5. Thymeleaf

---

1. 템플릿 엔진

- 템플릿 양식과 특정 데이터 모델에 따른 입력 자료를 합성하여 결과 문서(응답 화면)를 출력하는 것
  → 만들어둔 화면(html)에 데이터를 추가하여 하나의 html로 만들어서 응답
  (JSP도 템플릿 엔진)

https://www.thymeleaf.org/

2. Thymeleaf (타임리프)

- 웹 및 독립 실행형 환경 모두를 위한 최신 서버 측 Java 템플릿 엔진
- HTML 파일에서 th(Thymeleaf) 속성을 이용해 컨트롤러로 부터 전달 받은데이터를 이용해 동적 페이지를 만들 수 있음.
- Spring Boot에서는 JSP가 아닌 Thymeleaf 사용을 권장하고 있음.

3. Thymeleaf 사용 준비

[Maven] - pom.xml

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

[Gradle] - build.gradle

```
implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
```

application.properties

```
# thymeleaf 접두사, 접미사 설정
#미작성 시 기본값
#spring.thymeleaf.prefix=classpath:/templates/
#spring.thymeleaf.suffix=.html
```

4. Thymeleaf 문법(표현식)

- `<html lang="en" xmlns:th="http://www.thymeleaf.org">`
  th속성을 사용하기 위해 선언된 네임스페이스
  순수 HTML로만 이루어진 페이지의 경우, 선언하지 않아도 무관

Spring EL

`${key}` : 변수, Model 등으로 전달된 데이터의 key 입력 시 value 출력

```
<p th:text=${member}></p>
<p th:text=${member.memberNo}</p>
```

- `*{key}` : 선택 변수, 객체에 포함된 필드를 출력
  `th:object` 속성과 같이 사용

```
<!-- member 객체의 필드 memberNo와 memberPw 등을 출력 -->
<div th:obejct="${member}">
    <span th:text="*{memberNo}"></span>
    <span th:text="*{memberId}"></span>
</div>
```

- `#{key}` : Message Expression
  - src/main/resoources/messages.properties에 작성된 값을 얻어와 출력

- messages.properties

```
app.name=Board Project - boot
user.default.image=/images/user.png
```

- html파일

```
<p th:text="#{app.name}">프로젝트 이름</p>

<img th:src="#{user.default.image}" id="memberProfile">
```

- `@{url}` : 링크 URL, URL 형식 출력

```
<!-- 단순 url-->
<a th:href="@{/board}">단순 url</a>

<!-- 쿼리스트링 포함 url-->
<a th:href="@{/board(key=${key}, queyr=${query})}">쿼리스트링 포함</a>

<!-- PathVariable 포함 url-->
<a th:href="@{/board/{boardCode}/{boardNo}(boardCode=${boardCode}, boardNo=${boardNo})}">PathVariable 포함</a>
```

- `th:href, th:src, th:action` : @{url} 표기를 적용하는 속성
  - 일반 href, src, action은 처음 작성되어 있는 지정된 주소로만 요청 가능
  - th가 붙은 속성은 서버측에서 작성되어지기 때문에 동적으로 주소를 변경할 수 있음.

- `th:text, th:value`
  - 지정된 내용을 Text 형식으로 내용, input의 value로 출력

```
<!-- request scope에 key="memberName" value="홍길동" 이 있을 경우-->
<p th:text="${memberName}">회원이름</p>

<input type="text" th:value="${memberName}">

<!-- 랜더링 결과 -->
<p>홍길동</p>
<input type="text" th:value="홍길동">
```

- `th:text / th:utext`
  `[[...]] / [(...)]`
  - Thymeelaf의 텍스트를 HTML에 출력한다는 공통점이 있지만
  출력하려는 텍스트에 HTML 태그가 있을 경우
  이를 해석 할 것인지, 말 것인지를 구분하는 방법을 제공

  - th:text ， [[...]] : HTML 태그를 해석 X (JS의 innerText와 비슷)
  - th:utext ， [[...)] : HTML 태그를 해석 O (JS의 innerHTML과 비슷)

```
<!-- key="memberName" value="<b1>홍길동</b1>" 이 있을 경우-->

<h3> text 와 utext의 차이점 </h3>
<p th:text="${memberName}">회원이름</p>
<p th:utext="${memberName}">회원이름</p>

<h3> [[...]] 와 [(...)]의 차이점 </h3>
<p>[[${memberName}]]</p>
<p>[(${memberName})]</p>
```

text 와 utext의 차이점

홍길동

홍길동

... 와 ...의 차이점

홍길동

홍길동

- `|문자열|` : 리터럴 대체
  - th:text, th:value에 원하는 형식의 Text를 표기하는 방법

```
<!-- request scope에 key="memberName" value="홍길동" 이 있을 경우-->
<p th:text="|이름 : ${memberName} 님|">회원이름</p>

<!-- 랜더링 결과 -->
<p>이름 : 홍길동 님</p>
```

- `th:with` : 지역 변수 선언
  - th:with가 선언된 태그 내에서 사용할 변수명을 지정
  - ,로 구분하여 여러 변수 선언 가능
  - th:with="변수명=값, 변수명=값"

```
<ul th:with="num=100, name1=#{app.name}, name2=${memberName}">
    <li th:text="${num}">num</li>
    <li th:text="${name1}">name1</li>
    <li th:text="${name2}">name2</li>
</ul>
```

- `<th:block>`
  - Thymeleaf의 유일한 자체 태그
  - Thymeleaf는 HTML 태그 내에 th 속성을 작성하여 기능을 정의하는게 일반적이지만 마땅한 HTML 태그가 없을 경우에 사용

- `th:if="${조건}"  / th:unless="${조건}"` : if / else문
  - th:if == if문, th:unless == else문
  - th:unless 조건에 if와 같은 조건을 작성해야됨 (조건 자동으로 반대로 바꿔줌)

```
<th:block th:if="${session.loginMember == null}">
  <a href="/">메인 페이지</a> | <a href="/member/login">로그인</a>
</th:block>

<th:block th:unless="${session.loginMember == null}">
  <label for="header-menu-toggle"> 닉네임
    <i class="fa-solid fa-caret-down"></i>
  </label> <input type="checkbox" id="header-menu-toggle">

  <div id="header-menu">
    <a href="/member/info">내정보</a> <a href="/member/logout">로그아웃</a>
  </div>
</th:block>


<!-- if 조건식에 값만 작성한 경우 (있으면 true, 없으면 false)-->
<th:block th:unless="${session.loginMember}">
  로그인 x
</th:block>

<th:block th:if="${session.loginMember}">
  로그인 O
</th:block>
```

- `th:switch="${값}"  / th:case="switch의값 | *  "` : switch - case 문
  - th:switch에 작성된 값에 따라 일치하는 th:case가 실행

- th:case=”*” 은 case중 마지막에 작성되며, 앞선 case를 제외한 모든 경우(default)를 의미

```
<th:block th:switch="${num}">
  <p th:case="100">A</p>
  <p th:case="200">B</p>
  <p th:case="*">C</p>
</th:block>
```

- `비교 연산자 : lt(<), gt(>), le(<=), ge(>=), eq(==), ne(!=), not(!)`
  - 마크업 언어 태그 기호 <, >로 인해 오류가 발생할 가능성이 있으므로
    부등호가 포함된 연산자는 대체 연산자 사용

- `삼항 연산자` : 조건식 ? true일때 : false일 때

```
<h3>삼항 연산자</h3>
<p th:text="${memberName} ? ${memberName} : '데이터가 없습니다'">삼항 연산자</p>
```

- `Elvis 연산자` : 값 ?: 값이 없을 때
  - 삼항 연산자에서 조건식에 값만 작성한 후
    값이 있다면 작성한 값, 없으면 없을 경우에 대한 값이 출

```
<h3>Elvis 연산자</h3>
<p th:text="${memberName} ?: '데이터가 없습니다'">Elvis 연산자</p>
```

- `No-Operation`: 값 ?: _
  - 값이 없을 경우 Thymelaef가 실행이 안된 것 만듦
  - html 태그 내부에 작성된 값이 그대로 출력됨

```
<h3>No-Operation</h3>
<p th:text="${memberName222} ?: _">데이터가 없습니다</p>

<!-- 랜더링 결과 -->
<h3>No-Operation</h3>
<p>데이터가 없습니다</p>
```

- `th:each="item : ${list}"`
  - 해당 HTML 요소를 list의 길이 만큼 반복
  - list에 저장된 요소를 순차접근하여 item에 저장
  - 해당 코드가 작성된 HTML요소 내부에서 item 사용 가능

```
<th:block th:each="user : ${userList}">
```

fragment(조각)을 이용한 공통 영역 처리

- `th:fragment="조각 이름"`
  - html 중 공통된 부분(반복되는 코드)을 조각(fragment)으로 지정

  [templates/fragments/common.html]

```
<nav th:fragment="nav">  <!-- 해당 요소에 nav라는 조각 이름을 지정-->
    <ul>
        <li><a href="#">공지사항</a></li>
        <li><a href="#">자유 게시판</a></li>
        <li><a href="#">질문 게시판</a></li>
        <li><a href="#">FAQ</a></li>
        <li><a href="#">1:1문의</a></li>
        <li><a href="/chatting">채팅</a></li>
    </ul>
</nav>
```

`th:insert="~{조각파일경로 :: 조각이름}"`

- insert : 삽입하다
- 해당 속성이 작성된 요소 내부에 지정된 조각을 삽입하는 속성

```
<!-- 해당 요소를 조각으로 지정된 속성으로 변경-->
<div th:insert="~{fragments/commons :: nav}"></div>


<!-- 랜더링 결과 -->
<!-- div 요소 안에 nav라는 이름의 조각이 삽입됨 -->
<div>
  <nav>
      <ul>
          <li><a href="#">공지사항</a></li>
          <li><a href="#">자유 게시판</a></li>
          <li><a href="#">질문 게시판</a></li>
          <li><a href="#">FAQ</a></li>
          <li><a href="#">1:1문의</a></li>
          <li><a href="/chatting">채팅</a></li>
      </ul>
  </nav>
</div>
```

- `th:replace="~{조각파일경로 :: 조각이름}"`
  - replace : 바꾸다
  - 해당 속성이 작성된 요소를 지정된 조각으로 바꾸는 속성

  [templates/common/main.html]

```
<!-- 해당 요소를 조각으로 지정된 속성으로 변경-->
<div th:replace="~{fragments/commons :: nav}"></div>


<!-- 랜더링 결과 -->
<!-- div 요소가 nav라는 이름의 조각으로 변환됨 -->
<nav>
    <ul>
        <li><a href="#">공지사항</a></li>
        <li><a href="#">자유 게시판</a></li>
        <li><a href="#">질문 게시판</a></li>
        <li><a href="#">FAQ</a></li>
        <li><a href="#">1:1문의</a></li>
        <li><a href="/chatting">채팅</a></li>
    </ul>
</nav>
```

- `th:replace="~{html 파일 경로}"`
  - ::조각이름을 지정하지 않으면 html의 모든 내용을 얻어와 지정된 요소와 바꿈

  [templates/common/main.html]

```
<!-- common/header.html의 모든 내용을 읽어와 th:block 태그와 바꿈 -->
<th:block th:replace="~{common/header}"></th:block>



<!-- common/footer.html의 모든 내용을 읽어와 th:block 태그와 바꿈 -->
<th:block th:replace="~{common/footer}"></th:block>
```

- `<script th:inline="javascript">` : JS Inline
  - script 태그 내에 thymeleaf 문법을 작성 가능하도록 함
  - thymeleaf 문법으로 출력된 내용을 타입에 따라 알맞은 JS 형태로 변환

- `JS Inline - Text Rendering : [[...]]`
  - 출력하고자하는 값을 그대로 출력
- `JS Inline - Natural Template : /*[[...]]*/ "설명"`
  - HTML을 직접 열어도 동작하는 탬플릿

  + 주석을 통해서 thymelaef 컴파일 오류 해결

```
// message에 100 이라는 값이 저장되어 있을 경우
// message2에 "abc" 이라는 값이 저장되어 있을 경우


<script th:inline="javascript">
   const message = [[${message}]];
   const message2 = /*[[${message2}]]*/ "값";
</script>

// 랜더링 결과
// message 가 100으로 치환됨
<script th:inline="javascript">
   const message = 100;
   const message2 = "abc";

   // HTML을 직접 열었을 때
   const message2 = "값";
</script>
```

`${#numbers.sequence(시작, 끝 [,step])}`

- th:each문에서 컬렉션 반복 접근이 아닌
  시작, 끝을 지정해서 반복할 때 사

```
<p th:text=${member}></p>
<p th:text=${member.memberNo}</p>
```

- `th:classappend` : 클래스를 동적으로 추가

```
<!-- child-comment 클래스 추가 -->
<li class="comment-row" th:classappend="child-comment">

<!-- 랜더링 결과 -->
<li class="comment-row child-comment">



<!-- 조건이 true일 경우 child-comment 클래스 추가 -->
<li class="comment-row" th:classappend="${comment.parentNo} != 0 ? child-comment">

<!-- 랜더링 결과 -->
<!-- true인 경우 -->
<li class="comment-row child-comment">

<!-- false인 경우 -->
<li class="comment-row">
```

- `${객체?.필드}` : 안전 탐색 연산자(Safe Navigation Operator)

객체가 null인지 판별해서 null이 아닌 경우에 수행