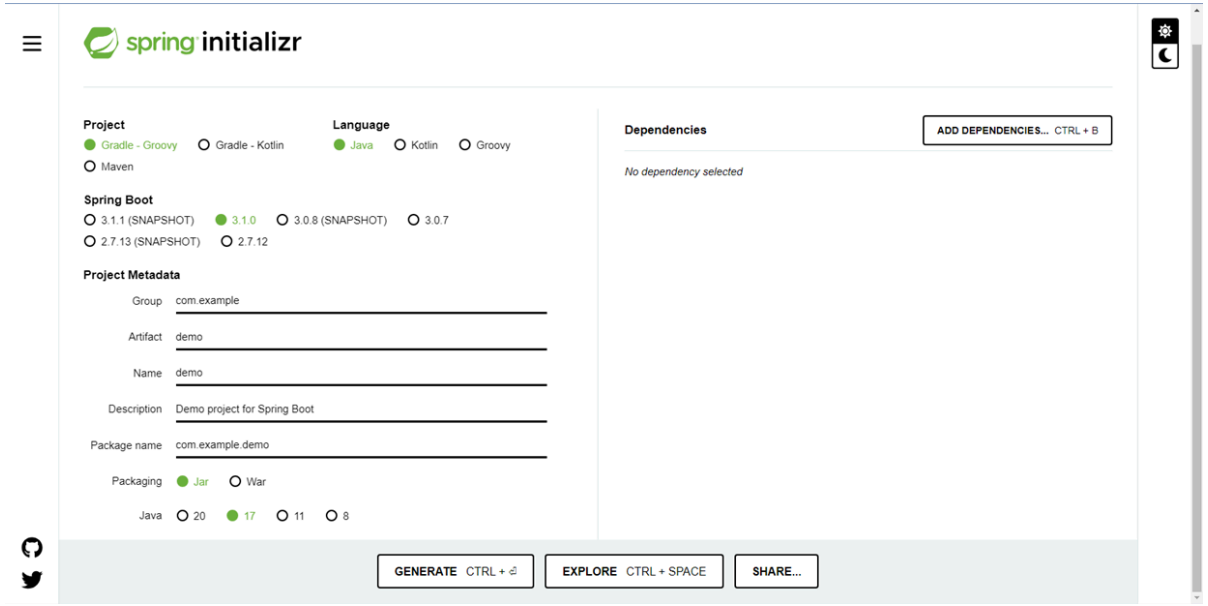
PBL기반의 Java Web 애플리케이션 개발

Spring Boot + Thymeleaf + DB(h2)

박종일

Spring Boot 프로젝트



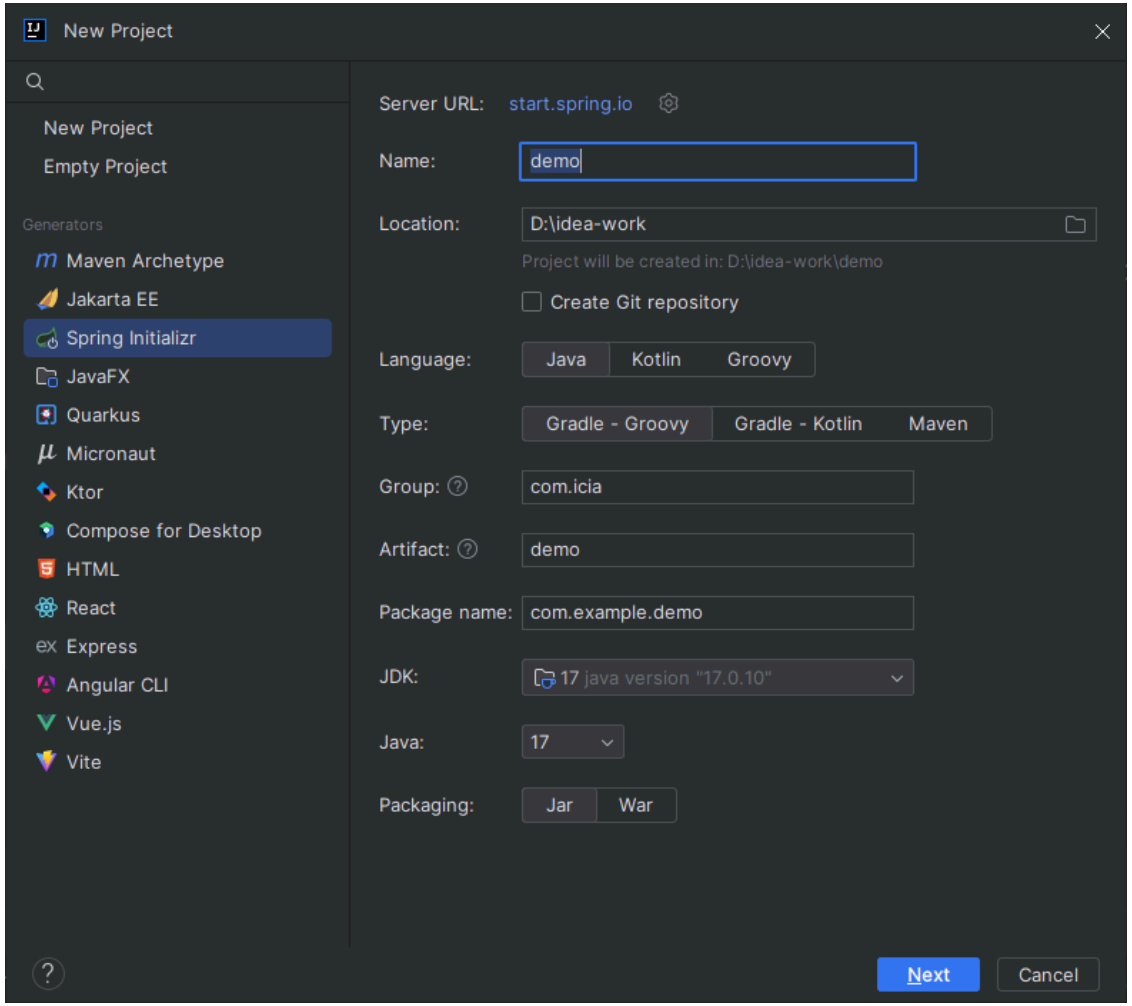
방식 1

https://start.spring.io/ 사이트를 통해 프로젝트를 생성하고, 다운로드하여 개발환경에서 불러오는 방법

- 모든 프로젝트는 이 사이트를 통해 생성되어 IDE로 다운로드된다.
- 즉, 인터넷이 연결되어 있지 않는 오프라인 환경에서는 프로젝트 생성도 불가

방식 2

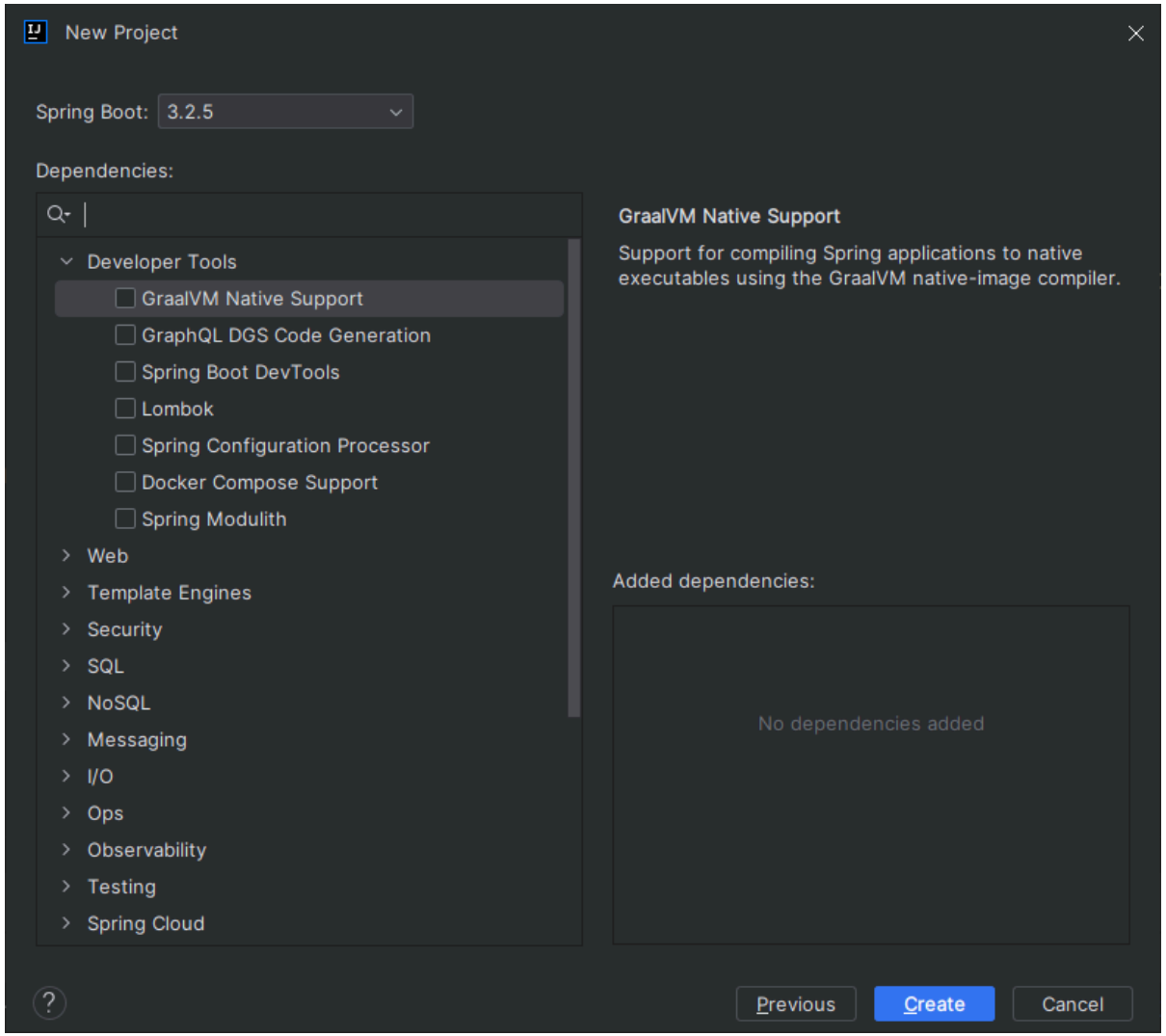
IDEA에서 프로젝트 생성



작성 항목

- Name : `start01`
- Location : `작업 폴더`
- Language : Java
- Type : Gradle - Groovy
- Group : com.`icia`
- Artifact : start01
- Package name : com.icia.start01
- JDK : 17
- Java : 17
- Packaging : Jar

 의 내용만 작성 및 지정 후 Next.

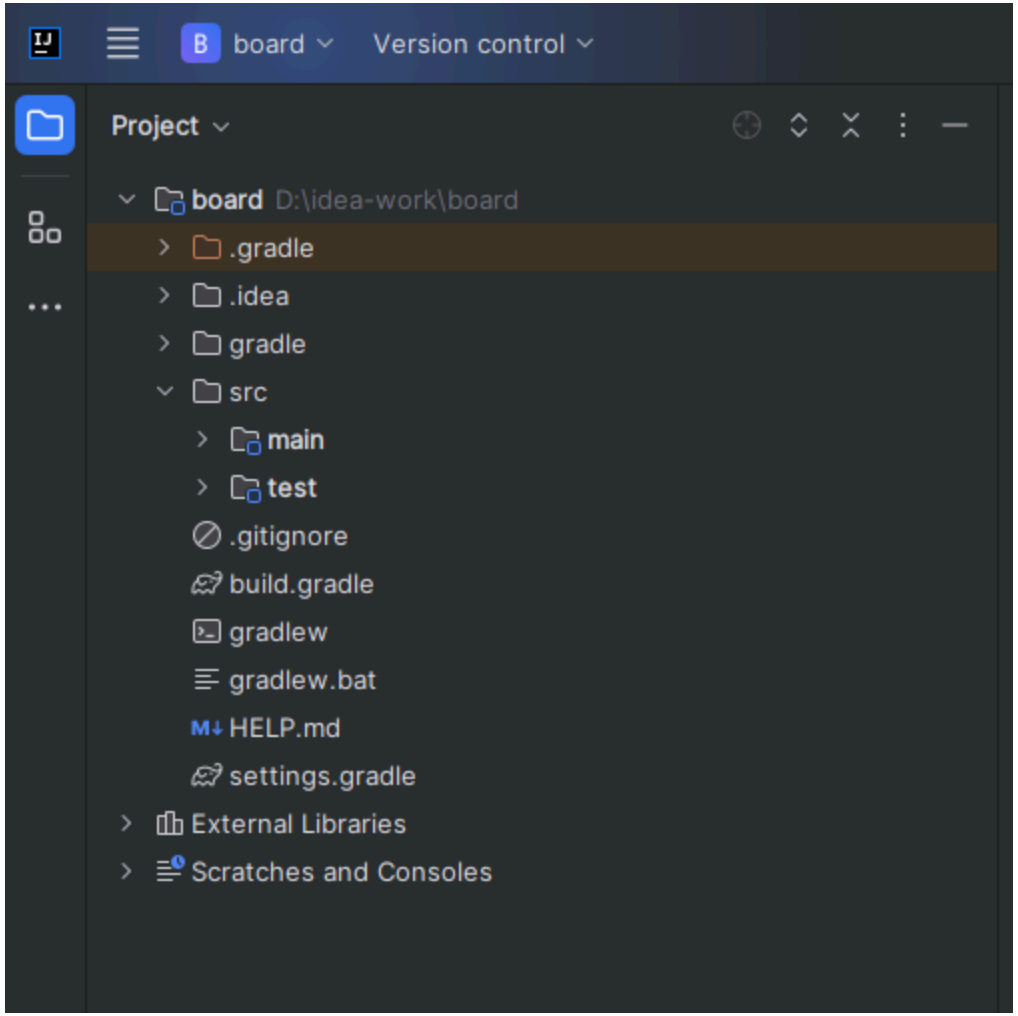


Dependency(초기 라이브러리 설정)

- Developer Tools
  - Spring Boot DevTools
  - Lombok
- Web
  - Spring Web
- Template Engines
  - Thymeleaf

선택 후 Create.

생성 완료 후 프로젝트 폴더 구조



초기 화면 처리 및 실행

> 작업 폴더 – src/main
> java 폴더 : java 소스 코드 작성.
> resources 폴더 : 이미지, 스타일시트, 자바스크립트, HTML 페이지 작성.

작성 내용

- ‘start01’에 Package 생성
  - controller
- controller 패키지에 Java Class 생성
  - HomeController
- static 폴더에 HTML file을 생성
  - index.html
- Start01Application 실행

Resources 폴더의 구성

static 폴더

배경이미지, 스타일시트, 자바스크립트 등 웹의 정적 자원(Static Resources)을 저장

static 폴더 밑에 각 자원을 위한 폴더를 생성하여 각각 저장

templates 폴더

HTML 템플릿을 저장하는 폴더로 접속자의 화면에 보여질 HTML 문서를 작성

Thymeleaf 등 템플릿 엔진을 사용하는 프로젝트에서 HTML 페이지를 작성

application.properties

정적 자원의 위치 지정, DB 설정, 파일 업로드, 에러 페이지 등 웹 프로그램 실행 시 필요한 설정 내용을 작성.

작성 코드

HomeController.java

```java
@Controller
public class HomeController {
    @GetMapping("/")
    public String home(){
        return "index.html";
    }
}
```

index.html

```html
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8" />
    <title>HOME</title>
  </head>
  <body>
    <h1>첫 페이지</h1>
    <p>처음으로 보이는 페이지</p>
  </body>
</html>
```

실행 환경 설정

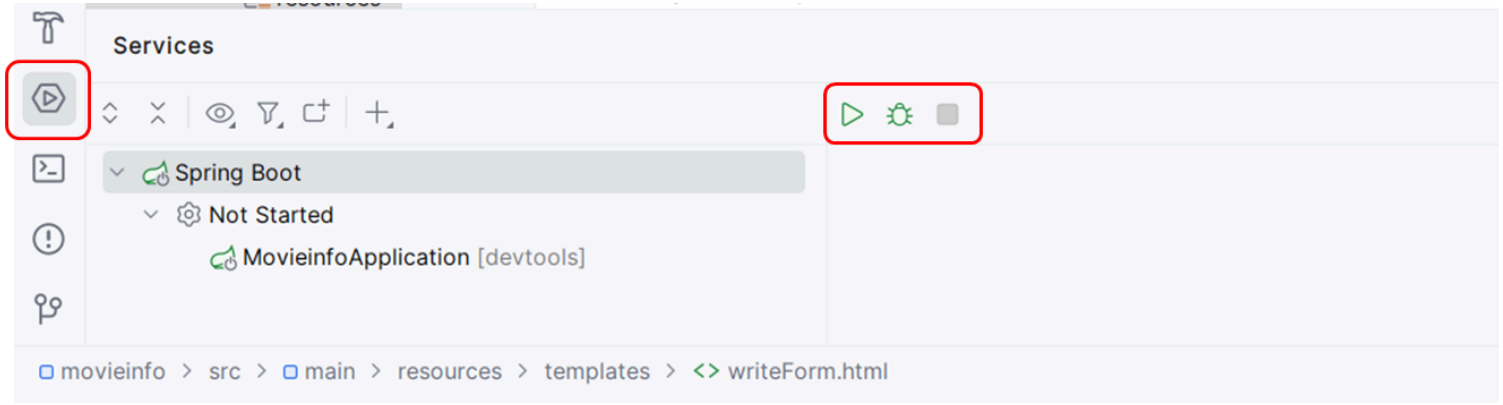
좌측하단의 `Services` 아이콘을 클릭.

- '+' (Add service) > Run Configuration Type > Spring Boot 선택.
- '▶️'를 눌러 실행.

> Lombok requires enabled annotation processing 창이 뜨면 Enable annotation processing 클릭

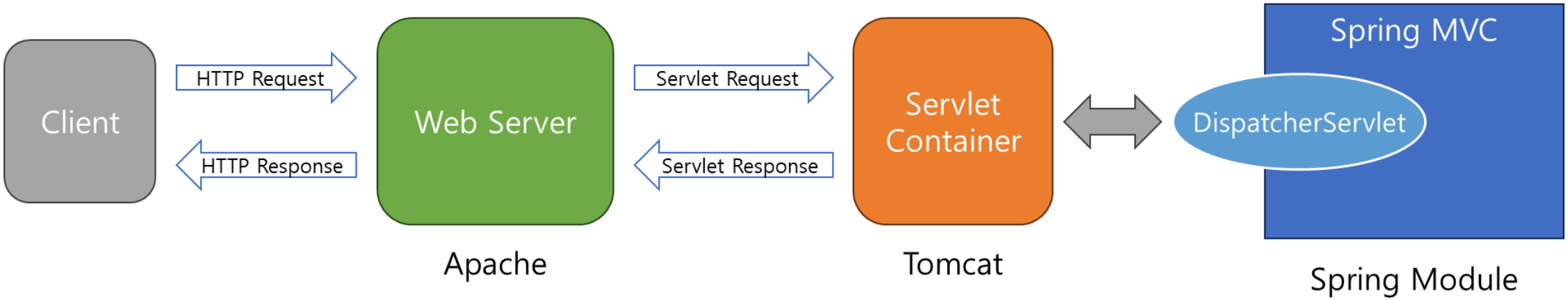
'Start01Application :8080/'에서 마지막 '8080/' 부분을 클릭.

- 브라우저로 실행

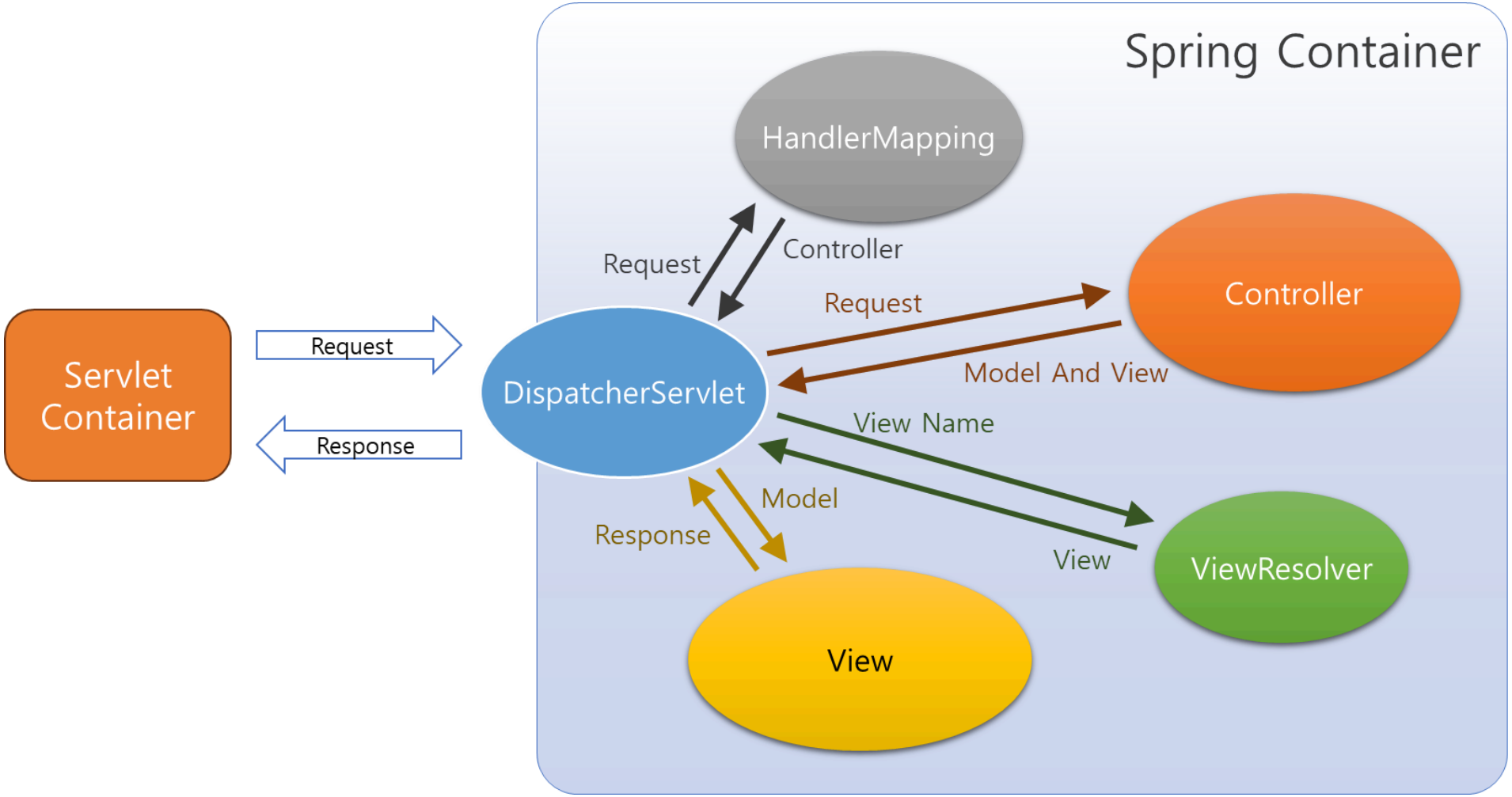


Spring MVC

Spring MVC 동작 구조는 다음과 같다.



사용자의 요청(Request)은 Web Server를 거쳐 Servlet Container의 Spring Module로 전달된다.



내부 동작 과정은 다음과 같다.

- DispatcherServlet은 HandlerMapping을 통해 해당 요청을 처리하는 Controller를 찾아 Request를 넘긴다.
- Controller는 요청 기능에 대한 서비스 로직을 수행하고 그 결과를 Model에 담아 DispatcherServlet에 반환하며, Model이 담길 View의 이름을 함께 반환한다.
- DispatcherServlet은 ViewResolver를 통해 해당 View를 찾아 Model을 담은 후 사용자에게 응답(Response)한다.

URI Mapping

사용자의 요청은 URI로 구분하여 Controller로 전달되며, Mapping된 메소드에서 처리된다.

Spring Framework는 Annotation 방식으로 URI와 메소드를 mapping한다.

- @RequestMapping
- @GetMapping
- @PostMapping

더 있지만 여기서는 주로 사용하는 3가지만 살펴본다.

@RequestMapping

Controller의 메소드를 Handler에 등록하기 위해 제공되는 annotation.

2가지 옵션으로 메소드의 mapping을 수행한다.

- value : mapping하는 uri를 작성. 필수 옵션
- method : http 전송 방식을 지정. 생략 시 GET 방식으로 설정.

```
@RequestMapping(value = "someUri", method = RequestMethod.GET)
public String ...
```

method 옵션으로 전송 방식(GET/POST)에 따라 메소드를 구분할 수 있으며, Get 방식의 전송일 경우 생략할 수 있다.
이 때는 vaule 키워드도 생략하여 uri만 작성한다.

```
@RequestMapping("someUri")
public String ...
```

@RequestMapping은 Controller Class에서도 활용가능하며, 이 때는 공통 uri를 지정할 수 있다.

예를 들어 customer site와 seller site로 controller를 구분하여 작성할 경우

```
@Controller
@RequestMapping("customer")
public class CustomerController {
...
```

```
@Controller
@RequestMapping("seller")
public class SellerController {
...
```

요청 uri는 `customer/xxx`, `seller/xxx`와 같으며, `xxx`부분은 각 메소드에서 따로 mapping하여 처리한다.

@GetMapping

Get 방식의 전송에 대응하는 메소드 mapping anntation.

value만 설정하면 된다.

```
@Controller
public class SomeController {
    @GetMapping("uri")
    public String uriMappingMethod(...
```

메소드 앞에만 붙일 수 있으며, class에 @RequestMapping이 붙은 경우 하위 uri와 mapping된다.

```
@Controller
@RequestMapping("customer")
public class CustomerController {
    @GetMapping("buy")
    public String uriMappingMethod(...
```

이 경우, 요청 uri는 `customer/buy`가 된다.

@PostMapping

Post 방식의 전송에 대응하는 메소드 mapping anntation.

@GetMapping과 마찬가지로 value만 설정하면 된다.

```
@Controller
public class SomeController {
    @PostMapping("uri")
    public String uriMappingMethod(...
```

메소드 앞에만 붙일 수 있으며, class에 @RequestMapping이 붙은 경우 하위 uri와 mapping된다.

```
@Controller
@RequestMapping("seller")
public class SellerController {
    @PostMapping("product")
    public String uriMappingMethod(...
```

이 경우, 요청 uri는 `seller/product`가 된다.

Model과 view

Back-end에서 Front-end로 데이터를 보낼 때 활용할 수 있는 두가지 객체를 제공한다.

Model 객체

Model 객체는 DispatcherServlet에서 보내주기 때문에 따로 생성하지 않고 해당 메소드의 매개변수로 받는다.

```
    @GetMapping("uri")
    public String someMethod(Model model){
        ...
        model.addAttribute("식별자", data);
        return "view_name";
    }
```

Model 객체는 return하지 않아도 DispatcherServlet에 전달된다.

따라서 메소드는 Model이 전달될 View의 이름만 return한다.

ModelAndView 객체

ModelAndView는 Model과 View를 합친 형태의 객체이다.

DispatcherServlet에서 보내주는 객체가 아니기 때문에 매개변수로 받지 않으며, 메소드 내에서 객체를 생성하여 사용한다.

```
    @GetMapping("uri")
    public ModelAndView someMethod(){
        ...
        ModelAndView mv = new ModelAndView();
        mv.addObject("식별자", data);
        mv.setViewName("view_name");

        return mv;
    }
```

Data를 담기위해 `addObject()`를 사용하며, View 이름의 지정을 위해 `setViewName()`을 사용한다.

Thymeleaf Templates

> Thymeleaf는 웹 및 독립형 환경 모두를 위한 최신 Server-Side Java 템플릿 엔진.
> Thymeleaf의 주요 목표는 개발 워크플로우에 우아하고 자연스러운 템플릿을 제공하는 것.

Spring Boot에 포함된 기본 Template 엔진 중 하나로 프로젝트 생성 시 바로 추가하여 사용할 수 있음.

Thymeleaf 활용

Namespace

Thymeleaf namespace를 <html> 태그에 명시해야 html 페이지 내에서 thymeleaf 코드가 동작한다.

```
<html lang=”ko” xmlns:th=”http://www.thymeleaf.org”>
```

기본 문법

html 태그 안에 th 문법을 추가

`<태그 th:[속성]=”데이터 및 표현식”></태그>`

표현식 기호

- ${식별자} : 일반적인 글자 데이터 출력(SpringEL)
- @{link} : a 태그의 href 속성이나 form 태그의 action 속성에서 uri 처리
- *{field} : server에서 dto 객체 형태로 전송할 때, dto 객체의 필드를 출력
  - 먼저 상위 요소에서 th:object로 객체를 지정
- 자바스크립트에서의 출력(html의 content에 직접 출력할 때도 사용)
  - `[[${식별자}]]`

링크 처리

- th:href - 링크 연결 속성. a 태그에서 사용
- th:action - form 태그의 action 속성 대신 사용

index.html

```
<p>처음으로 보이는 페이지</p>
<hr>
<a th:href="@{t_output}">타임리프 출력</a>
</body>
```

HomeController.java

```
    @GetMapping("/")
    public String home(){
        log.info("home()");
        return "index";
    }
```

Thymeleaf를 사용하는 경우 `return` 문에 작성하는 파일명은 **확장자**를 제외하고 작성한

데이터 출력

- innerText로 출력
  - th:text : html 시작태그 내에 사용
  - [[${식별자}]] : 시작태그와 종료태그 사이에서 사용
- innerHTML로 출력
  - th:utext - html 시작태그 내에 사용
  - [(${식별자})] : 시작태그와 종료태그 사이에서 사용

t_output.html

```
<h2>th:text와 th:utext의 차이</h2>
<div th:text="${testStr}"></div>
<div>[[${testStr}]]</div>
<div th:utext="${testStr}"></div>
<div>[(${testStr})]</div>
<hr />
```

HomeController.java

```
    @GetMapping("t_output")
    public String thymeleafOutput(Model model){
        log.info("thymeleafOutput()");

        //일반 데이터
        model.addAttribute("testStr",
                "<h3>h3로 만든 문자열</h3>");

        return "t_output";
    }
```

객체 처리

여러 데이터를 하나로 묶는 Map이나 Dto와 같은 객체의 처리

- Map 데이터
  - `map.put("data1", data);`
  - `model.addAttribute("식별자", map);`

t_output.html

```
<h2>Map 데이터 출력</h2>
<p th:text="${mapdata.data1}"></p>
<p th:text="${mapdata.data2}"></p>
<p th:text="${mapdata.data3}"></p>
```

HomeController.java

```
    @GetMapping("t_output")
    public String thymeleafOutput(Model model){
        ...
        //map 데이터 (묶음 데이터)
        Map<String, String> rmap = new HashMap<>();
        rmap.put("data1", "홍길동");
        rmap.put("data2", "20");
        rmap.put("data3", "인천시 미추홀구");

        model.addAttribute("mapdata", rmap);
        ...
    }
```

- DTO 객체
  - Map와 같은 형식으로 처리
  - th:object 활용
    - 하위 요소에서 *{field}로 각 데이터를 가져올 수 있음

t_output.html

```
<h2>Dto 데이터 출력 1</h2>
<p th:text="${dtoData.getName()}"></p>
<p th:text="${dtoData.age}"></p>
<p th:text="${dtoData.address}"></p>
<hr />
<h2>Dto 데이터 출력 2</h2>
<th:block th:object="${dtoData}">
  <p th:text="*{name}"></p>
  <p th:text="*{age}"></p>
  <p th:text="*{address}"></p>
</th:block>
```

HomeController.java

```
    @GetMapping("t_output")
    public String thymeleafOutput(Model model){
        ...
        //dto(또는 vo) 데이터
        DataDto dData = new DataDto();
        dData.setName("아무개");
        dData.setAge(30);
        dData.setAddress("인천시 동구");

        model.addAttribute("dtoData", dData);
        ...
    }
```

Dto/DataDto.java

```
@Setter
@Getter
public class DataDto {
    private String name;
    private int age;
    private String address;
}
```

스크립트 영역에서 처리

- th:inline - 스크립트 언어 지정
  - `<script th:inline=”javascript”>`
  - 스크립트 영역 내에서 Thymeleaf 문법을 활용할 수 있도록 함.

t_output.html

```
</body>
<script th:inline="javascript">
    let msg = [[${message}]];
    if(msg != null){
        alert(msg);
    }
</script>
</html>
```

HomeController.java

```
    @GetMapping("t_output")
    public String thymeleafOutput(Model model){
        ...
        //javascript 출력용 데이터
        model.addAttribute("message", "서버로부터의 메시지");
        ...
    }
```

Control

다음 페이지 이동을 위한 태그 및 코드, 페이지 추가

t_output.html

```
...
<hr>
<a th:href="@{t_control}">타임리프 제어문</a>
</body>
...
```

t_control.html

```
<body>
  <h1>Thymeleaf Control</h1>
</body>
```

HomeController.java

```
    @GetMapping("t_control")
    public String thymeleafControl(Model model){
        log.info("thymeleafControl()");

        return "t_control";
    }
```

제어용 태그

th:block

- 제어문(분기, 반복) 및 객체 설정에 사용

```
<th:block th:object="${dtoData}"></th:block>
```

제어 속성

1. 조건 제어 속성

- th:if - if문에 해당하는 속성(else는 없음)
- th:unless - if문의 반대에 해당하는 속성. 조건식을 th:if와 똑같이 작성하면 else문의 역할을 수행
- th:switch, th:case - switch, case에 해당하는 속성

t_control.html

```
...
<h2>th:if</h2>
<th:block th:if="${data != null}">
  <p>메시지 : [[${data}]]</p>
</th:block>
<th:block th:unless="${data != null}">
  <p>메시지가 없습니다.</p>
</th:block>
<hr />
<h2>th:switch</h2>
<th:block th:switch="${age/10}">
  <p th:case="2">20대입니다.</p>
  <p th:case="3">30대입니다.</p>
  <p th:case="4">40대입니다.</p>
  <p th:case="5">50대입니다.</p>
</th:block>
...
```

HomeController.java

```
    @GetMapping("t_control")
    public String thymeleafControl(Model model){
        ...
        model.addAttribute("data",
                "이 문자열이 보입니다.");
        model.addAttribute("age", 25);
        ...
    }
```

2. 반복 제어 속성

- th:each – for(foreach)문에 해당하는 속성
  - 반복 상태값(status) 활용 : `th:each="item,status:${list}"`
- 전송된 list를 출력하는 경우 th:if와 중첩하여 빈(empty) 목록인지 확인하는 형태로 활용

status

name	description
index	목록의 순번. 0부터 시작.
count	반복 횟수. 1부터 시작.
size	목록의 크기.
current	반복 중인 항목. th:if 속성과 결합하여 활용.
first	반복 중인 항목이 첫 번째인가의 여부. true/false
last	반복 중인 항목이 마지막인가의 여부. true/false
odd	반복 중인 항목이 홀수 번째인가의 여부. true/false
even	반복 중인 항목이 짝수 번째인가의 여부. true/false

t_control.html

```
...
<hr />
<h2>th:each</h2>
<table border="1">
  <thead>
    <tr>
      <th width="100">이름</th>
      <th width="50">나이</th>
      <th width="150">주소</th>
    </tr>
  </thead>
  <tbody>
    <th:block th:if="${#lists.isEmpty(dList)}">
      <tr>
        <th colspan="3">출력할 데이터가 없습니다.</th>
      </tr>
    </th:block>
  </tbody>
</table>
```

```
    <th:block th:unless="${#lists.isEmpty(dList)}">
      <th:block th:each="item:${dList}">
        <tr>
          <td th:text="${item.name}"></td>
          <td th:text="${item.age}"></td>
          <td th:text="${item.address}"></td>
        </tr>
      </th:block>
    </th:block>
  </tbody>
</table>
...
```

- `#lists.isEmpty(list)` : list가 비어있는지의 여부 확인.

HomeController.java

```
    @GetMapping("t_control")
    public String thymeleafControl(Model model){
        ...
        List<DataDto> d_list = new ArrayList<>();
        for(int i = 0; i < 5; i++){
            DataDto data = new DataDto();
            data.setName("이름" + i);
            data.setAge(25 + i);
            data.setAddress("주소" + i);
            d_list.add(data);
        }
        model.addAttribute("dList", d_list);
        ...
    }
```

Thymeleaf Expression Object - Basic, Utility Object

참고 : Thymeleaf Expression

Template 확장

th:fragment

- header나 nav, footer 같은 페이지 공통 부분의 처리를 위한 속성
- 하나의 html 파일에 여러 분할 영역을 작성
- 공통 부분이 필요한 페이지에서 활용

다음과 같은 형식으로 작성

```
th:fragment="fragment_name"
```

th:fragment 속성을 갖는 태그는 th:block, div, p 등 다양하게 활용 가능하다.

fragments.html

```
<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <title>Title</title>
  </head>
  <body>
    <th:block th:fragment="header1">
      <h2>Header1입니다.</h2>
    </th:block>
  </body>
</html>
```

Fragment 활용 속성

- th:insert - 태그 안으로 fragment 요소를 삽입

```
<header th:insert=..."></header>
```

```
▼<header> == $0
    <h2>테스트용 헤더</h2>
  </header>
```

- th:replace – 태그를 fragment 요소로 교체

```
<header th:replace=..."></header>
```

```
<h2>테스트용 헤더</h2>
```

Fragment 표현식

특정 페이지에서 fragment를 불러오는 표현식은 다음과 같다.

```
<tag th:insert"~{fragment_file::fragment_name}"></tag>
```

index.html

```
<body>
  ...
  <header th:insert="~{fragments::header1}"></header>
  ...
</body>
```

Fragment로 데이터 전송

th:fragment 속성의 fragment_name 뒤에 데이터를 받을 식별자를 작성한다.

th:text나 th:utext 등으로 데이터를 출력한다.

fragment_file.html

```
<th:block th:fragment="fragment_name(식별자)">
  <p th:text="${식별자}"></p>
</th:block>
```

fragment를 활용하는 페이지에서는 다음과 같이 데이터를 주입한다.

some.html

```
<header th:insert="~{fragment_file::fragment_name(data)}"></header>
```

fragments.html

```
<body>
  ...
  <th:block th:fragment="header2(data)">
    <h2 th:text="${data}"></h2>
  </th:block>
</body>
```

index.html

```
<body>
  ...
  <header th:replace="~{fragments::header2('2번째 헤더')}"></header>
  ...
</body>
```

Back-end에서 데이터 수신

@RequestParam

Query string과 method 매개변수를 매칭시키기 위한 어노테이션

```
    public String method(@RequestParam(“name”) type variableName) {
        ...
```

Html에서의 전송 문장이 다음과 같다면

```
<a th:href="“@{uri(n1" ="value)}">전송</a>
```

Controller는 다음과 같이 처리한다.

```
    public String methodName(@RequestParam(“n1”) String n){
        ...
```

예제

t_control.html에 전송 예제용 페이지로 이동하는 링크를 작성

```
...
<a th:href="@{sendData}">데이터전송 ></a>
</body>
```

HomeController.java에 해당 페이지로 이동하는 매핑 메소드를 작성

```
    @GetMapping("sendData")
    public String sendData(){
        return "sendData";
    }
```

sendData.html

```
<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <title>Send Form</title>
  </head>
  <body>
    <h1>Client -> Server</h1>
    <h2>a 태그 활용</h2>
    <p>전송 한 데이터 : 10, 20</p>
    <a th:href="@{a_send(num1=10,num2=20)}">전송</a>
  </body>
</html>
```

HomeController.java

```
    @GetMapping("a_send")
    public ModelAndView aTagDataSend(@RequestParam("num1") String num1,
                                     @RequestParam("num2") int num2){
        System.out.println("num1 : " + num1 + ", num2 : " + num2);
        ModelAndView mv = new ModelAndView();
        int n1 = Integer.parseInt(num1);
        mv.addObject("result", n1 + num2);
        mv.setViewName("s_result");
        return mv;
    }
```

Dto(전송 데이터 저장용) 객체 활용

Query string 또는 input 태그의 name 속성의 값과 동일한 멤버 변수명을 사용할 경우 같은 query string 또는 form으로 전송되는 모든 데이터를 한번에 처리할 수 있음

이 때는 @RequestParam을 작성하지 않는다.

```
    public String methodName(DtoClass dto) {
        ...
```

예제

sendData.html

```
...
<hr>
<h2>form 데이터 전송</h2>
<form th:action="@{noneDtoSend}">
    <input type="text" name="name"><br>
    <input type="number" name="age"><br>
    <input type="text" name="address"><br>
    <input type="submit" value="전송">
</form>
<hr>
<h2>form 데이터 전송 - Dto 사용</h2>
<form method="post" th:action="@{dtoSend}">
    <input type="text" name="name"><br>
    <input type="number" name="age"><br>
    <input type="text" name="address"><br>
    <input type="submit" value="전송">
</form>
</body>
```

HomeController.java

```
    @GetMapping("noneDtoSend")
    public String noneDtoSend(@RequestParam("name") String name,
                              @RequestParam("age") int age,
                              @RequestParam("address") String address,
                              Model model){
        System.out.println("name : " + name);
        System.out.println("age : " + age);
        System.out.println("address : " + address);

        model.addAttribute("result", "none dto send OK");
        return "s_result";
    }

    @PostMapping("dtoSend")
    public String dtoSend(DataDto data, Model model){
        System.out.println("name : " + data.getName());
        System.out.println("age : " + data.getAge());
        System.out.println("address : " + data.getAddress());

        model.addAttribute("result", "dto send OK");
        return "s_result";
    }
```

s_result.html

```
<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <title>Send Data Result</title>
  </head>
  <body>
    <h1>Send Data Result 페이지</h1>
    <th:block th:if="${result != null}">
      <p th:text="'결과 : ' + ${result}"></p>
    </th:block>
    <hr />
    <a th:href="@{sendData}">< 뒤로</a>
  </body>
</html>
```