Chapter 11 MVC 1

요청, 매핑,커맨드 객체, 리다이렉트, 폼 태그, 모델

정승혜

프로젝트 설정

Configuration 파일

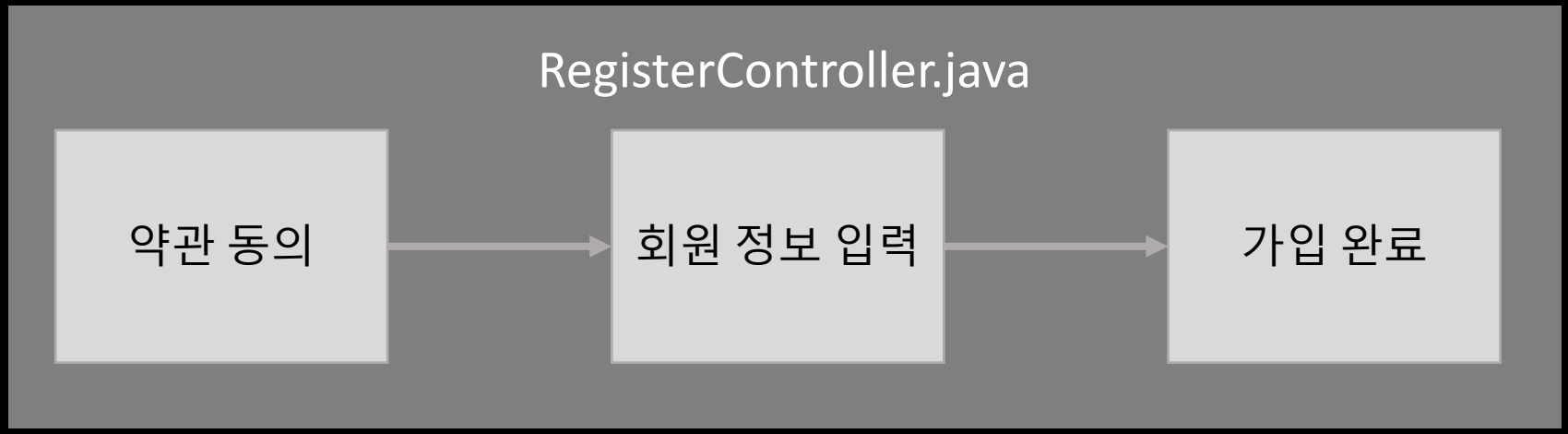
- Member(DB) : @EnableTransactionManagement
- Mvc : @EnableWebMvc
- Controller (컨트롤러 Bean, 커맨드 객체 의존주입)

MVC 구현 : 회원가입

회원 가입

- 약관동의 -> 회원 정보 입력 -> 가입완료

-> 여러 단계를 거쳐 완성되는 하나의 기능
-> 관련 요청 경로(URL)을 한 개의 컨트롤러 클래스에서



Controller

- Mapping

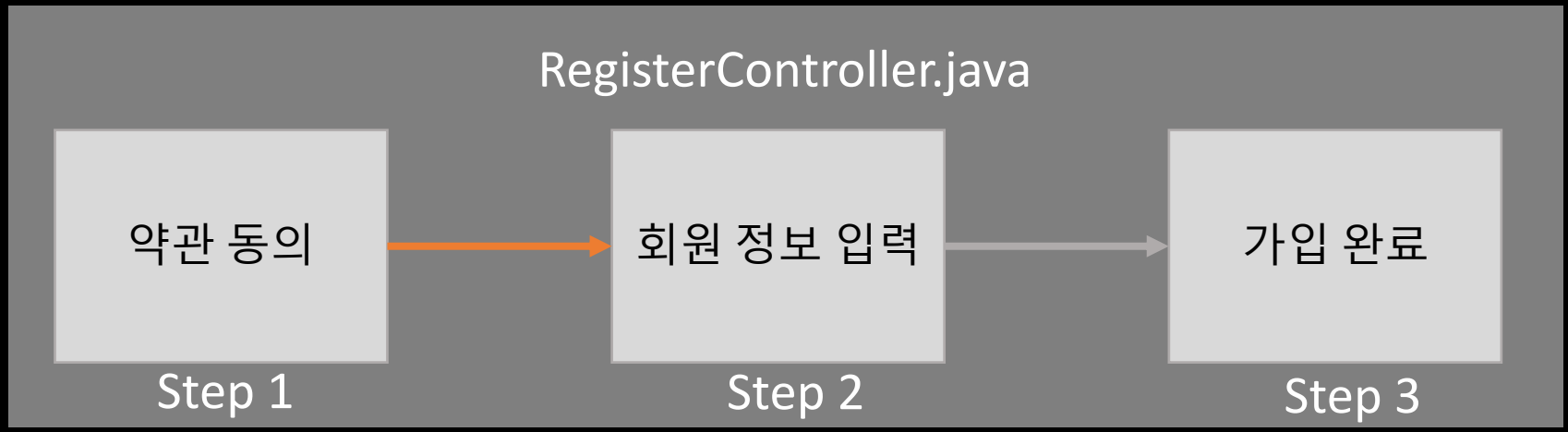
- @RequestMapping : get/post에 관계없이 처리
- 특정 요청만 처리하고 싶다면

@GetMapping, @PostMapping을 사용할 것

⇒ 동일한 경로에 대해 GET/POST 방식을

다른 메서드가 처리하도록 설정 가능

Controller - 1



Step1 -> Step2 에서 사용자가 약관 동의 체크 여부에 따라

1. Step1로 돌아감
2. Step2로 post

Controller에서 요청 파라미터 접근

체크 여부 확인하기

1. HttpServletRequest의 getParameter() 메서드

```
@PostMapping("/register/step2")
public String handleStep2(HttpServletRequest request) {
        String agreeParam = request.getParameter("agree"); // 체크 여부
        if (agreeParam == null || !agreeParam.equals("true")) {
                return "register/step1";
        }
        return "register/step2";
}
```

Controller에서 요청 파라미터 접근

체크 여부 확인하기

2. @RequestParam 어노테이션 사용
- 요청 파라미터 개수가 적은 경우 간단하게 구현 가능

```
@PostMapping("/register/step2")
public String handleStep2(
        @RequestParam(value="agree", defaultValue = "false") Boolean agreeVal) {
        if (!agreeVal) {
                return "register/step1";
        }
        return "register/step2";
}
```

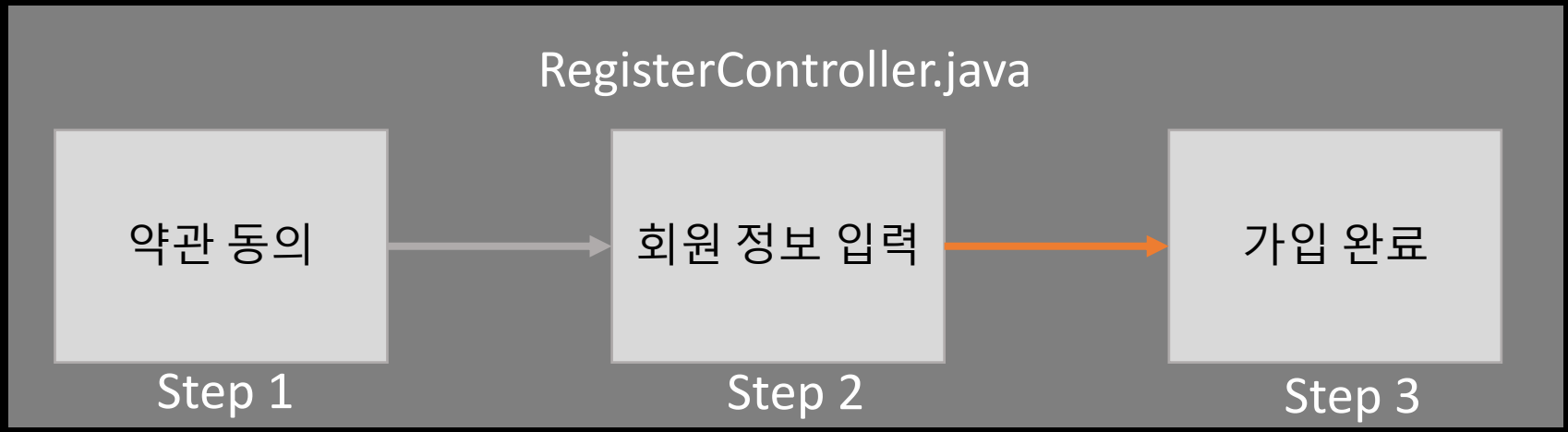
- value : 파라미터 이름
- defaultValue : 값이 없을 때 default 값
- required : 필수요소인데 값이 없으면 익셉션 발생

Controller - 1

Step2는 Step1 -> Step2로 넘어오는 PostMapping만 지원
만약 바로 step2 url로 넘어오려 했다면
=> 405 Error 발생, Redirect 필요

```
@GetMapping("/register/step2")
public String handleStep2Get( ) {
        return “redirect:/register/step2";
}
```

Controller - 2



Step2의 폼 (이메일,이름,비밀번호…등)을 서버에 전송

=> 변수 개수가 많아서 앞의 방법(getParameter, @RequestParam)
을 이용하면 코드 길이가 길어짐

커맨드 객체와 요청 파라미터

RegisterRequest 클래스

스프링의 요청 파라미터 값을 커맨드(command) 객체에 담아주는 기능

RegisterController.java

```
private MemberRegisterService memberRegisterService;

@PostMapping("/register/step3")
public String handleStep3( RegisterRequest regReq) {
        try {
                memberRegisterService.regist(regReq);
                 return "register/step3";
        } catch ( DuplicationMemberException ex) {
        return "register/step2";
        }
}
```

커맨드 객체와 요청 파라미터

ControllerConfig.java

```
@Autowired
private MemberRegisterService memberRegSvc;

@Bean
public RegisterController registerController( ) {
        RegisterController controller = new RegisterController();
        controller.setMemberRegisterService(memberRegSvc);
        return controller;
}
```

MemberConfig.java

```
@Bean
public MemberRegisterService memberRegSvc() {
return new MemberRegisterService(memberDao());
}
```

뷰 JSP 코드에서 커맨드 객체 사용

Step2.jsp

```
....
</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>
```

Step3.jsp

```
...
<body>
<p><strong>${registerRequest.name}님</strong>
회원 가입을 완료했습니다.</p>
</p><a href="<c:url value='/main'/>">
</body>
```

커맨드 객체 이름 바꾸기

```
Import org.springframework.web.bind.annotation.ModelAttribute;

@PostMapping(“/register/step3”)
Public String handleStep3(@ModelAttribute(“formData”) RegisterRequest regReq ) {
….
}
```

@ModelAttribute 어노테이션을 적용할 경우,
뷰 JSP 코드에서 registerRequest.~~~~가 아니라
formData.~~~로 접근할 수 있게 됨

뷰 JSP 코드에서 커맨드 객체 사용

Step2.jsp

```
....
</head>
<body>
<h2>회원 정보 입력</h2>
<form action = "step3" method="post">
<p>
<label>이메일:<br>
<input type = "text" name = "email" id = "email" value = "${registerRequest.email}">
</label>
</p>
```

Step3.jsp

```
...
<body>
<p><strong>${registerRequest.name}님</strong>
회원 가입을 완료했습니다.</p>
</p><a href="<c:url value='/main'/>">
</body>
```

커맨드 객체

커맨드 객체와 스프링 폼 연동

회원 정보를 입력했을 때,
중복됐거나 잘못된 정보로 폼을 재작성해야할 경우
(다시 돌아갔을때), 자동으로 값 채워 줌 기능

Step2.jsp

```
....
</head>
<body>
<h2>회원 정보 입력</h2>
<form action = "step3" method="post">
<p>
<label>이메일:<br>
<input type = "text" name = "email" id = "email" value = "${registerRequest.email}">
</label>
</p>
```

스프링MVC의 커스텀 태그 - 1

- 좀 더 간단한 방식으로 커맨드 객체의 값 출력 가능

```
<%@ taglib prefix=“form” uri=“http://www.springframwork.org/tags/form“ %>
```

- <form:form>
- <form:input>

```
<body>
        <h2>회원 정보 입력</h2>
        <form:form action="step3" modelAttribute="registerRequest">
        <p>
        <label>이메일:<br>
        <form:input path = "email"/>
        </label>
```

스프링MVC의 커스텀 태그 - 2

이 태그들은 커맨드 객체가 존재할 때 사용할 수 있으므로,
만약 Step2에서 <form:form> 태그를 사용하려면
Step1->Step2 과정에서 객체를 모델에 삽입하여 전송해야함

```
@PostMapping("/register/step2")
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";
}
```

스프링MVC의 커스텀 태그 - 2

파라미터로 바로 전송하여도 됨

```
@PostMapping("/register/step2")
public String handleStep2(
@RequestParam(value="agree", defaultValue = "false") Boolean agree,
                                                    RegisterRequest registerRequest ) {
        if (!agree) {
                return "register/step1";
        }
        return "register/step2";
}
```

커맨드 객체 : 중첩, 콜렉션 프로퍼티

AnsweredData Class

- 스프링 MVC는 커맨드 객체가 리스트/중첩 프로퍼티를 가진 경우에도 커맨드 객체에 설정해줌

Http 요청 파라미터 이름이

1. “프로퍼티[인덱스]” 형식이면 List 타입 프로퍼티의 값 목록으로 처리
2. “프로퍼티이름.프로퍼티이름“ 형식이면 중첩 프로퍼티 값 처리

WebMvcConfigurer

WebMvcConfigurer

회원 가입 완료 후 첫 화면으로 이동한 뒤
첫화면은 단순히 문구와 회원가입으로 이동하는 링크만 제공하고
이때 해당 컨트롤러는 처리할 것 없이 단순히 뷰를 연결하기만 함

⇒ 이때 컨트롤러 클래스를 생성할 필요없이 요청경로와 뷰 이름을 연결
⇒ WebConfigurer의 addViewControllers() 메서드

MvcConfig.java

```
@Override
public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/main").setViewName("main");
}
```

step3.jsp

```
<p><a href="<c:url value='/main'/>">[첫 화면 이동]</a></p>
```

MVC 구현 : 설문 조사

과정

1. Controller를 통해 경로에 대한 Mapping 설정
2. Configuration에 Bean 객체 등록
3. JSP로 뷰 작성

< 설문 조사 만들기 >

- Question Class 생성
- Question Class를 활용하여 설문항목을 컨트롤러에서 생성
- Model을 통해 뷰에 전달

```
public class Question {

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

컨트롤러에서 Data 작성 / 전달

SurveyController.java

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

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);
}
```

뷰 JSP 코드의 데이터 사용

surveyForm.jsp

```
<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>
```

ModelAndView 객체

@GetMapping을 적용한 메서드는 String 대신 ModelAndView 리턴 가능

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

```
@GetMapping
public ModelAndView form( ) {
        List<Question> questions = createQuestions();
        ModelAndView mav = new ModelAndView();
        mav.addObject("questions", questions);
        mav.setViewName("survey/surveyForm");
         return mav;
}
```