Thymeleaf Layout

1. 기본 원리 (한줄 요약)

레이아웃 = 공통된 틀(헤더/푸터/메타 등)을 **한 곳**에 두고, 각 페이지 콘텐츠(또는 프래그먼트)를 **그 틀의 지정된 위치에 삽입**해서 HTML 중복을 제거하는 기법.

---

2. 프로젝트 파일/폴더 구조 (추천)

```
src/main/resources/templates/
├─ layout.html          ← 기본 레이아웃 (또는 base.html)
├─ fragments/
│   ├─ head.html
│   ├─ header.html
│   ├─ footer.html
│   └─ nav.html
├─ pages/
│   ├─ home.html
│   ├─ about.html
│   └─ admin/
│      ├─ admin-layout.html
│      └─ admin-dashboard.html
```

---

3. 순수 Thymeleaf 방식 (`th:fragment` + `th:replace/insert/include`)

fragments/head.html

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

```
    <meta name="viewport" content="width=device-width,initial-scale=1"/>
    <title th:text="${title}">Default Title</title>

    <!-- 공통 CSS/JS -->
    <link rel="stylesheet" th:href="@{/css/main.css}" />
    <script th:src="@{/js/main.js}"></script>
  </head>
</html>
```

fragments/header.html

```
<!-- templates/fragments/header.html -->
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <body>
    <div th:fragment="header">
      <header>
        <h1><a th:href="@{/}">MySite</a></h1>
        <nav>
          <a th:href="@{/}">Home</a>
          <a th:href="@{/about}">About</a>
        </nav>
      </header>
    </div>
  </body>
</html>
```

layout.html (base)

```
<!-- templates/layout.html -->
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <!-- head fragment 호출 (매개변수 전달) -->
  <div th:replace="fragments/head :: head(${pageTitle})"></div>
</head>
<body>
```

```
  <!-- header -->
  <div th:replace="fragments/header :: header"></div>

  <!-- 페이지마다 바꿀 영역을 빈 fragment로 둠. 컨텐츠가 들어올 위치 -->
  <main>
    <!-- 실제 페이지에서 이 부분을 'replace' 하게 됨 -->
    <div th:fragment="content"></div>
  </main>

  <!-- footer -->
  <footer th:replace="fragments/footer :: footer"></footer>
</body>
</html>
```

pages/home.html (layout 사용)

```
<!-- templates/pages/home.html -->
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <body>
    <!-- layout.html의 content 프래그먼트를 완전히 교체 -->
    <div th:replace="~{layout :: content}">
      <!-- replacement content -->
      <section>
        <h2>홈</h2>
        <p>환영합니다.</p>
      </section>
    </div>
  </body>
</html>
```

핵심 차이 (th:replace vs th:insert vs th:include)

- `th:replace` : 해당 태그 자체를 프래그먼트의 내용으로 **완전히 교체**. 가장 많이 사용.
- `th:insert` : 현재 태그 내부에 프래그먼트 **내용을 삽입** (태그 구조 유지).
- `th:include` : `th:replace`와 유사하지만, 과거 버전 호환용. 현재는 `replace` / `insert` 권장.

4. Layout Dialect 사용 (권장 — 문법이 깔끔)

Layout Dialect는 `nz.net.ultraq.thymeleaf:thymeleaf-layout-dialect` 라이브러리로 문법을 단순화합니다.

의존성 (Maven / Gradle)

Gradle

```
implementation 'nz.net.ultraq.thymeleaf:thymeleaf-layout-dialect:3.4.0'  // 예시 버전, 실제는 최신 확인
```

(※ 실제 버전은 프로젝트에서 확인 — Gradle 설정 예시)

기본 레이아웃 (layout.html)

```
<!-- templates/layout.html -->
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
  <meta charset="UTF-8"/>
  <title layout:title-pattern="$CONTENT_TITLE - MySite">MySite</title>
  <div th:replace="fragments/head :: head(${pageTitle})"></div>
</head>
<body>
  <header th:replace="fragments/header :: header"></header>

  <!-- child 페이지가 이 이름(content)을 채움 -->
  <div layout:fragment="content"></div>

  <footer th:replace="fragments/footer :: footer"></footer>
</body>
</htm
```

home.html (Layout Dialect)

```
<!-- templates/pages/home.html -->
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
  <title>홈</title>
</head>
<body>
  <div layout:fragment="content">
    <h2>홈 페이지</h2>
    <p>내용...</p>
  </div>
</body>
</html>
```

장점

- `layout:decorate="~{layout}"` 으로 레이아웃 지정이 직관적.
- `layout:fragment="content"` 만으로 삽입 위치 지정.
- 다수의 fragment를 쉽게 덮어쓸 수 있음 (예: `sidebar`, `scripts` 등).

---

5. 프래그먼트에 매개변수(파라미터) 전달 — 실무에서 유용

타임리프 3.x 에서 `::` 문법으로 프래그먼트 이름과 파라미터 전달 가능.

head fragment에서 title 받기 (3번의 예시 재사용)

`fragments/head.html` :

```
<head th:fragment="head(title='Default')">
  <title th:text="${title}">Default</title>
```

```
  <!-- ... -->
</head>
```

호출 측에서:

```
<!-- layout.html -->
<div th:replace="fragments/head :: head('홈 페이지 타이틀')"></div>
```

또는 모델 속성 사용:

```
model.addAttribute("pageTitle", "Home");
```

```
<div th:replace="fragments/head :: head(${pageTitle})"></div>
```

또한 `th:with`와 조합해서 지역값을 만들어 전달할 수도 있음.

---

6. 중첩 레이아웃 (Nested Layouts) — ex) 일반사이트 vs 관리자

중첩 레이아웃은 매우 유용합니다. 기본 `layout.html` → `admin-layout.html` → `admin-page.html` 순으로 중첩.

`admin-layout.html`은 `layout.html`을 decorate 하거나 `layout:fragment`를 재정의.

ex)

```
<!-- admin-layout.html -->
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
  <div layout:fragment="content">
    <aside th:replace="fragments/admin-nav :: nav"></aside>
    <section>
      <div layout:fragment="adminContent"></div>
    </section>
  </div>
</html>
```

그리고 `admin-dashboard.html`에서는:

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
    layout:decorate="~{admin-layout}">
 <div layout:fragment="adminContent">
  <!-- 관리자 대시보드 내용 -->
 </div>
</html>
```

---

7. 폼, 바인딩, 재사용 가능한 입력 프래그먼트

폼 요소를 프래그먼트로 만들어 여러 폼에서 재사용 가능.

`fragments/form-input.html`

```
<div th:fragment="inputField(field, label)">
 <label th:for="${field}"><span th:text="${label}">Label</span></label>
 <input th:field="*{${field}}" />
 <div th:if="${#fields.hasErrors(field)}" th:errors="*{${field}}"></div>
</div>
```

호출:

```
<form th:object="${user}">
 <div th:replace="fragments/form-input :: inputField('username','사용자명')"></div>
 <div th:replace="fragments/form-input :: inputField('email','이메일')"></div>
</form>
```

(복잡한 바인딩은 `th:field`와 잘 조합해야 함 — 문자열로 필드 전달 시 주의)

---

8. Controller 연동 예시 (Spring MVC)

```
@Controller
public class HomeController {

    @GetMapping("/")
    public String home(Model model) {
        model.addAttribute("pageTitle", "홈");
        model.addAttribute("message", "환영합니다!");
        return "pages/home"; // templates/pages/home.html
    }
}
```

- layout.html에서 th:replace="fragments/head :: head(${pageTitle})" 를 쓰면 컨트롤러의 model 속성 pageTitle을 받아 제목으로 렌더링 됨.

---

9. Thymeleaf 설정 (Spring Boot 기본값과 개발 환경)

Spring Boot는 기본적으로 `spring-boot-starter-thymeleaf`를 넣으면 `classpath:/templates/`를 템플릿 위치로 사용.

application.properties (유용한 설정)

```
# 개발 중 템플릿 캐시 끄기 (바로 변경 반영)
spring.thymeleaf.cache=false

# 인코딩
spring.thymeleaf.encoding=UTF-8
spring.thymeleaf.mode=HTML
```

> 배포시에는 spring.thymeleaf.cache=true로 설정해 성능 높이기.

Layout Dialect는 dependency만 추가하면 대부분 별도 빈 등록 없이 동작. (특정 커스터마이징 원하면 `SpringTemplateEngine`에 dialect 추가 가능)

---

10. 성능 & 캐시 팁

- 개발: `spring.thymeleaf.cache=false` — 템플릿 변경을 바로 확인.
- 운영: `cache=true`, 템플릿(및 정적파일) CDN이나 버전 관리 적용.

- 많은 작은 프래그먼트를 과도하게 호출하면 (특히 `th:replace` 반복) 성능 영향이 있으니 빈번 호출되는 UI는 하나의 큰 fragment로 묶는 게 좋음.
- 복잡한 fragments 내에 무거운 연산(예: DB 조회)을 넣지 말 것 — 데이터는 컨트롤러에서 제공.

---

11. 디버깅 자주 발생하는 문제와 해결

- **정적 리소스(CSS / JS) 경로가 깨짐**

  → `th:href="@{/css/main.css}"` 사용. 상대경로 때문인 경우 절대 경로로 바꿔보기.

- **fragment not found**

  -> 경로/파일명 오타 또는 `templates` 폴더 위치 확인. `th:replace="fragments/header :: header"` 라면 `templates/fragments/header.html` 존재 확인.

- **모델 값이 안 나옴**

  -> 컨트롤러에서 `model.addAttribute()` 확인. fragment가 별도 `th:fragment` 로 분리되어 있는데 fragment 내부에서 `${...}` 접근 불가할 수 있음 — 전달된 매개변수 사용 또는 layout에서 `th:with` 로 전달.

- **캐싱 때문에 변경 안 보임**

  → `spring.thymeleaf.cache=false` 확인.

---

12. 권장 실무 관례 (팀/프로젝트 표준)

1. **레이아웃 1개 이상**: 기본 `layout.html` + 필요시 `admin-layout.html` 등.
2. **fragments 폴더**: head/header/footer/nav/form 등 카테고리별 정리.
3. **fragment 이름 규칙**: `header`, `footer`, `head(title)` 처럼 명확하게.
4. **프래그먼트 매개변수 최소화**: 너무 많은 매개변수는 복잡도를 높임.
5. **컨트롤러는 뷰에 필요한 데이터만 제공**: 뷰 로직은 템플릿 안에서 처리하되, DB 조회 같은 건 컨트롤러/서비스가 담당.
6. **템플릿 테스트**: Thymeleaf 템플릿은 통합 테스트(통상 SpringBootTest + MockMvc)로 렌더링 결과 일부를 검증.

---

13. 예시: 전체 예제 (한 번에 보는 완성본 요약)

- `layout.html` (Layout Dialect) : 공통 틀 + `layout:fragment="content"`
- `fragments/head.html` : `th:fragment="head(title='...')"`
- `fragments/header.html` : `th:fragment="header"`
- `pages/home.html` : `layout:decorate="~{layout}"` + `layout:fragment="content"`
- Controller : `model.addAttribute("pageTitle","홈")` + `return "pages/home";`

(위에서 이미 코드 스니펫으로 모두 구현했으니 그대로 복사해서 프로젝트에 붙이면 동작.)

---

14. 고급 팁(유용한 트릭)

- **페이지별 스크립트/스타일 삽입**: 레이아웃에 `layout:fragment="scripts"` 위치를 만들고, 페이지에서 해당 fragment를 채우면 각 페이지 스크립트를 레이아웃 하단에 모아 삽입 가능.
- **권한/로그인 정보 사용**: 헤더 프래그먼트에서 `#authentication` 또는 Spring Security의 `Principal`을 사용해 로그인 메뉴 노출 제어.
- **i18n**: `th:text="#{label.submit}"`처럼 메시지 코드 사용 — head fragment에서 locale 관련 메타 태그 관리.

---

15. 요약(빠른 체크리스트)

- 레이아웃 방식을 쓸 땐 Layout Dialect 권장(문법 간결).
- 공통 요소는 `fragments/`로 모아두고 `layout.html`로 데코레이트.
- 프래그먼트 매개변수로 title 등 전달 가능.
- 개발 환경에선 템플릿 캐시 비활성화. 운영에선 활성화.
- 중첩 레이아웃은 페이지 타입(관리자/사용자) 분리 시 유용.