1636
1637    4)Convert to Maven Project
1638      -project right-click > Configure > Convert to Maven Project > Finish
1639      -Project : /SpringWebDemo
1640      -Group Id : SpringWebDemo
1641      -Artifact Id : SpringWebDemo
1642      -version : 0.0.1-SNAPSHOT
1643      -Packaging : war
1644      -Finish
1645
1646    5)Add Spring Project Nature
1647      -project right-click > Spring Tools > Add Spring Project Nature
1648
1649    6)새로 생선된 pom.xmlfile에 필요한 library 추가 > Maven Clean > Maven Install
1650      <dependencies>
1651        <dependency>
1652          <groupId>org.springframework</groupId>
1653          <artifactId>spring-context</artifactId>
1654          <version>4.3.24.RELEASE</version>
1655        </dependency>
1656        <dependency>
1657            <groupId>junit</groupId>
1658            <artifactId>junit</artifactId>
1659            <version>4.12</version>
1660            <scope>test</scope>
1661        </dependency>
1662        <dependency>
1663          <groupId>org.springframework</groupId>
1664          <artifactId>spring-jdbc</artifactId>
1665          <version>4.3.24.RELEASE</version>
1666        </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 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 file 생성
    -Spring Perspective로 전환
    -config > right-click > New > Spring Bean Configuration File > 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
    -Business logic용의 Spring 설정 file (ex:applicationContext.xml)을 작성했기 때문에 listener로
    ContextLoaderListener class를 정의해야 한다.
    -DispatcherServlet이 생성되어 Spring Container를 구동하면 Controller들이 memory에 loading된다.
    -하지만 Controller들이 생성되기 전에 누군가가 먼저 config폴더에 있는 beans.xml file을 읽어 Business
    Component들을 memory에 생성해야 한다.
    -이때 사용하는 class가 ContextLoaderListener class이다.
    -<listener>태그 하위에 <listener-class> 태그를 이용하여 Spring에서 제공하는 ContextLoaderListener를
    등록하면 된다.
    -ContextLoaderListener class는 Spring 설정 file(default에서 file명 applicationContext.xml)을 load
    하면 ServletContextListener interface를 구현하고 있기 때문에 ServletContext instance 생성 시
    (Tomcat으로 application이 load된 때)에 호출된다.
    -즉, ContextLoaderListener class는 DispatcherServlet class의 loading보다 먼저 동작하여 business
    logic층을 정의한 Spring 설정 file을 load한다.
    -중요한 것은 ContextLoaderListener class는 Servlet Container가 web.xml file을 읽어서 구동할 때, 자동
    으로 memory에 생성된다.
    -ContextLoaderListener는 Client의 요청이 없어도 Container가 구동될 때 Pre-Loading 되는 객체이다.
    -web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener -
    ContextLoaderListener] 를 선택하면 아래의 code가 자동 삽입
```

```
1711
1712        <!-- needed for ContextLoaderListener -->
1713        <context-param>
1714          <param-name>contextConfigLocation</param-name>
1715          <param-value>location</param-value>
1716        </context-param>
1717
1718        <!-- Bootstraps the root web application context before servlet initialization -->
1719        <listener>
1720          <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class
              >
1721        </listener>
1722
1723      -아래 code로 변환
1724        <context-param>
1725          <param-name>contextConfigLocation</param-name>
1726          <param-value>classpath:beans.xml</param-value>
1727        </context-param>
1728
1729    11)DispatcherServlet Class 추가
1730      -web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet -
          DispatcherServlet declaration] 선택하면 아래의 code가 자동 추가된다.
1731
1732        <!-- The front controller of this Spring Web application, responsible for handling all
          application requests -->
1733        <servlet>
1734          <servlet-name>springDispatcherServlet</servlet-name>
1735          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
1736          <init-param>
1737            <param-name>contextConfigLocation</param-name>
1738            <param-value>location</param-value>
1739          </init-param>
1740          <load-on-startup>1</load-on-startup>
1741        </servlet>
1742
1743        <!-- Map all requests to the DispatcherServlet for handling -->
1744        <servlet-mapping>
1745          <servlet-name>springDispatcherServlet</servlet-name>
1746          <url-pattern>url</url-pattern>
1747        </servlet-mapping>
1748
1749      -아래의 code로 변환
1750        <init-param>
1751          <param-name>contextConfigLocation</param-name>
1752          <param-value>classpath:beans*.xml</param-value>
1753        </init-param>
1754
1755        <servlet-mapping>
1756          <servlet-name>springDispatcherServlet</servlet-name>
1757          <url-pattern>*.do</url-pattern>
1758        </servlet-mapping>
1759
1760    12)mvnrepository에서 'jstl'로 검색 후 설치
1761      -목록에서 2번째 : 1.2버전
```

```
1762
1763        <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
1764        <dependency>
1765            <groupId>javax.servlet</groupId>
1766            <artifactId>jstl</artifactId>
1767            <version>1.2</version>
1768        </dependency>
1769
1770      -pom.xml에 붙여넣고 Maven Clean > Maven Install
1771
1772    13)Controller와 JSP 호출순서
1773      -Controller와 JSP 호출순서.png 그림 참조
1774
1775    14)Hello Controller 작성
1776      -Client의 요청을 처리할 POJO 형태의 HelloController class를 작성
1777      -Controller class에 @Controller annotation 선언
1778      -요청을 처리할 method를 작성하고 @RequestMapping annotation을 선언
1779        --@RequestMapping : HTTP 요청 URL을 처리할 Controller method 정의
1780      -JSP를 이용한 View 영역의 코드를 작성
1781      -Browser 상에서 JSP를 실행
1782
1783
1784      -src/com.example.vo package 생성
1785      -src/com.example.vo.HelloVO class 생성
1786
1787        package com.example.vo;
1788
1789        public class HelloVO {
1790          private String name;
1791
1792          public void setName(String name) {
1793            this.name = name;
1794          }
1795
1796          public String sayHello() {
1797            return "Hello " + name;
1798          }
1799        }
1800
1801
1802      -src/com.example.controller package 생성
1803      -com.example.controller.HelloController class 생성
1804
1805        package com.example.controller;
1806
1807        import org.springframework.beans.factory.annotation.Autowired;
1808        import org.springframework.stereotype.Controller;
1809        import org.springframework.ui.Model;
1810        import org.springframework.web.bind.annotation.RequestMapping;
1811
1812        import com.example.vo.HelloVO;
1813
1814        @Controller
1815        public class HelloController {
```

```
1816          @Autowired
1817          private HelloVO helloBean;
1818
1819          @RequestMapping("/hello.do")
1820          public String hello(Model model) {
1821            String msg = helloBean.sayHello();
1822            model.addAttribute("greet", msg);
1823            return "hello.jsp";
1824          }
1825        }
1826
1827    15)View에 data를 전달하는 Model class
1828      -Controller에서 Service를 호출한 결과를 받아서 View에게 전달하기 위해, 전달받은 결과를 Model 객체에 저장
1829      -Model addAttribute(String name, Object value)
1830        --value객체를 name의 이름으롤 저장하고, view code에서는 name으로 지정한 이름을 통해서 value를 사용
1831
1832    16)beans.xml 수정
1833      <context:component-scan base-package="com.example" />
1834
1835      <bean id="helloVO" class="com.example.vo.HelloVO">
1836        <property name="name" value="한지민" />
1837      </bean>
1838
1839    17)WebContent/hello.jsp 생성
1840
1841      <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
1842      <!DOCTYPE html>
1843      <html>
1844        <head>
1845          <meta charset="UTF-8">
1846          <title>Insert title here</title>
1847        </head>
1848        <body>
1849          ${greet}
1850        </body>
1851      </html>
1852
1853    18)project > right-click > Run As > Run on Server > Finish
1854
1855    19)http://localhost:8080/SpringWebDemo/hello.do
1856
1857      Hello 한지민
1858
1859
1860  19. Lab
1861    1)In J2EE Perspective
1862    2)Project Explorer > right-click > New > Dynamic Web Project
1863    3)Project name : SpringWebDemo1 > Next > Check [Generate web.xml deployment
        descriptor] > Finish
1864
1865    4)Convert to Maven Project
1866      -project right-click > Configure > Convert to Maven Project > Finish
1867
```

```
1868    5)Add Spring Project Nature
1869      -project right-click > Spring Tools > Add Spring Project Nature
1870
1871    6)새로 생성된 pom.xmlfile에 필요한 library 추가 > Maven Clean > Maven Install
1872      <dependencies>
1873        <dependency>
1874          <groupId>org.springframework</groupId>
1875          <artifactId>spring-context</artifactId>
1876          <version>4.3.24.RELEASE</version>
1877        </dependency>
1878        <dependency>
1879            <groupId>junit</groupId>
1880            <artifactId>junit</artifactId>
1881            <version>4.12</version>
1882            <scope>test</scope>
1883        </dependency>
1884        <dependency>
1885          <groupId>org.springframework</groupId>
1886          <artifactId>spring-jdbc</artifactId>
1887          <version>4.3.24.RELEASE</version>
1888        </dependency>
1889        <dependency>
1890          <groupId>javax.servlet</groupId>
1891          <artifactId>jstl</artifactId>
1892          <version>1.2</version>
1893        </dependency>
1894        <dependency>
1895          <groupId>com.oracle</groupId>
1896          <artifactId>ojdbc6</artifactId>
1897          <version>12.2</version>
1898        </dependency>
1899        <dependency>
1900            <groupId>org.springframework</groupId>
1901            <artifactId>spring-webmvc</artifactId>
1902            <version>4.3.24.RELEASE</version>
1903        </dependency>
1904      </dependencies>
1905
1906      -Maven Clean > Maven Install
1907
1908    7)Build path에 config foler 추가
1909      -project right-click > Build Path > Configure Build Path > Select [Source] tab
1910      -Click [Add Folder] > Select 현재 project > Click [Create New Folder...]
1911      -Folder name : config > Finish > OK > Apply and Close
1912      -Java Resources > config 폴더 확인
1913
1914    8)config folder에 beans.xml file 생성
1915      -Spring Perspective로 전환
1916      -config right-click > New > Spring Bean Configuration File
1917      -File name : beans.xml
1918      -생성시 beans,context, mvc 체크
1919        <?xml version="1.0" encoding="UTF-8"?>
1920        <beans xmlns="http://www.springframework.org/schema/beans"
1921          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
1922          xmlns:context="http://www.springframework.org/schema/context"
1923          xsi:schemaLocation="http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans.xsd
1924            http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context-3.2.xsd">
1925
1926
1927        </beans>
1928
1929    9)ContextLoaderListener class 설정
1930      -비즈니스 로직용의 스프링 설정 file (ex:applicationContext.xml)을 작성했기 때문에 listener로
          ContextLoaderListener class를 정의해야 한다.
1931      -ContextLoaderListener class는 스프링 설정 file(디폴트에서 file명 applicationContext.xml)을 로드하면
          ServletContextListener 인터페이스를 구현하고 있기 때문에 ServletContext 인스턴스 생성 시(톰캣으로 어플리
          케이션이 로드된 때)에 호출된다. 즉, ContextLoaderListener class는 DispatcherServlet class의 로드보다
          먼저 동작하여 비즈니스 로직층을 정의한 스프링 설정 file을 로드한다.
1932      -web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener -
          ContextLoaderListener] 를 선택하면 아래의 코드가 자동 삽입
1933
1934        <!-- needed for ContextLoaderListener -->
1935        <context-param>
1936          <param-name>contextConfigLocation</param-name>
1937          <param-value>location</param-value>
1938        </context-param>
1939
1940        <!-- Bootstraps the root web application context before servlet initialization -->
1941        <listener>
1942          <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class
              >
1943        </listener>
1944
1945      -아래 코드로 변환
1946        <context-param>
1947          <param-name>contextConfigLocation</param-name>
1948          <param-value>classpath:beans.xml</param-value>
1949        </context-param>
1950
1951    10)DispatcherServlet Class 추가
1952      -web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet -
          DispatcherServlet declaration] 선택하면 아래의 코드가 자동 추가된다.
1953
1954        <!-- The front controller of this Spring Web application, responsible for handling all
            application requests -->
1955        <servlet>
1956          <servlet-name>springDispatcherServlet</servlet-name>
1957          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
1958          <init-param>
1959            <param-name>contextConfigLocation</param-name>
1960            <param-value>location</param-value>
1961          </init-param>
1962          <load-on-startup>1</load-on-startup>
1963        </servlet>
1964
1965        <!-- Map all requests to the DispatcherServlet for handling -->
```

```
1966        <servlet-mapping>
1967          <servlet-name>springDispatcherServlet</servlet-name>
1968          <url-pattern>url</url-pattern>
1969        </servlet-mapping>
1970
1971      -아래의 코드로 변환
1972        <init-param>
1973          <param-name>contextConfigLocation</param-name>
1974          <param-value>classpath:beans*.xml</param-value>
1975        </init-param>
1976
1977        <servlet-mapping>
1978          <servlet-name>springDispatcherServlet</servlet-name>
1979          <url-pattern>*.do</url-pattern>
1980        </servlet-mapping>
1981
1982    11)UserVO class 생성
1983      -src/com.example.vo package 생성
1984      -src/com.example.vo.UserVO class 생성
1985
1986        package com.example.vo;
1987
1988        public class UserVO {
1989          private String userId;
1990          private String name;
1991          private String gender;
1992          private String city;
1993          public UserVO() {}
1994          public UserVO(String userId, String name, String gender, String city) {
1995            this.userId = userId;
1996            this.name = name;
1997            this.gender = gender;
1998            this.city = city;
1999          }
2000          public String getUserId() {
2001            return userId;
2002          }
2003          public void setUserId(String userId) {
2004            this.userId = userId;
2005          }
2006          public String getName() {
2007            return name;
2008          }
2009          public void setName(String name) {
2010            this.name = name;
2011          }
2012          public String getGender() {
2013            return gender;
2014          }
2015          public void setGender(String gender) {
2016            this.gender = gender;
2017          }
2018          public String getCity() {
2019            return city;
```

```
2020          }
2021          public void setCity(String city) {
2022            this.city = city;
2023          }
2024          @Override
2025          public String toString() {
2026            return "UserVO [userId=" + userId + ", name=" + name + ", gender=" + gender + ",
                city=" + city + "]";
2027          }
2028        }
2029
2030    12)UserDao 객체 생성
2031      -src/com.example.dao package 생성
2032      -src/com.example.dao.UserDao interface
2033
2034        package com.example.dao;
2035
2036        import java.util.List;
2037
2038        import com.example.vo.UserVO;
2039
2040        public interface UserDao {
2041          void insert(UserVO user);
2042
2043          List<UserVO> readAll();
2044
2045          void update(UserVO user);
2046
2047          void delete(String id);
2048
2049          UserVO read(String id);
2050        }
2051
2052      -src/com.example.dao.UserDaoImplJDBC.java 생성
2053
2054        package com.example.dao;
2055
2056        import java.sql.ResultSet;
2057        import java.sql.SQLException;
2058        import java.util.List;
2059
2060        import javax.sql.DataSource;
2061
2062        import org.springframework.beans.factory.annotation.Autowired;
2063        import org.springframework.dao.EmptyResultDataAccessException;
2064        import org.springframework.jdbc.core.JdbcTemplate;
2065        import org.springframework.jdbc.core.RowMapper;
2066        import org.springframework.stereotype.Repository;
2067
2068        import com.example.vo.UserVO;
2069
2070        @Repository("userDao")
2071        public class UserDaoImplJDBC implements UserDao {
2072          private JdbcTemplate jdbcTemplate;
```

```
2073
2074        @Autowired
2075        public void setDataSource(DataSource dataSource) {
2076          this.jdbcTemplate = new JdbcTemplate(dataSource);
2077        }
2078
2079        class UserMapper implements RowMapper<UserVO> {
2080          public UserVO mapRow(ResultSet rs, int rowNum) throws SQLException {
2081            UserVO user = new UserVO();
2082            user.setUserId(rs.getString("userid"));
2083            user.setName(rs.getString("name"));
2084            user.setGender(rs.getString("gender"));
2085            user.setCity(rs.getString("city"));
2086            return user;
2087          }
2088        }
2089
2090        @Override
2091        public void insert(UserVO user) {
2092          String SQL = "INSERT INTO users (userid, name, gender,city) VALUES (?, ?, ?, ?)";
2093          jdbcTemplate.update(SQL, user.getUserId(), user.getName(), user.getGender(),
             user.getCity());
2094
2095          System.out.println("등록된 Record UserId=" + user.getUserId() + " Name=" +
             user.getName());
2096        }
2097
2098        @Override
2099        public List<UserVO> readAll() {
2100          String SQL = "SELECT * FROM users";
2101          List<UserVO>  userList = jdbcTemplate.query(SQL, new UserMapper());
2102          return userList;
2103        }
2104
2105        @Override
2106        public void update(UserVO user) {
2107          String SQL = "UPDATE users  SET name = ?, gender = ?, city = ? WHERE userid = ?";
2108          jdbcTemplate.update(SQL, user.getName(), user.getGender(),
             user.getCity(),user.getUserId());
2109          System.out.println("갱신된 Record with ID = " + user.getUserId() );
2110        }
2111
2112        @Override
2113        public void delete(String id) {
2114          String SQL = "delete from users where userid = ?";
2115          jdbcTemplate.update(SQL, id);
2116          System.out.println("삭제된 Record with ID = " + id );
2117        }
2118
2119        @Override
2120        public UserVO read(String id) {
2121          String SQL = "SELECT * FROM users WHERE userid = ?";
2122          try {
2123            UserVO user = jdbcTemplate.queryForObject(SQL,
```

```
                    new Object[] { id }, new UserMapper());
              return user;
            }catch(EmptyResultDataAccessException e){
              return null;
            }
          }
        }

13)UserService 객체 생성
  -src/com.example.service package 생성
  -src/com.example.service.UserService interface

    package com.example.service;

    import java.util.List;

    import com.example.vo.UserVO;

    public interface UserService {
      void insertUser(UserVO user);

      List<UserVO> getUserList();

      void deleteUser(String id);

      UserVO getUser(String id);

      void updateUser(UserVO user);
    }

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

    package com.example.service;

    import java.util.List;

    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Service;

    import com.example.dao.UserDao;
    import com.example.vo.UserVO;

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

      @Autowired
      UserDao userDao;

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

      @Override
```

```
2178          public List<UserVO> getUserList() {
2179            return this.userDao.readAll();
2180          }
2181
2182          @Override
2183          public void deleteUser(String id) {
2184            this.userDao.delete(id);
2185          }
2186
2187          @Override
2188          public UserVO getUser(String id) {
2189            return this.userDao.read(id);
2190          }
2191
2192          @Override
2193          public void updateUser(UserVO user) {
2194            this.userDao.update(user);
2195          }
2196        }
2197
2198    14)UserController 객체 생성
2199      -src/com.example.controller package 생성
2200      -com.example.controller.UserController class 생성
2201
2202        package com.example.controller;
2203
2204        import org.springframework.beans.factory.annotation.Autowired;
2205        import org.springframework.stereotype.Controller;
2206        import org.springframework.ui.Model;
2207        import org.springframework.web.bind.annotation.RequestMapping;
2208        import org.springframework.web.bind.annotation.RequestParam;
2209
2210        import com.example.service.UserService;
2211        import com.example.vo.UserVO;
2212
2213        @Controller
2214        public class UserController {
2215          @Autowired
2216          private UserService userService;
2217
2218          @RequestMapping("/userInfo.do")
2219          public String getUserList(@RequestParam("userId") String userId, Model model) {
2220            UserVO user = userService.getUser(userId);
2221            //System.out.println(user);
2222            model.addAttribute("user", user);
2223            return "userInfo.jsp";
2224          }
2225        }
2226
2227    15)config/dbinfo.properties file 생성
2228
2229      db.driverClass=oracle.jdbc.driver.OracleDriver
2230      db.url=jdbc:oracle:thin:@localhost:1521:XE
2231      db.username=scott
```

```
2232      db.password=tiger
2233
2234    16)beans.xml 수정
2235
2236      <context:component-scan base-package="com.example" />
2237
2238      <context:property-placeholder location="classpath:dbinfo.properties" />
2239      <bean id="dataSource"
          class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
2240        <property name="driverClass" value="${db.driverClass}" />
2241        <property name="url" value="${db.url}" />
2242        <property name="username" value="${db.username}" />
2243        <property name="password" value="${db.password}" />
2244      </bean>
2245
2246    17)WebContent/index.jsp 생성
2247
2248      <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2249      <c:redirect url="userinfo.do" />
2250
2251    18)WebContent/userInfo.jsp 생성
2252
2253      <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
2254        <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2255        <c:set var="user" value="${user}"/>
2256        <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
          "http://www.w3.org/TR/html4/loose.dtd">
2257        <html>
2258        <head>
2259        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
2260        <title>Insert title here</title>
2261        </head>
2262        <body>
2263          <h1>userInfo.jsp</h1>
2264          <h2>사용자 정보</h2>
2265          아이디 : ${user.userId} <br />
2266          이름 : ${user.name} <br />
2267          성별 : ${user.gender} <br />
2268          도시 : ${user.city} <br />
2269        </body>
2270        </html>
2271
2272    19)project > right-click > Run As > Run on Server > Finish
2273
2274    20)http://localhost:8080/SpringWebDemo/userinfo.do?userId=jimin
2275
2276
2277  20. File Upload
2278    1)Server에 File을 저장할때 고려해야 할 사항들
2279      -File Upload 방식 결정하기.
2280        --Post 방식으로 전송할지 아니면 Ajax 방식으로 전송할지 결정해야 한다.
2281        --아마 요즘은 주로 Ajax방식을 사용하는것 같다.
2282      -File이름 중복문제.
```

--DB에 File을 저장할수도 있지만, 일반적으로 FileSystem에 File을 저장하게된다.
--따라서 upload 되는 file의 이름의 중복을 해결할 방법이 필요하다. -> UUID로 해결가능
-File 저장경로에 대한문제.
--Windows나 Linux등 운영체제에서 folder내의 file 개수가 너무 많아지게 되면, 속도저하 문제가 발생하게 된다.
--특히 Windows의 FileSystem의 경우 folder내 최대 file 개수의 제한이 있다.(100만단위가 넘어가긴 하지만 …)
--위 문제를 해결하기 위해서 보통 file이 upload 되는 시점별로 folder를 관리한다.
--예를 들어 2018년 9월 6일 file이 upload 되면, 그 file은 특정 folder의 경로의 /2018/09/06/ 경로에 저장하면 위 문제를 해결 할수있다.
--즉 upload 할때 file을 저장할 folder의 유무에 따라 folder 생성 logic이 필요하다.
-Image file의 경우 thumbnail 생성.
--Image file인 경우 저장된 file을 다시 화면에 보여줄때, 보통 그 image file의 thumbnail file을 보여주게된다.
--따라서 image file이 server에 저장될때는 추가적으로 그 image file의 thumbnail file을 생성해 주어야 한다.
--imgscalr-lib library가 image의 thumbnail 생성을 해준다.

21. Lab
1)In J2EE Perspective
2)Project Explorer > right-click > New > Dynamic Web Project
3)Project name : FileUploadDemo > 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.xmlfile에 필요한 library 추가 > Maven Clean > Maven Install

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.3.20.RELEASE</version>
  </dependency>
  <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-webmvc</artifactId>
     <version>4.3.20.RELEASE</version>
  </dependency>
</dependencies>
```

7)ContextLoaderListener class 설정
-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>
```

```
2331        <!-- Bootstraps the root web application context before servlet initialization -->
2332        <listener>
2333          <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class
              >
2334        </listener>
2335
2336      -아래 코드로 변환
2337        <context-param>
2338          <param-name>contextConfigLocation</param-name>
2339          <param-value>classpath:applicationContext.xml</param-value>
2340        </context-param>
2341
2342    8)DispatcherServlet Class 추가
2343      -web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet -
          DispatcherServlet declaration] 선택하면 아래의 코드가 자동 추가된다.
2344
2345        <!-- The front controller of this Spring Web application, responsible for handling all
          application requests -->
2346        <servlet>
2347          <servlet-name>springDispatcherServlet</servlet-name>
2348          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
2349          <init-param>
2350            <param-name>contextConfigLocation</param-name>
2351            <param-value>location</param-value>
2352          </init-param>
2353          <load-on-startup>1</load-on-startup>
2354        </servlet>
2355
2356        <!-- Map all requests to the DispatcherServlet for handling -->
2357        <servlet-mapping>
2358          <servlet-name>springDispatcherServlet</servlet-name>
2359          <url-pattern>url</url-pattern>
2360        </servlet-mapping>
2361
2362      -아래의 코드로 변환
2363        <init-param>
2364          <param-name>contextConfigLocation</param-name>
2365          <param-value>classpath:beans.xml</param-value>
2366        </init-param>
2367
2368        <servlet-mapping>
2369          <servlet-name>springDispatcherServlet</servlet-name>
2370          <url-pattern>/</url-pattern>
2371        </servlet-mapping>
2372
2373    9)FileUpload library 추가
2374      -Apache에서 제공하는 Common FileUpload library를 사용하여 file upload를 처리하기 위한 library
2375      -mvnrepository에서 'common fileupload'라고 검색하여 library 추가
2376        <dependency>
2377           <groupId>commons-fileupload</groupId>
2378           <artifactId>commons-fileupload</artifactId>
2379           <version>1.3.3</version>
2380        </dependency>
2381
```

-mvnrepository에서 'commons io'라고 검색하여 library 추가
<dependency>
<groupId>commons-io</groupId>
<artifactId>commons-io</artifactId>
<version>2.4</version>
</dependency>

-Maven Clean > Maven Install

10)thumbnail image library 추가
-mvnrepository에서 'imgscalr-lib'라고 검색하여 libary 추가
<dependency>
<groupId>org.imgscalr</groupId>
<artifactId>imgscalr-lib</artifactId>
<version>4.2</version>
</dependency>

-Maven Clean > Maven Install

11)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

12)config folder에 beans.xml file 생성
-Spring Perspective로 전환
-beans.xml

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<context:component-scan base-package="com.example" />
<mvc:annotation-driven />
</beans>

13)Spring mulipartResolver
-화면단에서 mutipart/form-date방식으로 server에 전송되는 data를 Spring MVC의 mulipartResolver로 처리할수 있다.
-File Upload library를 추가했다면, CommonMultipartResolver를 bean에 등록해야 한다.
-MultipartResolver는 Muiltpart 객체를 controller에 전달하는 역할을 한다.
-이 설정에서 아주 중요한 점은, CommonMultipartResolver class의 id나 dlfma값이다.
-이 class는 이름이 정해져 있다.
-정확하게는 DispatcherServlet이 특정 이름으로 등록된 CommonsMultipartResolver 객체만 인식하도록 되어 있다.
-따라서 반드시 CommonsMultipartResolver의 id값은 'multipartResolver'를 사용해야 한다.

<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
<!-- 최대 upload 가능한 byte크기 -->
<property name="maxUploadSize" value="10240000" />

```
2432          <!-- 디스크에 임시 file을 생성하기 전에 메모리에 보관할수있는 최대 바이트 크기 -->
2433          <!-- property name="maxInMemorySize" value="52428800" / -->
2434          <!-- defaultEncoding -->
2435          <property name="defaultEncoding" value="utf-8" />
2436        </bean>
2437
2438      -maxUploadSize에 대한 setter injection은 upload할 수 있는 file의 크기를 제한하기 위한 설정이다.
2439      -지정하지 않으면 기본으로 -1이 설정되는데, 이는 file의 크기가 무제한이라는 의미이다.
2440
2441    14)web.xml에 한글file encoding 처리하기
2442      -한글 file이 upload될 때 file 명이 깨지는 것을 해결하기 위해 web.xml에 아래 내용을 추가한다.
2443
2444        <filter>
2445          <filter-name>encodingFilter</filter-name>
2446          <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
2447          <init-param>
2448            <param-name>encoding</param-name>
2449            <param-value>UTF-8</param-value>
2450          </init-param>
2451          <init-param>
2452            <param-name>forceEncoding</param-name>
2453            <param-value>true</param-value>
2454          </init-param>
2455        </filter>
2456        <filter-mapping>
2457          <filter-name>encodingFilter</filter-name>
2458          <url-pattern>/*</url-pattern>
2459        </filter-mapping>
2460
2461    15)View 작성
2462      -WebContent/form.jsp
2463
2464        <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
2465        <!DOCTYPE html>
2466        <html>
2467        <head>
2468        <meta charset="UTF-8">
2469        <title>Insert title here</title>
2470        </head>
2471        <body>
2472          <h1>file 업로드 예제</h1>
2473          <form method="post" action="upload" enctype="multipart/form-data">
2474            <label>email:</label> <input type="text" name="email"> <br>
2475            <br> <label>file:</label> <input type="file" name="file1">
2476            <br>
2477            <br> <input type="submit" value="upload">
2478          </form>
2479        </body>
2480        </html>
2481
2482    16)Service 작성
2483      -src/com.example.service package
2484      -src/com.example.service.FileUploadService.java
```

```
2485
2486        package com.example.service;
2487
2488        import java.io.FileOutputStream;
2489        import java.io.IOException;
2490        import java.util.Calendar;
2491
2492        import org.springframework.stereotype.Service;
2493        import org.springframework.web.multipart.MultipartFile;
2494
2495        @Service
2496        public class FileUploadService {
2497          // 리눅스 기준으로 file 경로를 작성 ( 루트 경로인 /으로 시작한다. )
2498          // 윈도우라면 workspace의 드라이브를 파악하여 JVM이 알아서 처리해준다.
2499          // 따라서 workspace가 C드라이브에 있다면 C드라이브에 upload 폴더를 생성해 놓아야 한다.
2500          private static final String SAVE_PATH = "/upload";
2501          private static final String PREFIX_URL = "/upload/";
2502
2503          public String restore(MultipartFile multipartFile) {
2504            String uri = null;
2505
2506            try {
2507              // file 정보
2508              String originFilename = multipartFile.getOriginalFilename();
2509              String extName = originFilename.substring(originFilename.lastIndexOf("."),
                  originFilename.length());
2510              Long size = multipartFile.getSize();
2511
2512              // 서버에서 저장 할 file 이름
2513              String saveFileName = genSaveFileName(extName);
2514
2515              System.out.println("originFilename : " + originFilename);
2516              System.out.println("extensionName : " + extName);
2517              System.out.println("size : " + size);
2518              System.out.println("saveFileName : " + saveFileName);
2519
2520              writeFile(multipartFile, saveFileName);
2521              uri = PREFIX_URL + saveFileName;
2522            }
2523            catch (IOException e) {
2524              // 원래라면 RuntimeException 을 상속받은 예외가 처리되어야 하지만
2525              // 편의상 RuntimeException을 던진다.
2526              // throw new FileUploadException();
2527              throw new RuntimeException(e);
2528            }
2529            return uri;
2530          }
2531
2532
2533          // 현재 시간을 기준으로 file 이름 생성
2534          private String genSaveFileName(String extName) {
2535            String fileName = "";
2536
2537            Calendar calendar = Calendar.getInstance();
```

```
          fileName += calendar.get(Calendar.YEAR);
          fileName += calendar.get(Calendar.MONTH);
          fileName += calendar.get(Calendar.DATE);
          fileName += calendar.get(Calendar.HOUR);
          fileName += calendar.get(Calendar.MINUTE);
          fileName += calendar.get(Calendar.SECOND);
          fileName += calendar.get(Calendar.MILLISECOND);
          fileName += extName;

          return fileName;
        }


        // file을 실제로 write 하는 메서드
        private boolean writeFile(MultipartFile multipartFile, String saveFileName)
                    throws IOException{
          boolean result = false;

          byte[] data = multipartFile.getBytes();
          FileOutputStream fos = new FileOutputStream(SAVE_PATH + "/" + saveFileName);
          fos.write(data);
          fos.close();

          return result;
        }
      }
```

-SAVE_PATH는 file을 저장할 위치를 가리킨다.
  --일반적으로 server는 Linux 기반이므로 Linux 경로명을 사용하는 것이 좋다.
  --즉 file을 root 경로인 / 아래의 upload folder에 저장하겠다는 의미인데, Windows에서는 JVM이 알아서 workspace가 존재하는 drive의 위치를 찾아서 drive를 root 경로로 하여 upload folder에 저장한다.
  --예를들어 Eclipse workspace가 C drive에 있다면 C drive의 upload folder에 file이 저장될 것이다.
-PREFIX_URL은 저장된 file을 JSP에서 불러오기 위한 경로를 의미한다.
-MultipartFile 객체는 file의 정보를 담고 있다.
-uri을 반환하는 이유는 view page에서 바로 image file을 보기 위함이다.
  --만약 DB에서 image 경로를 저장 해야 한다면, 이와 같이 uri을 반환하면 좋을 것이다.
-현재 시간을 기준으로 file 이름을 바꾼다.
  --이렇게 하는 이유는, 여러 사용자가 올린 file의 이름이 같을 경우 덮어 씌어지는 문제가 발생하기 때문이다.
  --따라서 file 이름이 중복될 수 있는 문제를 해결하기 위해 ms단위의 시스템 시간을 이용하여 file 이름을 변경한다.
-FileOutputStream 객체를 이용하여 file을 저장한다.

17)Controller 작성
-com.example.controller package
-com.example.controller.FileUploadController.java

```
  package com.example.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.RequestMethod;
  import org.springframework.web.bind.annotation.RequestParam;
  import org.springframework.web.multipart.MultipartFile;
```

```
2591
2592        import com.example.service.FileUploadService;
2593
2594        @Controller
2595        public class FileUploadController {
2596          @Autowired
2597          FileUploadService fileUploadService;
2598
2599          @RequestMapping("/form")
2600          public String form() {
2601            return "form.jsp";
2602          }
2603
2604          @RequestMapping(value = "/upload", method = RequestMethod.POST)
2605          public String upload(@RequestParam("email") String email, @RequestParam("file1")
              MultipartFile file, Model model) {
2606            String uri = fileUploadService.restore(file);
2607            model.addAttribute("uri", uri);
2608            return "result.jsp";
2609          }
2610        }
2611
2612    18)WebContent/result.jsp
2613      <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
2614      <!DOCTYPE html>
2615      <html>
2616      <head>
2617      <meta charset="UTF-8">
2618      <title>Insert title here</title>
2619      </head>
2620      <body>
2621        <h1>Upload completed</h1>
2622        <div class="result-images">
2623          <img src="${pageContext.request.contextPath }${uri }" style="width: 150px">
2624        </div>
2625        <p>
2626          <a href='/FileUploadDemo/form'> 다시 업로드 하기 </a>
2627        </p>
2628      </body>
2629      </html>
2630
2631    19)Project > right-click > Run As > Run on Server
2632      http://localhost:8080/FileUploadDemo/form
2633
2634    20)문제점 및 해결
2635      -upload folder를 보면 file이 upload된 것을 확인할 수 있지만, 결과 화면을 보면 image가 제대로 출력 되지 않을
          것이다.
2636      -엑박이 뜬 image file을 right-click하여 경로를 보면 아마 다음과 같을 것이다.
2637      -http://localhost:8080/FileUploadDemo/upload/201822632335607.PNG
2638      -File을 저장할 때 upload라는 folder에 저장을 했는데, file을 저장할 때의 upload는 C drive 내의 upload
          folder이고,
2639      -위 URL에서 upload는 application 상 경로에 있는 upload이므로 WEB-INF 폴더의 하위 folder로서의 upload
          를 의미한다.
```

-즉 실제 file이 저장된 server 상의 위치(물리 주소)와 application에서 보여주고자 하는 file 경로(가상 주소)가 일치하지 않은 것이다.
-따라서 실제 file이 저장되어 있는 위치와 application 상의 위치를 일치시키는 작업이 필요하다.
-beans.xml에 물리 주소와 가상 주소를 mapping 해주는 code를 추가하도록 해야한다.

```
<!-- resource mapping -->
<!-- location : 물리적 주소 / mapping : 가상 주소 -->
<mvc:resources location="file:///C:/upload/" mapping="/upload/*"/>
```

-이제 정상적으로 result.jsp에서 image가 출력될 것이다.

21)Multiple file upload
-이번에는 여러 개의 file을 upload 할 수 있는 multiple upload를 알아보자.
-수정할 부분은 <input> tag와 Controller에서 MultipartFile 객체를 받는 parameter 부분 두 곳인데, 필요한 부분만 보자.
-form.jsp

```
<input type="file" name="files" multiple>
--<input> 태그에서는 multiple 속성만 추가하면 된다.
--"File선택"을 클릭하면 ctrl 키를 눌러서 여러 개의 file을 선택할 수 있다.
```

-FileUploadController

```
@RequestMapping( "/upload" )
public String upload(@RequestParam String email,
   @RequestParam(required=false) List<MultipartFile> files, Model model) {

     ...

}
```

-Controller에서는 여러 개의 file을 받기 때문에 MultipartFile을 List로 받아야 한다.

22. Apache Log4j

1)Log4j : A logging library for Java.
-log문의 출력을 다양한 대상으로 할 수 있도록 도와주는 도구, Opensource
http://logging.apache.org/log4j/1.2/
http://logging.apache.org/log4j/2.x/index.html

2)History of log4j
-Started in early 1996 as tracing API for the E.U. SEMPER (Secure Electronic Marketplace for Europe) project.
-After countless enhancements and several incarnations, the initial API has evolved to become log4j, a popular logging package for Java.
-The package is distributed under the Apache Software License, a full-fledged open source license certified by the open source initiative.
-The latest log4j version, including its full-source code, class files, and documentation can be found at http://logging.apache.org/log4j/

3)Loggers, Appenders and Layouts
-Log4j has three main components: loggers, appenders and layouts.
-These three types of components work together to enable developers to log messages according to message type and level, and to control at runtime how these messages are formatted and where they are reported.
-log4j has three main components:

--loggers: Responsible for capturing logging information.
--appenders: Responsible for publishing logging information to various preferred destinations.
--layouts: Responsible for formatting logging information in different styles.

4)log4j Features
- It is optimized for speed.
- It is based on a named logger hierarchy.
- It is fail-stop. However, although it certainly strives to ensure delivery, log4j does not guarantee that each log statement will be delivered to its destination.
- It is thread-safe.
- log4j는 융통성이 풍부
- 설정 file은 property file과 XML 형식으로 실행 중 수정 적용 가능
--Logging behavior can be set at runtime using a configuration file.
- It is designed to handle Java Exceptions from the start.
- It supports multiple output appenders per logger.
--log4j는 출력을 file, console, java.io.OutputStream, java.io.Writer, TCP를 사용하는 원격 server, 원격 Unix Syslog daemon, - 원격 JMS 구독자, Windows NT EventLog로 보낼 수 있고, 심지어는 e-mail로 보낼 수도 있음
-It uses multiple levels, namely ALL, TRACE, DEBUG, INFO, WARN, ERROR and FATAL.
--log4j는 다음 6단계의 장애 level을 사용. < TRACE(추가), DEBUG, INFO, WARN, ERROR, FATAL >
- The format of the log output can be easily changed by extending the Layout class.
- The target of the log output as well as the writing strategy can be altered by implementations of the Appender interface.
- log4j는 logger 하나에 다수의, 출력을 담당하는 appender를 할당할 수 있음
- It supports internationalization.

5)Log4j 구조
-Logger(Category)
--logging message를 Appender에 전달
--log4J의 심장부에 위치
--개발자가 직접 log출력 여부를 runtime에 조정
--logger는 loglevel을 가지고 있으며, log의 출력 여부는 log문의 level과 logger의 level을 가지고 결정
-Appender
--Log의 출력위치를 결정(File, Console, DB 등)
--log4J API문서의 XXXAppender로 끝나는 class들의 이름을 보면, 출력위치를 어느정도 짐작가능
-Layout
--Appender가 어디에 출력할 것인지 결정했다면 어떤 형식으로 출력할 것이지 출력 layout을 결정

6)Log4j level
-These are defined in the org.apache.log4j.Level class.
-FATAL
--아주 심각한 error가 발생한 상태.
--System적으로 심각한 문제가 발생해서 application작동이 불가능할 경우 해당하는데, 일반적으로는 application에서는 사용할 일이 없음
-ERROR
--요청을 처리하는 중 문제가 발생한 상태를 나타냄
-WARN
--처리 가능한 문제이지만, 향후 System error의 원인이 될 수 있는 경고성 message를 나타냄
-INFO
--Login, 상태변경과 같은 정보성 message를 나타냄
-DEBUG
--개발시 debug 용도로 사용한 message를 나타냄
-TRACE

--log4j1.2.12에서 신규 추가된 level로서, DEBUG level이 너무 광범위한 것을 해결하기 위해서 좀 더 상세한 상태를 나타냄
-FATAL > ERROR > WARN > INFO > DEBUG > TRACE
--DEBUG level로 했다면 INFO~FATAL까지 모두 logging이 되어진다.
-Here is an example of this rule.

```
// get a logger instance named "com.foo"
Logger  logger = Logger.getLogger("com.foo");

// Now set its level. Normally you do not need to set the
// level of a logger programmatically. This is usually done
// in configuration files.
logger.setLevel(Level.INFO);

Logger barlogger = Logger.getLogger("com.foo.Bar");

// This request is enabled, because WARN >= INFO.
logger.warn("Low fuel level.");

// This request is disabled, because DEBUG < INFO.
logger.debug("Starting search for nearest gas station.");

// The logger instance barlogger, named "com.foo.Bar",
// will inherit its level from the logger named
// "com.foo" Thus, the following request is enabled
// because INFO >= INFO.
barlogger.info("Located nearest gas station.");

// This request is disabled, because DEBUG < INFO.
barlogger.debug("Exiting gas station search");
```

7)Log4j Pattern Option
-일반적으로 PatternLayout(org.apache.log4j.PatternLayout, http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html)을 사용하는 것이 debugging에 가장 적합함
-***로 표시된 항목은 실행속도에 영향이 있음
-%p: debug, info, warn, error, fatal 등의 priority 출력
-%m : log내용 출력
-%d *** : logging event가 발생한 시간을 출력, ex)포맷은 %d{HH:mm:ss} 같은 형태의 SimpleDateFormat
-%t : log event가 발생된 thread의 이름 출력
-%F ***: logging이 발생한 program file명 출력
-%l ***: logging이 발생한 caller의 정보 출력
-%L ***: logging이 발생한 caller의 line수 출력
-%M ***: logging이 발생한 method 이름 출력
-% : % 표시 출력
-%n : platform 종속적인 개행문자 출력
-%c : category 출력, ex)Category가 a.b.c 처럼 되어있다면 %c{2}는 b.c 출력
-%C ***: class명 출력, ex)Class구조가 org.apache.xyz.SomeClass 처럼 되어있다면 %C{2}는 xyz.SomeClass 출력
-%r : Application 시작 이후 부터 logging이 발생한 시점의 시간(milliseconds) 출력
-%x : logging이 발생한 thread와 관련된 NDC(nested diagnostic context) 출력
-%X : logging이 발생한 thread와 관련된 MDC(mapped diagnostic context) 출력

-For example, the PatternLayout with the conversion pattern

"%r [%t] %-5p %c - %m%n" will output something akin to:

176 [main] INFO org.foo.Bar - Located nearest gas station.

-The first field is the number of milliseconds elapsed since the start of the program.
-The second field is the thread making the log request.
-The third field is the level of the log statement.
-The fourth field is the name of the logger associated with the log request.
-The text after the '-' is the message of the statement.

8)Log4j 주요 class
-org.apache.log4j.ConsoleAppender
--Console에 log message 출력
-org.apache.log4j.FileAppender
--File에 log message 기록
-org.apache.log4j.rolling.RollingFileAppender
--File 크기가 일정 수준 이상이 되면 기존 file을 backup file로 바꾸고 처음부터 기록
-org.apache.log4j.DailyRollingFileAppender
--일정 기간 단위로 log file을 생성하고 기록
-org.apache.log4j.jdbc.JDBCAppender
--DB에 log를 출력. 하위에 Driver, URL, User, Password, Sql과 같은 parameter를 정의할 수 있음
-SMTPAppender
--Log message를 Email로 전송
-NTEventAppender
--Windows System Event Log로 message 전송

9)Configuration
-Library 추가

```
<dependency>
   <groupId>log4j</groupId>
   <artifactId>log4j</artifactId>
   <version>1.2.17</version>
</dependency>
```

-또는
--http://logging.apache.org/log4j/1.2/download.html 로 이동 후 library download.
--Download받은 library를 project의 WebContent > WEB-INF > lib 에 복사

-설정 file 만들기
--설정 file은 xml 또는 properties 로 설정 가능함(xml 권장)
--File 위치는 설정 file들이 있는 곳에 자유롭게 배치해도됨
--설정 file 내용은 간단하게 console에 log를 출력하는 것만 작성함

-Example

```
import com.foo.Bar;

// Import log4j classes.
import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;

public class MyApp {
```

```
2837          // Define a static logger variable so that it references the
2838          // Logger instance named "MyApp".
2839          static Logger logger = Logger.getLogger(MyApp.class);
2840
2841          public static void main(String[] args) {
2842
2843            // Set up a simple configuration that logs on the console.
2844            BasicConfigurator.configure();
2845
2846            logger.info("Entering application.");
2847            Bar bar = new Bar();
2848            bar.doIt();
2849            logger.info("Exiting application.");
2850          }
2851        }
2852
2853        --MyApp begins by importing log4j related classes.
2854        --It then defines a static logger variable with the name MyApp which happens to be the
          fully qualified name of the class.
2855        --MyApp uses the Bar class defined in the package com.foo.
2856
2857          package com.foo;
2858          import org.apache.log4j.Logger;
2859
2860          public class Bar {
2861            static Logger logger = Logger.getLogger(Bar.class);
2862
2863            public void doIt() {
2864              logger.debug("Did it again!");
2865            }
2866          }
2867
2868        --The invocation of the BasicConfigurator.configure method creates a rather simple log4j
          setup.
2869        --This method is hardwired to add to the root logger a ConsoleAppender.
2870        --The output will be formatted using a PatternLayout set to the pattern
2871          "%-4r [%t] %-5p %c %x - %m%n".
2872
2873        --Note that by default, the root logger is assigned to Level.DEBUG.
2874        --The output of MyApp is:
2875
2876          0    [main] INFO  MyApp  - Entering application.
2877          36   [main] DEBUG com.foo.Bar  - Did it again!
2878          51   [main] INFO  MyApp  - Exiting application.
2879
2880    10)Default Initialization under Tomcat
2881      -Define the following servlet in the web.xml file for your web-application.
2882
2883        <servlet>
2884          <servlet-name>log4j-init</servlet-name>
2885          <servlet-class>com.foo.Log4jInit</servlet-class>
2886          <init-param>
2887            <param-name>log4j-init-file</param-name>
2888            <param-value>WEB-INF/classes/log4j.lcf</param-value>
```

```
2889          </init-param>
2890          <load-on-startup>1</load-on-startup>
2891        </servlet>
2892
2893      -It is also possible to use a special servlet for log4j initialization. Here is an example,
2894
2895        package com.foo;
2896
2897        import org.apache.log4j.PropertyConfigurator;
2898        import javax.servlet.http.HttpServlet;
2899        import javax.servlet.http.HttpServletRequest;
2900        import javax.servlet.http.HttpServletResponse;
2901        import java.io.PrintWriter;
2902        import java.io.IOException;
2903
2904        public class Log4jInit extends HttpServlet {
2905
2906          public void init() {
2907            String prefix =  getServletContext().getRealPath("/");
2908            String file = getInitParameter("log4j-init-file");
2909            // if the log4j-init-file is not set, then no point in trying
2910            if(file != null) {
2911              PropertyConfigurator.configure(prefix+file);
2912            }
2913          }
2914
2915          public void doGet(HttpServletRequest req, HttpServletResponse res) {
2916
2917          }
2918        }
2919
2920    11)Semi-Lab
2921      -Spring Legacy Project > Spring MVC >
2922      -Project name : Log4jDemo > Next
2923      -Enter a topLevelPackage : com.example.biz > Finish
2924      -pom.xml에서 slf4j에 관한 library 제거 : slf4j-api, jcl-over-slf4j, slf4j-log4j12
2925      -log4j version을 1.2.17
2926
2927        <properties>
2928          <java-version>1.8</java-version>
2929          <org.springframework-version>4.3.24.RELEASE</org.springframework-version>
2930          <org.aspectj-version>1.9.4</org.aspectj-version>
2931        </properties>
2932          ...
2933        <!-- Spring -->
2934        <dependency>
2935          <groupId>org.springframework</groupId>
2936          <artifactId>spring-context</artifactId>
2937          <version>${org.springframework-version}</version>
2938
2939        -아래 <exclusions> 제거할 것
2940          <exclusions>
2941            <!-- Exclude Commons Logging in favor of SLF4j -->
2942            <exclusion>
```

```
2943              <groupId>commons-logging</groupId>
2944              <artifactId>commons-logging</artifactId>
2945            </exclusion>
2946          </exclusions>
2947        </dependency>
2948
2949        <!-- Logging -->
2950        <dependency>
2951          <groupId>log4j</groupId>
2952          <artifactId>log4j</artifactId>
2953          <version>1.2.17</version>
2954          <scope>runtime</scope>
2955        </dependency>
2956
2957      -HomeController.java에서
2958        --다음 code를 제거
2959          private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
2960
2961        --대신 다음 code 추가
2962          protected Log log = LogFactory.getLog(HomeController.class);
2963
2964          @RequestMapping(value = "/", method = RequestMethod.GET)
2965          public String home(Locale locale, Model model) {
2966
2967            log.info("Logging 시작하기");   <-- 추가
2968
2969      -기본 프로젝트에 있는 log4j.xml이다.
2970      -서버를 실행 시키면 콘솔창에 빨간 글씨들 사이에 하얀 글씨들이 보일 것이다.
2971
2972          INFO : com.example.biz.HomeController - Logging 시작하기
2973
2974      -그게 바로 여기서 설정한 로그 메시지들이다.
2975      -Root Logger의 레벨이 warn 이고, Appender Logger들의 레벨은 info다.
2976      -전부 warn레벨을 상속받지만, 현재 info 레벨이 적용된다.
2977      -그리고 ConsoleAppender에 로그 메시지를 전송한다.
2978      -현재는 이렇게 찍히지만, 이제 우리 입맛에 맞게, 로그의 종류에 따라 다르게 출력되게 logj4.xml을 변경할 것이다.
2979
2980      -log4j.xml 수정
2981
2982        <?xml version="1.0" encoding="UTF-8"?>
2983        <!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd">
2984        <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
2985
2986          <!-- Appenders -->
2987          <appender name="console" class="org.apache.log4j.ConsoleAppender">
2988            <param name="Target" value="System.out" />
2989            <layout class="org.apache.log4j.PatternLayout">
2990              <param name="ConversionPattern" value="%d %5p [%c] %m%n" />
2991            </layout>
2992          </appender>
2993
2994          <!-- *추가* -->
2995          <appender name="console-infolog" class="org.apache.log4j.ConsoleAppender">
2996            <layout class="org.apache.log4j.PatternLayout">
```

```
2997                <param name="ConversionPattern" value="%d %5p %m%n"/>
2998             </layout>
2999          </appender>
3000          <!--  -->
3001
3002          <!-- Application Loggers -->
3003          <logger name="com" additivity="false">
3004             <level value="debug" />
3005             <appender-ref ref="console"/>
3006          </logger>
3007
3008          <!-- 3rdparty Loggers -->
3009          <logger name="org.springframework.core">
3010             <level value="info" />
3011          </logger>
3012
3013          <logger name="org.springframework.beans">
3014             <level value="info" />
3015          </logger>
3016
3017          <logger name="org.springframework.context">
3018             <level value="info" />
3019          </logger>
3020
3021          <logger name="org.springframework.web">
3022             <level value="info" />
3023          </logger>
3024
3025          <!-- Root Logger -->
3026          <root>
3027             <priority value="off" />
3028             <appender-ref ref="console" />
3029          </root>
3030
3031       </log4j:configuration>
3032
3033    -2가지의 Appender ( console, console-infolog )와 기존의 4가지 Logger와 추가된 Logger가 있다.
3034    -기존의 4가지 Logger(28~42행, <!-- 3rdparty Loggers --> ~ <!-- Root Logger --> 전까지)는 삭제해
       줘도 된다.
3035    -Logger마다 메세지를 전달하는 Appender가 다르니 잘 확인하자.
3036    -그리고 상속을 하지 않는(Additivity=false) 이유는 로그가 2번 출력되서 보통 상속하지 않는다.
3037    -서버를 돌리면 기존 4가지 Logger를 지우지 않았다면 그대로 출력되고, 지웠다면 빨간 글씨만 출력될 것이다. log4j의
       설정 끝!
3038
3039
3040 23. Logging
3041   1)Log
3042    -Log는 기록을 남기는 것을 의미한다.
3043    -구체적으로는 program 개발이나 운영시 발생하는 문제점을 추적 하거나 운영 상태를 monitoring 하는 정보를 기록
       하는 것.
3044    -또한 분석을 통해 통계를 낼 수도 있기 때문에 기록을 남기는 것은 중요하다고 할 수 있다.
3045    -하지만 log를 남기면 성능이 나빠진다는 단점이 있는데, 그보다 log를 통해 얻는 정보가 훨씬 많기 때문에 필요한 부분
       에 file로써 log를 남기는 것이 중요하다.
3046
```

2)Logback
-Spring에서는 기본적으로 commons.logging libarary (Apache의 JCL, Jakarta Commons Logging) 을 사용한다.
-즉 Spring 개발을 할 때 Spring이 뿜어내는 message는 JCL에 의존하여 log를 남기는 것이다.
-실제로 spring-context library를 설치할 때 project folder의 Maven Libarary를 확인해보면 commons.logging library(commons-logging-1.2.jar)를 확인할 수 있다.
-예전에는 Spring에서 log를 남길 때 Log4J를 사용했었는데, 성능 및 기능상의 이유로 대체 logger들이 많아졌고, 현재 대부분은 SLF4J interface를 구현한 Logback을 사용한다.
-Spring이 기존에 사용하던 log libtrary JCL 대신, 새로운 library Logback을 사용하도록 하기 위해서는 SLF4J가 필요하다.
-즉 SLF4J는 JCL과 Log4J의 징검다리 역할을 한다고 보면 된다.
Log4J <-- jcl-over-slf4j --> SLF4J <-- logback-classic --> Logback

3)환경 설정
-Spring Legacy Project > LogDemo > Spring MVC Project > com.example.biz >
-pom.xml

```
<properties>
<java-version>1.8</java-version>
<org.springframework-version>4.3.24.RELEASE</org.springframework-version>
<org.aspectj-version>1.9.4</org.aspectj-version>
<org.slf4j-version>1.7.26</org.slf4j-version>
</properties>
<dependencies>
<!-- Spring core -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>${org.springframework-version}</version>

<!-- JCL 제외 -->
<exclusions>
<exclusion>
<groupId>commons-logging</groupId>
<artifactId>commons-logging</artifactId>
</exclusion>
</exclusions>
</dependency>

<!-- Logging -->
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-api</artifactId>
<version>${org.slf4j-version}</version>
</dependency>
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>jcl-over-slf4j</artifactId>
<version>${org.slf4j-version}</version>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
```

```
3097            <version>${org.slf4j-version}</version>
3098            <scope>runtime</scope>
3099          </dependency>
3100          ...
3101          <dependency>
3102            <groupId>javax.servlet</groupId>
3103            <artifactId>javax.servlet-api</artifactId>
3104            <version>4.0.1</version>
3105            <scope>provided</scope>
3106          </dependency>
3107          <dependency>
3108            <groupId>javax.servlet.jsp</groupId>
3109            <artifactId>javax.servlet.jsp-api</artifactId>
3110            <version>2.3.3</version>
3111            <scope>provided</scope>
3112          </dependency>
3113          <dependency>
3114            <groupId>junit</groupId>
3115            <artifactId>junit</artifactId>
3116            <version>4.12</version>
3117            <scope>test</scope>
3118          </dependency>
3119
3120        -앞서 언급했듯이 spring-context에서는 기본적으로 commons-logging library를 사용하고 있으므로
           Logback library로 대체하기 위해서는 spring-context library를 추가할 때 commons-logging library를
           제외 시켜야 한다.
3121        -JCL을 제외시켰기 때문에 기존에 JCL을 통해 log를 남기던 code들은 error를 발생 시킬 것이다.
3122        -그래서 필요한 것이 jcl-over-slf4j library이며, 일종의 다리 역할을 하는 것이다.
3123        -실제로는 SLF4J을 구현한 logback-classic library가 log를 남기게 된다.
3124
3125    4)/src/main/resources/logback.xml
3126      -Log를 어떻게 남길지에 대한 설정 file이다.
3127
3128        <?xml version="1.0" encoding="UTF-8"?>
3129        <configuration>
3130          <!-- Console로 log를 남김 -->
3131          <appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">
3132            <encoder>
3133              <charset>UTF-8</charset>
3134              <!-- Log message pattern -->
3135              <Pattern>
3136                %d{HH:mm:ss.SSS} [%thread] %-5level %logger{5} - %msg%n
3137              </Pattern>
3138            </encoder>
3139          </appender>
3140
3141          <!-- file로 log를 남김 -->
3142          <appender name="fileAppender"
           class="ch.qos.logback.core.rolling.RollingFileAppender">
3143            <file>c:\LogExample\logexample2.log</file>
3144            <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
3145              <Pattern>
3146                   %d{HH:mm:ss.SSS} [%thread] %-5level %logger{5} - %msg%n
3147              </Pattern>
```

```
            </encoder>

            <!-- log를 남기는 file의 용량이 50KB가 넘으면 이를 압축 file로 만들고 새로 log file로 만들라는 정책
            -->
            <triggeringPolicy
              class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
              <MaxFileSize>10KB</MaxFileSize>
            </triggeringPolicy>

            <!-- file을 덮어쓰는 정책 -->
            <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
              <FileNamePattern>C:\LogExample\logexample2.%i.log.zip</FileNamePattern>
              <!--
                 MinIndex가 1이고, MaxIndex가 10이므로, 위의 file 이름 pattern에 따라 아래의 로그 file이 생길
                 것이다.
                 logexample2.1.log.zip   logexample2.2.log.zip  .... logexample2.10.log.zip
                 이 상태에서 또 10KB가 넘으면 logexample2.1.log.zip이 된다.
              -->
              <MinIndex>1</MinIndex>
              <MaxIndex>10</MaxIndex>
            </rollingPolicy>
          </appender>
            <!--
              com.victolee.logExample 아래 package log들만  consoleAppender, fileAppender 방법으로
              log를 남긴다.
              물론 <appender-ref ref="consoleAppender" />를 추가하여 console로도 log를 남길 수 있다.

            -->
            <logger name="com.victolee.logExample"  level="info" additivity="false">
              <appender-ref ref="fileAppender" />
            </logger>

            <!-- root는 global logger를 의미하며, 위의 logger에 해당하지 않으면 root logger가 실행된다. -->
            <root level="warn">
              <appender-ref ref="consoleAppender" />
            </root>
        </configuration>

    -Console로 log를 남기는 방법이 있고, file로 log를 남기는 방법이 있다. ( 두 가지 방법을 함께 사용할 수도 있다. )
    -File로 log를 남길 경우, 어느 경로에 file을 남길 것인지, file의 용량을 어느 정도 크기로 제한을 할 것인지, file이 제
    한한 용량을 초과했을 시 어떻게 할 것인지에 대한 정책들을 작성한다.
    -그리고 어느 package 또는 class에 대해서 log를 남길 것인지, 어느 정도 수준에 대해서 log를 남길 것인지도 설정할
    수 있다.

  5)Controller 작성
    -src/com.example.controller package
    -com.example.controller.LogController.java

      package com.example.controller;

      import org.apache.commons.logging.Log;
      import org.apache.commons.logging.LogFactory;
      import org.springframework.stereotype.Controller;
      import org.springframework.web.bind.annotation.RequestMapping;
```

```
3196    import org.springframework.web.bind.annotation.ResponseBody;
3197
3198    @Controller
3199    public class LogController {
3200      private static final Log LOG = LogFactory.getLog(LogController.class);
3201
3202      @RequestMapping("/log")
3203      @ResponseBody
3204      public String logExam() {
3205        LOG.debug("#ex1 - debug log");
3206        LOG.info("#ex1 - info log");
3207        LOG.warn("#ex1 - warn log");
3208        LOG.error("#ex1 - error log");
3209        return "콘솔 또는 file경로 확인";
3210      }
3211    }
3212
3213  -com.example.controller package를 생성한 이유는 위의 설정 file의 <logger>에서 범위를 지정했기 때문이
      다.
3214  -그리고 file로 log를 남기겠다고 했으므로 요청을 했을 시 C:\LogExample\logexample2.log 경로에 file이 작성
      되는지 확인해본다.
```