Thymeleaf - Form 데이터 전송

1. Text, CheckBox, Multi Checkbox

1) Text 데이터 전송

- 가장 단순한 Text 데이터 전송 예제입니다.
- 먼저 Controller 측 코드입니다.
- /form으로 요청이 왔을 때 폼을 들고 있는 view를 랜더링시키는 메서드를 갖고 있습니다.
- 이 때 model에 비어있는 객체를 넘기는데 이 부분이 중요합니다.

▶ Controller

```
@Getter @Setter
public class FormDto {
    private String name;
}

@Controller
public class UseFormController {
    @GetMapping("/form")
    public String showForm(Model model) {
        model.addAttribute("data", new FormDto());
        return "form-test/form";
    }
}
```

▶ 기본 폼 : form.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
                                        ..................
</head>
<body>
<div class="container">
    <div class="py-4 text-center">
        <h2>Test Form</h2>
```

```
    </div>
    <form th:action th:object="${data}" method="post">
        <div>
            <label for="name">이름</label>
            <input type="text" id="name" th:field="*{name}" class="form-control">
        </div>
        <button class="btn-primary" type="submit">전송</button>
    </form>
</div>
</body>
</html>
```

이제 전송해볼까요.
post 요청을 받아줄 컨트롤러 메서드 코드입니다.

참고로 Thymeleaf에 대한 내용은 아니지만 Model 객체를 사용하지 않고 view 단에 데이터를 넘길 수 있는 이유는 **@ModelAttribute** 덕분입니다.
이 어노테이션은 Model 객체를 생성하여 전달받은 객체를 넣어주는 역할까지 해줍니다.

▶ 폼에 전달된 내용을 확인하기 위해 로그설치
→ Controller 맨 처음에 **@Slf4j** 추가

▶ Controller

```
    @PostMapping("/form")
    public String form(@ModelAttribute("data") FormDto formDto) {
        log.info("formDTO.name = " + formDto.getName());
        return "form-test/form";
    }
```

```
o.s.web.servlet.DispatcherServlet       : Completed initialization in 0 ms
c.e.t.controller.UseFormController      : formDTO.name = 장원영
```

2) CheckBox 데이터 전송

- CheckBox 부분 html 코드입니다.
- 일반 text데이터를 전송하는 것과 거의 동일합니다.
- Thymeleaf의 th:field가 여기서 굉장히 많은 역할을 해줍니다.
- 내부적으로 hidden 필드를 만들어서 미체크시 문제가 되는 null 문제를 해결해주는 등의 역할을 해줍니다. 랜더링 후에 브라우저에서 소스 보기를 해보시면 입력하지 않은 히든타입 체크박스가 추가

되어 있을 것 입니다.

▶ 기본 폼 : form.html

```
.....
<div>CheckBox Test</div>
<div>
    <div class="form-check">
        <input type="checkbox" th:field="*{trueOrFalse}" class="form-check-input">
        <label for="trueOrFalse" class="form-check-label">True or False</label>
    </div>
</div>
<button class="btn-primary" type="submit">전송</button>
```

▶ Controller 및 Dto

```
@Getter @Setter
public class FormDto {
    private String name;
    private boolean trueOrFalse;
}
@PostMapping("/form")
    public String form(@ModelAttribute("data") FormDto formDto) {
        log.info("formDTO.name = " + formDto.getName());
        log.info("formDTO.trueOrFalse = " + formDto.isTrueOrFalse());
        return "form-test/form";
}
```

3) Multi-Checkbox 데이터 전송

- multi-checkbox 데이터를 받기 위한 필드를 폼 객체에 추가합니다.
- 여러 데이터가 들어올 수 있기 때문에 List 타입으로 해주었습니다.

▶ FormDto

```
@Getter @Setter
public class FormDto {
    private String name;
    private boolean trueOrFalse;
    private List<String> hobbies; // multi-checkbox
}
```

- 체크박스 key값과 value 값을 초기화하여 Model 객체에 담아주기 위해 메서드 레벨
- @ModelAttribute 를 사용해 봅니다.
- 메서드 레벨 @ModelAttribute를 컨트롤러에 작성해주었습니다.
- @ModelAttribute를 메서드 레벨에 붙이게 되면 리턴되는 값을 @ModelAttribute("name") 설정한 이름을 key값으로 하여 Model 객체에 담아줍니다.
- 각 매핑 메서드에서 model.addAttribute("hobbies", map); 를 한 것과 동일한 것 입니다.

▶ Controller

```
@ModelAttribute("hobbies")
private Map<String, String> favorite() {
    Map<String, String> map = new HashMap<>();
    map.put("movie", "영화보기");
    map.put("music", "음악듣기");
    map.put("running", "런닝하기");
    map.put("game", "게임하기");
    return map;
}
```

- HTML 부분을 작성해줍니다.
- 메서드 레벨 @ModelAttribute를 통해 구성한 map을 th:each 를 사용하여 반복하며 한 개씩 체크박스를 만들어줍니다.
- th:value에는 map의 key값이 담기게 되고 실제 서버로 전송되는 값이 됩니다.
- label 의 for부분에서 생소한 문법이 사용되었습니다.
- #{ids.prev('hobbies')} 는 th:field="*{hobbies}"에 의해 동적으로 생성되는 id 값에 맞춰서 값을 세팅해주는 기능을 제공합니다.
- 앞서 말했듯 th:field는 id를 자동으로 생성해줍니다.
- label의 for는 input의 id와 반드시 맞춰줘야 하기 때문에 위와 같은 코드를 사용하는 것입니다.
- 보통 ids 는 label태그에 사용하는데, 이 label이 form의 앞에 위치하는가 뒤에 위치하는가를 기준으로 next와 prev를 선택해서 사용합니다. label이 input 태그보다 먼저 오는 경우 next를, label이 input 태그보다 뒤에오는 경우 prev 를 사용합니다.
- #ids 는 타임리프가 자체적으로 제공하는 객체로 prev, next, seq 등의 메서드를 지원합니다.
- 마지막으로 사용자에게 보여줄 텍스트는 앞서 구성한 map의 value값을 사용하였습니다.

▶ HTML

```
<hr class="my-4">
        <div>
            <div>취미 선택 (다중 선택 가능)</div>
            <div th:each="hobby : ${hobbies}" class="form-check">
                <input type="checkbox" th:field="*{hobbies}" th:value="${hobby.key}"
                                                    class="form-check-input">
                <label th:for="${#ids.prev('hobbies')}" th:text="${hobby.value}"
                                                    class="form-check-label"></label>
            </div>
        </div>
```

▶ Post Controller

```
    @PostMapping("/form")
    public String form(@ModelAttribute("data") FormDto formDto) {
        log.info("formDTO.name = " + formDto.getName());
        log.info("formDTO.trueOrFalse = " + formDto.isTrueOrFalse());
        List<String> hobbies = formDto.getHobbies();
        for (String hobby : hobbies) {
            log.info("formDto.hobby = {}", hobby);
        }
        return "form-test/form";
    }
```

2. Radio Button, Select box

1) RadioButton 데이터 전송

- radio button 은 단일 선택만 지원하는 input태그의 옵션입니다.
- RadioButton 부분 HTML 코드입니다.
- 이전과 마찬가지로 th:field에 의해 동적으로 생성되는 input의 id를 label과 맞추기 위해 #ids.prev를 사용하였습니다.
- 이전에는 Map을 사용하였지만 이번에는 ENUM 타입을 사용하였습니다.

▶ FormDto

```
@Getter @Setter
public class FormDto {
    private String name;
    private boolean trueOrFalse;
    private List<String> hobbies; // multi-checkbox
    private String language; // radio-button
}
```

▶ Enum - Language

```
@Getter
public enum Language {
    JAVA("자바"), C("C언어"), CPP("C++"), PYTHON("파이썬");
    private final String description;
    Language(String description){
        this.description = description;
    }
}
```

▶ Controller

```
    @ModelAttribute("language")
    private Language[] languages() {
        return Language.values();
    }

    @PostMapping("/form")
    public String form(@ModelAttribute("data") FormDto formDto) {
        log.info("formDTO.name = " + formDto.getName());
        log.info("formDTO.trueOrFalse = " + formDto.isTrueOrFalse());
```

```
        List<String> hobbies = formDto.getHobbies();
        for (String hobby : hobbies) {
            log.info("formDto.hobby = {}", hobby);
        }
        log.info("Language = " + formDto.getLanguage());
        return "form-test/form";
    }
```

▶ Form

```
<div>
  <div>전공 선택 (한 개만 선택 가능)</div>
  <div th:each="lang : ${language}" class="form-check form-check-inline">
    <input type="radio" th:field="*{language}" th:value="${lang.name()}" class="form-check-input">
    <label th:for="${#ids.prev('language')}"
                    th:text="${lang.description}" class="form-check-label"></label>
  </div>
</div>
```

2) SelectBox 데이터 전송

- select-box는 radio-button과 마찬가지로 단건의 데이터 전송을 지원하는 태그입니다.
- select-box를 이용해서 국가 정보를 받아보겠습니다.
- 먼저 HTML코드 입니다.
- th:each(반복문)을 이용해 select의 각 값이 되는 option을 생성합니다.

▶ FormDto

```
@Getter @Setter
public class FormDto {
    private String name;
    private boolean trueOrFalse;
    private List<String> hobbies; // multi-checkbox
    private String language; // radio-button
    private String country; // select
}
```

▶ Enum - Country

```
@Getter
public enum Country {
    KOREA("대한민군"), USA("미국"), JAPAN("일본"),
    CHINA("중국"), SINGAPORE("싱가폴");

    private final String description;
    Country(String description) {
        this.description = description;
    }
}
```

- 정의한 ENUM 클래스를 view단에 넘겨 select-option 의 값으로 사용하기 위해 메서드 레벨의 @ModelAttribute를 사용합니다.

▶ Controller

```
    @ModelAttribute("country")
    private Country[] countries() {
        return Country.values();
    }

    @PostMapping("/form")
    public String form(@ModelAttribute("data") FormDto formDto) {
        log.info("formDTO.name = " + formDto.getName());
        log.info("formDTO.trueOrFalse = " + formDto.isTrueOrFalse());
        List<String> hobbies = formDto.getHobbies();
        for (String hobby : hobbies) {
            log.info("formDto.hobby = {}", hobby);
        }
        log.info("Language = " + formDto.getLanguage());
        log.info("Country = " + formDto.getCountry());
        return "form-test/form";
    }
```

▶ Form

```
<hr class="my-4">
<div>국가 선택</div>
    <select th:field="*{country}" class="form-select">
        <option value="">***** 국 가 를 선 택 하 세 요 *****</option>
        <option th:each="c : ${country}" th:value="${c.name()}"
                th:text="${c.description}"></option>
    </select>
</div>
```

▶ 다음을 폼으로 디자인 한 후 그 결과를 출력하는 프로그램을 작성하시오.

- FormControlController
- formControlView.html
- formControlResultView.html