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";
  }
```

-src/main/webapp/WEB-INF/views/confirm3.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="${userInfo}"/>
<!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>
  <h1>confirm3.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/jimin/1234/한지민/25/여성

confirm3.jsp

사용자 정보

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

15. Lab : @RequestMapping Parameter

1)GET 방식과 POST 방식

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

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

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

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

```
<!DOCTYPE html>
```

```
718         <html>
719         <head>
720         <meta charset="UTF-8">
721         <title>로그인 폼</title>
722         </head>
723         <body>
724           <form method="GET" action="/biz/login">
725             아이디 : <input type="text" name="userid" /><br />
726             패스워드 : <input type="password" name="passwd" /><br />
727             <input type="submit" value="로그인하기" />
728           </form>
729         </body>
730         </html>
731
732       -http://localhost:8080/biz/resources/login.html에서 submit 하면 405 error 발생
733       -왜냐하면 서로의 method가 불일치하기 때문
734       -해결방법
735       -src/main/java/com.example.biz/HomeController.java 수정
736       -즉 login method(요청 처리 method)의 이름은 같지만 parameter의 type과 return type이 틀리기 때문에
          Method Overloading 됨.
737
738         @RequestMapping(value="/login", method=RequestMethod.POST)
739         public String login(@RequestParam("userid") String userid,
740                             @RequestParam("passwd") String passwd,
741                             Model model) {
742
743           model.addAttribute("userid", userid);
744           model.addAttribute("passwd", passwd);
745           return "login";
746         }
747         @RequestMapping(value="/login", method=RequestMethod.GET)
748         public ModelAndView login(@RequestParam("userid") String userid,
749                             @RequestParam("passwd") String passwd) {
750
751           ModelAndView mav = new ModelAndView();
752           mav.addObject("userid", userid);
753           mav.addObject("passwd", passwd);
754           mav.setViewName("login");
755           return mav;
756         }
757
758       -src/main/webapp/WEB-INF/views/login.jsp
759
760         <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
761         <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
          "http://www.w3.org/TR/html4/loose.dtd">
762         <html>
763         <head>
764         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
765         <title>Insert title here</title>
766         </head>
767         <body>
768           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       -Select Spring MVC Project
919       -Project name : MVCDemo1 > Next
920       -Enter a topLevelPackage : com.example.biz > Finish
921       -Create Table in MariaDB
922         CREATE TABLE Member
923         (
924           userid        VARCHAR(20),
925           username        VARCHAR(20)   NOT NULL,
```

```
926         userage       TINYINT   NOT NULL,
927         gender      VARCHAR(10)   NOT NULL,
928         city          VARCHAR(50),
929         CONSTRAINT member_userid_pk  PRIMARY KEY(userid)
930       );
931
932   2)src/main/webapp/static folder 생성
933     -src/main/webapp/static/css folder
934     -src/main/webapp/static/images folder
935     -src/main/webapp/static/js folder
936       --jquery-1.12.4.js
937     -src/main/webapp/static/register.html
938       <!DOCTYPE html>
939       <html lang="en">
940       <head>
941         <meta charset="UTF-8">
942         <title>회원 가입</title>
943       </head>
944       <body>
945         <h1>회원 가입 창</h1>
946         <form action="/biz/create" method="post">
947           <ul>
948             <li>ID : <input type="text" name="userid" /></li>
949             <li>이름 : <input type="text" name="username" /></li>
950             <li>나이 : <input type="number" name="userage" /></li>
951             <li>성별 : <input type="radio" name="gender" value="남성"/>남성
952                        <input type="radio" name="gender" value="여성"/>여성</li>
953             <li>거주지 : <input type="text" name="city" /></li>
954             <li><input type="submit" value="가입하기" /></li>
955           </ul>
956         </form>
957       </body>
958       </html>
959
960   3)src/main/webapp/WEB-INF/spring/appServlet/sevlet-context.xml 수정
961     <resources mapping="/static/**" location="/static/" /> 추가
962
963     <context:component-scan base-package="com.example" /> 수정
964
965   4)src/main/resources/mariadb.properties
966     db.driverClass=org.mariadb.jdbc.Driver
967     db.url=jdbc:mariadb://localhost:3306/test
968     db.username=root
969     db.password=javamariadb
970
971   5)src/main/webapp/WEB-INF/spring/root-context.xml
972     <?xml version="1.0" encoding="UTF-8"?>
973     <beans xmlns="http://www.springframework.org/schema/beans"
974       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
975       xmlns:context="http://www.springframework.org/schema/context"
976       xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
977         http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-4.3.xsd">
```

```
978
979         <!-- Root Context: defines shared resources visible to all other web components -->
980         <context:property-placeholder location="classpath:mariadb.properties"/>
981         <bean id="dataSource"
            class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
982           <property name="driverClass" value="${db.driverClass}" />
983           <property name="url" value="${db.url}" />
984           <property name="username" value="${db.username}" />
985           <property name="password" value="${db.password}" />
986         </bean>
987
988         <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
989           <property name="dataSource" ref="dataSource" />
990         </bean>
991       </beans>
992
993     6)src/test/java/com.example.biz/TestApp class 생성(JUnit Test Case)
994       package com.example.biz;
995
996       import org.junit.Before;
997       import org.junit.Test;
998       import org.springframework.context.ApplicationContext;
999       import org.springframework.context.support.GenericXmlApplicationContext;
1000      import org.springframework.jdbc.core.JdbcTemplate;
1001
1002      public class TestApp {
1003        private ApplicationContext ctx;
1004
1005        @Before
1006        public void init() {
1007          this.ctx = new
            GenericXmlApplicationContext("file:src/main/webapp/WEB-INF/spring**/root-context.x
            ml");
1008        }
1009        @Test
1010        public void test() {
1011          JdbcTemplate jdbcTemplate = this.ctx.getBean("jdbcTemplate", JdbcTemplate.class);
1012          System.out.println(jdbcTemplate);
1013        }
1014      }
1015      -Run as > JUnit Test > Green bar
1016
1017    7)package 생성
1018      -src/main/java/com.example.vo
1019      -src/main/java/com.example.dao
1020      -src/main/java/com.example.service
1021
1022    8)com/example.dao
1023      -MemberDao interface
1024        package com.example.dao;
1025
1026        import java.util.List;
1027
1028        import com.example.vo.MemberVO;
```

```
1029
1030        public interface MemberDao {
1031          int create(MemberVO memberVo);
1032          MemberVO read(String userid);
1033          List<MemberVO> readAll();
1034          int update(MemberVO memberVo);
1035          int delete(String userid);
1036        }
1037
1038      -MemberDaoImple.java
1039        package com.example.dao;
1040
1041        import java.sql.ResultSet;
1042        import java.sql.SQLException;
1043        import java.util.List;
1044
1045        import org.springframework.beans.factory.annotation.Autowired;
1046        import org.springframework.jdbc.core.JdbcTemplate;
1047        import org.springframework.jdbc.core.RowMapper;
1048        import org.springframework.stereotype.Repository;
1049
1050        import com.example.vo.MemberVO;
1051
1052        @Repository("memberDao")
1053        public class MemberDaoImpl implements MemberDao {
1054          @Autowired
1055          JdbcTemplate jdbcTemplate;
1056
1057          @Override
1058          public int create(MemberVO memberVo) {
1059            String sql = "INSERT INTO Member VALUES(?,?,?,?,?)";
1060            return this.jdbcTemplate.update(sql, memberVo.getUserid(),
1061                memberVo.getUsername(), memberVo.getUserage(),
1062                memberVo.getGender(), memberVo.getCity());
1063          }
1064
1065          @Override
1066          public MemberVO read(String userid) {
1067            String sql = "SELECT * FROM Member WHERE userid = ?";
1068            return this.jdbcTemplate.queryForObject(sql, new Object[] {userid},
1069                        new MyRowMapper());
1070          }
1071
1072          class MyRowMapper implements RowMapper<MemberVO>{
1073            @Override
1074            public MemberVO mapRow(ResultSet rs, int rowNum) throws SQLException {
1075              MemberVO memberVo = new MemberVO(rs.getString("userid"),
1076                  rs.getString("username"), rs.getInt("userage"),
1077                  rs.getString("gender"), rs.getString("city"));
1078              return memberVo;
1079            }
1080          }
1081          @Override
1082          public List<MemberVO> readAll() {
```

```
1083            String sql = "SELECT * FROM Member ORDER BY userid DESC";
1084            return this.jdbcTemplate.query(sql, new MyRowMapper());
1085          }
1086
1087          @Override
1088          public int update(MemberVO memberVo) {
1089            String sql = "UPDATE Member SET userage = ?, gender = ?, city = ? " +
1090                         "WHERE userid = ?";
1091            return this.jdbcTemplate.update(sql, memberVo.getUserage(),
1092                 memberVo.getGender(), memberVo.getCity(), memberVo.getUserid());
1093          }
1094
1095          @Override
1096          public int delete(String userid) {
1097            String sql = "DELETE FROM Member WHERE userid = ?";
1098            return this.jdbcTemplate.update(sql, userid);
1099          }
1100        }
1101
1102    9)com.example.service
1103      -MemberService interface
1104        package com.example.service;
1105
1106        import java.util.List;
1107
1108        import com.example.vo.MemberVO;
1109
1110        public interface MemberService {
1111          int create(MemberVO memberVo);
1112          MemberVO read(String userid);
1113          List<MemberVO> readAll();
1114          int update(MemberVO memberVo);
1115          int delete(String userid);
1116        }
1117
1118      -MemberServiceImpl.java
1119        package com.example.service;
1120
1121        import java.util.List;
1122
1123        import org.springframework.beans.factory.annotation.Autowired;
1124        import org.springframework.stereotype.Service;
1125
1126        import com.example.dao.MemberDao;
1127        import com.example.vo.MemberVO;
1128
1129        @Service("memberService")
1130        public class MemberServiceImpl implements MemberService {
1131          @Autowired
1132          MemberDao memberDao;
1133
1134          @Override
1135          public int create(MemberVO memberVo) {
1136            return this.memberDao.create(memberVo);
```

```
        }

        @Override
        public MemberVO read(String userid) {
          return this.memberDao.read(userid);
        }

        @Override
        public List<MemberVO> readAll() {
          return this.memberDao.readAll();
        }

        @Override
        public int update(MemberVO memberVo) {
          return this.memberDao.update(memberVo);
        }

        @Override
        public int delete(String userid) {
          return this.memberDao.delete(userid);
        }

      }

  10)com.example.biz
    -HomeController.java
      package com.example.biz;

      import java.util.List;

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

      import com.example.service.MemberService;
      import com.example.vo.MemberVO;

      /**
       * Handles requests for the application home page.
       */
      @Controller
      public class HomeController {
        @Autowired
        MemberService memberService;

        @RequestMapping(value = "/create", method = RequestMethod.POST)
        public String home(MemberVO memberVo, Model model) {
          int row = this.memberService.create(memberVo);
          if(row == 1) model.addAttribute("status", "Insert Success");
          else model.addAttribute("status", "Insert Failure");
```

```
1191            return "create";    //   /WEB-INF/views/create.jsp
1192          }
1193
1194          @RequestMapping(value = "/list", method = RequestMethod.GET)
1195          public String list(Model model) {
1196            List<MemberVO> list = this.memberService.readAll();
1197            model.addAttribute("userlist", list);
1198            return "list";    // /WEB-INF/views/list.jsp
1199          }
1200
1201          @RequestMapping(value = "/view/{userid}", method = RequestMethod.GET)
1202          public String view(@PathVariable String userid, Model model) {
1203            MemberVO memberVo = this.memberService.read(userid);
1204            model.addAttribute("member", memberVo);
1205            return "view";
1206          }
1207
1208          @RequestMapping(value = "/delete/{userid}", method = RequestMethod.GET)
1209          public String delete(@PathVariable String userid) {
1210            this.memberService.delete(userid);
1211            return "redirect:/list";
1212          }
1213
1214          @RequestMapping(value = "/update", method = RequestMethod.POST)
1215          public String update(@RequestParam("userid") String userid,
1216              @RequestParam("userage") int userage,
1217              @RequestParam("gender") String gender,
1218              @RequestParam("city") String city) {
1219            this.memberService.update(
1220                new MemberVO(userid, "", userage, gender, city));
1221            return "redirect:/list";
1222          }
1223        }
1224
1225    11)views/create.jsp
1226      <%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
1227      <!DOCTYPE html>
1228      <html>
1229      <head>
1230      <meta charset="UTF-8">
1231      <title>Insert title here</title>
1232      </head>
1233      <body>
1234        <h1>${status}</h1>
1235      </body>
1236      </html>
1237
1238    12)views/list.jsp
1239      <%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
1240      <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
1241      <!DOCTYPE html>
1242      <html>
```

```
1243      <head>
1244      <meta charset="UTF-8">
1245      <title>Insert title here</title>
1246      </head>
1247      <body>
1248        <h1>Member List</h1>
1249        <table border='1'>
1250          <thead>
1251            <tr>
1252              <th>아이디</th><th>이름</th><th>나이</th><th>성별</th><th>거주지</th>
1253            </tr>
1254          </thead>
1255          <tbody>
1256            <c:forEach items="${userlist}" var="user">
1257              <tr>
1258                <td><a
                  href="/biz/view/${user.userid}">${user.userid}</a></td><td>${user.username
                  }</td>
1259                <td>${user.userage}</td><td>${user.gender }</td>
1260                <td>${user.city }</td>
1261              </tr>
1262            </c:forEach>
1263          </tbody>
1264        </table>
1265      </body>
1266      </html>
1267
1268    13)views/view.jsp
1269      <%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
1270      <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
1271      <c:set var="user" value="${member}" />
1272      <!DOCTYPE html>
1273      <html>
1274      <head>
1275        <meta charset="UTF-8">
1276        <title>Insert title here</title>
1277        <script src="/biz/static/js/jquery.min.js"></script>
1278        <script>
1279          $(function(){
1280            $("#btnList").bind("click", function(){
1281              location.href = "/biz/list";
1282            });
1283            $("#btnDelete").bind("click", function(){
1284              location.href = "/biz/delete/${user.userid}";
1285            });
1286          });
1287        </script>
1288      </head>
1289      <body>
1290        <h1>${user.username}의정보</h1>
1291        <form action="/biz/update" method="post">
1292          <input type="hidden" name="userid" value = "${user.userid}" />
1293          <ul>
```

```
1294            <li>아이디 : ${user.userid }</li>
1295            <li>나이 : <input type='number' name="userage" value='${user.userage}' /></li>
1296            <li>성별 : <c:if test='${user.gender eq "남성"}'>
1297                 <input type="radio" name="gender" value="남성" checked />남성  
1298                 <input type="radio" name="gender" value="여성" />여성
1299                 </c:if>
1300                 <c:if test='${user.gender eq "여성"}'>
1301                 <input type="radio" name="gender" value="남성" />남성  
1302               <input type="radio" name="gender" value="여성" checked />여성
1303            </c:if>
1304            </li>
1305            <li>거주지 : <input type="text" name="city" value="${user.city }" /></li>
1306            <li><input type='submit' value='수정하기' /></li>
1307            <li><input type='button' value='삭제하기'  id="btnDelete"/></li>
1308            <li><input type='button' value='목록으로'  id="btnList"/></li>
1309          </ul>
1310        </form>
1311      </body>
1312      </html>
1313
1314    14)pom.xml
1315      <properties>
1316        <java-version>1.8</java-version>
1317        <org.springframework-version>4.3.20.RELEASE</org.springframework-version>
1318        <org.aspectj-version>1.9.2</org.aspectj-version>
1319      </properties>
1320      <dependencies>
1321        ...
1322        <dependency>
1323          <groupId>javax.servlet</groupId>
1324          <artifactId>javax.servlet-api</artifactId>
1325          <version>3.1.0</version>
1326          <scope>provided</scope>
1327        </dependency>
1328        <dependency>
1329          <groupId>javax.servlet.jsp</groupId>
1330          <artifactId>javax.servlet.jsp-api</artifactId>
1331          <version>2.3.1</version>
1332          <scope>provided</scope>
1333        </dependency>
1334        <dependency>
1335          <groupId>javax.servlet</groupId>
1336          <artifactId>jstl</artifactId>
1337          <version>1.2</version>
1338        </dependency>
1339        <dependency>
1340          <groupId>junit</groupId>
1341          <artifactId>junit</artifactId>
1342          <version>4.12</version>
1343          <scope>test</scope>
1344        </dependency>
1345        <dependency>
1346          <groupId>org.springframework</groupId>
1347          <artifactId>spring-jdbc</artifactId>
```

```
1348          <version>4.3.20.RELEASE</version>
1349        </dependency>
1350        <dependency>
1351          <groupId>org.mariadb.jdbc</groupId>
1352          <artifactId>mariadb-java-client</artifactId>
1353          <version>2.3.0</version>
1354        </dependency>
1355        <dependency>
1356          <groupId>org.springframework</groupId>
1357          <artifactId>spring-test</artifactId>
1358          <version>4.3.20.RELEASE</version>
1359          <scope>test</scope>
1360        </dependency>
1361      </dependencies>
1362
1363
1364  17. Lab : Form Data Validation
1365    1)Package Explorer > right-click > New > Spring Legacy Project
1366    2)Select Spring MVC Project
1367    3)Project name : 1208 > Next
1368    4)Enter a topLevelPackage : com.example.biz > Finish
1369    5)UserVO 객체 생성
1370      -src/main/java/com.example.vo package 생성
1371      -src/main/java/com.example.vo.UserVO class
1372
1373        package com.example.vo;
1374
1375        public class UserVO {
1376          private String name;
1377          private int age;
1378          private String userid;
1379          public String getName() {
1380            return name;
1381          }
1382          public void setName(String name) {
1383            this.name = name;
1384          }
1385          public int getAge() {
1386            return age;
1387          }
1388          public void setAge(int age) {
1389            this.age = age;
1390          }
1391          public String getUserid() {
1392            return userid;
1393          }
1394          public void setUserid(String userid) {
1395            this.userid = userid;
1396          }
1397          @Override
1398          public String toString() {
1399            return "UserVO [name=" + name + ", age=" + age + ", userid=" + userid + "]";
1400          }
1401        }
```

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

```
package com.example.biz;

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

import com.example.vo.UserVO;

public class UserValidator implements Validator {

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

  @Override
  public void validate(Object obj, Errors errors) {
    System.out.println("검증시작");
    UserVO userVO = (UserVO)obj;

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

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

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

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

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

@RequestMapping(value = "/register", method=RequestMethod.POST)
```

```
1456        public String register(@ModelAttribute("userVO") UserVO userVO, BindingResult result) {
1457          String page = "register_ok";
1458          UserValidator validator = new UserValidator();
1459          validator.validate(userVO, result);
1460          if(result.hasErrors()) {
1461            page = "register";
1462          }
1463          return page;
1464        }
1465
1466      -src/main/webapp/WEB-INF/views/register.jsp
1467        <%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
1468        <!DOCTYPE html>
1469        <html>
1470        <head>
1471        <meta charset="UTF-8">
1472        <title>회원 가입 폼</title>
1473        </head>
1474        <body>
1475          <form action="/biz/register" method="post">
1476            Name : <input type="text" name="name" /><br />
1477            Age : <input type="number" name="age" /><br />
1478            ID : <input type="text" name="userid" /><br />
1479            <input type="submit" value="가입하기" />
1480          </form>
1481        </body>
1482        </html>
1483
1484      -src/main/webapp/WEB-INF/views/register_ok.jsp
1485        <%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
1486        <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
1487        <c:set var="user" value="${userVO}" />
1488        <!DOCTYPE html">
1489        <html>
1490        <head>
1491        <meta charset="UTF-8">
1492        <title>회원 가입 결과 창</title>
1493        </head>
1494        <body>
1495          <ul>
1496            <li>이름 : ${user.name}</li>
1497            <li>나이 : ${user.age}</li>
1498            <li>아이디 : ${user.userid}</li>
1499          </ul>
1500        </body>
1501        </html>
1502
1503
1504    7)ValidataionUtils class를 이용한 검증
1505      -ValidatationUtils class는 validate() method를 좀 더 편리하게 사용할 수 있게 해줌.
1506      -UserValidator.java 수정
1507
1508        /*String username = userVO.getName();
```

```
1509        if(username == null || username.trim().isEmpty()) {
1510          System.out.println("이름의 값이 빠졌습니다.");
1511          errors.rejectValue("name", "No Value");
1512        }*/
1513
1514        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "No Value");
1515
1516    8)@Valid와 @InitBinder 이용하기
1517      -Spring Framework이 대신 검증해 줌
1518      -mvnrepository에서 'hibernate validator'로 검색
1519
1520        <dependency>
1521            <groupId>org.hibernate.validator</groupId>
1522            <artifactId>hibernate-validator</artifactId>
1523            <version>6.0.5.Final</version>
1524        </dependency>
1525
1526      -pom.xml에 넣고 Maven Clean > Maven Install
1527      -HomeController.java 수정
1528
1529        @RequestMapping(value = "/register", method=RequestMethod.POST)
1530        public String register(@ModelAttribute("userVO") @Valid UserVO userVO, BindingResult
          result) {
1531          String page = "register_ok";
1532          //UserValidator validator = new UserValidator();
1533          //validator.validate(userVO, result);
1534          if(result.hasErrors()) {
1535            page = "register";
1536          }
1537
1538          return page;
1539        }
1540
1541        @InitBinder
1542        protected void initBinder(WebDataBinder binder) {
1543          binder.setValidator(new UserValidator());
1544        }
1545
1546
1547  18. Lab
1548    1)In J2EE Perspective
1549    2)Project Explorer > right-click > New > Dynamic Web Project
1550    3)Project name : SpringWebDemo > Next > Check [Generate web.xml deployment descriptor]
        > Finish
1551
1552    4)Convert to Maven Project
1553      -project right-click > Configure > Convert to Maven Project > Finish
1554
1555    5)Add Spring Project Nature
1556      -project right-click > Spring Tools > Add Spring Project Nature
1557
1558    6)새로 생선된 pom.xmlfile에 필요한 library 추가 > Maven Clean > Maven Install
1559      <dependencies>
1560        <dependency>
```

```
1561          <groupId>org.springframework</groupId>
1562          <artifactId>spring-context</artifactId>
1563          <version>4.3.13.RELEASE</version>
1564        </dependency>
1565        <dependency>
1566            <groupId>junit</groupId>
1567            <artifactId>junit</artifactId>
1568            <version>4.12</version>
1569            <scope>test</scope>
1570        </dependency>
1571        <dependency>
1572          <groupId>org.springframework</groupId>
1573          <artifactId>spring-jdbc</artifactId>
1574          <version>4.3.13.RELEASE</version>
1575        </dependency>
1576      </dependencies>
1577
1578    7)Spring mvc library 검색 및 설치
1579      -http://mvnrepository.com에서 'spring mvc'로 검색
1580      -pom.xml에 추가
1581
1582        <dependency>
1583            <groupId>org.springframework</groupId>
1584            <artifactId>spring-webmvc</artifactId>
1585            <version>4.3.13.RELEASE</version>
1586        </dependency>
1587
1588      -Maven Clean > Maven Install
1589
1590    8)Build path에 config foler 추가
1591      -project right-click > Build Path > Configure Build Path > Select [Source] tab
1592      -Click [Add Folder] > Select 현재 project > Click [Create New Folder...]
1593      -Folder name : config > Finish > OK > Apply and Close
1594      -Java Resources > config 폴더 확인
1595
1596    9)config folder에 beans.xml file 생성
1597      -Spring Perspective로 전환
1598      -beans.xml
1599      -생성시 beans,context, mvc 체크
1600        <?xml version="1.0" encoding="UTF-8"?>
1601        <beans xmlns="http://www.springframework.org/schema/beans"
1602          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1603          xmlns:context="http://www.springframework.org/schema/context"
1604          xsi:schemaLocation="http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans.xsd
1605            http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context-3.2.xsd">
1606
1607
1608        </beans>
1609
1610    10)ContextLoaderListener class 설정
1611      -비즈니스 로직용의 스프링 설정 file (ex:applicationContext.xml)을 작성했기 때문에 listener로
          ContextLoaderListener class를 정의해야 한다.
```

-DispatcherServlet이 생성되어 Spring Container를 구동하면 Controller들이 메모리에 로딩된다.
-하지만 Controller들이 생성되기 전에 누군가가 먼저 config폴더에 있는 beans.xml file을 읽어 Business Component들을 메모리에 생성해야 한다.
-이때 사용하는 class가 ContextLoaderListener class이다.
-<listener>태그 하위에 <listener-class> 태그를 이용하여 Spring에서 제공하는 ContextLoaderListener를 등록하면 된다.
-ContextLoaderListener class는 스프링 설정 file(디폴트에서 file명 applicationContext.xml)을 로드하면 ServletContextListener 인터페이스를 구현하고 있기 때문에 ServletContext 인스턴스 생성 시(톰캣으로 어플리케이션이 로드된 때)에 호출된다.
-즉, ContextLoaderListener class는 DispatcherServlet class의 로드보다 먼저 동작하여 비즈니스 로직층을 정의한 스프링 설정 file을 로드한다.
-중요한 것은 ContextLoaderListener class는 Servlet Container가 web.xml file을 읽어서 구동할 때, 자동으로 메모리에 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 코드가 자동 삽입

```
  <!-- needed for ContextLoaderListener -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>location</param-value>
  </context-param>

  <!-- Bootstraps the root web application context before servlet initialization -->
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class
    >
  </listener>
```

-아래 코드로 변환

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

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

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

  <!-- Map all requests to the DispatcherServlet for handling -->
  <servlet-mapping>
    <servlet-name>springDispatcherServlet</servlet-name>
```

```
1656          <url-pattern>url</url-pattern>
1657        </servlet-mapping>
1658
1659      -아래의 코드로 변환
1660        <init-param>
1661          <param-name>contextConfigLocation</param-name>
1662          <param-value>classpath:beans*.xml</param-value>
1663        </init-param>
1664
1665        <servlet-mapping>
1666          <servlet-name>springDispatcherServlet</servlet-name>
1667          <url-pattern>*.do</url-pattern>
1668        </servlet-mapping>
1669
1670    12)mvnrepository에서 'jstl'로 검색 후 설치
1671      -목록에서 2번째 : 1.2버전
1672
1673        <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
1674        <dependency>
1675            <groupId>javax.servlet</groupId>
1676            <artifactId>jstl</artifactId>
1677            <version>1.2</version>
1678        </dependency>
1679
1680      -pom.xml에 붙여넣고 Maven Clean > Maven Install
1681
1682    13)Controller와 JSP 호출순서
1683      -Controller와 JSP 호출순서.png 그림 참조
1684
1685    14)Hello Controller 작성
1686      -Client의 요청을 처리할 POJO 형태의 HelloController class를 작성
1687      -Controller class에 @Controller annotation 선언
1688      -요청을 처리할 method를 작성하고 @RequestMapping annotation을 선언
1689        --@RequestMapping : HTTP 요청 URL을 처리할 Controller method 정의
1690      -JSP를 이용한 View 영역의 코드를 작성
1691      -Browser 상에서 JSP를 실행
1692
1693
1694      -src/com.example.vo package 생성
1695      -src/com.example.HelloVO class 생성
1696
1697        package com.example.vo;
1698
1699        public class HelloVO {
1700          private String name;
1701
1702          public void setName(String name) {
1703            this.name = name;
1704          }
1705
1706          public String sayHello() {
1707            return "Hello " + name;
1708          }
1709        }
```

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

```
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 com.example.vo.HelloVO;

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

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

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

16)beans.xml 수정

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

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

17)WebContent/hello.jsp 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
  </head>
  <body>
    ${greet}
  </body>
</html>
```

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

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

Hello 한지민

19. Lab

1)In J2EE Perspective

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

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

4)Convert to Maven Project

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

5)Add Spring Project Nature

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

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

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.3.13.RELEASE</version>
  </dependency>
  <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>4.3.13.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>
  <dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.1</version>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>4.3.13.RELEASE</version>
  </dependency>
</dependencies>
```

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

8)config folder에 beans.xml file 생성
-Spring Perspective로 전환
-config right-click > New > Spring Bean Configuration File
-File name : beans.xml
-생성시 beans,context, mvc 체크

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.2.xsd">


</beans>
```

9)ContextLoaderListener class 설정
-비즈니스 로직용의 스프링 설정 file (ex:applicationContext.xml)을 작성했기 때문에 listener로 ContextLoaderListener class를 정의해야 한다.
-ContextLoaderListener class는 스프링 설정 file(디폴트에서 file명 applicationContext.xml)을 로드하면 ServletContextListener 인터페이스를 구현하고 있기 때문에 ServletContext 인스턴스 생성 시(톰켓으로 어플리케이션이 로드된 때)에 호출된다. 즉, ContextLoaderListener class는 DispatcherServlet class의 로드보다 먼저 동작하여 비즈니스 로직층을 정의한 스프링 설정 file을 로드한다.
-web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener - ContextLoaderListener] 를 선택하면 아래의 코드가 자동 삽입

```
<!-- needed for ContextLoaderListener -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>location</param-value>
</context-param>

<!-- Bootstraps the root web application context before servlet initialization -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class
  >
</listener>
```

-아래 코드로 변환

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

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

```
1861
1862        <!-- The front controller of this Spring Web application, responsible for handling all
            application requests -->
1863        <servlet>
1864          <servlet-name>springDispatcherServlet</servlet-name>
1865          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
1866          <init-param>
1867            <param-name>contextConfigLocation</param-name>
1868            <param-value>location</param-value>
1869          </init-param>
1870          <load-on-startup>1</load-on-startup>
1871        </servlet>
1872
1873        <!-- Map all requests to the DispatcherServlet for handling -->
1874        <servlet-mapping>
1875          <servlet-name>springDispatcherServlet</servlet-name>
1876          <url-pattern>url</url-pattern>
1877        </servlet-mapping>
1878
1879      -아래의 코드로 변환
1880        <init-param>
1881          <param-name>contextConfigLocation</param-name>
1882          <param-value>classpath:beans*.xml</param-value>
1883        </init-param>
1884
1885        <servlet-mapping>
1886          <servlet-name>springDispatcherServlet</servlet-name>
1887          <url-pattern>*.do</url-pattern>
1888        </servlet-mapping>
1889
1890    11)UserVO class 생성
1891      -src/com.example.vo package 생성
1892      -src/com.example.vo.UserVO class 생성
1893
1894        package com.example.vo;
1895
1896        public class UserVO {
1897          private String userId;
1898          private String name;
1899          private String gender;
1900          private String city;
1901          public UserVO() {}
1902          public UserVO(String userId, String name, String gender, String city) {
1903            this.userId = userId;
1904            this.name = name;
1905            this.gender = gender;
1906            this.city = city;
1907          }
1908          public String getUserId() {
1909            return userId;
1910          }
1911          public void setUserId(String userId) {
1912            this.userId = userId;
1913          }
```

```
1914          public String getName() {
1915            return name;
1916          }
1917          public void setName(String name) {
1918            this.name = name;
1919          }
1920          public String getGender() {
1921            return gender;
1922          }
1923          public void setGender(String gender) {
1924            this.gender = gender;
1925          }
1926          public String getCity() {
1927            return city;
1928          }
1929          public void setCity(String city) {
1930            this.city = city;
1931          }
1932          @Override
1933          public String toString() {
1934            return "UserVO [userId=" + userId + ", name=" + name + ", gender=" + gender + ",
                city=" + city + "]";
1935          }
1936        }
1937
1938    12)UserDao 객체 생성
1939      -src/com.example.dao package 생성
1940      -src/com.example.dao.UserDao interface
1941
1942        package com.example.dao;
1943
1944        import java.util.List;
1945
1946        import com.example.vo.UserVO;
1947
1948        public interface UserDao {
1949          void insert(UserVO user);
1950
1951          List<UserVO> readAll();
1952
1953          void update(UserVO user);
1954
1955          void delete(String id);
1956
1957          UserVO read(String id);
1958        }
1959
1960      -src/com.example.dao.UserDaoImplJDBC.java 생성
1961
1962        package com.example.dao;
1963
1964        import java.sql.ResultSet;
1965        import java.sql.SQLException;
1966        import java.util.List;
```

```
1967
1968        import javax.sql.DataSource;
1969
1970        import org.springframework.beans.factory.annotation.Autowired;
1971        import org.springframework.dao.EmptyResultDataAccessException;
1972        import org.springframework.jdbc.core.JdbcTemplate;
1973        import org.springframework.jdbc.core.RowMapper;
1974        import org.springframework.stereotype.Repository;
1975
1976        import com.example.vo.UserVO;
1977
1978        @Repository("userDao")
1979        public class UserDaoImplJDBC implements UserDao {
1980          private JdbcTemplate jdbcTemplate;
1981
1982          @Autowired
1983          public void setDataSource(DataSource dataSource) {
1984            this.jdbcTemplate = new JdbcTemplate(dataSource);
1985          }
1986
1987          class UserMapper implements RowMapper<UserVO> {
1988            public UserVO mapRow(ResultSet rs, int rowNum) throws SQLException {
1989              UserVO user = new UserVO();
1990              user.setUserId(rs.getString("userid"));
1991              user.setName(rs.getString("name"));
1992              user.setGender(rs.getString("gender"));
1993              user.setCity(rs.getString("city"));
1994              return user;
1995            }
1996          }
1997
1998          @Override
1999          public void insert(UserVO user) {
2000            String SQL = "INSERT INTO users (userid, name, gender,city) VALUES (?, ?, ?, ?)";
2001            jdbcTemplate.update(SQL, user.getUserId(), user.getName(), user.getGender(),
                user.getCity());
2002
2003            System.out.println("등록된 Record UserId=" + user.getUserId() + " Name=" +
                user.getName());
2004          }
2005
2006          @Override
2007          public List<UserVO> readAll() {
2008            String SQL = "SELECT * FROM users";
2009            List<UserVO>  userList = jdbcTemplate.query(SQL, new UserMapper());
2010            return userList;
2011          }
2012
2013          @Override
2014          public void update(UserVO user) {
2015            String SQL = "UPDATE users  SET name = ?, gender = ?, city = ? WHERE userid = ?";
2016            jdbcTemplate.update(SQL, user.getName(), user.getGender(),
                user.getCity(),user.getUserId());
2017            System.out.println("갱신된 Record with ID = " + user.getUserId() );
```

```
2018          }
2019
2020          @Override
2021          public void delete(String id) {
2022            String SQL = "delete from users where userid = ?";
2023            jdbcTemplate.update(SQL, id);
2024            System.out.println("삭제된 Record with ID = " + id );
2025          }
2026
2027          @Override
2028          public UserVO read(String id) {
2029            String SQL = "SELECT * FROM users WHERE userid = ?";
2030            try {
2031              UserVO user = jdbcTemplate.queryForObject(SQL,
2032                   new Object[] { id }, new UserMapper());
2033              return user;
2034            }catch(EmptyResultDataAccessException e){
2035              return null;
2036            }
2037          }
2038        }
2039
2040    13)UserService 객체 생성
2041      -src/com.example.service package 생성
2042      -src/com.example.service.UserService interface
2043
2044        package com.example.service;
2045
2046        import java.util.List;
2047
2048        import com.example.vo.UserVO;
2049
2050        public interface UserService {
2051          void insertUser(UserVO user);
2052
2053          List<UserVO> getUserList();
2054
2055          void deleteUser(String id);
2056
2057          UserVO getUser(String id);
2058
2059          void updateUser(UserVO user);
2060        }
2061
2062      -src/com.example.service.UserServiceImpl.java
2063
2064        package com.example.service;
2065
2066        import java.util.List;
2067
2068        import org.springframework.beans.factory.annotation.Autowired;
2069        import org.springframework.stereotype.Service;
2070
2071        import com.example.dao.UserDao;
```

```
2072        import com.example.vo.UserVO;
2073
2074        @Service("userService")
2075        public class UserServiceImpl implements UserService {
2076
2077          @Autowired
2078          UserDao userDao;
2079
2080          @Override
2081          public void insertUser(UserVO user) {
2082            this.userDao.insert(user);
2083          }
2084
2085          @Override
2086          public List<UserVO> getUserList() {
2087            return this.userDao.readAll();
2088          }
2089
2090          @Override
2091          public void deleteUser(String id) {
2092            this.userDao.delete(id);
2093          }
2094
2095          @Override
2096          public UserVO getUser(String id) {
2097            return this.userDao.read(id);
2098          }
2099
2100          @Override
2101          public void updateUser(UserVO user) {
2102            this.userDao.update(user);
2103          }
2104        }
2105
2106    14)UserController 객체 생성
2107      -src/com.example.controller package 생성
2108      -com.example.controller.UserController class 생성
2109
2110        package com.example.controller;
2111
2112        import org.springframework.beans.factory.annotation.Autowired;
2113        import org.springframework.stereotype.Controller;
2114        import org.springframework.ui.Model;
2115        import org.springframework.web.bind.annotation.RequestMapping;
2116        import org.springframework.web.bind.annotation.RequestParam;
2117
2118        import com.example.service.UserService;
2119        import com.example.vo.UserVO;
2120
2121        @Controller
2122        public class UserController {
2123          @Autowired
2124          private UserService userService;
2125
```

```
2126          @RequestMapping("/getUser.do")
2127          public String getUserList(@RequestParam("userId") String userId, Model model) {
2128            UserVO user = userService.getUser(userId);
2129            //System.out.println(user);
2130            model.addAttribute("user", user);
2131            return "userInfo.jsp";
2132          }
2133        }
2134
2135    15)config/dbinfo.properties file 생성
2136
2137      db.driverClass=oracle.jdbc.driver.OracleDriver
2138      db.url=jdbc:oracle:thin:@localhost:1521:XE
2139      db.username=scott
2140      db.password=tiger
2141
2142    16)beans.xml 수정
2143
2144      <context:component-scan base-package="com.example" />
2145
2146      <context:property-placeholder location="classpath:dbinfo.properties" />
2147      <bean id="dataSource"
         class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
2148        <property name="driverClass" value="${db.driverClass}" />
2149        <property name="url" value="${db.url}" />
2150        <property name="username" value="${db.username}" />
2151        <property name="password" value="${db.password}" />
2152      </bean>
2153
2154    17)WebContent/index.jsp 생성
2155
2156      <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2157      <c:redirect url="userinfo.do" />
2158
2159    18)project > right-click > Run As > Run on Server > Finish
2160
2161    19)http://localhost:8080/SpringWebDemo/userinfo.do?userId=jimin
2162
2163
2164  20. File Upload
2165    1)Server에 File을 저장할때 고려해야 할 사항들
2166      -File Upload 방식 결정하기.
2167        --Post 방식으로 전송할지 아니면 Ajax 방식으로 전송할지 결정해야 한다.
2168        --아마 요즘은 주로 Ajax방식을 사용하는것 같다.
2169      -File이름 중복문제.
2170        --DB에 File을 저장할수도 있지만, 일반적으로 FileSystem에 File을 저장하게된다.
2171        --따라서 upload 되는 file의 이름의 중복을 해결할 방법이 필요하다. -> UUID로 해결가능
2172      -File 저장경로에 대한문제.
2173        --Windows나 Linux등 운영체제에서 folder내의 file 개수가 너무 많아지게 되면, 속도저하 문제가 발생하게 된다.
2174        --특히 Windows의 FileSystem의 경우 folder내 최대 file 개수의 제한이 있다.(100만단위가 넘어가긴 하지만
           …)
2175        --위 문제를 해결하기 위해서 보통 file이 upload 되는 시점별로 folder를 관리한다.
2176        --예를 들어 2018년 9월 6일 file이 upload 되면, 그 file은 특정 folder의 경로의 /2018/09/06/ 경로에 저장
           하면 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>

<!-- Bootstraps the root web application context before servlet initialization -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

-아래 코드로 변환

```
<context-param>
  <param-name>contextConfigLocation</param-name>
```

```
2226          <param-value>classpath:applicationContext.xml</param-value>
2227        </context-param>
2228
2229    8)DispatcherServlet Class 추가
2230      -web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet -
          DispatcherServlet declaration] 선택하면 아래의 코드가 자동 추가된다.
2231
2232        <!-- The front controller of this Spring Web application, responsible for handling all
          application requests -->
2233        <servlet>
2234          <servlet-name>springDispatcherServlet</servlet-name>
2235          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
2236          <init-param>
2237            <param-name>contextConfigLocation</param-name>
2238            <param-value>location</param-value>
2239          </init-param>
2240          <load-on-startup>1</load-on-startup>
2241        </servlet>
2242
2243        <!-- Map all requests to the DispatcherServlet for handling -->
2244        <servlet-mapping>
2245          <servlet-name>springDispatcherServlet</servlet-name>
2246          <url-pattern>url</url-pattern>
2247        </servlet-mapping>
2248
2249      -아래의 코드로 변환
2250        <init-param>
2251          <param-name>contextConfigLocation</param-name>
2252          <param-value>classpath:beans.xml</param-value>
2253        </init-param>
2254
2255        <servlet-mapping>
2256          <servlet-name>springDispatcherServlet</servlet-name>
2257          <url-pattern>/</url-pattern>
2258        </servlet-mapping>
2259
2260    9)FileUpload library 추가
2261      -Apache에서 제공하는 Common FileUpload library를 사용하여 file upload를 처리하기 위한 library
2262      -mvnrepository에서 'common fileupload'라고 검색하여 library 추가
2263        <dependency>
2264           <groupId>commons-fileupload</groupId>
2265           <artifactId>commons-fileupload</artifactId>
2266           <version>1.3.3</version>
2267        </dependency>
2268
2269      -mvnrepository에서 'commons io'라고 검색하여 library 추가
2270        <dependency>
2271           <groupId>commons-io</groupId>
2272           <artifactId>commons-io</artifactId>
2273           <version>2.4</version>
2274        </dependency>
2275
2276      -Maven Clean > Maven Install
2277
```

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" />
  <!-- 디스크에 임시 file을 생성하기 전에 메모리에 보관할수있는 최대 바이트 크기 -->
  <!-- property name="maxInMemorySize" value="52428800" / -->
  <!-- defaultEncoding -->
  <property name="defaultEncoding" value="utf-8" />
</bean>
```

-maxUploadSize에 대한 setter injection은 upload할 수 있는 file의 크기를 제한하기 위한 설정이다.
-지정하지 않으면 기본으로 -1이 설정되는데, 이는 file의 크기가 무제한이라는 의미이다.

14)web.xml에 한글file encoding 처리하기
-한글 file이 upload될 때 file 명이 깨지는 것을 해결하기 위해 web.xml에 아래 내용을 추가한다.

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

15)View 작성
-WebContent/form.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>
  <h1>file 업로드 예제</h1>
  <form method="post" action="upload" enctype="multipart/form-data">
    <label>email:</label> <input type="text" name="email"> <br>
    <br> <label>file:</label> <input type="file" name="file1">
    <br>
    <br> <input type="submit" value="upload">
  </form>
</body>
</html>
```

16)Service 작성
-src/com.example.service package
-src/com.example.service.FileUploadService.java

```
package com.example.service;

import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Calendar;

import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
```

```
2381
2382        @Service
2383        public class FileUploadService {
2384          // 리눅스 기준으로 file 경로를 작성 ( 루트 경로인 /으로 시작한다. )
2385          // 윈도우라면 workspace의 드라이브를 파악하여 JVM이 알아서 처리해준다.
2386          // 따라서 workspace가 C드라이브에 있다면 C드라이브에 upload 폴더를 생성해 놓아야 한다.
2387          private static final String SAVE_PATH = "/upload";
2388          private static final String PREFIX_URL = "/upload/";
2389
2390          public String restore(MultipartFile multipartFile) {
2391            String uri = null;
2392
2393            try {
2394              // file 정보
2395              String originFilename = multipartFile.getOriginalFilename();
2396              String extName = originFilename.substring(originFilename.lastIndexOf("."),
                  originFilename.length());
2397              Long size = multipartFile.getSize();
2398
2399              // 서버에서 저장 할 file 이름
2400              String saveFileName = genSaveFileName(extName);
2401
2402              System.out.println("originFilename : " + originFilename);
2403              System.out.println("extensionName : " + extName);
2404              System.out.println("size : " + size);
2405              System.out.println("saveFileName : " + saveFileName);
2406
2407              writeFile(multipartFile, saveFileName);
2408              uri = PREFIX_URL + saveFileName;
2409            }
2410            catch (IOException e) {
2411              // 원래라면 RuntimeException 을 상속받은 예외가 처리되어야 하지만
2412              // 편의상 RuntimeException을 던진다.
2413              // throw new FileUploadException();
2414              throw new RuntimeException(e);
2415            }
2416            return uri;
2417          }
2418
2419
2420          // 현재 시간을 기준으로 file 이름 생성
2421          private String genSaveFileName(String extName) {
2422            String fileName = "";
2423
2424            Calendar calendar = Calendar.getInstance();
2425            fileName += calendar.get(Calendar.YEAR);
2426            fileName += calendar.get(Calendar.MONTH);
2427            fileName += calendar.get(Calendar.DATE);
2428            fileName += calendar.get(Calendar.HOUR);
2429            fileName += calendar.get(Calendar.MINUTE);
2430            fileName += calendar.get(Calendar.SECOND);
2431            fileName += calendar.get(Calendar.MILLISECOND);
2432            fileName += extName;
2433
```

```
2434          return fileName;
2435        }
2436
2437
2438        // file을 실제로 write 하는 메서드
2439        private boolean writeFile(MultipartFile multipartFile, String saveFileName)
2440                    throws IOException{
2441          boolean result = false;
2442
2443          byte[] data = multipartFile.getBytes();
2444          FileOutputStream fos = new FileOutputStream(SAVE_PATH + "/" + saveFileName);
2445          fos.write(data);
2446          fos.close();
2447
2448          return result;
2449        }
2450      }
2451
2452    -SAVE_PATH는 file을 저장할 위치를 가리킨다.
2453      --일반적으로 server는 Linux 기반이므로 Linux 경로명을 사용하는 것이 좋다.
2454      --즉 file을 root 경로인 / 아래의 upload folder에 저장하겠다는 의미인데, Windows에서는 JVM이 알아서
          workspace가 존재하는 drive의 위치를 찾아서 drive를 root 경로로 하여 upload folder에 저장한다.
2455      --예를들어 Eclipse workspace가 C drive에 있다면 C drive의 upload folder에 file이 저장될 것이다.
2456    -PREFIX_URL은 저장된 file을 JSP에서 불러오기 위한 경로를 의미한다.
2457    -MultipartFile 객체는 file의 정보를 담고 있다.
2458    -uri을 반환하는 이유는 view page에서 바로 image file을 보기 위함이다.
2459      --만약 DB에서 image 경로를 저장 해야 한다면, 이와 같이 uri을 반환하면 좋을 것이다.
2460    -현재 시간을 기준으로 file 이름을 바꾼다.
2461      --이렇게 하는 이유는, 여러 사용자가 올린 file의 이름이 같을 경우 덮어 씌어지는 문제가 발생하기 때문이다.
2462      --따라서 file 이름이 중복될 수 있는 문제를 해결하기 위해 ms단위의 시스템 시간을 이용하여 file 이름을 변경한다.
2463    -FileOutputStream 객체를 이용하여 file을 저장한다.
2464
2465  17)Controller 작성
2466    -com.example.controller package
2467    -com.example.controller.FileUploadController.java
2468
2469      package com.example.controller;
2470
2471      import org.springframework.beans.factory.annotation.Autowired;
2472      import org.springframework.stereotype.Controller;
2473      import org.springframework.ui.Model;
2474      import org.springframework.web.bind.annotation.RequestMapping;
2475      import org.springframework.web.bind.annotation.RequestMethod;
2476      import org.springframework.web.bind.annotation.RequestParam;
2477      import org.springframework.web.multipart.MultipartFile;
2478
2479      import com.example.service.FileUploadService;
2480
2481      @Controller
2482      public class FileUploadController {
2483        @Autowired
2484        FileUploadService fileUploadService;
2485
2486        @RequestMapping("/form")
```

```
        public String form() {
          return "form.jsp";
        }

        @RequestMapping(value = "/upload", method = RequestMethod.POST)
        public String upload(@RequestParam("email") String email, @RequestParam("file1")
        MultipartFile file, Model model) {
          String uri = fileUploadService.restore(file);
          model.addAttribute("uri", uri);
          return "result.jsp";
        }
      }
```

18)WebContent/result.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>
  <h1>Upload completed</h1>
  <div class="result-images">
    <img src="${pageContext.request.contextPath }${uri }" style="width: 150px">
  </div>
  <p>
    <a href='/FileUploadDemo/form'> 다시 업로드 하기 </a>
  </p>
</body>
</html>
```

19)Project > right-click > Run As > Run on Server

http://localhost:8080/FileUploadDemo/form

20)문제점 및 해결

-upload folder를 보면 file이 upload된 것을 확인할 수 있지만, 결과 화면을 보면 image가 제대로 출력 되지 않을 것이다.

-엑박이 뜬 image file을 right-click하여 경로를 보면 아마 다음과 같을 것이다.

-http://localhost:8080/FileUploadDemo/upload/201822632335607.PNG

-File을 저장할 때 upload라는 folder에 저장을 했는데, file을 저장할 때의 upload는 C drive 내의 upload folder이고,

-위 URL에서 upload는 application 상 경로에 있는 upload이므로 WEB-INF 폴더의 하위 folder로서의 upload를 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
```

```
2629        // level of a logger programmatically. This is usually done
2630        // in configuration files.
2631        logger.setLevel(Level.INFO);
2632
2633        Logger barlogger = Logger.getLogger("com.foo.Bar");
2634
2635        // This request is enabled, because WARN >= INFO.
2636        logger.warn("Low fuel level.");
2637
2638        // This request is disabled, because DEBUG < INFO.
2639        logger.debug("Starting search for nearest gas station.");
2640
2641        // The logger instance barlogger, named "com.foo.Bar",
2642        // will inherit its level from the logger named
2643        // "com.foo" Thus, the following request is enabled
2644        // because INFO >= INFO.
2645        barlogger.info("Located nearest gas station.");
2646
2647        // This request is disabled, because DEBUG < INFO.
2648        barlogger.debug("Exiting gas station search");
2649
2650    7)Log4j Pattern Option
2651      -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
2652      -***로 표시된 항목은 실행속도에 영향이 있음
2653      -%p: debug, info, warn, error, fatal 등의 priority 출력
2654      -%m : log내용 출력
2655      -%d *** : logging event가 발생한 시간을 출력, ex)포맷은 %d{HH:mm:ss} 같은 형태의
          SimpleDateFormat
2656      -%t : log event가 발생된 thread의 이름 출력
2657      -%F ***: logging이 발생한 program file명 출력
2658      -%l ***: logging이 발생한 caller의 정보 출력
2659      -%L ***: logging이 발생한 caller의 line수 출력
2660      -%M ***: logging이 발생한 method 이름 출력
2661      -% : % 표시 출력
2662      -%n : platform 종속적인 개행문자 출력
2663      -%c : category 출력, ex)Category가 a.b.c 처럼 되어있다면 %c{2}는 b.c 출력
2664      -%C ***: class명 출력, ex)Class구조가 org.apache.xyz.SomeClass 처럼 되어있다면 %C{2}는
          xyz.SomeClass 출력
2665      -%r : Application 시작 이후 부터 logging이 발생한 시점의 시간(milliseconds) 출력
2666      -%x : logging이 발생한 thread와 관련된 NDC(nested diagnostic context) 출력
2667      -%X : logging이 발생한 thread와 관련된 MDC(mapped diagnostic context) 출력
2668
2669      -For example, the PatternLayout with the conversion pattern
2670        "%r [%t] %-5p %c - %m%n" will output something akin to:
2671
2672        176 [main] INFO  org.foo.Bar - Located nearest gas station.
2673
2674      -The first field is the number of milliseconds elapsed since the start of the program.
2675      -The second field is the thread making the log request.
2676      -The third field is the level of the log statement.
2677      -The fourth field is the name of the logger associated with the log request.
2678      -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 {

   // Define a static logger variable so that it references the
   // Logger instance named "MyApp".
   static Logger logger = Logger.getLogger(MyApp.class);

   public static void main(String[] args) {

      // Set up a simple configuration that logs on the console.
      BasicConfigurator.configure();
```

```
2733            logger.info("Entering application.");
2734            Bar bar = new Bar();
2735            bar.doIt();
2736            logger.info("Exiting application.");
2737          }
2738        }
2739
2740        --MyApp begins by importing log4j related classes.
2741        --It then defines a static logger variable with the name MyApp which happens to be the
            fully qualified name of the class.
2742        --MyApp uses the Bar class defined in the package com.foo.
2743
2744          package com.foo;
2745          import org.apache.log4j.Logger;
2746
2747          public class Bar {
2748            static Logger logger = Logger.getLogger(Bar.class);
2749
2750            public void doIt() {
2751              logger.debug("Did it again!");
2752            }
2753          }
2754
2755        --The invocation of the BasicConfigurator.configure method creates a rather simple log4j
            setup.
2756        --This method is hardwired to add to the root logger a ConsoleAppender.
2757        --The output will be formatted using a PatternLayout set to the pattern
2758          "%-4r [%t] %-5p %c %x - %m%n".
2759
2760        --Note that by default, the root logger is assigned to Level.DEBUG.
2761        --The output of MyApp is:
2762
2763          0    [main] INFO  MyApp  - Entering application.
2764          36   [main] DEBUG com.foo.Bar  - Did it again!
2765          51   [main] INFO  MyApp  - Exiting application.
2766
2767    10)Default Initialization under Tomcat
2768      -Define the following servlet in the web.xml file for your web-application.
2769
2770        <servlet>
2771          <servlet-name>log4j-init</servlet-name>
2772          <servlet-class>com.foo.Log4jInit</servlet-class>
2773          <init-param>
2774            <param-name>log4j-init-file</param-name>
2775            <param-value>WEB-INF/classes/log4j.lcf</param-value>
2776          </init-param>
2777          <load-on-startup>1</load-on-startup>
2778        </servlet>
2779
2780      -It is also possible to use a special servlet for log4j initialization. Here is an example,
2781
2782        package com.foo;
2783
2784        import org.apache.log4j.PropertyConfigurator;
```

```
2785        import javax.servlet.http.HttpServlet;
2786        import javax.servlet.http.HttpServletRequest;
2787        import javax.servlet.http.HttpServletResponse;
2788        import java.io.PrintWriter;
2789        import java.io.IOException;
2790
2791        public class Log4jInit extends HttpServlet {
2792
2793          public void init() {
2794            String prefix =  getServletContext().getRealPath("/");
2795            String file = getInitParameter("log4j-init-file");
2796            // if the log4j-init-file is not set, then no point in trying
2797            if(file != null) {
2798              PropertyConfigurator.configure(prefix+file);
2799            }
2800          }
2801
2802          public void doGet(HttpServletRequest req, HttpServletResponse res) {
2803
2804          }
2805        }
2806
2807    11)Semi-Lab
2808      -Spring Legacy Project > Spring MVC > com.example.biz
2809      -pom.xml에서 slf4j에 관한 library 제거 : slf4j-api, jcl-over-slf4j, slf4j-log4j12
2810      -log4j version을 1.2.17
2811
2812        <properties>
2813          <java-version>1.8</java-version>
2814          <org.springframework-version>4.3.20.RELEASE</org.springframework-version>
2815          <org.aspectj-version>1.9.2</org.aspectj-version>
2816        </properties>
2817          ...
2818        <!-- Spring -->
2819        <dependency>
2820          <groupId>org.springframework</groupId>
2821          <artifactId>spring-context</artifactId>
2822          <version>${org.springframework-version}</version>
2823
2824        -아래 <exclusions> 제거할 것
2825          <exclusions>
2826            <!-- Exclude Commons Logging in favor of SLF4j -->
2827            <exclusion>
2828              <groupId>commons-logging</groupId>
2829              <artifactId>commons-logging</artifactId>
2830             </exclusion>
2831          </exclusions>
2832        </dependency>
2833
2834        <!-- Logging -->
2835        <dependency>
2836          <groupId>log4j</groupId>
2837          <artifactId>log4j</artifactId>
2838          <version>1.2.17</version>
```

```
        <scope>runtime</scope>
      </dependency>

    -HomeController.java에서
      --다음 code를 제거
        private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

      --대신 다음 code 추가
        protected Log log = LogFactory.getLog(HomeController.class);

        @RequestMapping(value = "/", method = RequestMethod.GET)
        public String home(Locale locale, Model model) {

          log.info("Logging 시작하기");   <-- 추가

    -기본 프로젝트에 있는 log4j.xml이다.
    -서버를 실행 시키면 콘솔창에 빨간 글씨들 사이에 하얀 글씨들이 보일 것이다.

        INFO : com.example.biz.HomeController - Logging 시작하기

    -그게 바로 여기서 설정한 로그 메시지들이다.
    -Root Logger의 레벨이 warn 이고, Appender Logger들의 레벨은 info다.
    -전부 warn레벨을 상속받지만, 현재 info 레벨이 적용된다.
    -그리고 ConsoleAppender에 로그 메시지를 전송한다.
    -현재는 이렇게 찍히지만, 이제 우리 입맛에 맞게, 로그의 종류에 따라 다르게 출력되게 logj4.xml을 변경할 것이다.

    -log4j.xml 수정

      <?xml version="1.0" encoding="UTF-8"?>
      <!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd">
      <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

        <!-- Appenders -->
        <appender name="console" class="org.apache.log4j.ConsoleAppender">
          <param name="Target" value="System.out" />
          <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d %5p [%c] %m%n" />
          </layout>
        </appender>

        <!-- *추가* -->
        <appender name="console-infolog" class="org.apache.log4j.ConsoleAppender">
          <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d %5p %m%n"/>
          </layout>
        </appender>
        <!--  -->

        <!-- Application Loggers -->
        <logger name="com" additivity="false">
          <level value="debug" />
          <appender-ref ref="console"/>
        </logger>
```

```
2893          <!-- 3rdparty Loggers -->
2894          <logger name="org.springframework.core">
2895             <level value="info" />
2896          </logger>
2897
2898          <logger name="org.springframework.beans">
2899             <level value="info" />
2900          </logger>
2901
2902          <logger name="org.springframework.context">
2903             <level value="info" />
2904          </logger>
2905
2906          <logger name="org.springframework.web">
2907             <level value="info" />
2908          </logger>
2909
2910          <!-- Root Logger -->
2911          <root>
2912             <priority value="off" />
2913             <appender-ref ref="console" />
2914          </root>
2915
2916       </log4j:configuration>
2917
2918    -2가지의 Appender ( console, console-infolog )와 기존의 4가지 Logger와 추가된 Logger가 있다.
2919    -기존의 4가지 Logger(28~42행, <!-- 3rdparty Loggers --> ~ <!-- Root Logger --> 전까지)는 삭제해
       줘도 된다.
2920    -Logger마다 메세지를 전달하는 Appender가 다르니 잘 확인하자.
2921    -그리고 상속을 하지 않는(Additivity=false) 이유는 로그가 2번 출력되서 보통 상속하지 않는다.
2922    -서버를 돌리면 기존 4가지 Logger를 지우지 않았다면 그대로 출력되고, 지웠다면 빨간 글씨만 출력될 것이다. log4j의
       설정 끝!
2923
2924
2925 23. Logging
2926   1)Log
2927    -Log는 기록을 남기는 것을 의미한다.
2928    -구체적으로는 program 개발이나 운영시 발생하는 문제점을 추적 하거나 운영 상태를 monitoring 하는 정보를 기록
       하는 것.
2929    -또한 분석을 통해 통계를 낼 수도 있기 때문에 기록을 남기는 것은 중요하다고 할 수 있다.
2930    -하지만 log를 남기면 성능이 나빠진다는 단점이 있는데, 그보다 log를 통해 얻는 정보가 훨씬 많기 때문에 필요한 부분
       에 file로써 log를 남기는 것이 중요하다.
2931
2932   2)Logback
2933    -Spring에서는 기본적으로 commons.logging libarary ( Apache의 JCL, Jakarta Commons Logging )
       을 사용한다.
2934    -즉 Spring 개발을 할 때 Spring이 뿜어내는 message는 JCL에 의존하여 log를 남기는 것이다.
2935    -실제로 spring-context library를 설치할 때 project folder의 Maven Libarary를 확인해보면
       commons.logging library(commons-logging-1.2.jar)를 확인할 수 있다.
2936    -예전에는 Spring에서 log를 남길 때 Log4J를 사용했었는데, 성능 및 기능상의 이유로 대체 logger들이 많아졌고, 현
       재 대부분은 SLF4J interface를 구현한 Logback을 사용한다.
2937    -Spring이 기존에 사용하던 log libtrary JCL 대신, 새로운 library Logback을 사용하도록 하기 위해서는 SLF4J
       가 필요하다.
2938    -즉 SLF4J는 JCL과 Log4J의 징검다리 역할을 한다고 보면 된다.
```

```
2939        Log4J <-- jcl-over-slf4j --> SLF4J <-- logback-classic --> Logback
2940
2941    3)환경 설정
2942      -Spring Legacy Project > LogDemo > Spring MVC Project > com.example.biz >
2943      -pom.xml
2944
2945        <properties>
2946          <java-version>1.8</java-version>
2947          <org.springframework-version>4.3.20.RELEASE</org.springframework-version>
2948          <org.aspectj-version>1.9.2</org.aspectj-version>
2949          <org.slf4j-version>1.7.25</org.slf4j-version>
2950        </properties>
2951        <dependencies>
2952          <!-- Spring core -->
2953          <dependency>
2954          <groupId>org.springframework</groupId>
2955          <artifactId>spring-context</artifactId>
2956          <version>${org.springframework-version}</version>
2957
2958          <!-- JCL 제외 -->
2959          <exclusions>
2960            <exclusion>
2961              <groupId>commons-logging</groupId>
2962              <artifactId>commons-logging</artifactId>
2963            </exclusion>
2964          </exclusions>
2965          </dependency>
2966
2967          <!-- Logging -->
2968          <dependency>
2969            <groupId>org.slf4j</groupId>
2970            <artifactId>slf4j-api</artifactId>
2971            <version>${org.slf4j-version}</version>
2972          </dependency>
2973          <dependency>
2974            <groupId>org.slf4j</groupId>
2975            <artifactId>jcl-over-slf4j</artifactId>
2976            <version>${org.slf4j-version}</version>
2977            <scope>runtime</scope>
2978          </dependency>
2979          <dependency>
2980            <groupId>org.slf4j</groupId>
2981            <artifactId>slf4j-log4j12</artifactId>
2982            <version>${org.slf4j-version}</version>
2983            <scope>runtime</scope>
2984          </dependency>
2985
2986        -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.
2987        -JCL을 제외시켰기 때문에 기존에 JCL을 통해 log를 남기던 code들은 error를 발생 시킬 것이다.
2988        -그래서 필요한 것이 jcl-over-slf4j library이며, 일종의 다리 역할을 하는 것이다.
2989        -실제로는 SLF4J을 구현한 logback-classic library가 log를 남기게 된다.
2990
```

4)/src/main/resources/logback.xml
-Log를 어떻게 남길지에 대한 설정 file이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <!-- 콘솔로 로그를 남김 -->
  <appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <charset>UTF-8</charset>
      <!-- 로그 메시지 pattern -->
      <Pattern>
        %d{HH:mm:ss.SSS} [%thread] %-5level %logger{5} - %msg%n
      </Pattern>
    </encoder>
  </appender>

  <!-- file로 로그를 남김 -->
  <appender name="fileAppender"
  class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>c:\LogExample\logexample2.log</file>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <Pattern>
           %d{HH:mm:ss.SSS} [%thread] %-5level %logger{5} - %msg%n
      </Pattern>
    </encoder>

    <!-- 로그를 남기는 file의 용량이 50KB가 넘으면 이를 압축 file로 만들고 새로 로그 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 패키지 로그들만  consoleAppender, fileAppender 방법으로 로
      그를 남긴다.
      물론 <appender-ref ref="consoleAppender" />를 추가하여 콘솔로도 로그를 남길 수 있다.
    -->
    <logger name="com.victolee.logExample"  level="info" additivity="false">
      <appender-ref ref="fileAppender" />
    </logger>
```

```
        <!-- root는 글로벌 로거를 의미하며, 위의 logger에 해당하지 않으면 root 로거가 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 제한한 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;
    import org.springframework.web.bind.annotation.ResponseBody;

    @Controller
    public class LogController {
      private static final Log LOG = LogFactory.getLog(LogController.class);

      @RequestMapping("/log")
      @ResponseBody
      public String logExam() {
        LOG.debug("#ex1 - debug log");
        LOG.info("#ex1 - info log");
        LOG.warn("#ex1 - warn log");
        LOG.error("#ex1 - error log");
        return "콘솔 또는 file경로 확인";
      }
    }
```

-com.example.controller 패키지를 생성한 이유는 위의 설정 file의 <logger>에서 범위를 지정했기 때문이다.
-그리고 file로 log를 남기겠다고 했으므로 요청을 했을 시 C:\LogExample\logexample2.log 경로에 file이 작성되는지 확인해본다.