# 타임리프 - 기본기능

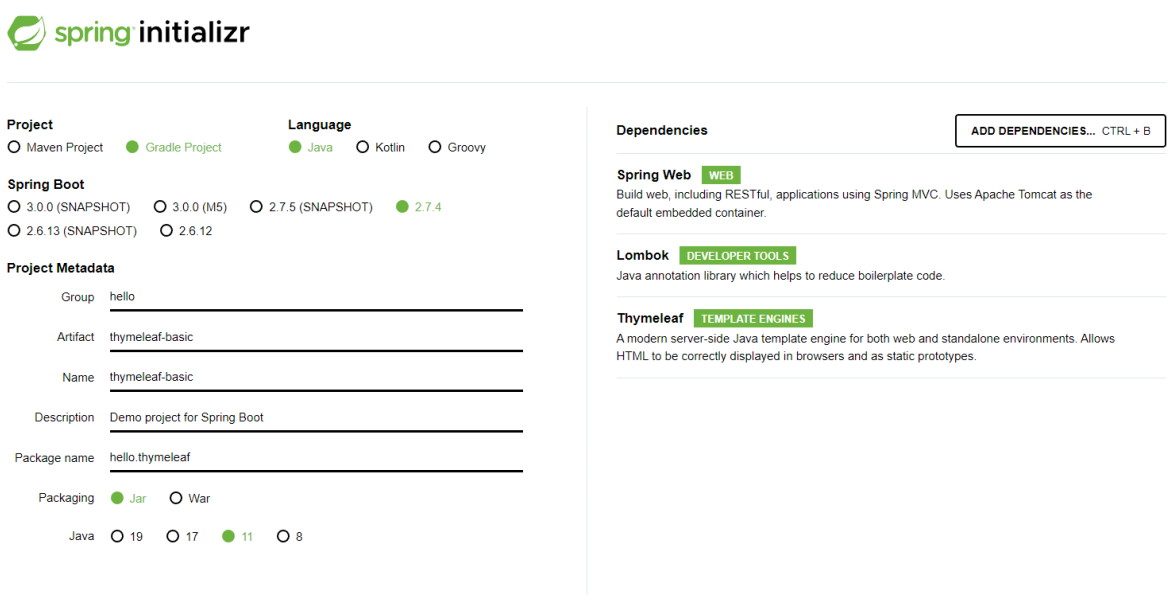
## 프로젝트 생성

사전 준비물

- Java 11
  - 컴퓨터 설정이 Java 8이기에 강의와 달리 버전 8로 실습하고자 한다.
- IDE : intelliJ

**스프링 부트 스타터 사이드로 이동해서 스프링 프로젝트 생성

https://start.spring.io/




Spring Boot가 자동으로 home으로 인식하는 path : /resources/static/index.html

Lombok 적용

1. Preferences plugin lombok 검색 실행 (재시작)
2. Preferences Annotation Processors 검색 Enable annotation processing 체크 (재시작)

3. 임의의 테스트 클래스를 만들고 @Getter, @Setter 확인

# 타임리프 소개

공식 사이트: https://www.thymeleaf.org/
공식 메뉴얼 - 기본 기능:
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html
공식 메뉴얼 - 스프링 통합:
https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html

## 타임리프 특징

- 서버 사이드 HTML 렌더링 (SSR)
- 네츄럴 템플릿
- 스프링 통합 지원

1. 서버 사이드 HTML 렌더링 (SSR)

: 타임리프는 백엔드 서버에서 HTML을 동적으로 렌더링 하는 용도로 사용된다.(JSP 같은 거)

** 클라이언트 서버 렌더링은 JavaScript 같은 거

2. 네츄럴 템플릿

- 타임리프는 순수 HTML을 최대한 유지하는 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 템플릿들은 해당 파일을 열면, 예를 들어서 JSP 파일 자체를 그대로 웹 브라우저에서 열어보면 JSP 소스코드와 HTML이 뒤죽박죽 섞여서 웹 브라우저에서 정상적인 HTML 결과를 확인할 수 없다. 오직 서버를 통해서 JSP가 렌더링 되고 HTML 응답 결과를 받아야 화면을 확인할 수 있다.
- 반면에 타임리프로 작성된 파일은 해당 파일을 그대로 웹 브라우저에서 열어도 정상적인 HTML 결과를 확인할 수 있다. 물론 이 경우 동적으로 결과가 렌더링 되지는 않는다.

하지만 HTML 마크업 결과가 어떻게 되는지 파일만 열어도 바로 확인할 수 있다.

이렇게 순수 HTML을 그대로 유지하면서 뷰 템플릿도 사용할 수 있는 타임리프의 특징을 네츄럴 템플릿(natural templates)이라 한다.

3. 스프링 통합 지원

타임리프는 스프링과 자연스럽게 통합되고, 스프링의 다양한 기능을 편리하게 사용할 수 있게 지원한다. 이 부분은 스프링 통합과 폼 장에서 자세히 알아보겠다.

# 텍스트 - text, utext

- HTML의 콘텐츠(content)에 데이터를 출력할 때는 다음과 같이 th:text 를 사용하면 된다.
  - `<span th:text="${data}">`
- HTML 테그의 속성이 아니라 HTML 콘텐츠 영역안에서 직접 데이터를 출력하고 싶으면 다음과 같이 [[...]] 를 사용하면 된다.
  - 컨텐츠 안에서 직접 출력하기 = `[[${data}]]`

## HTML 엔티티

HTML 엔티티 : 웹 브라우저는 < 를 HTML 태그의 시작으로 인식한다. 따라서 < 를 태그의 시작이 아니라 문자로 표현하는 방법

이스케이프(escape) : HTML에서 사용하는 특수 문자를 HTML 엔티티로 변경하는 것

이스케이프 기능을 사용하지 않으려면 어떻게 해야 할까? (Unescape)
타임리프는 다음 두 기능을 제공한다.

- th:text ⇒ th:utext
- [[...]] ⇒ [(...)]

```
<!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>
```

이스케이프 처리를 왜 해야할까 ?

게시판을 예로 들면 이해하기 쉽다. 사용자가 <,>를 게시물로 작성하였을 때 이스케이프 처리가 안되어있다면 HTML이 다 깨질 것이다.

# 변수 - SpringEL

변수 표현식 : ${…}

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 ⇒ list.get(0).getUsername()
- users[0]['username'] : 위와 같음
- users[0].getUsername() : List에서 첫 번째 회원을 찾고 메서드 직접 호출

Map

- userMap['userA'].username : Map에서 userA를 찾고, username 프로퍼티 접근 ⇒ map.get("userA").getUsername()

- userMap['userA']['username'] : 위와 같음
- userMap['userA'].getUsername() : Map에서 userA를 찾고 메서드 직접 호

### 지역변수 선언

`th:with`를 사용하면 지역 변수를 선언해서 사용할 수 있다. 지역 변수는 선언한 태그 안에서만 사용할 수 있다.

scope를 벗어나면 사용할 수 없다. 벗어나서 사용할 경우 오류가 발생하여 서버가 돌아가지 않는다.

`th:with="first=${users[0]}"` : first에 users[0] 객체가 들어간다.

```
<div th:with="first=${users[0]}">
    <p>처음 사람의 이름은 <span th:text="${first.username}"></span></p>
</div>
```

# 기본 객체들

타임리프는 기본 객체들을 제공한다.

```
# 기본 객체

${#request}
${#response}
${#session}
${#servletContext}
${#locale}
```

그런데 `#request` 는 `HttpServletRequest` 객체가 그대로 제공되기 때문에 데이터를 조회하려면 `request.getParameter("data")`처럼 불편하게 접근해야 한다.

이런 점을 해결하기 위해 편의 객체도 제공한다.

```
# 편의 객체

HTTP 요청 파라미터 접근: param
예) ${param.paramData}
```

```
HTTP 세션 접근: session
예) ${session.sessionData}

스프링 빈 접근: @
예) ${@helloBean.hello('Spring!')}
```

# 유틸리티 객체와 날짜

> 참고
> 이런 유틸리티 객체들은 대략 이런 것이 있다 알아두고, 필요할 때 찾아서 사용하면 된다

타임리프 유틸리티 객체
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#expression-utilityobjects

유틸리티 객체 예시
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#appendix-b-expressionutility-objects

### 자바8 날짜

타임리프에서 자바8 날짜인 LocalDate , LocalDateTime , Instant 를 사용하려면 추가 라이브러리가
필요하다. 스프링 부트 타임리프를 사용하면 해당 라이브러리가 자동으로 추가되고 통합된다.
타임리프 자바8 날짜 지원 라이브러리 : thymeleaf-extras-java8time
자바8 날짜용 유틸리티 객체 : #temporals

# URL 링크

단순한 URL
: @{/hello} ⇒ /hello

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

경로 변수
: @{/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.

상대경로, 절대경로, 프로토콜 기준을 표현할 수 도 있다.
/hello : 절대 경로
hello : 상대 경로

참고: https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#link-urls

# 리터럴

: 리터럴 - 소스 코드 상에서 고정된 값을 말하는 용어

타임리프에서 문자 리터럴은 항상 ' (작은 따옴표)로 감싸야 한다.
`<span th:text="'hello'">`

그런데 문자를 항상 ' 로 감싸는 것은 너무 귀찮은 일이다. 공백 없이 쭉 이어진다면 하나의 의미있는 토큰으로 인지해서 다음과 같이 작은 따옴표를 생략할 수 있다.

이어지는 문자 종류 : A-Z , a-z , 0-9 , [] , . , - , _

흔히 하는 실수 : 띄어쓰기는 포함되지 않는다.
`<span th:text="hello">`