Thymeleaf는 View Template Engine이다. 컨트롤러에서 전달받은 데이터를 추출해 동적인 페이지를 만들 수 있다. 태그의 속성으로 Thymeleaf 명령어를 사용할 수 있으며 html 파일 내에서 사용이 가능하다. JSP와 같이 서버 사이드 렌더링 방식이다.

Thymeleaf 의 궁극적인 목표는 내츄럴 템플릿을 사용해 개발하는 것이다. 여기서 내츄럴 템플릿이란 Thymeleaf 로 작성된 HTML 템플릿을 말하는데 순수 HTML 으로도 웹 브라우저에서 동작하고 렌더링을 통해 뷰 템플릿으로 동작 가능하다.

Thymeleaf 의 장점

Java, Spring 기반에서 개발하기 쉽고 기존에 JSP 를 경험한적이 있다면 진입 장벽이 낮다.

순수 HTML구조를 유지하여 서버상에서 동작시키지 않아도 되므로 웹 퍼블리셔들과의 협업이 용이하다. 기존의 템플릿 기술들은 항상 서버를 구동시켜 결과물을 확인해야 하지만 Thymeleaf의 경우 static 파일을 사용하듯 해당 내용을 브라우저에서 바로 확인할 수 있다는 장점이 있다, 이것이 가능한 이유는 Thymeleaf가 HTML 태그의 속성(Attribute)으로 작성되므로에 기존의 HTML구조를 건드리지 않기 때문이다.

Natural Template (내추럴 템플릿)
서버를 구동하지 않으면 순수 HTML로 구성되는 정적인 페이지를, 서버를 구동하면 동적으로 페이지가 생성된다. 이렇게 Thymeleaf 는 순수 HTML을 유지하기 때문에 내추럴 템플릿으로도 불린다.

```
<h2 th:text="${title}">서버없이 브라우저로 오픈하여 랜더링 하면 보임</h2>
<h2 data-th-text="${title}">서버없이 브라우저로 오픈하여 랜더링 하면 보임</h2>
```

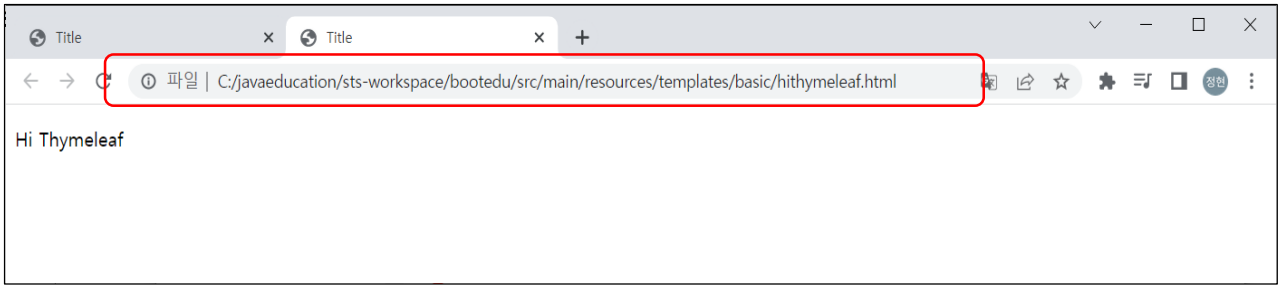
Spring 진영에서도 Thymeleaf 사용을 공식적으로 권장하고 있다

Thymeleaf의 default prefix는 src/main/resources/templates이며 suffix는 .html 이다.

Thymeleaf 파일은 일반 HTML 파일과 다를 것이 없다. 원하는 HTML 태그에 'th' 라는 prefix를 사용하기 위해 최상위 태그인 <HTML> 태그에 정해진 네임스페이스만 추가하면 된다. 다음 소스는 th 속성을 사용하여 간단하게 텍스트 값을 출력하고 있다.

```
<!DOCTYPE HTML>
<HTML xmlns:th="http://www.Thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<p><span th:text="${say}">Hi</span>Thymeleaf</p>
</body>
</HTML>
```

위와 같이 템플릿 소스를 작성했다면 이 HTML 파일을 Spring Boot 서버를 기동하지 않고 그냥 브라우저에서 열어보면 다음과 같이 'Hi Thymeleaf' 라는 메세지가 출력된다.



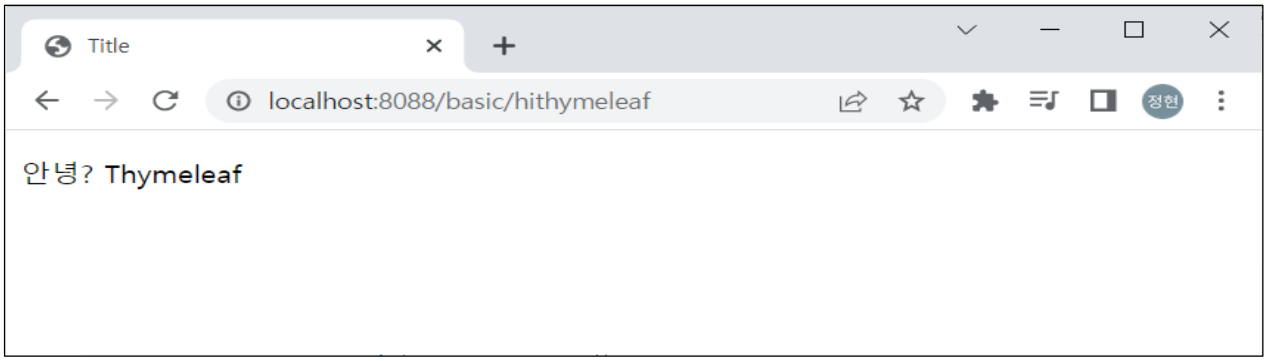
그러나 Spring Boot 서버를 기동 시키고 브라우저로 정해진 URL 문자열로 요청하면 다음과 같이 컨트롤러를 거쳐서 Thymeleaf 템플릿이 서버상에서 처리되고 그 결과가 브라우저로 출력된다.

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

    @GetMapping("/hithymeleaf")
```

```
public String hiThymeleaf(Model model) {
    model.addAttribute("say", "안녕?");
    return "basic/hithymeleaf";
}
```

```
<p><span th:text="${say}">Hi</span> Thymeleaf</p>
```



JSP

```
<input type="text" name="userName" value="${user.name}">
```

Thymeleaf

```
<input type="text" name="userName" value="unico" th:value="${user.name}">
```

Thymeleaf 문법 정리

SpringEL

Spring Expression Language라는 뜻의 SpringEL (SpEL)은 런타임 시 메서드 호출 및 기본 문자열 템플릿 등의 기능을 제공한다.

1) ${...} 표현식 - 변수 표현식

${...} 표현식을 이용해 컨트롤러에서 전달받은 변수에 접근할 수 있으며 th:속성명 에서만 사용 가능하다.

2) @{...} 표현식 - URL 표현식

@{...} 표현식은 서버의 contextPath를 추가한 URI 로 변경된다.

@{/} -> "/contextPath/"

@{/images/1.png} -> "/contextPath/images/1.png"

@{/vdelete(id=${vo.id})} -> "/contextPath/vdelete?id=2"

3) 문자 합치기

합치고 싶은 문자열을 "|" 으로 감싸거나 + 연산자를 이용해 문자를 합칠 수 있다.

```
<div th:text="|My name is ${info.name} !! |"></div>
<div th:text="'My name is '+ ${info.name} + ' !! '"></div>
```

4) 비교 연산자

```
<!-- 이항 연산자 -->
<div th:text="${info.name != 'kim'}"></div>
<div th:text="${info.age >= 30}"></div>

<!-- 삼항 연산자 -->
 <div th:text="${info.name == 'kim' ? 'ok' : 'no'}"></div>
```

5) HTML 태그의 컨텐트 설정 - th:text

```
<div th:text="${info.name}">유니코</div>
```

6) HTML 태그의 value 속성의 값 설정 - th:value

```
<input type='text' th:value="${info.name}" value="둘리" >
```

7) th:if, th:unless

if~else 구문과 비슷하다. 조건을 채크하여 참이면<th:if> 그리고 거짓이면<th:unless> 컨텐트를 표현한다.

```
<p th:if="${info.age > 18}">입장 가능</p>
<p th:unless="${info.age <= 18}">입장 가능</p>
```

```
<th:block th:if="${info.age > 18}"><hr><p>입장 가능</p><hr></th:block>
<th:block th:unless="${info.age <= 18}"><hr><p>입장 가능</p><hr></th:block>
```

8) th:switch, th:case

switch 구문과 비슷하다. th:case 속성에 지정된 값과 동일한 서브 태그를 표현한다.

```
<th:block th:switch="${info.name}">
  <div th:case="올라프">겨울왕국</div>
  <div th:case="또치">아기공룡둘리</div>
</th:block>
```

9) th:each

for 반복문과 비슷하다.

```
<th:block th:each="data:${datas}">
    <h1 th:text="${data}"></h1>
</th:block>
```

변수명 앞에 status 변수를 추가해 row에 대한 추가정보를 얻을 수 있다.

```
<th:block th:each="data,status:${datas}">
    <h1 th:text="|${status.count} ${data}|"></h1>
</th:block>
```

[status 속성]

index : 0부터 시작

count : 1부터 시작

size : 총 개수

current : 현재 index의 변수

event/odd : 짝수/홀수 여부

first/last : 처음/마지막 여부

10) 링크될 대상 URL : th:href="@{}"

```
<a th:href="@{/vdelete(id=${vo.id})}">
```

11) th:with="${}"

<div th:with="userId=${number}" th:text="${usesrId}">

변수형태의 값을 재정의하는 속성이다. th:with를 이용하여 새로운 변수값을 생성할 수 있다.

[구현 예 1]

```
<th:block th:if="${ list }" >
  <h1 onclick="location.href='/meeting'">미팅 스케쥴</h1>
  <hr>
  <table>
    <tr th:each="vo : ${list}">
        <td th:class="${vo.id}">[[${ vo.name }]]</td>
        <td th:class="${vo.id}" th:onclick="|displayReply(${vo.id})|" >[[${ vo.title }]]</td>
        <td th:class="${vo.id}">[[${ vo.meetingDate }]]</td>
        <td><a th:href ="@{/meeting/delete(id=${vo.id})}">
                    <img src = "/images/delete.png" width ='20'></a></td>
        <td><img th:onclick="|displayUpdateForm(${vo.id})|"
                    src="/images/edit.png" width ='20'></td>
        <td><img th:onclick="|insertReply(${vo.id})|"
                    src="/images/comment.png" width ='25'></td>
        </tr>
    </table>
</th:block>
```

Thymeleaf 구문 요약 정리

[표현식]

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

[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)

[특별한 토큰]
No-Operation: _

Expressions Basic Objects (표현식 기본 객체)

타임리프는 자주 사용하는 객체들을 간편하게 조회할 수 있게 아래와 같이 기본 객체들을 지원한다

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

유틸리티 객체

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

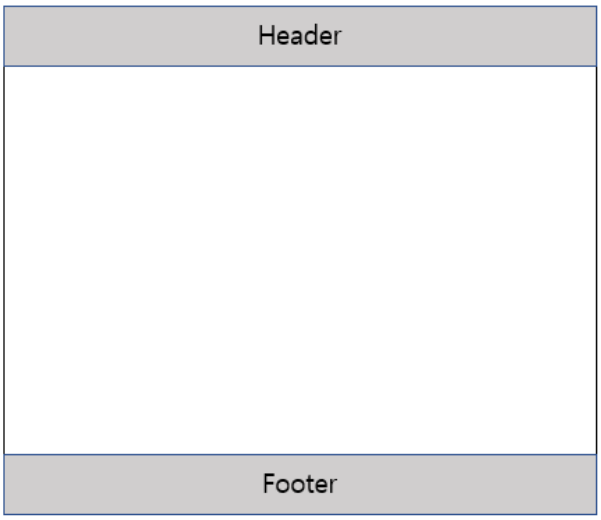
표현식은 #{...}의 표현식을 사용한다, 사실 타임리프가 제공하는 유틸리티는 다양하고 많기 때문에 간단하게 어떤 것들이 있는지 리스트만 살펴본다.

#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 (날짜는 주로 temporals를 사용)
#numbers : 숫자 서식 지원
#strings : 문자 관련 편의 기능
#objects : 객체 관련 기능
#bools : boolean 관련 기능
#arrays : 배열 관련 기능
#lists : 컬렉션 관련 기능
#sets : 컬렉션 관련 기능
#maps : 컬렉션 관련 기능
#ids : 아이디 처리 관련 기능

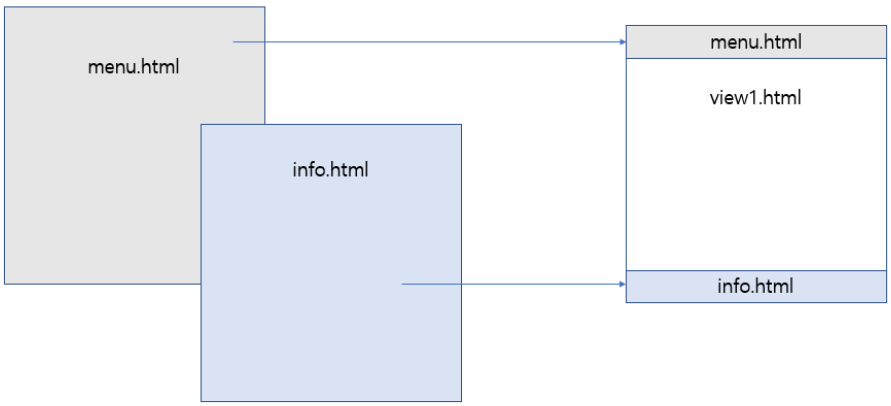
Fragment(부분 컨텐트) 삽입과 대체

Fragment 이용하기

HTML을 이용해 화면을 그리다 보면 항상 중복된 메뉴나 정보 등을 나타내고자 하는 경우가 생긴다.



여러 페이지에서 반복되는 Header의 메뉴나 Footer의 정보 등을 표현할 때 사용한다.



위의 그림과 같이 중복된 메뉴나 정보등을 나타낼때 Thymeleaf에서 지원하는 도구가 "Fragment" 이다.

~{HTML 파일}

메뉴나 footer처럼 반복되는 HTML을 미리 만들어 두고 필요한 페이지에 추가 구성으로 넣을 수

있는 기능이다.

[menu.html]

```
<body>
    <div th:fragment="header">
        <a href=" https://spring.io/">스프링페이지</a>
         <a href=" https://www.thymeleaf.org/">타임리프페이지</a>
    </div>
</body>
```

[info.html]

```
<body>
    <div th:fragment="info">
        문의 사항은 unicodaum@hanmail.net 으로 메일 보내세요.
    </div>
</body>
```

[fragmentMain1.html]

```
<body>
   <header th:insert="basic/fragment/menu :: header">여기는 menu</header>
   <section>
            여기는 fragmentMain1.html 페이지 입니다.
   </section>
   <footer th:insert="basic/fragment/info :: info">여기는 info</footer>
</body>
```