01. 프로젝트 준비

- src/main/java : 자바 코드 위치
- src/main/resource : 설정 파일 위치
- src/main/webapp : HTML, CSS, JS 등이 위치할 폴더
- src/main/webapp/WEB-INF : web.xml 파일이 위치할 폴더
- src/main/webapp/WEB-INF/view : 컨트롤러의 결과를 보여줄 뷰 코드 위치

예제 1) sp4-chap11 폴더에 pom.xml 파일을 작성한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>sp4</groupId>
        <artifactId>sp4-chap11</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>war</packaging>

        <dependencies>
                <dependency>
                        <groupId>javax.servlet.jsp</groupId>
                        <artifactId>jsp-api</artifactId>
                        <version>2.2</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>javax.servlet-api</artifactId>
                        <version>3.0.1</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>jstl</artifactId>
                        <version>1.2</version>
                </dependency>
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-webmvc</artifactId>
```

```
            <version>4.1.0.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>4.1.0.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>com.mchange</groupId>
            <artifactId>c3p0</artifactId>
            <version>0.9.2.1</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.30</version>
        </dependency>

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

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                    <encoding>utf-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

<dependency> 부분을 보면 웹을 위한 모듈과 DB연동을 위한 모듈을 설정한다.

- AlreadyExistingMemberException.java
- ChangPasswordService.java
- IdPasswordNotMatchingException.java
- Member.java
- MemberDao.java
- MemberNoFoundException.java
- MemberRegisterService.java
- RegisterRequest.java

예제 2) 서비스 클래스와 DAO클래스를 위한 스프링 설정  클래스를 작성한다.
sp4-chap11\src\main\resources\spring-member.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:tx="http://www.springframework.org/schema/tx"
        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
                http://www.springframework.org/schema/tx
                http://www.springframework.org/schema/tx/spring-tx.xsd">

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

        <bean id="dataSource"
                        class="com.mchange.v2.c3p0.ComboPooledDataSource"
                        destroy-method="close">
                        <property name="driverClass"
                                        value="oracle.jdbc.driver.OracleDriver" />
                        <property name="jdbcUrl"
                                        value="jdbc:oracle:thin:@localhost:1521:xe" />
                        <property name="user" value="smrit" />
                        <property name="password" value="oracle" />
                <property name="maxPoolSize" value="100" />
                <property name="maxideTime" value="600" />
                <property name="idleConnectionTestPeriod" value="300" />
        </bean>
```

```
        <bean id="transactionManager"

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

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

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

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

</beans>
```

예제 3) 스프링 MVC를 위한 기본 설정 파일 작성
sp4-chap11\src\main\resources\spring-mvc.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        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
                http://www.springframework.org/schema/mvc
                http://www.springframework.org/schema/mvc/spring-mvc.xsd">

        <mvc:annotation-driven />

        <mvc:default-servlet-handler />

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

```
</beans>
```

예제 4) web.xml 파일을 작성한다.  src\main\webapp\WEB-INF\web.xml
contextConfiguration 초기화 파라미터 설정 부분이 있다.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns="http://java.sun.com/xml/ns/javaee"
	xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
		http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
	version="3.0">

	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-class>
			org.springframework.web.servlet.DispatcherServlet
		</servlet-class>
		<init-param>
			<param-name>contextConfigLocation</param-name>
			<param-value>
				classpath:spring-mvc.xml
				classpath:spring-controller.xml
				classpath:spring-member.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>

</web-app>
```

13~19행 : contextConfiguration 초기화 파리미터.

02. @RequestMapping을 이용한 경로 매핑

- 특정 요청 URL을 처리할 코드 작성
- 처리 결과를 HTML과 같은 형식으로 응답

@Controller 애노테이션을 사용한 컨트롤러 클래스를 이용해서 구현한다.
컨트롤러 클래스는 @RequestMapping 애노테이션을 사용해서 메서드에서 처리할 요청 경로를 지정한다.

예제 5) 회원 가입의 첫 번째 과정인 약관을 보여주는 요청 경로를 처리하는 컨트롤러 코드를 작성해보자. src\main\java\controller\RegisterController.java

```
package controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import spring.MemberRegisterService;

@Controller
public class RegisterController {
        private MemberRegisterService memberRegisterService;
```

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

	@RequestMapping("/register/step1.do")
	public String handleStep1() {
		return "register/step1";
	}

}
```

요청경로가 /Register/step1 인 경우 별다른 처리 없이 약관 내용을 보여주면 되기 때문에 handleStep1() 메서드는 단순히 약관 내용을 보여줄 뷰 이름을 리턴하도록 구현 한다.
예제 6) 예제5의 24행에서 리턴하는 뷰 이름에 해당하는 JSP코드를 작성한다.
src\main\webapp\WEB-INF\view\register\step1.jsp

`

```
<%@ page contentType="text/html; charset=utf-8" %>
<!DOCTYPE html>
<html>
<head>
	<title>회원가입</title>
</head>
<body>
	<h2>약관</h2>
	<p>약관 내용</p>
	<form action="step2.do" method="post">
	<label>
		<input type="checkbox" name="agree" value="true"> 약관 동의
	</label>
	<input type="submit" value="다음 단계" />
	</form>
</body>
</html>
```

예제 7) RegisterController클래스를 빈으로 등록한다.
src\main\resources\spring-controller.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
	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
		http://www.springframework.org/schema/mvc
		http://www.springframework.org/schema/mvc/spring-mvc.xsd">


	<bean class="controller.RegisterController">
	</bean>

</beans>
```

실행 : http://localhost:8088/sp4-chap11/register/step1.do

예제 8) 약관 동의 여부를 확인하기 위해 HTTP요청 파라미터의 값을 사용하게 한다.
약관 동의 화면의 다음 요청을 처리하는 코드를 RegisterController클래스에 추가한다.
src\main\java\controller\RegisterController.java

```
package controller;

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

import spring.RegisterRequest;

@Controller
public class RegisterController {
	@RequestMapping("/register/step1.do")
	public String handleStep1() {
		return "register/step1";
	}
```

```
@RequestMapping(value = "/register/step2.do", method = RequestMethod.POST)
	public String handleStep2(
			@RequestParam(value = "agree", defaultValue = "false")
Boolean agree,
			Model model) {
		if (!agree) {
			return "register/step1";
		}
		return "register/step2";
	}
}
```

handleStep2() 메서드는 agree요청 파라미터의 값이 true가 아니면 약관 동의 폼을 보여주기 위해 “register/step1"뷰 이름을 리턴한다.
약관에 동의했다면 입력 폼을 보여주기 위해 “register/step2"를 뷰 이름으로 리턴한다.

예제 9) src\main\webapp\WEB-INF\view\register\step2.jsp

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

05. 리다이렉트 처리

웹 브라우저에서 http://localhost:8088/sp4-chap11/register/step2를 입력하면 POST방식만을 처리하기 때문에 웹브라우저에 직접 주소를 입력할 때 사용하는 GET방식 요청은 처리하지 않는다.

잘못된 전송 방식으로 요청이 왔을 때 알맞은 경로로 리다이렉트하는 것이 좋다.

예제 10) src\main\java\controller\RegisterController.java

```
@Controller
public class RegisterController {
... (중략)

        @RequestMapping(value = "/register/step2.do", method =
RequestMethod.GET)
        public String handleStep2Get() {
                return "redirect:step1";
        }

}
```

RegisterController클래스에 코드를 추가했다면, 서버를 재 시작한 뒤에 웹브라우저에 http://localhost:8088/sp4-chap11/register/step2 주소를 직접 입력해보자.
지정한 경로로 리다이렉트되는 것을 확인 할 수 있다.

@RequestMapping 적용 메서드가 "redirect:"로 시작하는 경로를 리턴하면, 나머지 경로를 이용해서 리다이렉트 할 경로를 구한다.

"redirect:"뒤의 문자열이 “/”로 시작하면 웹 어플리케이션 기준으로 이동 경로를 생성한다.

즉 웹서버에 설정되어 있는 “root"경로부터 시작 경로를 리턴하게 된다.

“/”로 시작하지 않으면, 현재경로를 기준으로 상대경로를 사용한다.

06. 커맨드 객체를 이용해서 요청 파라미터 사용하기

```
@RequestMapping(value = "/register/step3.do", method = RequestMethod.POST)
public String handleStep3(RegisterRequest regReq) {
    String email=request.getParameter("email");
    String name=request.getParameter("name");
    String password=request.getParameter("password");
```

위 코드가 나쁜 것은 아니지만, 요청 파라미터 개수가 증가할 때마다 handleStep3()메서드의 코드 길이도 함께 길어지는 담점이 있다.

다소 복잡한 폼의 경우 파라미터 개수가 20개를 넘고 파라미터 값을 읽어와야 하는 코드만 40줄이 없을 수도 있다.

스프링은 이런 불편함을 줄이기 위해 커맨드 객체라는 것을 지원하고 있다.
스프링은 요청 파라미터의 값을 커맨드 객체에 담아주는 기능을 제공한다.

파라미터 중에 이름이 name인 파라미터의 값을 커맨드 객체의 setName() 메서드를 사용해서 커맨드 객체에 전달하는 기능을 제공한다.
요청 파라미터의 값을 전달받을 수 있는 set메서드를 포함하고 있는 객체를 커맨드 객체로 사용하면 된다.

커맨드 객체는 @RequestMapping 애노테이션이 적용된 메서드의 파라미터로 사용된다.

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

RegisterRequest 클래스는 setEmail(), setName, setPassword(), setConfirmPassword() 메서드를 정의하고 있느데, 스프링은 이들 메서드를 사용해서 email, name, password, confirmPassword 요청파라미터의 값을 커맨드 객체에 복사한 뒤 regReq 파라미터를 전달한다.
스프링은 MVC가 handleStep3()메서드에 전달할 RegisterRequest객체를 생성하고, 그 객체의 set메서드를 이용해서 일치하는 요청 파라미터의 값을 전달하게 된다.

예제 11) RegisterController클래스에 커맨드 객체를 이용해서 폼에 입력한 값을 전달받아 회원 가입을 처리하는 코드를 추가해보자.

```
package controller;

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

import spring.AlreadyExistingMemberException;
import spring.MemberRegisterService;
import spring.RegisterRequest;

@Controller
public class RegisterController {
	private MemberRegisterService memberRegisterService;

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

	@RequestMapping("/register/step1.do")
	public String handleStep1() {
		return "register/step1";
	}

	@RequestMapping(value = "/register/step2.do", method = RequestMethod.POST)
	public String handleStep2(
			@RequestParam(value = "agree", defaultValue = "false") Boolean agree,
			Model model) {
		if (!agree) {
			return "register/step1";
		}
		model.addAttribute("registerRequest", new RegisterRequest());
		return "register/step2";
	}
```

```
	@RequestMapping(value = "/register/step2.do", method = RequestMethod.GET)
	public String handleStep2Get() {
		return "redirect:step1";
	}

	@RequestMapping(value = "/register/step3.do", method = RequestMethod.POST)
	public String handleStep3(RegisterRequest regReq) {
		try {
			memberRegisterService.regist(regReq);
			return "register/step3";
		} catch (AlreadyExistingMemberException ex) {
			return "register/step2";
		}
	}
}
```

handleStep3()메서드는 MemberRegisterService를 이용해서 회원 가입을 처리한다.
회원 가입이 성공하면 뷰 이름으로 “register/step3”을 리턴 한다.

예제 12) RegisterController클래스는 MemberRegisterService 타입에 빈에 대한 의존을 가지므로 spring-controller.xml 파일에 의존 주입을 설정해 준다.
memberRegSvc빈은 spring-member.xml 파일에 정의되어 있다.
src\main\resources\spring-controller.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
	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
		http://www.springframework.org/schema/mvc
		http://www.springframework.org/schema/mvc/spring-mvc.xsd">


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

```
</beans>
```

예제 13) 가입 성공했을 때 응답결과를 보여 줄 step3.jsp
src\main\webapp\WEB-INF\view\register\step3.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
	<title>회원가입</title>
</head>
<body>
	<p>회원 가입을 완료했습니다.</p>
	<p><a href="<c:url value='/main.do'/>">[첫 화면 이동]</a></p>
</body>
</html>
```

컨트롤러 클래스를 수정하고 설정파일을 수정했으므로 서버를 재 시작

참조)MemberRegisterService.java에 굵은 부분 추가

```
package spring;

import java.util.Date;

import java.text.SimpleDateFormat;
import org.springframework.transaction.annotation.Transactional;

public class MemberRegisterService {
	private MemberDao memberDao;

	public MemberRegisterService(MemberDao memberDao) {
		this.memberDao = memberDao;
	}

	@Transactional
	public void regist(RegisterRequest req) {

		Member member = memberDao.selectByEmail(req.getEmail());
		if (member != null) {
			throw new AlreadyExistingMemberException("dup email " +
req.getEmail());
```

```
                }
                Member newMember = new Member(
                                req.getEmail(), req.getPassword(), req.getName(),
                                new Date());

                SimpleDateFormat dateFormat = new
SimpleDateFormat("MMddHHmmss");
                String prefix = dateFormat.format(new Date());
                long prefix1= Integer.valueOf(prefix);
                newMember.setId(prefix1);

                memberDao.insert(newMember);
        }
}
```

참조)
데이터베이스 연결이 안되는 경우 톰캣 라이브러리 폴더에 jdbc6.jar가 복사되어 있는 지 확인해보자.

07. 뷰 JSP 코드에서 커맨드 객체 사용하기
회원 가입 완료화면에서 가입할 때 사용한 이메일 주소와 이름을 보여주면 사용자에게 조금 더 친절하게 보일 것이다.

http요청 파라미터를 이용해서 회원 정보를 전달했으므로 JSP의 표현식 등을 이용해서 정보를 표시해도 되지만, 스프링을 사용한다면 커맨드 객체를 사용해서 정보를 표시할 수 있다.

예제 14) src\main\webapp\WEB-INF\view\register\step3.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
        <title>회원가입</title>
</head>
<body>
        <p><strong>${registerRequest.name}님</strong>
                회원 가입을 완료했습니다.</p>
        <p><a href="<c:url value='/main.do'/>">[첫 화면 이동]</a></p>
</body>
</html>
```

9행을 보면 ${registerRequest.name}코드를 볼 수 있다.
**${registerRequest.name} 대신** ${registerRequest.email}처럼 name 대신 email로 변경해도 됨. 이전 페이지의 name속성명에 의해 전달 받게 된다.

여기서 registerRequest가 커맨드 객체에 접근할 때 사용한 속성명이 된다.
RegisterController.java 의 public String handleStep3(RegisterRequest regReq)의 인자인 RegisterRequest를 사용하여 뷰에 전달된다.
RegisterRequest는 **${registerRequest.name}의 registerRequest에 의해 전달 됨.**
스프링MVC는 커맨드 객체의(첫 글자를 소문자로 바꾼) 클래스 이름과 동일한 속성 이름을 사용해서 커맨드 객체를 뷰에 전달한다.

커맨드 객체에 접근할 때 사용할 속성명을 변경하고 싶다면, 커맨드 객체로 사용할 파라미터에 @ModelAttribute 애노테이션을 적용하면된다.

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

※ <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>에 오류가 발생하는 경우 pom.xml의 <dependencys>에

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
</dependency>
```

를 추가해주면 된다.

08. 커맨드 객체와 스프링 폼 연동
회원 정보 입력 폼에서 중복된 이메일 주소를 입력하면 다시 입력 폼을 보여주는데, 이 경우 앞에서 입력했던 내용이 사라진 텅 빈 폼 화면을 보게 된다.
그러나 다시 입력 폼을 보여 줄 때 커맨드 객체의 값을 폼에 보여주면 될 것이다.

커맨드 객체를 사용할 경우, 스프링 MVC가 제공하는 커스텀 태그를 사용하면 좀더 간단하게 커맨드 객체의 값을 출력할 수 있다.

예제 15) 스프링은 <form:form> 태그와 <form:input> 태그를 제공하고 있으며, 이 두 태그를 사용하면 step2.jsp 코드를 커맨드 객체의 값을 폼에 출력할 수 있게 할 수 있다.

```
<%@ page contentType="text/html; charset=utf-8" %>
```

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
<head>
	<title>회원가입</title>
</head>
<body>
	<h2>회원 정보 입력</h2>
	<form:form action="step3.do" commandName="registerRequest">
	<p>
		<label>이메일:<br>
		<form:input path="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="가입 완료">
	</form:form>

</body>
</html>
```

02행에서 스프링이 제공하는 폼 태그를 사용하기 위해 taglib 디렉티브를 설정했다.
10행의 <form:form> 태그는 <form>태그를 생성해 준다.
<form:form> 태그의 속성
• action : <form> 태그의 action 속성과 동일한 값을 사용한다.
• commandName : 커맨드 객체의 속성 이름을 지정한다. 설정하지 않을 경우 "command"를 기본 값으로 사용한다.

커맨드 객체의 속성 이름이 "registerRequest"이므로 commandName속성의 값으로 설정했다.

<form:input>태그는 <input>태그를 생성해주는데, 이때 path 경로로 지정한 커맨드 객체의 프로퍼티 값을 <input> 태그의 value 속성의 값으로 사용한다.
<form:input path="name" />의 경우 커맨드 객체의 name 프로퍼티 값을 value 속성으로 사용한다.
<form:password> 태그도 <form:input>태그와 유사하지만, password 타입의 <input> 태그를 생성하므로 value속성의 값을 빈 문자열로 설정한다.

<form:form>태그를 사용하려면 반드시 커맨드 객체가 존재해야 한다.
예제 16) 존재하는 이메일을 입력하여 다시 입력 페이지로 돌아가는 지 확인하고 돌아왔을 때 입력한 내용이 올바르게 출력되는지 확인해 보자.
src\main\java\controller\RegisterController.java

```
package controller;

import org.springframework.ui.Model;


	@RequestMapping(value = "/register/step2.do", method = RequestMethod.POST)
	public String handleStep2(
			@RequestParam(value = "agree", defaultValue = "false") Boolean agree,
			Model model) {
		if (!agree) {
			return "register/step1";
		}
		model.addAttribute("registerRequest", new RegisterRequest());
		return "register/step2";
	}
```

09. 컨트롤러 구현 없는 경로 매핑
<mvc:view-controller> 태그는 요청 경로와 뷰 이름을 연결해주는 역할을 하도록 해 준다.
<mvc:view-controller path="/main" view-name="main"/>
이 태그는 /main 요청 경로에 대해 뷰 이름으로 main을 사용한다는 것을 의미한다.

예제 17) src\main\java\resources\spring-controller.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        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
                http://www.springframework.org/schema/mvc
                http://www.springframework.org/schema/mvc/spring-mvc.xsd">

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

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

</beans>
```

mvc네임스페이스를 이용하므로 04행과 08~09행의 네임스페이스 관련 설정을 함께 추가했다.

예제 18) 뷰 이름으로 main을 사용하므로, 이에 해당하는 main.jsp파일을 작성해 보자.
src\main\webapp\WEB-INF\view\main.jsp

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

http://localhost:8088/sp4-chap11/main.do

10. 주요 에러 발생 상황

10.1 @RequestMapping과 관련된 주요 익셉션

가장 흔하게 발생하는 에러는 매핑된 경로가 없는 경우이다. 요청경로를 처리할 컨트롤러가 존재하지 않거나 <mvc:view-controller> 등의 설정이 없다면 404오류가 발생할 것이다.

- 요청경로가 올바른지 여부
- 컨트롤러에 설정한 경로가 올바른지 여부
- 컨트롤러 클래스를 빈으로 등록했는지 여부
- 컨트롤러 클래스에 @Controller 애노테이션을 적용했는지 여부

11. 커맨드 객체: 중첩 • 콜렉션 프로퍼티

예제 19) 세 개의 항목이 있고 응답자 정보를 담기 위해

src\main\java\survey\Respondent.java

```
package survey;

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;
	}
}
```

예제 20) 설문 항목에 대한 답변과 응답자 정보를 함께 저장
src\main\java\survey\AnsweredData.java

```
package survey;

import java.util.List;

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;
	}

}
```

AnsweredData클래스는 답변 목록을 저장하기 위해 List타입의 responses프로퍼티를 사용했고 응답자의 정보를 담기 위해 Respondent타입의 res프로퍼티를 사용했다.

• 리스트 타입의 프로퍼티가 존재한다. responses프로퍼티는 String 타입의 값을 갖는 List 컬렉션이다.
• 중첩 프로퍼티를 갖는다. res 프로퍼티는 Responsedent 타입으로서, res프로퍼티는 다시 age와 location프로퍼티를 갖는다.
이를 중첩된 형식으로 표시하면, res.age프로퍼티와 res.location 프로퍼티로 표현할 수 있다.

예제 21) List타입의 프로퍼티와 중첩 타입을 위해 사용될 파라미터의 이름을 지정하는 방법은
commandObj.getRes().setName(request.getParameter("res.name"));
이다.

AnsweredData클래스를 커맨드 객체로 사용하는 예제를 작성해 보자.
src\main\java\survey\SurveyController.java

```
package survey;

import java.util.List;

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


@Controller
@RequestMapping("/survey")
public class SurveyController<Question> {

	@RequestMapping(method = RequestMethod.GET)
	public String form(Model model) {
		List<Question> questions = createQuestions();
		model.addAttribute("questions", questions);
		return "survey/surveyForm";
	}

	private List<Question> createQuestions() {
		// TODO Auto-generated method stub
		return null;
	}

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

}
```

form()메서드와 submit()메서드의 @RequestMapping은 전송 방식만을 설정하고, 클래스의 @RequestMapping에만 처리할 경로를 지정했다.
이 경우 form()메서드와 submit()메서드가 처리하는 경로는 "/survey'요청을 처리하고, submit()메서드는 POST방식의 "/survey"요청을 처리한다.
submit()메서드는 커맨드 객체로 AnsweredData객체를 사용한다,

예제 22) spring-controller.xml파일에 컨트롤러 클래스를 빈으로 추가한다.
src\main\resources\spring-controller.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
	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
		http://www.springframework.org/schema/mvc
		http://www.springframework.org/schema/mvc/spring-mvc.xsd">

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

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

	<bean class="survey.SurveyController">
	</bean>
</beans>
```

예제 23) SurveyController 클래스의 form()메서드는 각각 뷰 이름으로 "survey/surveyForm"과 "survey/submitted"를 사용하므로 이 두 뷰 이름에 해당하는 JSP 파일을 만들 차례이다.
src\main\webapp\WEB-INF\view\survey\surveyForm.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
	<title>설문조사</title>
</head>
<body>
	<h2>설문조사</h2>
	<form method="post">
	<p>
		1. 당신의 역할은?<br/>
		<label><input type="radio" name="responses[0]" value="서버">
			서버개발자</label>
		<label><input type="radio" name="responses[0]" value="프론트">
			프론트개발자</label>
		<label><input type="radio" name="responses[0]" value="풀스택">
			풀스택개발자</label>
	</p>
	<p>
		2. 가장 많이 사용하는 개발도구는?<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="Sublime">
			Sublime</label>
	</p>
	<p>
		3. 하고싶은 말<br/>
		<input type="text" name="responses[2]">
	</p>
	<p>
		<label>응답자 위치:<br>
		<input type="text" name="res.location">
		</label>
	</p>
	<p>
		<label>응답자 나이:<br>
		<input type="text" name="res.age">
		</label>
	</p>
```

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

각 <input> 태그의 name속성의 값을 보면 List타입 프로퍼티와 중첩 프로퍼티에 요청 파라미터 값을 전달하기 위한 이름을 사용한 것이다.

예제 24) 전송 후 결과를 보여주기 위한 submitted.jsp를 작성하자.
src\main\webapp\WEB-INF\view\survey\submitted.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
        <title>응답 내용</title>
</head>
<body>
        <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>
</body>
</html>
```

11~14행은 커맨드 객체 (useData)의 List타입 프로퍼티인 responses에 담겨 있는 각 값을 출력한다.
16~17행은 중첩 프로퍼티인 res.location과 res.age의 값을 출력한다.

http://localhost:8088/sp4-chap11/survey 주소를 입력한다.

12 model을 통해 컨트롤러에서 뷰에 데이터 전달하기
컨트롤러는 뷰가 응답 화면을 구성하는 데 필요한 데이터를 생성해서 전달해주어야 하는데,

이때 사용하는 것이 바로 Model이다.

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class HelloController {

	@RequestMapping("/hello")
	public String hello(Model model,
			@RequestParam(value = "name", required = false) String
name) {
		model.addAttribute("greeting", "안녕하세요, " + name);
		return "hello";
	}
}
```

뷰에 데이터를 전달해야 하는 컨트롤러는 hello()메서드처럼 @RequestMapping이 적용된 메서드의 파라미터로 Model을 추가하고, Model파라미터의 addAttribute()메서드를 사용해서 뷰에서 사용할 데이터를 전달해주면 된다.

addAttribute()메서드의 첫 번째 파라미터는 속성 이름으로서 뷰 코드에서는 이 이름을 사용해서 데이터에 접근한다,
JSP에서는 다음과 같이 표현식을 사용해서 속성값에 접근 할 수 있다.
${greeting}

예제 25) src\main\java\survey\Question.java
SurveyController 예제는 surveyForm.jsp에 설문 항목을 하드 코딩으로 등록했는데, 설문 항목을 컨트롤러에서 생성해서 뷰에 전달하는 방식으로 변경해 본다
개별 설문 항목 데이터를 담기 위한 클래스를 작성한다.

```
package survey;

import java.util.Collections;
import java.util.List;

public class Question {

	private String title;
	private List<String> options;
```

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


	public Question(String title) {
		this(title, Collections.<String>emptyList());
	}

	public String getTitle() {
		return title;
	}

	public List<String> getOptions() {
		return options;
	}

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

예제 26) src\main\java\survey\SurveyController.java
SurveyController가 Question객체의 목록을 생성해서 뷰에 전달하도록 구현하는 것이다.
실제로 DB와 같은 곳에서 정보를 읽어와 Question 목록을 생성하겠지만, 예제에서는 컨트롤에서 직접 생성하도록 구현한다.

```
package survey;

import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
```

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

	@RequestMapping(method = RequestMethod.GET)
	public String form(Model model) {
		List<Question> questions = createQuestions();
		model.addAttribute("questions", questions);
		return "survey/surveyForm.do";
	}

	private List<Question> createQuestions() {
		Question q1 = new Question("당신의 역할은 무엇입니까?",
				Arrays.asList("서버", "프론트", "풀스택"));
		Question q2 = new Question("많이 사용하는 개발도구는 무엇입니까?",
				Arrays.asList("이클립스", "인텔리J", "서브라임"));
		Question q3 = new Question("하고 싶은 말을 적어주세요.");
		return Arrays.asList(q1, q2, q3);
	}

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

}
```

17행에서 보듯 form()메서드에 Model타입의 파라미터를 추가했고, 18행을 통해 생성한 Question리스트를
19행에서는 "question"라는 이름으로 모델을 추가했다.

예제 27) src\main\webapp\WEB-INF\view\survey\surveyForm.jsp
컨트롤러가 전달한 Question리스트를 사용해서 폼 화면을 생성하도록 JSP코드를 수정한다.

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
	<title>설문조사</title>
</head>
<body>
```

```
        <h2>설문조사</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="${q.options}">
                        <label><input type="radio"
                                                name="responses[${status.index}]"
value="${option}">
                                ${option}</label>
                        </c:forEach>
                </c:if>
                <c:if test="${! q.choice }">
                <input type="text" name="responses[${status.index}]">
                </c:if>
        </p>
        </c:forEach>

        <p>
                <label>응답자 위치:<br>
                <input type="text" name="res.location">
                </label>
        </p>
        <p>
                <label>응답자 나이:<br>
                <input type="text" name="res.age">
                </label>
        </p>
        <input type="submit" value="전송">
        </form>

</body>
</html>
```

참조) http://localhost:8088/sp4-chap11/로 실행
src\main\webapp\index.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
```

```
<html>
<head>
    <title>인덱스</title>
</head>
<body>
    <p>11장 예제</p>
    <ul>
        <li><a href="main.do">/main으로 이동</a></li>
        <li><a href="survey">/survey로 이동</a></li>
    </ul>
</body>
</html>
```