10강 MVC 1

프로젝트 준비

- spring legacy project로 프로젝트를 생성한다.
- com.green.ex00

- POM.xml에 다음 내용을 채운다.

프로젝트 준비

- 기존의 프로젝트에서 다음 코드를 복사한다.

- spring.exception
    - AlreadyExistingMemberException.java
    - IdPasswordNotMatchingException.java
    - MemberNotFoundException.java
- spring.vo
    - Member.java
    - RegisterRequest.java
- spring.dao
    - MemberDao.java
-spring.service
    - MemberRegisterService.java
    - ChangePasswordService.java

프로젝트 준비

- 데이터베이스 연결을 준비한다. (기존의 데이터베이스를 사용한다.)
- spring_member.xml을 작성한다.

```
<tx:annotation-driven transaction-manager="transactionManager"/>

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
    <property name="driverClass" value="oracle.jdbc.OracleDriver"/>
    <property name="jdbcUrl" value="jdbc:oracle:thin:@db.interstander.com:41521:XE"/>
    <property name="user" value="green9999"/>
    <property name="password" value="1234"/>
    <property name="maxPoolSize" value="30"/>
</bean>

<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>

<bean id="memberDao" class="spring.dao.MemberDao">
    <constructor-arg ref="dataSource"/>
</bean>

<bean id="changePwdSvc" class="spring.service.ChangePasswordService">
    <constructor-arg ref="memberDao"/>
</bean>
<bean id="memberRegSvc" class="spring.service.MemberRegisterService">
    <constructor-arg ref="memberDao"/>
</bean>
```

프로젝트 준비

- 데이터베이스 연결을 준비한다. (기존의 데이터베이스를 사용한다.)
- spring-mvc.xml을 작성한다.

```
<mvc:annotation-driven/>

<mvc:default-servlet-handler/>

<mvc:view-resolvers>
    <mvc:jsp prefix="/WEB-INF/views/"/>
</mvc:view-resolvers>
```

프로젝트 준비

- 데이터베이스 연결을 준비한다. (기존의 데이터베이스를 사용한다.)
- web.xml을 작성한다.

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>spring.profiles.active</param-name>
        <param-value>test</param-value>
    </init-param>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            classpath:spring-mvc.xml
            classpath:spring-member.xml
            classpath:spring-controller.xml
        </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

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

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

<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

@RequestMapping을 이용한 경로 매핑

- 웹 어플리케이션을 개발한다는 것은 결국 다음을 위한 코드를 작성하는 것과 같다.

    1) 특정 요청의 URL을 처리할 코드 작성
    2) 처리 결과를 HTML과 같은 형식으로 응답

1)의 경우 @Controller 애노테이션을 사용한 컨트롤러 클래스를 이용해서 구한다.

컨트롤러 클래스는 @RequestMapping애노테이션을 사용해서 메서드가 처리할 요청 경로를 지정한다.

@RequestMapping을 이용한 경로 매핑

- 예제 1
- spring.controller.RegisterController.java

```java
@Controller
public class RegisterController {

    @RequestMapping("/register/step1")
    public String handlerStep1() {
        return "register/step1";    // WEB-INF/views/register/step1.jsp
    }
```

@RequestMapping을 이용한 경로 매핑

- 예제 1
- views/register/step1.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원가입</title>
</head>
<body>
<h2>약관</h2>
<p>약관 내용~~</p>
<form action="step2" method="POST">
    <label>
        <input type="checkbox" name="agree" value="true">약관 동의
    </label>

    <input type="submit" value="다음 단계">
</form>
</body>
</html>
```

@RequestMapping을 이용한 경로 매핑

- 예제 1
- spring-controller.xml

```
<bean id="registerController" class="spring.controller.RegisterController" >
</bean>
```

GET과 POST 구분

- HTML 폼 태그의 전송방식은 Get방식과 Post 방식으로 구분된다.

- 스프링 MVC는 get과 Post 구분없이 처리를 한다.
- 그러나 특정 방식의 요청을 처리하고자 한다면 @RequestMapping 애노테이션을 사용해서 전송방식을 한정할 수 있다..

예제 2

- RegisterController.java

```
@RequestMapping(value = "/register/step2", method = RequestMethod.POST)
public String handleStep2(){
    return "register/step2";
}
```

요청 파라미터 접근

- 클라이언트가 요청을 할 때는 단순히 경로만으로 요청하는 경우도 있지만 어떤 처리를 목적으로 데이터를 전달하는 경우도 있다.
- 이런 데이터를 요청 파라미터라고 하는데 컨트롤러는 이러한 파라미터는 얻어와서 처리해야 한다.

- 이때 파라미터를 얻어오는 방법는 두가지 있다

1) httpServletRequest 객체에서 직접 얻어오는 방법
2) @RequestParam 애노테이션은 사용하는 방법

요청 파라미터 접근

예제 3 : httpServletRequest 객체에서 직접 얻어오는 방법

```
@RequestMapping(value = "/register/step2", method = RequestMethod.POST)
public String handleStep2(HttpServletRequest request) {
    String agreeParam = request.getParameter("agree");
    if(agreeParam == null || !agreeParam.equals("true")) {
        return "register/step1";
    }
    return "register/step2";
}
```

예제 3-2 : @RequestParam 애노테이션은 사용하는 방법

```
@RequestMapping(value="/register/step2", method=RequestMethod.POST)
public String handlerStep2(
        @RequestParam(value="agree", defaultValue="false")Boolean agree) {
    if(!agree) {
        return "register/step1";
    }

    return "register/step2";
}
```

요청 파라미터 접근

- @RequestParam(value = "agree", defaultValue = "false") Boolean agreeVal

- agree 요청 파라미터의 값을 읽어와서 agreeVal 매개변수에 할당한다.
- defaultValue에 의해서 요청 파라미터가 없으면 false로 처리한다.

- 파라미터를 읽어와서 자동으로 기본 타입의 형변환을 지원한다.

속성	타입	설명
value	String	HTTP 요청 파라미터의 이름을 지정한다.
required	Boolean	필수 여부를 지정한다. 기본값은 true이며 true일 경우 해당 요청 파라미터에 값이 없으면 스프링 MVC는 예외를 발생시킨다.
defaultValue	String	요청 파라미터의 값이 없을 경우 사용할 문자열 값을 지정한다.

요청 파라미터 접근

예제 4 : @RequestParam 애노테이션은 사용하는 방법

```
@RequestMapping(value="/register/step2", method=RequestMethod.POST)
public String handlerStep2(
        @RequestParam(value="agree", defaultValue="false")Boolean agree) {
    if(!agree) {
        return "register/step1";
    }

    return "register/step2";
}
```

요청 파라미터 접근

예제 5 : step2.jsp

```
<title>회원가입</title>
</head>
<body>
<h2>회원 정보 입력</h2>
    <form action="step3" method="post">
        <p>
            <label>이메일:<br>
            <input type="text" name="email" id="email" >
            </label>
        </p>
        <p>
            <label>이름:<br>
            <input type="text" name="name" id="name">
            </label>
        </p>
        <p>
            <label>비밀번호:<br>
            <input type="password" name="password" id="password">
            </label>
        </p>
        <p>
            <label>비밀번호 확인:<br>
            <input type="password" name="confirmPassword" id="confirmPassword">
            </label>
        </p>
        <input type="submit" value="가입 완료">
    </form>

</body>
```

@RequestMapping을 이용한 경로 매핑

- HelloController 클래스의 @RequestMapping 애노테이션 적용 메서드를 한 개만 정의 했지만 @RequestMapping 애노테이션 적용 메소드를 두개 이상 정의 할 수도 있다.

```
@Controller
public class RegistController{
    @RequestMapping("/register/step1"){
    public String handlerStep1(){
        return "register/step1";
    }
    @RequestMapping("/register/step2"){
    public String handlerStep2(){
        return "register/step2";
    }

....
}
```

```
@Controller
@RequestMapping("/register")
public class RegistController{
    @RequestMapping("/step1"){
    public String handlerStep1(){
        return "register/step1";
    }
    @RequestMapping("/step2"){
    public String handlerStep2(){
        return "register/step2";
    }

....
```

- 공통의 경로를 클래스에 축약할 수 있다 => 오른쪽 의 경로도 /register/step1 로 정해진다.

@RequestMapping을 이용한 경로 매핑

- 공통의 경로를 클래스에 축약할 수 있다 => 오른쪽 의 경로도 /register/step1 로 정해진다.

```
@Controller
@RequestMapping("/register")
public class RegisterController {
```

```
//@RequestMapping("/register/step1") //  =>   /register/step1
@RequestMapping("/step1")
public String handlerStep1() {
```

```
//  @RequestMapping(value = "/register/step2", method = RequestMethod.POST)
    @RequestMapping(value="/step2", method=RequestMethod.POST)   //  =>   /register/step2
    public String handlerStep2(
```

리다이렉트 처리

- 서버를 구동하고 주소창에 직접/register/step2를 입력하면 404에러 페이지를 보여준다.
- 이유는 주소창에 직접 요청하는 방식이 Get 방식이기 때문이다.

- 이렇듯 잘못된 전송방식으로 요청이 왔을 때 알맞은 경로로 리다이렉트 하는 것이 좋다.

- 이때 @RequestMapping적용 메서드가 redirect: 로 시작되는 경로를 반환하면 나머지 경로를 이용해서 자동으로 리다이렉트 할 경로를 구한다.

리다이렉트 처리

- 예제 5

```
@RequestMapping(value="/step2", method=RequestMethod.GET)
public String handlerStep2Get() {
    return "/register/step1";// 이 방식의 단점 : 이렇게 보내면 주소창은 정상적으로 표현, 보여지는 페이지는 step1
}
```

```
@RequestMapping(value="/step2", method=RequestMethod.GET)
public String handlerStep2Get() {
    //return "/register/step1";// 이 방식의 단점 : 이렇게 보내면 주소창은 정상적으로 표현, 보여지는 페이지는 step1
    return "redirect:register/step1";    // register/step1 매핑 시작
}
```

```
@RequestMapping(value="/step2", method=RequestMethod.GET)
public String handlerStep2Get() {
    //return "/register/step1";// 이 방식의 단점 : 이렇게 보내면 주소창은 정상적으로 표현, 보여지는 페이지는 step1
    //return "redirect:register/step1"; // register/step1 매핑 시작
    // 전체 주소를 작성할 수도 있다.
    return "redirect:http://localhost:8090/ex00/register/step1";
}
```

커맨드 객체를 이용해서 요청 파라미터 사용하기

- 앞서 본 예제에서는 하나의 요청 파라미터를 가져오는 내용을 살펴 보았다.
- 일반적인 방법으로 여러 데이터를 읽어오기
- 예제 6

```java
@RequestMapping("/step3")    //  =>   /register/step3
public String handlerStep3(HttpServletRequest request) {
    String email = request.getParameter("email");
    String name = request.getParameter("name");
    String password = request.getParameter("password");
    String confirmPassword = request.getParameter("confirmPassword");

    //~~~~~~~~
    return "register/step3";
}
```

```java
@RequestMapping("/step3")    //  =>   /register/step3
public String handlerStep3(
        @RequestParam(value="email")String email,
        @RequestParam(value="name")String name,
        @RequestParam(value="password")String password,
        @RequestParam(value="confirmPassword")String confirmPassword
        ) {
    //~~~~~~~~
    return "register/step3";
}
```

커맨드 객체를 이용해서 요청 파라미터 사용하기

- 그러나 요청 파라미터의 개수가 늘어나면 늘어날 수록 코드의 복잡도는 점점 늘어난다.
- 이런 불편함을 줄이고자 스프링은 커맨드 객체를 사용한다.
- 커맨드 객체는 파라미터를 전달하는 기능을 가지므로 set메소드가 존재하는 클래스 여야 한다.
- 예제 7

```
@RequestMapping(value="/step3", method=RequestMethod.POST)
public String handlerStep3(RegisterRequest regReq) {

    //~~~~~~~
    return "register/step3";
}
```

커맨드 객체를 이용해서 요청 파라미터 사용하기

- 커맨드 객체를 활용해서 실제 회원 가입하는 코드를 작성해보자

1) 서비스 객체를 주입받아서 사용하도록 처리한다.

```
private MemberRegisterService memberRegisterService;

public void setMemberRegisterService(MemberRegisterService memberRegisterService) {
    this.memberRegisterService = memberRegisterService;
}
```

```
@RequestMapping(value="/register/step3", method=RequestMethod.POST)
public String handlerStep3(RegisterRequest regReq) {
    try {
        memberRegisterService.regist(regReq);
        return "register/step3";
    }catch(AlreadyExistingMemberException e) {
        return "register/step2";
    }
}
```

커맨드 객체를 이용해서 요청 파라미터 사용하기

- 커맨드 객체를 활용해서 실제 회원 가입하는 코드를 작성해보자

2) 객체 주입을 위한 설정 : spring-controller.xml

```
<bean class="spring.controller.RegisterController" >
    <property name="memberRegisterService" ref="memberRegSvc" />
</bean>
```

커맨드 객체를 이용해서 요청 파라미터 사용하기

- 커맨드 객체를 활용해서 실제 회원 가입하는 코드를 작성해보자

3) 처리 후 이동할 페이지 작성 : step3.jsp

```
<title>회원 가입</title>
</head>
<body>
    <p>회원 가입을 완료했습니다.</p>
    <p><a href="<c:url value='/main'/>">[첫 화면 이동]</a></p>
</body>
```

뷰 JSP 코드에서 커맨드 객체 사용하기

- 가입 완료 화면에서 가입할 때 사용한 이메일 주소와 이름을 보여주면 좋을 것이다.
- 이때 커맨드 객체를 사용해서 JSP를 통해 정보를 표시할 수 있다.
- 예제 7

```
@RequestMapping(value = "/register/step3", method = RequestMethod.POST)
        public String handleStep3(RegisterRequest regReq) { ...}
```

```
<p><strong>${registerRequest.name}님</strong>
                회원 가입을 완료했습니다.</p>
```

```
<body>
    <%-- <p>회원 가입을 완료했습니다.</p>
    <p><a href="<c:url value='/main'/>">[첫 화면 이동]</a></p> --%>
    <p><strong>${registerRequest.name}</strong>님</strong>회원 가입을 완료했습니다.</p>
    <p><a href="<c:url value='/main'/>">[첫 화면 이동]</a></p>
</body>
```

뷰 JSP 코드에서 커맨드 객체 사용하기

- 커맨드 객체의 속성명을 변경하고 싶다면 @ModelAttribute 애노테이션을 사용하면 된다.
  예제 8

```
@RequestMapping(value = "/register/step3", method = RequestMethod.POST)
public String handleStep3(@ModelAttribute("formData")RegisterRequest regReq) {...}
```

```
<p><strong>${formData.name}님</strong>
                    회원 가입을 완료했습니다.</p>
```

커맨드 객체와 스프링 폼 연동

- 회원 정보 입력 폼에서 중복된 이메일을 입력할 경우 회원 정보를 입력하면 안되고 다시 정보를 요구해야 한다.

- 이때 앞서 본 예제는 기존의 입력 내용이 전부 사라지고 텅 빈 폼을 보게 되는데 이때 기존에 입력 내용을 커맨드 객체의 값을 보여 줄 수 있다.

커맨드 객체와 스프링 폼 연동

- 예제 8

```
<form action="step3" method="POST">
<p>
    <label>이메일:<br>
        <input type="text" name="email" id="email" value="${formData.email}">
    </label>
</p>
<p>
    <label>이름:<br>
        <input type="text" name="name" id="name" value="${formData.name}">
    </label>
</p>
<p>
    <label>비밀번호:<br>
        <input type="password" name="password" id="password">
    </label>
</p>
<p>
    <label>비밀번호 확인:<br>
        <input type="password" name="confirmPassword" id="confirmPassword">
    </label>
</p>
<input type="submit" value="가입완료">
</form>
```

커맨드 객체와 스프링 폼 연동

- 커맨드 객체를 사용하는 경우 스프링 MVC가 제공하는 커스텀 태그를 사용하면 좀 더 간편하게 커맨드 객체의 값을 출력할 수 있다.

- 스프링 MVC가 제공하는 커스텀 태그

  <form:form>,<form:input>,<form:password>

커맨드 객체와 스프링 폼 연동

- 예제 9

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html>
<html>
```

```
<h2>회원 정보 입력</h2>
<form:form action="step3" commandName="formData">
    <p>
        <label>이메일 : <br>
            <form:input path="email" />
            <%-- <input type="text" name="email" id="email" value="${formData.email}"> --%>
        </label>
    </p>
    <p>
        <label>이름 : <br>
            <form:input path="name" />
        </label>
    </p>
    <p>
        <label>비밀번호 : <br>
            <form:password path="password" />
        </label>
    </p>
    <p>
        <label>비밀번호 확인 : <br>
            <form:password path="confirmPassword" />
        </label>
    </p>
    <input type="submit" value="가입완료">
</form:form>
```

커맨드 객체와 스프링 폼 연동

- <form:form>태그는 form을 생성한다.
  - action : form 태그의 action과 같은 역할 => submit 클릭시 이동할 페이지
  - commandName : 커맨드 객체 속성 이름을 지정한다. 미설정시 command로 기본값 사용

- <form:input>태그는 input 태그를 생성한다.
  - <form:input path="name" /> 의 경우 커맨드 객체의 name 프로퍼티 값을 value에 속성으로 사용한다.
  - 예를 들어 커맨드 객체에서 name = "홍길동"이라고 한다면

  <input id="name" name="name" type="text" value="홍길동">이 생성된다.

커맨드 객체와 스프링 폼 연동

- <form:form>태그를 사용하려면 반드시 커맨드 객체가 필요하다.
  - step1에서 step2로 넘어오는 단계에서 커맨드 객체를 모델에 담아 주어야 한다.
  - 예제 10

```
@RequestMapping(value="/register/step2", method=RequestMethod.POST)   //  =>   /register/step2
public String handlerStep2(
        @RequestParam(value="agree", defaultValue="false")Boolean agree,
        Model model) {
    if(!agree) {
        return "register/step1";
    }

    model.addAttribute("formData",new RegisterRequest());
    return "register/step2";
}
```

컨트롤러 구현 없는 경로 매핑

- 컨트롤러 중에서는 별도의 처리없이 단순히 뷰 이름만 반환하는 컨트롤러의 경우

예제 11

```java
@Controller
public class MainController {

    @RequestMapping("/main")
    public String main() {
        return "main";
    }
}
```

```xml
<bean class="spring.controller.MainController"/>
```

컨트롤러 구현 없는 경로 매핑

- 컨트롤러 중에서는 별도의 처리없이 단순히 뷰 이름만 반환하는 컨트롤러의 경우

특별한 로직도 없는 컨트롤러를 만드는 성가심을 해소하기 위해 스프링 MVC는 <mvc:view-controller>태그를 제공한다.

예제 12

```
<mvc:view-controller path="/main" view-name="main"/>
```

컨트롤러 구현 없는 경로 매핑

예제 13 : main.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>메인</title>
</head>
<body>
    <p>환영합니다</p>
    <p><a href="<c:url value='/register/step1'/>">[회원 가입하기]</a></p>
</body>
</html>
```

커맨드 객체 : 중첩, 콜렉션 프로퍼티

- 예 : 간단한 설문 조사를 한다, 3가지 설문항목이 있고, 응답자의 지역과 나이를 입력받아야 한다고 가정한다.

⇒관련 클래스를 제작한다 :예제 12 : spring.survey.Respondent.java

```
public class Respondent {

    private int age;
    private String location;

    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getLocation() {
        return location;
    }
    public void setLocation(String location) {
        this.location = location;
    }
```

커맨드 객체 : 중첩, 콜렉션 프로퍼티

- 예 : 간단한 설문 조사를 한다, 3가지 설문항목이 있고, 응답자의 지역과 나이를 입력받아야 한다고 가정한다.

⇒관련 클래스를 제작한다 :예제 12 : spring.survey.AnsweredData.java

```
public class AnsweredData {

    private List<String> responses;
    private Respondent res;

    public List<String> getResponses() {
        return responses;
    }
    public void setResponses(List<String> responses) {
        this.responses = responses;
    }
    public Respondent getRes() {
        return res;
    }
    public void setRes(Respondent res) {
        this.res = res;
    }
}
```

커맨드 객체 : 중첩, 콜렉션 프로퍼티

- 위 클래스는 이전 커맨드 객체와 2가지 다른 점이 존재한다.

-> 리스트타입의 프로퍼티

-> 중첩 프로퍼티

커맨드 객체 : 중첩, 콜렉션 프로퍼티

- 스프링은 위와 같은 프로퍼티를 가진 경우에도 요청 파라미터의 값을 알맞게 설정해준다.

-> HTTP 요청 파라미터 이름이 “프로퍼티이름[인덱스]”형식이면 List 타입의 프로퍼티 값으로

-> HTTP 요청 파라미터 이름이 “프로퍼티이름.프로퍼티이름”형식으면 중첩 프로퍼티 값으로

커맨드 객체 : 중첩, 콜렉션 프로퍼티

예제 13-1 : 컨트롤러 SurveyController.java

```java
@Controller
@RequestMapping("/survey")
public class SurveyController {

    @RequestMapping(method= RequestMethod.GET)
    public String form() {
        return "survey/surveyForm";
    }

    @RequestMapping(method= RequestMethod.POST)
    public String submit(@ModelAttribute("ansData") AnsweredData data) {
        return "survey/submitted";
    }
```

커맨드 객체 : 중첩, 콜렉션 프로퍼티

예제 13-2 : 빈 등록

```
<bean class="spring.controller.SurveyController" />
```

커맨드 객체 : 중첩, 콜렉션 프로퍼티

예제 13-3 : 매핑에 대응되는 jsp 생성 : surveyForm.jsp

```
<h2>설문조사</h2>
<form method="POST">
        <p>
            1. 당신의 역할은?<br>
            <label><input type="radio" name="responses[0]" value="서버">서버 개발자</label>
            <label><input type="radio" name="responses[0]" value="프론트">프론트 개발자</label>
            <label><input type="radio" name="responses[0]" value="풀스택">풀스택 개발자</label>
        </p>
        <P>
            2. 가장 많이 사용하는 개발 도구는? <br>
            <label><input type="radio" name="responses[1]" value="Eclipse">Eclipse</label>
            <label><input type="radio" name="responses[1]" value="IntelliJ">IntelliJ</label>
            <label><input type="radio" name="responses[1]" value="NetBeans">NetBeans</label>
        </P>
        <p>
            3. 하고 싶은 말<br>
            <input type="text" name="responses[2]">
        </p>
        <p>
            <label>응답자 위치:
                <input type="text" name="res.location">
            </label>
        </p>
        <p>
            <label>응답자 나이:
                <input type="text" name="res.age">
            </label>
        </p>
        <input type="submit" value="전송">

    </form>
```

커맨드 객체 : 중첩, 콜렉션 프로퍼티

예제 13-4 : 매핑에 대응되는 jsp 생성 : submitted.jsp

```
<p>응답 내용</p>
<ul>
    <c:forEach var="response" items="${ansData.responses}" varStatus="status">
        <li>${status.index+1}번 문항 : ${response}</li>
    </c:forEach>
</ul>

<p>응답자 위치 : ${ansData.res.location}</p>
<p>응답자 나이 : ${ansData.res.age}</p>
```

Model을 통해 컨트롤러에서 뷰에 데이터 전달하기

- 컨트롤러는 뷰가 응답 화면을 구성하는 데 필요한 데이터를 생성해서 보내 주어야 한다.
- 이때 사용되는 것이 Model이다.

- 이전 설문조사 코드에서 설문 제목과 답변 옵션을 객체로 만들어서 전달하는 예제를 만들어 본다.

Model을 통해 컨트롤러에서 뷰에 데이터 전달하기

예제 14 -1 : Question.java

```java
public class Question {

    private String title;
    private List<String> option;

    public Question(String title, List<String> option) {
        this.title = title;
        this.option = option;
    }

    public Question(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public List<String> getOption() {
        return option;
    }

    public boolean isChoice() {
        return option !=null && !option.isEmpty();
    }
}
```

Model을 통해 컨트롤러에서 뷰에 데이터 전달하기

예제 14 -2 : ServeyController.java 추가

```
private List<Question> createQuestions(){// 설문 내용을 만들기 위한 메소드
    Question q1 = new Question(
            "당신의 역할은 무엇입니까?",
            Arrays.asList("서버","프론트","풀스택"));
    Question q2 = new Question(
            "많이 사용하는 개발 도구는 무엇입니까?",
            Arrays.asList("이클립스","인텔리J","넷빈즈"));
    Question q3 = new Question(
            "하고 싶은 말을 적어 주세요?");

    return Arrays.asList(q1,q2,q3);
}

@RequestMapping(method=RequestMethod.GET)//    컨텍스트 패스/survey
public String form(Model model) {// 폼으로 이동하는 컨트롤러
                    // 컨트롤러에서 생성된 데이터를 JSP 뷰로 전달하기 위한 객체 : Model
    List<Question> questions = createQuestions();
    model.addAttribute("questions",questions);
    return "survey/surveyForm";
}
```

Model을 통해 컨트롤러에서 뷰에 데이터 전달하기

예제 14 -3 : surveyForm.jsp 변경

```
<h2>설 문 조 사</h2>
<form method="POST">
    <c:forEach var="q" items="${questions}" varStatus="status">
        <p>
            ${status.index+1}.${q.title}<br>
            <c:if test="${q.choice}">
                <c:forEach var="option" items="${q.options}">
                    <label>
                        <input type="radio"
                          name="response[${status.index}]" value="${option}">${option}
                    </label>
                </c:forEach>
            </c:if>
            <c:if test="${!q.choice}">
                <input type="text" name="response[${status.index}]">
            </c:if>
        </p>
    </c:forEach>
    <p>
        <label>응답자의 위치 : <br>
        <input type="text" name="red.location">
        </label>
    </p>
    <p>
        <label>응답자의 나이 : <br>
        <input type="text" name="red.age">
        </label>
    </p>
    <input type="submit" value="전송">
</form>
```

ModelAndView를 통한 뷰 선택과 모델 전달

- 컨트롤러의 기본 기능은 두가지 특징을 가진다.
  - Model을 이용해서 뷰에 전달할 데이터 설정
  - 결과를 보여줄 뷰 이름을 반환
- 이때 위 두가지 기능을 한번에 처리 해줄 ModelAndView 객체를 사용할 수 있다.
- 예제 15

```
@RequestMapping(method=RequestMethod.GET)
public ModelAndView form() {
    ModelAndView mav = new ModelAndView();

    List<Question> questions = createQuestions();

    mav.addObject("questions",questions);   //Model의 역할
    mav.setViewName("survey/surveyForm");   //View의 역할

    return mav;
}
```

주요 폼 태그 -

- <form> => <form:form>
- <input> => <form:input>, <form:password>, <form:hidden>
- <select> => <form:select>,<form:options>, <form:option>
- <input type="checkbox"> => <form:checkboxs>, <form:checkbox>
- <input type="radio"> => <form:radiobuttons>, <form:radiobutton>
- <textarea> => <form:textarea>

주요 폼 태그 - <form:form>

- <form:form> => <form>태그 구성

```
<form:form>
    ~~
</form:form>
```

속성 =>

- method 전송방식 : 기본값 :POST
- action 이동할 페이지 : 기본값은 기본 현재 요청페이지와 일치한다.
- commandName 커맨드 객체 : 기본값은 command

```
<form:form commandName="커맨드이름">
    ~~
</form:form>
```

주요 폼 태그 - <form:input>

- <form:input>, <form:password>, <form:hidden> => <input> 태그 구성

- Input => text 타입의 input 태그
- password=> password 타입의 input 태그
- hidden => hidden타입의 input 태그

```
<form:input  path="email">
```

```
<input id="email" name="email" type="text" value="" >
```

주요 폼 태그 - <form:select>

- <form:select>,<form:options>, <form:option> => <select> 태그 구성

- Form:select => select 태그를 생성, Options 태그를 생성하는 데 필요한 콜렉션을 전달 받을 수 있다.
- Form:options => 지정한 콜렉션 객체를 이용하여 option 태그를 생성
- Form:option => 한개의 option 태그를 생성

```
메소드(){
    List<String> list = ~~~
    list.add("AA");
    list.add("BB");
    list.add("CC");
    ...
}
model.addAttribute("list",list);...
```

```
<form:select path="list" items="${list}" >
```

```
<select id="list" name="list">
    <option value="AA">AA </option>
    <option value="BB">BB </option>
    <option value="CC">CC </option>
</select>
```

주요 폼 태그 - <form:select>

- <form:select>,<form:options>, <form:option> => <select> 태그 구성

```
<form:select path="list">
    <option value="">~~</option>
    <form:options items="${list}">
</form:select>
```

```
<form:select path="list">
    <form:option value="AA"/>
    <form:option value="BB">BB</form:option>
    <form:option value="CC" label="CC" />
</form:select>
```

주요 폼 태그 - <form:checkboxes>

- <form:checkboxes>, <form:checkbox> => <input type="checkbox">태그 구성

- <form:checkboxes> => 커맨드 객체의 특정 프로퍼티와 관련된 checkbox 타입의 input태그목록을 생성한다.
- <form:checkbox> => 커맨드 객체의 특정 프로퍼티와 관련된 한개의 checkbox 타입의 input 태그를 생성

```
<form:checkboxes items="${favoriteOnNames}" path="favoriteOs">
```

```
<span>
    <input id="favoriteOs1" name="favoriteOs1" type="checkbox" value="window ME">
    <label for="favoriteOs1" >window ME</label>
</span>
<span>
    <input id="favoriteOs2" name="favoriteOs2" type="checkbox" value="window Vista">
    <label for="favoriteOs2" >window Vista</label>
</span>
<input type="hidden" name="_favoriteOs" value="on" />
```

주요 폼 태그 - <form:radiobutton>

- <form: radiobuttons>, <form:radiobutton> => <input type="radio">태그 구성
  => 여러 옵션 중 하나를 선택해야 하는 경우
- <form:radiobuttons> => 커맨드 객체의 특정 프로퍼티와 관련된 radio타입의 input태그목록을 생성한다.
- <form: radiobutton > => 커맨드 객체의 특정 프로퍼티와 관련된 한개의 radio타입의 input 태그 목록을 생성한다.

```
<form:radiobuttons items="${tools}" path="tool">
```

```
<span>
    <input id="tool1" name="tool1" type="radio" value="Eclipse">
    <label for="tool1" >Eclipse</label>
</span>
<span>
    <input id="tool2" name="tool2" type="radio" value="IntelliJ">
    <label for="tool2" > IntelliJ </label>
</span>
```

주요 폼 태그 - <form:textarea>

- <form:textarea> => <textarea> 태그 구성

```
<form:text area path="etc" cols="20" rows="3" />
```

```
<textarea id="etc" name="etc" cols="20" rows="3"> </textarea>
```