Thymeleaf(타임리프) 기본 사용법 정리

1. 프로젝트 생성 및 기본 세팅

2. 타임리프 소개

3. 기본 표현식

4. 텍스트 - text, utext

5. 변수 - SpringEL

6. 유틸리티 객체와 날짜

7. URL 링크

8. 리터럴

9. 연산

10. 속성 값 설정

11. 반복

12. 조건부 평가

13. 주석

14. 블록

15. 자바스크립트 인라인

1. 프로젝트 생성 및 기본 세팅

- 프로젝트 만들기

1) Spring initializr: https://start.spring.io/

Project: Gradle-Groovy
Language: Java
Spring Boot: 3.1.10
Project Metadata:
- Group: com.example
- Artifact: thymeleafTest
- Name: thymeleafTest
- Description: Demo project for Spring Boot
- Package name: com.example.thymeleafTest
- Packaging: Jar
- Java: 17

Dependencies:
- Spring Web
- Lombok
- Thymeleaf
- spring-boot-devtools

2) Build,Execution,Deployment → Build Tools → Gradle → Build and run using && Run tests using → IntenlliJ IDEA

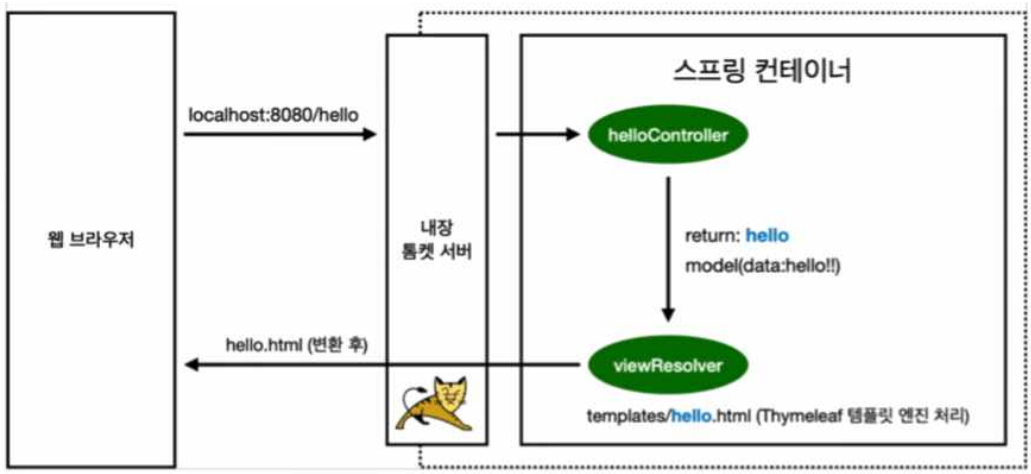
- spring-boot-devtools를 사용하여 소스코드 자동반영 설정하기(intelliJ)

1) File → Settings → Advanced Settings →
[Allow auto-make to start even if developed application is currently running] → check

2) Build,Execution,Deployment → Compiler → [Build project automatically] → check
3) 크롬 웹스토어에서 LiveReload ++ 설치하기
→ 크롬 우 상단 햄버거메뉴 → 확장프로그램 → Chrome 웹스토어 방문하기 → 개발자도구 → liveReload 검색 → LiveReload++ 설치

- Encoding 설정

Settings → encoding 검색 → File Encodings → Globla Encoding (UTF8) → Project Encoding (UTF8) → Default encoding for properties files (UTF8) → 아래 Transparent native-to-ascii conversion [[[check]]]



2. 타임리프 소개

1) Thymeleaf란?
- Thymeleaf는 웹 및 독립실행형 환경 모두를 위한 Java 템플릿 엔진이다.
- Thymeleaf로 작성된 HTML 템플릿은 애플리케이션을 통한 실행이 아니더라도 브라우저에서 열었을 때 HTML 내용을 그대로 확인이 가능하여 네츄럴 템플릿(Natural Templates)이라고 불린다.

※ 공식 사이트: https://www.thymeleaf.org/

2) Thymeleaf의 특징
- 서버 사이드 HTML 렌더링(SSR)
- 타임리프는 백엔드 서버에서 HTML을 동적으로 렌더링하는 용도로 사용된다.
- 러닝커브가 높지 않아 금방 학습할 수 있으며, 페이지가 어느 정도 동적이고 빠른 생산성이 필요한 화면을 만드는 경우 좋은 선택지가 될 수 있다.

3) 네츄럴 템플릿
- 타임리프는 순수한 HTML을 최대한 유지하려는 특징이 있다.
- 타임리프로 작성한 파일은 HTML을 유지하기 때문에 웹 브라우저에서 파일을 직접 열어도 내용을 확인할 수 있고, 서버를 통해 뷰 템플릿을 거치면 동적으로 변경된 결과를 확인할 수 있다.
- JSP를 포함한 다른 뷰 템플릿은 해당 파일 자체를 웹 브라우저로 열어보면 JSP 소스코드와 HTML이 섞여있어 웹 브라우저에서 정상적인 HTML 결과를 확인할 수 없다. 오직 서버를 통해서 JSP가 렌더링 되고 HTML 응답 결과를 받아야 화면이 확인할 수 있다.
- 순수 HTML을 그대로 유지하면서 뷰 템플릿도 사용할 수 있는 특징을 Natural Template이라고 한다.

4) 스프링 통합 지원
- 타임리프는 스프링과 자연스럽게 통합되어 스프링의 다양한 기능을 쉽게 사용할 수 있다.

3. 기본 표현식

- 기본 표현식
  - 타임리프는 다음과 같은 기본 표현식들을 제공합니다.

- 간단한 표현:
  - 변수 표현식: ${...}
  - 선택 변수 표현식: *{...}
  - 메시지 표현식: #{...}
  - 링크 URL 표현식: @{...}
  - 조각 표현식: ~{...}
- 리터럴
  - 텍스트: 'one text', 'Another one!',...
  - 숫자: 0, 34, 3.0, 12.3,...
  - 불린: true, false
  - 널: null
  - 리터럴 토큰: one, sometext, main,...
- 문자 연산:
  - 문자 합치기: +
  - 리터럴 대체: |The name is ${name}|
- 산술 연산:
  - Binary operators: +, -, *, /, %
  - Minus sign (unary operator): -
- 불린 연산:
  - Binary operators: and, or
  - Boolean negation (unary operator): !, not
- 비교와 동등:
  - 비교: >, <, >=, <= (gt, lt, ge, le)
  - 동등 연산: ==, != (eq, ne)
- 조건 연산:
  - If-then: (if) ? (then)
  - If-then-else: (if) ? (then) : (else)
  - Default: (value) ?: (defaultvalue)

공식 문서: https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#standard-expression-syntax

4. 텍스트 - text, utext

타임리프는 기본적으로 HTML 태그의 속성에 기능을 정의해서 동작한다.
HTML의 컨텐츠에 데이터를 출력할 때는 HTML 속성에 타임리프 속성만 추가, 만약, HTML 태그의 속성이 아닌 HTML 콘텐츠 영역 안에서 직접 데이터를 출력하고 싶다면 '[[...]]'를 사용한다.

※ 주의: th:text와 [[...]]은 기본적으로 이스케이프(escape)를 제공합니다. 즉, 서버에서 데이터로 HTML 태그 내용을 담아서 보낸 경우 '<', '>'와 같이 치환되어 내용이 출력되게 됩니다. 이러한 경우 타임리프는 Unescape 하게 사용하여 HTML 태그 자체로 출력하기 위한 th:utext와 [(...)] 을 제공합니다.

이스케이프: 웹 브라우저는 '<'를 HTML 태그의 시작으로 인식한다. 때문에, '<'를 태그의 시작이 아니라 문자로 표현할 수 있는 방법이 필요한데 이것을 HTML 엔티티라고 한다. 그리고 HTML에서 사용하는 특수 문자를 HTML 엔티티로 변경하는 것을 이스케이프(escape)라고 한다.

참고사이트 : https://dev.w3.org/html5/html-author/charref

● th:text, th:utext

```
<span th:text="${data}">
<span th:utext="${data}">
```

● [[...]], [(...)]

```
<span>hello [[${data}]]</span>
<span>hello [(${data})]</span>
```

▶ 예제코드

```
@Controller
@RequestMapping("/basic")
public class BasicController {

    @GetMapping("/text-basic")
    public String textBasic(Model model) {
        model.addAttribute("data", "Hello <b>Spring!</b>");
        return "basic/text-basic";
    }
}
```

th:inline="none": 타임리프는 [[...]]를 해석하기 때문에, 화면에 [[...]] 글자를 보여줄 수 없다. 때문에, 해당 태그 안에서는 타임리프가 해석안하도록 하는 옵션이다.

▶ text-basic.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>text vs utext</h1>
<ul>
  <li>th:text = <span th:text="${data}"></span></li>
  <li>th:utext = <span th:utext="${data}"></span></li>
</ul>
<h1><span th:inline="none">[[...]] vs [(...)]</span></h1>
<ul>
  <li><span th:inline="none">[[...]] = </span>[[${data}]]</li>
  <li><span th:inline="none">[(...)] = </span>[(${data})]</li>
</ul>
</body>
</html>
```

5. 변수 - SpringEL

1) 타임리프에서 Model에 담은 값을 꺼내거나, 타임리프 내부에 선언된 변수 사용 등 변수를 사용할 때는 변수 표현식을 사용한다.

2) 스프링이 제공하는 변수 표현식(스프링 EL)

- 단순 변수
  - data

- Object
  - user.username : user의 username을 프로퍼티 접근 user.getUsername()
  - user['username'] : 위와 같음 user.getUsername()
  - user.getUsername() : user의 getUsername() 을 직접 호출

- List
  - users[0].username : List에서 첫 번째 회원을 찾고 username 프로퍼티 접근
  - users[0]['username'] : 위와 같음
  - users[0].getUsername() : List에서 첫 번째 회원을 찾고 메서드 직접 호출
  - list.get(0).getUsername(): List의 get 메서드를 통해 데이터를 찾아 username 프로퍼티 접근

- Map
  - userMap['userA'].username : Map에서 userA를 찾고, username 프로퍼티 접근
  - userMap['userA']['username'] : 위와 같음
  - userMap['userA'].getUsername() : Map에서 userA를 찾고 메서드 직접 호출
  - map.get("userA").getUsername(): Map에서 get 메서드를 통해 userA 데이터를 찾아 username 프로퍼티 접근

- 3) 지역 변수 선언(**th:with**) : th:with를 사용하면 지역 변수를 선언해서 사용할 수 있다. 하지만, 지역 변수이기 때문에 선언한 태그 안에서만 사용 가능하다.

아래 코드는 'item'이라는 변수에 list[0]의 데이터를 담아 사용하는 예제이다.

```
<div th:with="item=${list[0]}">
        <ul>
                <li>이름 : <span th:text="${item.username}"></span></li>
                <li>나이 : [[${item.age}]]</li>
        </ul>
</div>
```

▶ 예제 코드

```
    @Controller
    @RequestMapping("/basic")
    public class BasicController {

    @GetMapping("/variable")
    public String variable(Model model) {
        User userA = new User("userA", 10);
        User userB = new User("userB", 20);

        List<User> list = new ArrayList<>(Arrays.asList(userA, userB));

        Map<String, User> map = new HashMap<>();
        map.put("userA", userA);
        map.put("userB", userB);

        model.addAttribute("user", userA);
        model.addAttribute("users", list);
        model.addAttribute("userMap", map);

        return "basic/variable";
    }
------------------------------  package name : vo ------------------------------------------
    @Data
    static class User {
        private String username;
        private int age;
        public User(String username, int age) {
            this.username = username;
            this.age = age;
        }
    }
}
```

▶ variable.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>SpringEL 표현식</h1>
<ul>Object
  <li>${user.username} = <span th:text="${user.username}"></span></li>
  <li>${user['username']} = <span th:text="${user['username']}"></span></li>
  <li>${user.getUsername()} = <span th:text="${user.getUsername()}"></span></li>
</ul>
<ul>List
  <li>${users[0].username} = <span th:text="${users[0].username}"></span></li>
  <li>${users[0]['username']} = <span th:text="${users[0]['username']}"></span></li>
  <li>${users[0].getUsername()} = <span th:text="${users[0].getUsername()}"></span></li>
</ul>
<ul>Map
  <li>${userMap['userA'].username} = <span th:text="${userMap['안유진'].username}"></span></li>
  <li>${userMap['userA']['username']} = <span th:text="${userMap['장원영']['username']}">
                                        </span></li>
  <li>${userMap['userA'].getUsername()} = <span th:text="${userMap['안유진'].getUsername()}">
                                        </span></li>
</ul>
<div th:with="item=${users[0]}">
  <ul>
    <li>이름 : <span th:text="${item.username}"></span></li>
    <li>나이 : [[${item.age}]]</li>
  </ul>
</div>
</body>
</html>
```

6. 유틸리티 객체와 날짜

타임리프는 문자, 숫자, 날짜, URI 등을 편리하게 다루는 다양한 유틸리티 객체들을 제공한다.

- 공식 문서:
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#expression-utility-objects

- 공식 문서 예제:
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#appendix-b-expression-utility-objects

```
#message : 메시지, 국제화 처리
#uris : URI 이스케이프 지원
#dates : java.util.Date 서식 지원
#calendars : java.util.Calendar 서식 지원
#temporals : 자바8 날짜 서식 지원
#numbers : 숫자 서식 지원
#strings : 문자 관련 편의 기능
#objects : 객체 관련 기능 제공
#bools : boolean 관련 기능 제공
#arrays : 배열 관련 기능 제공
#lists , #sets , #maps : 컬렉션 관련 기능 제공
```

1) Java8 날짜

- 스프링 부트 타임리프를 사용하면 해당 라이브러리가 자동으로 추가되고 통합
- '**#temporals**'로 LocalDateTime 객체를 포맷팅하거나 여러가지 원하는 형식으로 바꿔서 출력 가능

▶ Java8 날짜 예제 코드

```
@GetMapping("/date")
    public String date(Model model) {
        model.addAttribute("localDateTime", LocalDateTime.now());
        return "date";
    }
```

▶ date.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
```

```
  <title>Title</title>
</head>
<body>
<h1>LocalDateTime</h1>
<ul>
  <li>default = <span th:text="${localDateTime}"></span></li>
  <li>yyyy-MM-dd HH:mm:ss =
<span th:text="${#temporals.format(localDateTime, 'yyyy-MM-dd HH:mm:ss')}"></span></li>
  <li>yyyy-MM-dd/a hh:mm =
<span th:text="${#temporals.format(localDateTime, 'yyyy-MM-dd/a hh:mm')}"></span></li>
</ul>
<h1>LocalDateTime - Utils</h1>
<ul>
  <li>${#temporals.day(localDateTime)} =
                <span th:text="${#temporals.day(localDateTime)}"></span></li>
  <li>${#temporals.month(localDateTime)} =
                <span th:text="${#temporals.month(localDateTime)}"></span></li>
  <li>${#temporals.monthName(localDateTime)} =
                <span th:text="${#temporals.monthName(localDateTime)}"></span></li>
  <li>${#temporals.monthNameShort(localDateTime)} =
                <span th:text="${#temporals.monthNameShort(localDateTime)}"></span></li>
  <li>${#temporals.year(localDateTime)} =
                <span th:text="${#temporals.year(localDateTime)}"></span></li>
  <li>${#temporals.dayOfWeek(localDateTime)} =
                <span th:text="${#temporals.dayOfWeek(localDateTime)}"></span></li>
  <li>${#temporals.dayOfWeekName(localDateTime)} =
                <span th:text="${#temporals.dayOfWeekName(localDateTime)}"></span></li>
  <li>${#temporals.dayOfWeekNameShort(localDateTime)} =
                <span th:text="${#temporals.dayOfWeekNameShort(localDateTime)}"></span></li>
  <li>${#temporals.hour(localDateTime)} =
                <span th:text="${#temporals.hour(localDateTime)}"></span></li>
  <li>${#temporals.minute(localDateTime)} =
                <span th:text="${#temporals.minute(localDateTime)}"></span></li>
  <li>${#temporals.second(localDateTime)} =
                <span th:text="${#temporals.second(localDateTime)}"></span></li>
  <li>${#temporals.nanosecond(localDateTime)} =
                <span th:text="${#temporals.nanosecond(localDateTime)}"></span></li>
</ul>
</body>
</html>
```

7. URL 링크

- 타임리프에서 URL을 생성할 때는 '@{...}' 문법을 사용
- 공식 문서: https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#link-urls

- 단순한 URL
  - @{/hello}
    → /hello

- 쿼리 파라미터 (Query Parameter)
  - @{/hello(param1=${param1}, param2=${param2})}
    → /hello?param1=data1&param2=data2
    → () 에 있는 부분은 쿼리 파라미터로 처리된다.

- 경로 변수 (Path Variable)
  - @{/hello/{param1}/{param2}(param1=${param1}, param2=${param2})}
    → /hello/data1/data2
    → URL 경로상에 변수가 있으면 () 부분은 경로 변수로 처리된다.

- 경로 변수 + 쿼리 파라미터
  - @{/hello/{param1}(param1=${param1}, param2=${param2})}
    → /hello/data1?param2=data2
    → 경로 변수와 쿼리 파라미터를 함께 사용할 수 있다.
    → 경로 변수로 정의되지 않고 남은 변수들은 자동으로 쿼리 파라미터로 붙는다.

- 경로 표현 방법: 상대경로, 절대경로, 프로토콜 기준을 표현할 수 도 있다.

/hello : 절대 경로
hello : 상대 경로

▶ 예제 코드

```
@GetMapping("/link")
public String link(Model model) {
    model.addAttribute("name", "장원영");
    model.addAttribute("age", 21);
    return "link";
}
```

▶link.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>URL 링크</h1>
<ul>
  <li><a th:href="@{/hello}">basic url</a></li>
  <li><a th:href="@{/hello(name=${name}, age=${age})}">query param</a></li>
  <li><a th:href="@{/hello/{name}/{age}(name=${name}, age=${age})}">path variable</a></li>
  <li><a th:href="@{/hello/{name}(name=${name},
                                  age=${age})}">path variable + query parameter</a></li>
</ul>
</body>
</html>
```

8. 리터럴

- 리터럴은 소스 코드상에서 고정된 값을 말함.
- 타임리프는 다음과 같은 리터럴이 존재

- 문자: 'hello'
- 숫자: 10
- 불린: true , false
- null: null

문자 리터럴은 항상 작은 따옴표(')로 감싸주어야 한다. 단, 리터럴 내에서 공백이 없다면 작은 따옴표 생략 가능

```
<!-- 문자 리터럴 표현 -->
<span th:text="'text'"></span>

<!-- 작은 따옴표 생략 가능 -->
<span th:text="HelloWorld"></span>

<!-- 작은 따옴표 생략 불가: 문자 내에 공백이 존재하기 때문에 생략 불가 -->
<span th:text="'Hello World'"></span>
```

리터럴 대체 문법

- '|'(세로바) 을 감싸서 사용하면 리터럴 대체(Literal substitutions)가 가능하여 마치 템플릿을 사용하는 것 처럼 편리하게 사용 가능

```
<span th:text="|hello ${data}|"></span>
```

▶ 예제코드

```
@GetMapping("/literal")
public String literal(Model model) {
    model.addAttribute("data", "Spring!");
    return "literal";
}
```

▶ literal.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body><h1>리터럴</h1>
<ul>
  <!--주의! 다음 주석을 풀면 예외가 발생함-->
<!--        <li>"hello world!" = <span th:text="'hello world!'"></span></li>-->
  <li>'hello' + ' world!' = <span th:text="'hello' + ' world!'"></span></li>
  <li>'hello world!' = <span th:text="'hello world!'"></span></li>
  <li>'hello ' + ${data} = <span th:text="'hello ' + ${data}"></span></li>
  <li>리터럴 대체 |hello ${data}| = <span th:text="|hello ${data}|"></span></li>
</ul>
</body>
</html>
```

9. 연산

- 타임리프의 연산은 자바의 연산과 크게 다르지 않으며, 다만 HTML 엔티티를 사용하는 부분만 주의해서 사용하면 된다.

- 산술 연산
  - 자바의 연산과 유사하다.
  -
  -

- 비교연산
  - HTML 엔티티를 사용해야 하는 부분을 주의하자
  → (gt), < (lt), >= (ge), <= (le), ! (not), == (eq), != (neq, ne)
  -
  -
  -
  -

- 조건식
  - 자바의 조건식과 유사하다.
  - [삼항 연산자]:

- Elvis 연산자
  - 데이터가 없을 때 사용할 수 있는 연산자 (조건식의 편의 버전)
  - 데이터가 있으면 데이터를 출력하고, 없으면 우측에 있는 값을 출력
  -

- No-Operation
  - _ 인 경우 마치 타임리프가 실행되지 않는 것처럼 동작한다.
  - 이것을 잘 사용하면 HTML의 내용 그대로 활용할 수 있다.
  - 즉, No-Operation('_')을 사용하면 타임리프 태그를 렌더링하지 않고 안에 내용이 그대로 출력된다.
  - Default Value

▶ 예제코드

```
@GetMapping("/operation")
public String operation(Model model) {
    model.addAttribute("nullData", null);
    model.addAttribute("name", "안유진");
    return "operation";
}
```

▶ operation.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<ul>
  <li>산술 연산
    <ul>
      <li>10 + 2 = <span th:text="10 + 2"></span></li>
      <li>10 % 2 == 0 = <span th:text="10 % 2 == 0"></span></li>
    </ul>
  </li>
  <li>비교 연산
    <ul>
      <li>1 < 10 = <span th:text="1 lt 10"></span></li>
      <li>1 &gt; 10 = <span th:text="1 gt 10"></span></li>
      <li>1 lt 10 = <span th:text="1 < 10"></span></li>
      <li>1 >= 10 = <span th:text="1 >= 10"></span></li>
      <li>1 ge 10 = <span th:text="1 ge 10"></span></li>
      <li>1 == 10 = <span th:text="1 == 10"></span></li>
      <li>1 != 10 = <span th:text="1 != 10"></span></li>
    </ul>
  </li>
  <li>조건식
    <ul>
      <li>(10 % 2 == 0)? '짝수':'홀수' =
            <span th:text="(10 % 2 == 0)? '짝수':'홀수'"></span></li>
    </ul>
```

```
  </li>
  <li>Elvis 연산자
    <ul>
      <li>${name}?: '데이터가 없습니다.' =
              <span th:text="${name}?: '데이터가 없습니다.'"></span></li>
      <li>${nullData}?: '데이터가 없습니다.' =
              <span th:text="${nullData}?: '데이터가 없습니다.'"></span></li>
    </ul>
  </li>
  <li>No-Operation
    <ul>
      <li>${name}?: _ = <span th:text="${name}?: _">데이터가 없습니다.</
        span>
      </li>
      <li>${nullData}?: _ = <span th:text="${nullData}?: _">이곳은 비어있습니다.</span></li>
    </ul>
  </li>
</ul>
</body>
</html>
```

10. 속성 값 설정

- 타임리프는 주로 HTML 태그에 'th:*' 속성을 지정하는 방식으로 동작하며, 'th:*'로 속성을 적용하면 기존 속성을 대체하며 기존 속성이 없으면 새로 생성한다.
- 'th:attrappend', 'th:attrprepend' 는 문자 그대로 추가하기 때문에 띄어쓰기에 주의 필요!

- th:attrappend : 속성 값의 뒤에 값을 추가한다.
- th:attrprepend: 속성 값의 앞에 값을 추가한다.
- th:classappend: class 속성에 자연스럽게 추가한다.
- th:속성명: 해당 속성을 대체한다.

<예시>
- 타임리프 HTML 코드: <input type="text" name="mock" th:name="userA" />
- 타임리프 렌더링 후: <input type="text" name="userA" />

▶ 예제 코드

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

▶ attribute

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
</head>
<body>

<h1>속성 설정</h1>
<input type="text" name="mock" th:name="userA"/>

<h1>속성 추가</h1>
- th:attrappend = <input type="text" class="text" th:attrappend="class=' large'"/><br/>
- th:attrprepend = <input type="text" class="text" th:attrprepend="class='large '"/><br/>
- th:classappend = <input type="text" class="text" th:classappend="large"/><br/>
```

```
</body>
</html>
```

▶ 브라우저 소스보기

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

<h1>속성 설정</h1>
<input type="text" name="userA"/>

<h1>속성 추가</h1>
- th:attrappend = <input type="text" class="text large"/><br/>
- th:attrprepend = <input type="text" class="large text"/><br/>
- th:classappend = <input type="text" class="text large"/><br/>
</body>
</html>
```

11. 반복

- 타임리프에서 반복은 '**th:each**'를 사용하며, 추가적으로 반복 루틴에서 사용할 수 있는 여러 상태 값을 지원합니다.

- 기본 사용법

1) 반복 시 오른쪽 컬렉션의 값을 하나씩 꺼내서 왼쪽 변수에 담아서 태그를 반복 실행
2) 'th:each'는 List 뿐만 아니라 Array, java.util.Iterable, java.util.Enumeration을 구현한 모든 객체를 반복에 사용할 수 있다.
3) Map도 사용할 수 있는데 이 경우 Map.Entry가 사용

```
<tr th:each="아이템 : ${반복할리스트}">
    <td th:text="${아이템.프로퍼티1}">default value</td>
    <td th:text="${아이템.프로퍼티2}">default value</td>
</tr>
```

- 반복 상태 유지

1) th:each를 사용하여 반복을 할 때, 두 번째 파라미터를 설정해서 반복의 상태를 확인
2) 두 번째 파라미터는 생략이 가능하며, 생략하면 지정한 변수명에 Stat가 추가된 변수로 컬렉션 상태 객체가 제공 됨.

```
index : 0부터 시작하는 값
count : 1부터 시작하는 값
size : 전체 사이즈
even , odd : 홀수, 짝수 여부(boolean)
first , last : 처음, 마지막 여부(boolean)
current : 현재 객체

<tr th:each="아이템, 컬렉션상태: ${반복할리스트}">
    <td th:text="${아이템.프로퍼티1}">default value</td>
    <td th:text="${아이템.프로퍼티2}">default value</td>
</tr>

<!-- 아래의 경우 두 번째 파라미터를 생략하여도 userStat으로 사용 가능 -->
<tr th:each="user, userStat : ${users}">
    컬렉션 카운트: <span th:text="${userStat.count}"></span>
    컬렉션 사이즈: <span th:text="${userStat.size}"></span>
    유저이름: <span th:text="${user.username"></span>
</tr>
```

▶ 예제코드

```
@GetMapping("/each")
public String each(Model model) {
	addUsers(model);
	return "each";
}
private void addUsers(Model model) {
		List<User> users = new ArrayList<>(
				Arrays.asList(
						new User("안유진", 22),
						new User("장원영", 21),
						new User("이서", 18),
						new User("가을", 23),
						new User("리즈", 21),
						new User("레이", 21)
				));

		model.addAttribute("users", users);
}
```

▶ each.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

<h1>기본 테이블</h1>
<table border="1">
  <tr>
    <th>username</th>
    <th>age</th>
  </tr>
  <tr th:each="user : ${users}">
    <td th:text="${user.username}">username</td>
    <td th:text="${user.age}">0</td>
  </tr>
```

```
</table>

<h1>반복 상태 유지</h1>
<table border="1">
  <tr>
    <th>count</th>
    <th>username</th>
    <th>age</th>
    <th>etc</th>
  </tr>
  <tr th:each="user, userStat : ${users}">
    <td th:text="${userStat.count}">username</td>
    <td th:text="${user.username}">username</td>
    <td th:text="${user.age}">0</td>
    <td>
      index = <span th:text="${userStat.index}"></span>
      count = <span th:text="${userStat.count}"></span>
      size = <span th:text="${userStat.size}"></span>
      even? = <span th:text="${userStat.even}"></span>
      odd? = <span th:text="${userStat.odd}"></span>
      first? = <span th:text="${userStat.first}"></span>
      last? = <span th:text="${userStat.last}"></span>
      current = <span th:text="${userStat.current}"></span>
    </td>
  </tr>
</table>

</body>
</html>
```

12. 조건부 평가

- 타임리프에서는 'if', 'unless', 'switch' 조건문을 제공
- 타임리프는 해당 조건이 맞지 않으면 태그 자체가 렌더링 되지 않으며, 즉 조건에 충족하지 않으면 태그 자체가 없는 것처럼 된다.

```
if, unless(not if)
 - th:if="조건식"
 - th:unless="조건식"

switch-case
 - * 은 만족하는 조건이 없을 때 사용하는 디폴트이다.
<div th:switch="${조건대상 변수}">
    <span th:case="비교변수1">value1</span>
    <span th:case="비교변수2">value2</span>
    <span th:case="*">default</span>
</div>
```

▶ 예제코드

```
@GetMapping("/condition")
public String condition(Model model) {
    addUsers(model);
    return "condition";
}

private void addUsers(Model model) {
    List<User> users = new ArrayList<>(
            Arrays.asList(
                    new User("안유진", 22),
                    new User("장원영", 21),
                    new User("이서", 18),
                    new User("가을", 23),
                    new User("리즈", 21),
                    new User("레이", 21)
            ));

    model.addAttribute("users", users);
}
```

▶ condition.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

<h1>if, unless</h1>
<table border="1">
  <tr>
    <th>count</th>
    <th>username</th>
    <th>age</th>
  </tr>
  <tr th:each="user, userStat : ${users}">
    <td th:text="${userStat.count}">1</td>
    <td th:text="${user.username}">username</td>
    <td>
      <span th:text="${user.age}">0</span>
      <span th:text="'미성년자'" th:if="${user.age lt 20}"></span>
      <span th:text="'미성년자'" th:unless="${user.age ge 20}"></span>
    </td>
  </tr>
</table>

<h1>switch</h1>
<table border="1">
  <tr>
    <th>count</th>
    <th>username</th>
    <th>age</th>
  </tr>
  <tr th:each="user, userStat : ${users}">
    <td th:text="${userStat.count}">1</td>
    <td th:text="${user.username}">username</td>
    <td th:switch="${user.age/10}">
      <span th:case="1">10대</span>
      <span th:case="2">20대</span>
```

```
        <span  th:case="*">기타</span>
      </td>
    </tr>
</table>

</body>
</html>
```

13. 주석

- 주석에는 표준 HTML 주석, 타임리프 파서 주석, 타임리프 프로토타입 주석 등 여러 종류의 주석이 제공

1) 표준 HTML 주석

- 자바스크립트의 표준 HTML 주석은 타임리프가 렌더링 하지 않고 그대로 남겨둔다.

```
<h1>1. 표준 HTML 주석</h1>
<!-- <span th:text="${data}">html data</span> -->
```

2) 타임리프 파서 주석

- 타임리프에 적용되는 주석으로 해당 주석 내용은 렌더링에서 주석 부분이 제거된다.

```
<h1>2. 타임리프 파서 주석</h1>
<!--/* [[${data}]] */-->

<!--/*-->
<span th:text="${data}">html data</span>
<!--*/-->
```

3) 타임리프 프로토타입 주석

- 해당 주석은 HTML 파일을 그대로 열면 렌더링되지 않고, 타임리프를 통해 화면 렌더링을 하게 되면 보이게 하는 기능이다. 즉, HTML 파일로 열면 주석 처리되고, 서버를 통한 렌더링되면 주석 처리 되지 않는다.

```
<h1>3. 타임리프 프로토타입 주석</h1>
<!--/*/
<span th:text="${data}">html data</span>
/*/-->
```

▶ 예제코드

```
@GetMapping("/comments")
public String comments(Model model) {
    model.addAttribute("data", "Spring!");
    return "comments";
}
```

▶ comments

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>

<body> <h1>예시</h1>
<span th:text="${data}">html data</span>

<h1>1. 표준 HTML 주석</h1>
<!--
<span th:text="${data}">html data</span>
-->

<h1>2. 타임리프 파서 주석</h1>
<!--/* [[${data}]] */-->

<!--/*-->
<span th:text="${data}">html data</span>
<!--*/-->

<h1>3. 타임리프 프로토타입 주석</h1>
<!--/*/
<span th:text="${data}">html data</span>
/*/-->

</body>
</html>
```

14. 블록

- 타임리프는 자체 태그인 <th:block> 태그를 제공
- 해당 태그는 HTML 태그가 아닌, 타임리프에서만 사용하는 태그로 해결하기 어려운 것들을 해결하기 위해 유일한 자체 태그인 블록을 제공한다.
- <th:block>은 렌더링시 제거되는 태그이며, 안의 내용은 제거되지 않고, <th:block> 태그만 제거된다.
- 대표적으로 th:each를 통해 반복을 할 때, 반복의 대상이 한 요소가 아닌 동등한 레벨의 여러 요소를 그룹화하여 반복하고자 할 때 th:block이 유용하다.

▶ 예제코드

```
@GetMapping("/block")
public String block(Model model) {
    addUsers(model);
    return "block";
}
```

▶ block.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

<th:block th:each="user : ${users}">
  <div>
    멤버이름 : <span th:text="${user.username}"></span>
    , 멤버나이 :<span th:text="${user.age}"></span>
  </div>
  <div>
    요약 : <span th:text="${user.username} + ' / ' + ${user.age}"></span>
  </div>
</th:block>

</body>
</html>
```

15. 자바스크립트 인라인

타임리프는 자바스크립트에서 타임리프를 편리하게 사용할 수 있는 자바스크립트 인라인 기능을 제공한다.
타임리프의 자바스크립트 인라인 기능을 이용하면 Javascript에 변수를 담거나, 객체를 담는 경우 쉽게 사용이 가능하다.
문자열은 쌍따옴표(")를 자동으로 추가하여 대입해주고, 객체인 경우 JSON 형태로 변환하여 대입해준다.
또한, 변수에 포함된 쌍따옴표(") 같은 Javascript에 문제가 될 수 있는 문자가 있으면 이스케이프(escape) 처리도 자동으로 해줍니다. (ex: " -> ₩")

```
1. 유저명을 넣는 경우
var username = [[${user.username}]]; // var username = 이름; 으로 에러가 발생한다.
var username = "[[${user.username}]]"; // 쌍따옴표를 이용하여 변수를 넣어주어야 한다.

2. 객체를 넣는 경우
 - 객체명을 직접 호출하면 toString 메소드가 호출되어 그대로 들어가게 된다.
 - 즉, 객체를 넣고 싶다면 서버에서 ObjectMapper 등으로 객체를 JSON 문자열로 만들어 Model에 담아 사용해야 한다.
 - 아니면, 프로퍼티를 하나씩 꺼내 JSON을 직접 구현해야 한다.
var user = [[${user}]]; // var user = BasicController.User(username=UserA, age=10);
var user = {
    username: '[[${user.username}]]',
    age: [[${user.age}]]
}
```

- 자바스크립트 인라인 each : 자바스크립트안에서 loop(반복)을 하여야 하는 경우에도 자바스크립트 인라인 안에서 each를 제공

```
<script th:inline="javascript">
    [# th:each="user, stat : ${users}"]
    var user[[${stat.count}]] = [[${user}]];
    [/]
</script>
```

▶ 예제코드

```
@GetMapping("/javascript")
public String javascript(Model model) {
    model.addAttribute("user", new User("UserA", 10));
    addUsers(model);
```

```
    return "basic/javascript";
}
```

▶

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

<!-- 자바스크립트 인라인 사용 전 -->
<script>
    var username = [[${user.username}]];
    var age = [[${user.age}]];

    //자바스크립트 내추럴 템플릿
    var username2 = /*[[${user.username}]]*/ "test username";

    //객체
    var user = [[${user}]];
</script>

<!-- 자바스크립트 인라인 사용 후 -->
<script th:inline="javascript">
    var username = [[${user.username}]];
    var age = [[${user.age}]];

    //자바스크립트 내추럴 템플릿
    var username2 = /*[[${user.username}]]*/ "test username";

    //객체
    var user = [[${user}]];
</script>

<!-- 자바스크립트 인라인 each -->
<script th:inline="javascript">
    [# th:each="user, stat : ${users}"]
```

```
    var  user[[${stat.count}]]  =  [[${user}]];
    [/]
</script>

</body>
</html>
```