

# 스프링 부트 웹 프로젝트

chapter05

## Thymeleaf

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

# Contents

part.1

thymeleaf 소개

part.2

thymeleaf의 기본 사용법

part.3

템플릿 레이아웃1

part.4

템플릿 레이아웃2

part.5

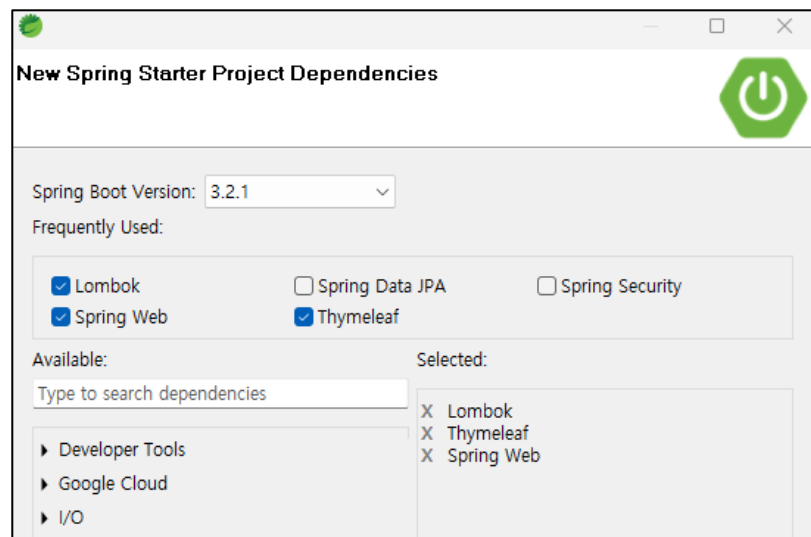
부트스트랩

# thymeleaf 소개

## 프로젝트 생성

실습을 위해 프로젝트를 생성한다.

라이브러리는 'Lombok, Thymeleaf, Spring Web'를 선택한다.



application.properties 파일에 타임리프 설정을 추가한다.

프로젝트 변경 후 만들어진 결과를 저장하지 않도록 설정한다.

application.properties

```
spring.thymeleaf.cache=false
```

실습 파일의 내용을  
복사해서 넣어주세요!

### Thymeleaf란?

- 타임리프는 동적인 페이지를 만드는 템플릿 엔진이다.
- 자바 코드를 삽입하여 컨트롤러에서 전달한 데이터를 처리할 수 있다.
- 조건문, 반복문, 객체 등 다양한 기능을 제공한다.

### JSP와 Thymeleaf의 차이점

- JSP는 과거에는 많이 사용되었지만, 지금은 성능과 기능이 다른 기술이보다 떨어진다.
- JSP는 자바코드와 HTML이 혼합되어, 유지보수가 어려울 수 있다.
- 반면 타임리프는 비즈니스로직과 완전히 분리되어, JSP와 달리 순수한 HTML 문법을 사용한다.



Thymeleaf



JSP

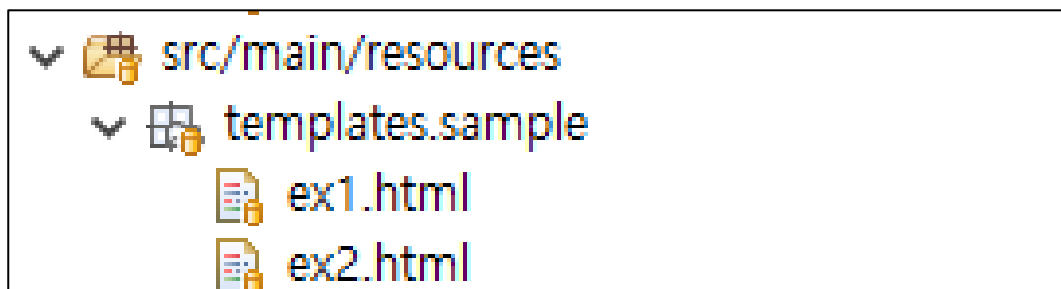
### 타임리프 템플릿 파일 생성 방법

- 타임리프는 기본적으로 template 폴더를 사용하므로, template 폴더 밑에 html파일을 생성한다.

### 타임리프 템플릿 파일 작성 방법

- 타임리프를 사용할 때는 HTML 파일의 상단에 타임리프의 네임스페이스를 정의해야 한다. (최신 버전에서는 생략 가능)
- 타임리프 속성은 'th:' 접두사를 사용한다.
- 타임리프를 사용하여 컨트롤러에서 전달받은 데이터를 처리할 수 있다.

템플릿 파일 위치



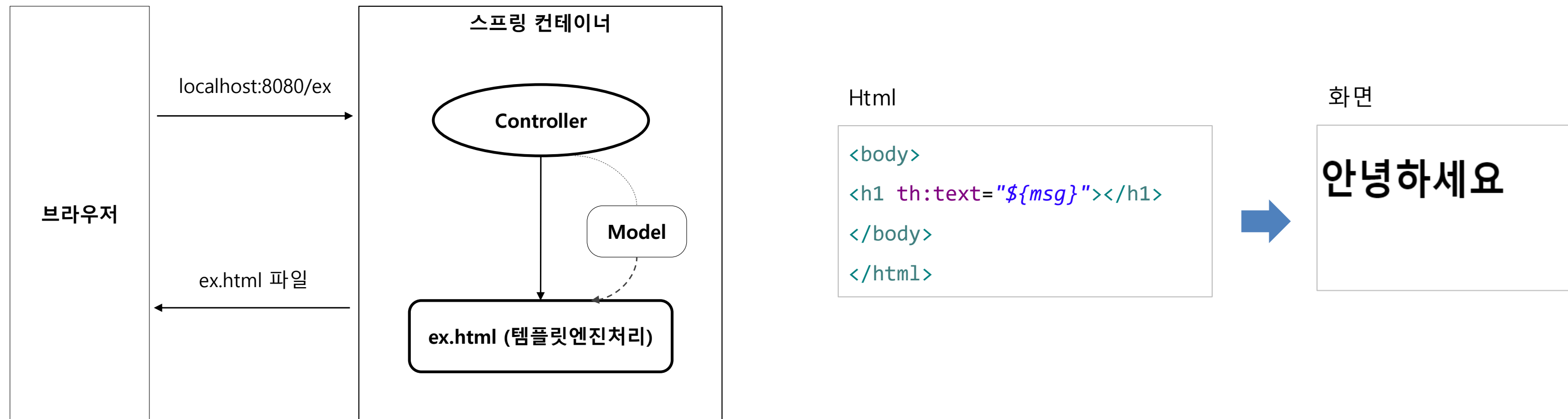
템플릿 파일 예시

```
<html xmlns:th="http://www.thymeleaf.org">
<body>
<h1 th:text="${msg}"></h1>
</body>
</html>
```

# thymeleaf 소개

## mvc와 타임리프의 동작과정

1. 컨트롤러는 사용자에게 보여줄 뷰(템플릿)을 선택한다
2. 뷰에 데이터를 전달한다.
3. 뷰에서는 타임리프가 동작하여 전달받은 데이터를 사용하여 동적인 페이지를 생성한다.



표현식	종류
변수 표현	<ul style="list-style-type: none"><li>• 변수 표현식: <code>\${...}</code></li><li>• 선택 변수 표현식: <code>*{...}</code></li><li>• 메세지 표현식: <code>#{...}</code></li><li>• 링크 표현식: <code>@{...}</code></li><li>• 조각 표현식: <code>~{...}</code></li></ul>
객체 표현	<ul style="list-style-type: none"><li>• 객체의 필드에 접근하기<ul style="list-style-type: none"><li>- <code>obj.field</code></li><li>- <code>obj['field']</code></li><li>- <code>obj.getField()</code></li></ul></li><li>• 리스트의 요소에 접근하기<ul style="list-style-type: none"><li>- <code>list[0]</code></li><li>- <code>list.get(0)</code></li></ul></li><li>• 맵의 요소에 접근하기<ul style="list-style-type: none"><li>- <code>map['key']</code></li></ul></li></ul>

### 텍스트 출력하기

- 타임리프는 html 태그의 속성에 th: 붙여서, 데이터를 출력하거나 조작할 수 있다.
- th:text 속성 : html 태그 내에 사용하여 데이터를 출력할 수 있다.
- '[[ ... ]]' 표현식 : html태그 없이 데이터를 출력할 수 있다.

h1 태그의 텍스트로 데이터 출력하기

```
<h1 th:text="${msg}"></h1>
```

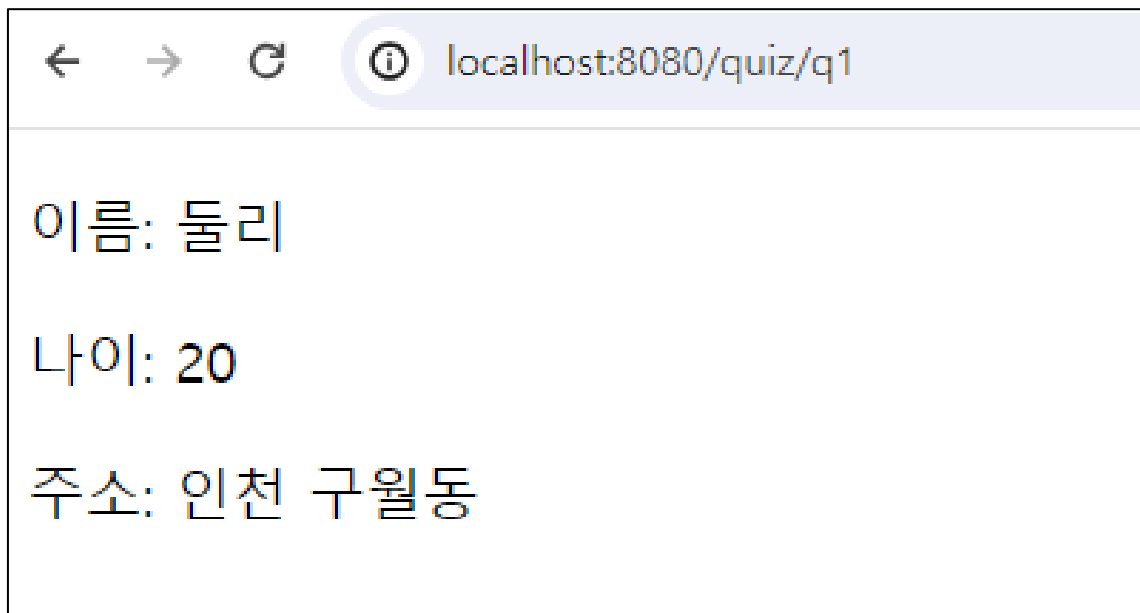
html 태그 없이 데이터 출력하기

```
[[${msg}]]
```

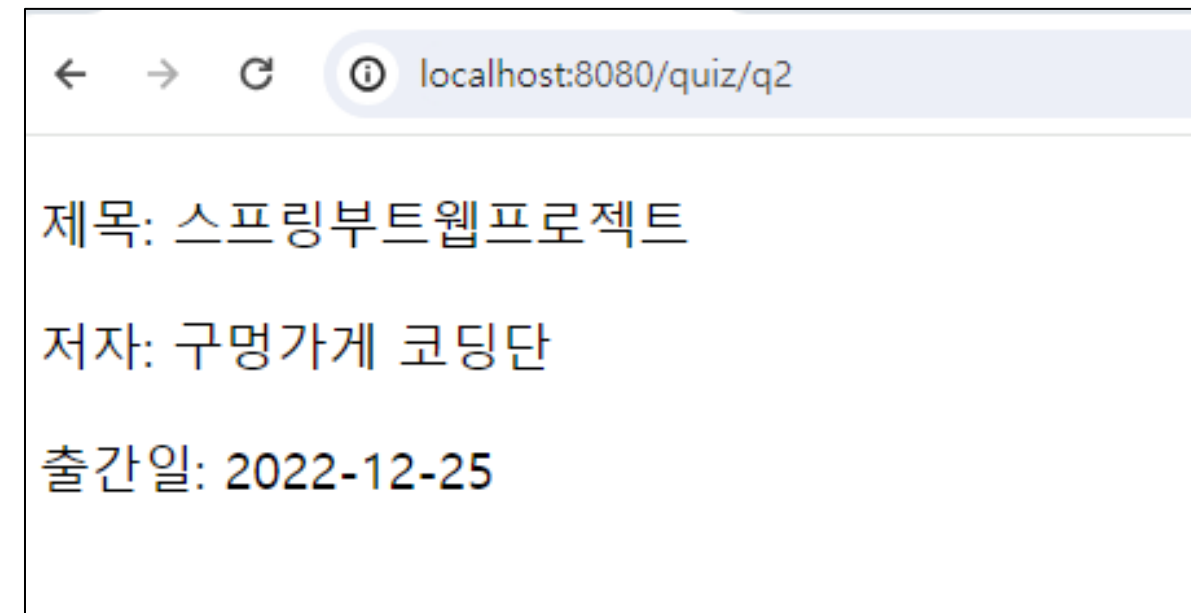


**Q1.** localhost:8080/quiz/q1 주소에 접속했을 때 아래와 같이 화면을 표시되도록 코드를 구현하세요.  
컨트롤러에서 사람의 정보(이름, 나이, 주소)를 화면으로 전달하고 화면에서는 해당 정보를 출력하세요.

**Q2.** localhost:8080/quiz/q2 주소에 접속했을 때 아래와 같이 화면을 표시되도록 코드를 구현하세요.  
컨트롤러에서 책의 정보(제목, 저자, 출간일)를 화면으로 전달하고 화면에서는 해당 정보를 출력하세요.



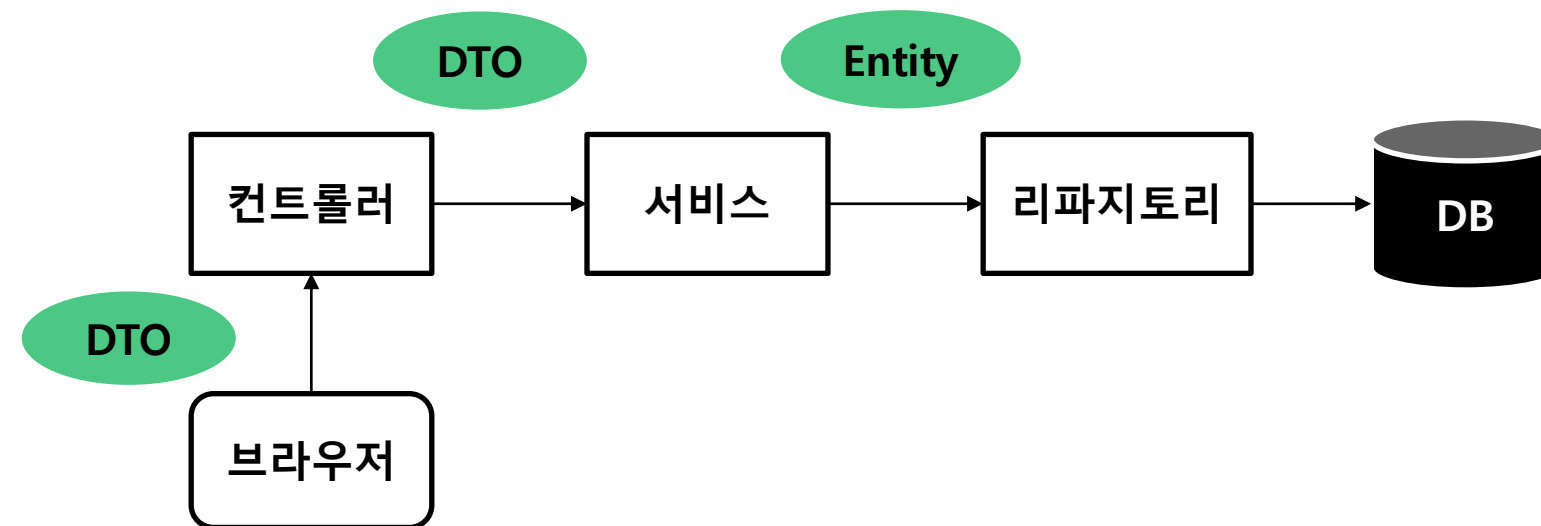
1번 문제 결과 화면



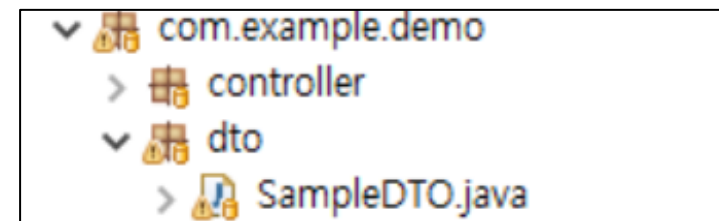
2번 문제 결과 화면

## DTO 클래스

- DTO는 다른 계층에 데이터를 전달하기 위한 클래스이다.
- 컨트롤러와 뷰가 데이터를 주고 받을 때 사용한다.



클래스 위치



DTO 클래스 예시

```
@Getter
@Setter
@ToString
@NoArgsConstructor
@AllArgsConstructor
public class SampleDTO {

    private int no; //번호

    private String text; //글내용

    private LocalDateTime regDate; //등록일자

}
```

**Q3.** localhost:8080/quiz/q3 주소에 접속했을 때 아래와 같이 화면을 표시되도록 코드를 구현하세요..

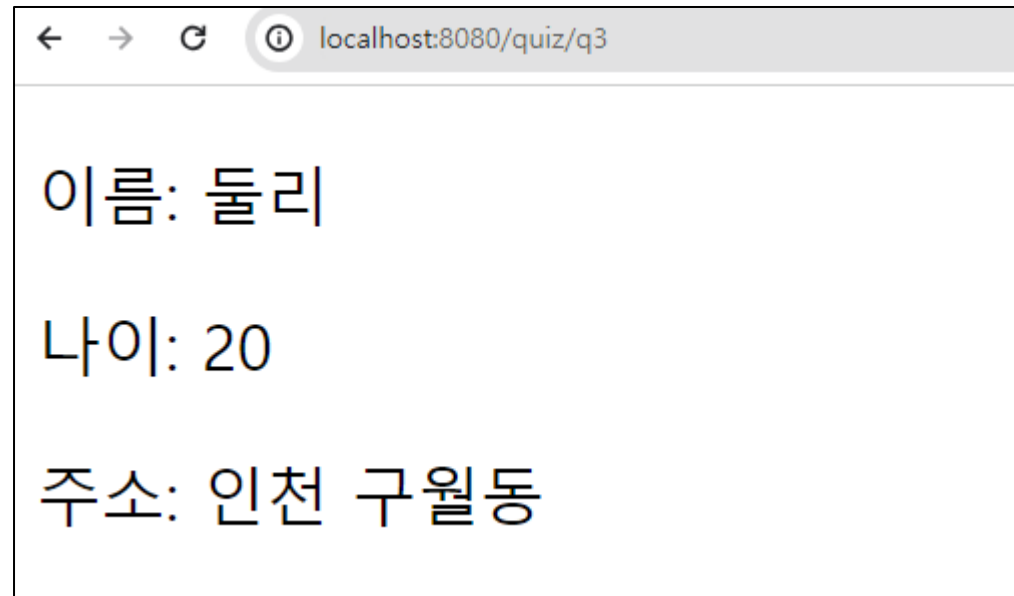
사람의 정보를 저장할 PersonDTO클래스를 만드세요.

컨트롤러에서 PersonDTO 타입의 객체를 생성하고 화면으로 전달하세요.

화면에서는 해당 정보를 출력하세요.

```
public class PersonDTO {  
  
    private String name;  
  
    private int age;  
  
    private String address;  
  
}
```

DTO 클래스



이름: 둘리  
나이: 20  
주소: 인천 구월동

결과 화면

**Q4.** localhost:8080/quiz/q4 주소에 접속했을 때 아래와 같이 화면을 표시되도록 코드를 구현하세요.

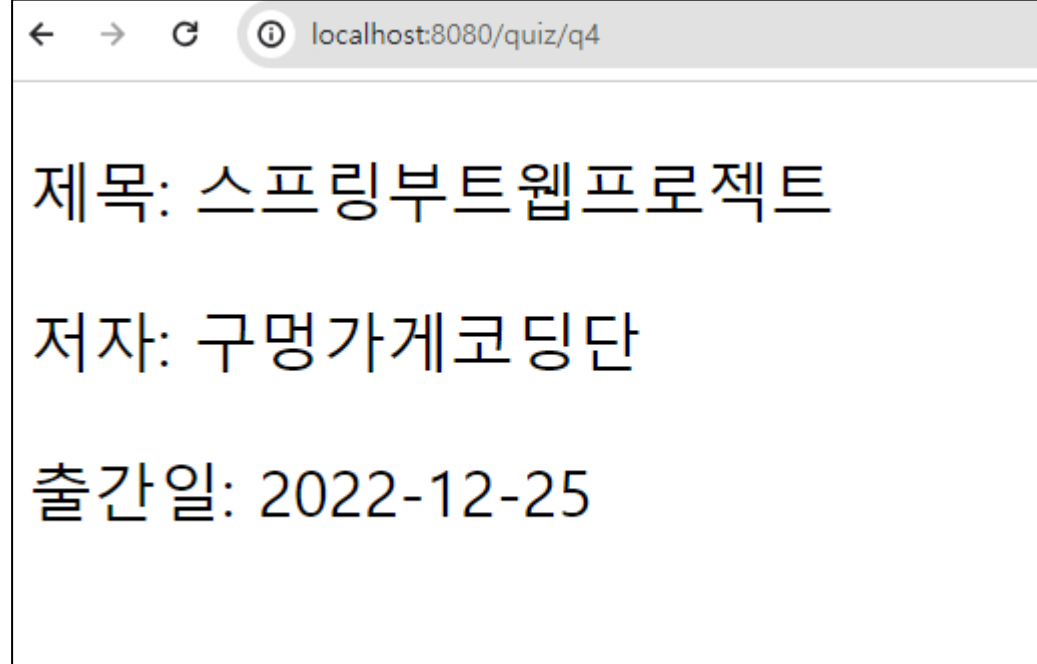
책의 정보를 저장할 BookDTO 클래스를 만드세요.

컨트롤러에서 BookDTO 타입의 객체를 생성하고 화면으로 전달하세요.

화면에서는 해당 정보를 출력하세요.

```
public class BookDTO {  
  
    String title; //제목  
  
    String author; //저자  
  
    LocalDate publicationDate; //출간일  
  
}
```

DTO 클래스



A screenshot of a web browser window. The address bar shows 'localhost:8080/quiz/q4'. The main content area displays the following text:

제목: 스프링부트웹프로젝트

저자: 구멍가게코딩단

출간일: 2022-12-25

결과 화면

### 반복문

- th:each를 사용하여 리스트 안에 있는 요소를 반복적으로 처리하고, 각 요소에 대한 태그를 생성할 수 있다.
- 리스트, 배열의 값을 하나씩 꺼내서 왼쪽 변수에 저장한다.

### 반복 상태

- 반복문에서 제공하는 상태객체를 사용하여 반복상태를 확인 할 수 있다.

반복문

```
th:each = "변수: ${목록}"
```

반복문의 상태객체

```
th:each="dto, status : ${list}"
```

반복문 렌더링 후

- {index = 0, count = 1, size = 3, current = SampleDTO(no=1, text=aaa, regDate=2024-01-03)}
- {index = 1, count = 2, size = 3, current = SampleDTO(no=2, text=bbb, regDate=2024-01-03)}
- {index = 2, count = 3, size = 3, current = SampleDTO(no=3, text=ccc, regDate=2024-01-03)}

### 조건문

- th:if는 해당 조건을 만족하면 HTML태그를 화면에 나타낸다.  
조건이 맞지 않으면 태그 자체를 렌더링 하지 않는다.
- th:unless는 반대로 조건을 만족하지 않으면 HTML태그를 화면에 나타낸다.

### 조건문 사용시

- 코드: `<p th:if="{10%2 == 0}">10은 짝수 입니다</p>`
- 코드 렌더링 후: `<p> 10은 짝수 입니다 </p>`

th:if = "{조건식}"

th:unless = "{조건식}"

## 블록

- HTML 요소 없이 타임리프 속성을 사용할 수 있다.
- `<p>` `<h1>` 같은 HTML 요소없이 타임리프 속성을 사용할 수 있다.

html요소에 타임리프 사용

```
< li th:each="dto : ${list}" >
```

블록태그로 타임리프

사용

```
< th:block th:each="dto: ${list}" >
```

### Inline

- 자바스크립트에서 인라인 기능을 사용하려면 th:inline="javascript " 속성을 사용해야 한다.
- 인라인 기능을 사용하면 자바스크립트 코드에서 타임리프 표현식을 사용할 수 있다.

### Inline을 사용하지 않은 경우 발생하는 문제점

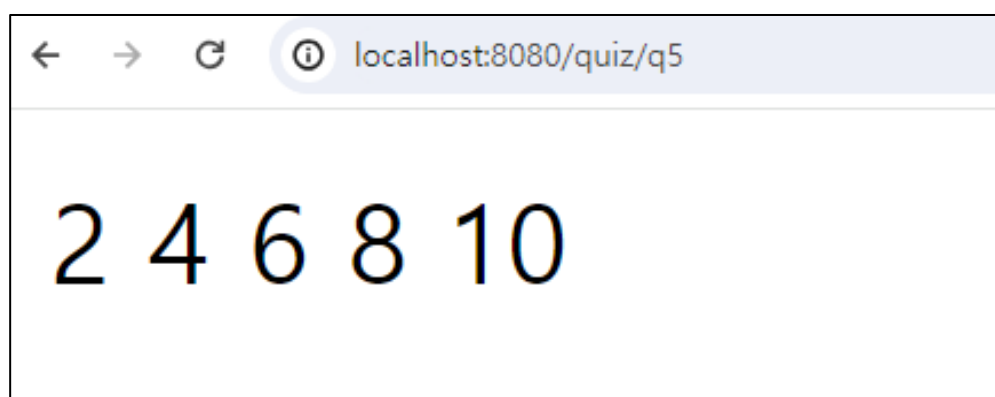
- 인라인 기능을 사용하면, 서버에서 전달받은 데이터를 자바스크립트에서 사용할 수 있는 자료형으로 변환해준다.
- 인라인 기능을 사용하지 않으면, 순수한 텍스트로 해석하여 데이터를 바로 사용할 수 없다.

```
<!-- 인라인 사용 전 -->
<script>
    var msg = [[${result}]]; //success
    var dto = [[${dto}]]; //SampleDTO(no=1, text=aaa, regDate=2024-05-28)
</script>

<!-- 인라인 사용 후 -->
<script th:inline="javascript">
    var msg = [[${result}]]; //"success"
    var dto = [[${dto}]]; //{ "no":1, "text":"aaa", "regDate":"2024-05-28" }
</script>
```



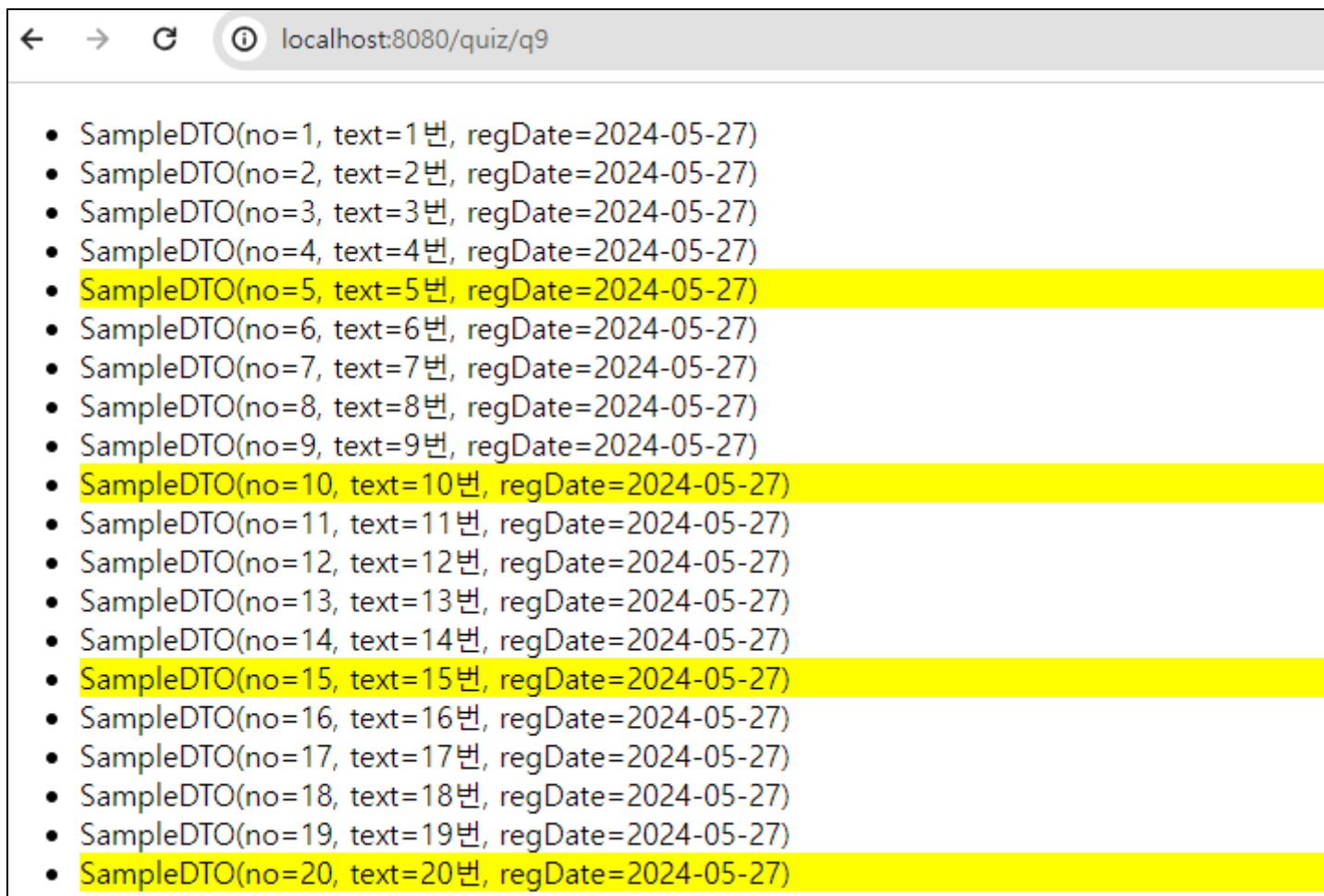
**Q5.** localhost:8080/quiz/q5 주소에 접속했을 때 아래와 같이 화면을 표시되도록 코드를 구현하세요.  
컨트롤러에서 10개 크기의 배열을 생성하고, 1~10을 담아서 화면으로 전달하세요.  
화면에서는 반복문과 조건문을 사용하여 배열의 요소 중 짝수만 출력하세요.



**Q6.** localhost:8080/quiz/q6 주소에 접속했을 때 아래와 같이 화면을 표시되도록 코드를 구현하세요.  
컨트롤러에서 3개 크기의 PersonDTO형 리스트를 생성하고 화면으로 전달하세요.  
화면에서는 반복문을 사용하여 사람 목록을 테이블 형태로 출력하세요.

localhost:8080/quiz/q6		
이름	나이	주소
둘리	20	인천 구월동
또치	30	서울 신림동
도우너	40	부산 문래동

**Q7.** localhost:8080/quiz/q7 주소에 접속했을 때 아래와 같이 화면을 표시되도록 코드를 구현하세요.  
컨트롤러에서 20개 크기의 SampleDTO형 리스트를 생성하고 화면으로 전달하세요.  
화면에서는 목록을 출력하고, 번호가 5의 배수일 경우에는 노란색 배경을 적용하세요.



## 타임리프의 기본객체

- 타임리프는 문자, 숫자, 날짜 등 다양한 유형의 데이터를 처리할 수 있는 객체들을 제공한다.

### 타임리프 객체들

#message : 메시지를 다국어로 처리하는데 사용됨

#uris : URI 주소에 특수문자가 포함된 경우에 사용함

#dates, #calendars, #temporals : 날짜와 시간을 처리하는데 사용됨 (날짜 형식, 날짜 계산 등)

#numbers : 숫자를 처리하는데 사용됨 (숫자 형식, 계산 등)

#strings : 문자를 처리하는데 사용됨 (문자열 가르기, 대소문자 변환 등)

#objects : 객체를 처리하는데 사용됨 (객체의 속성에 접근하거나 객체 출력 등)

#bools : boolean 처리하는데 사용됨

#arrays : 배열 처리하는데 사용됨 (배열요소에 접근하거나 배열의 길이 처리 등)

#lists , #sets , #maps : 컬렉션 처리하는데 사용됨

**Q8.** localhost:8080/quiz/q8 주소에 접속했을 때 아래와 같이 화면을 표시되도록 코드를 구현하세요.  
컨트롤러에서 5개 크기의 PersonDTO형 리스트를 생성하고 화면으로 전달하세요.  
화면에서는 모든 사람의 정보를 출력하고, 인천 거주 여부를 표시하세요.

성	이름	나이	주소	인천 거주 여부
박	하나	25	인천 구월동	거주
홍	재범	17	서울 신림동	비거주
문	유리	31	부산 문래동	비거주
김	재민	8	인천 연수동	거주
장	유라	33	부산 문래동	비거주

**Q9.** localhost:8080/quiz/q9 주소에 접속했을 때 아래와 같이 화면을 표시하도록 코드를 구현하세요.  
컨트롤러에서 5개 크기의 PersonDTO형 리스트를 생성하고 화면으로 전달하세요.  
화면에서는 미성년자만 찾아서 출력하세요.

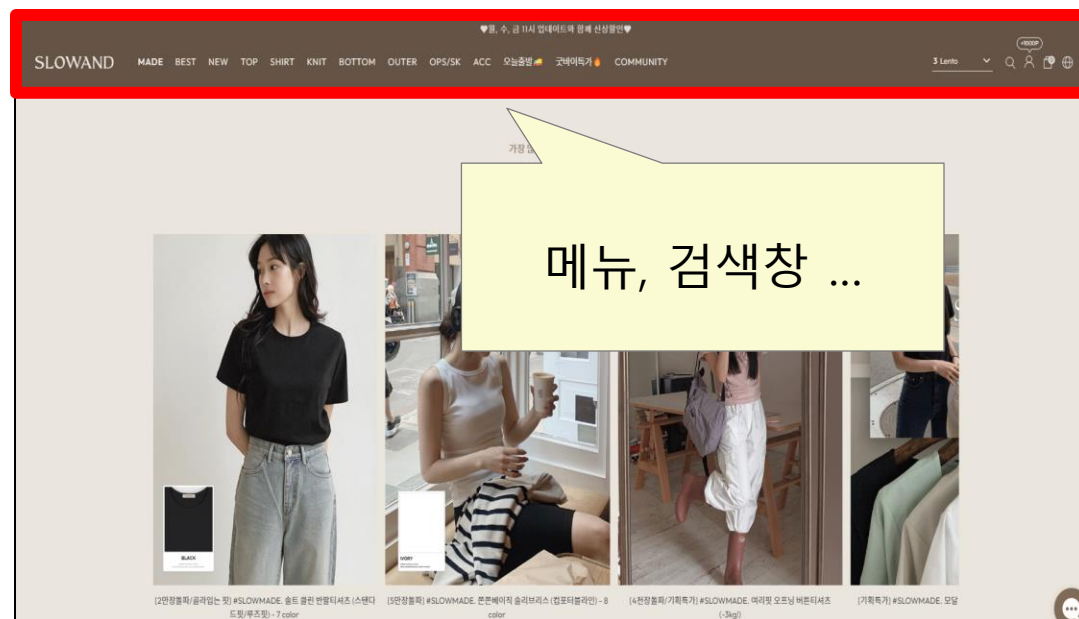
localhost:8080/quiz/q8			
성	이름	나이	주소
홍	재범	17	서울 신림동
김	재민	8	인천 연수동

# 템플릿 레이아웃1

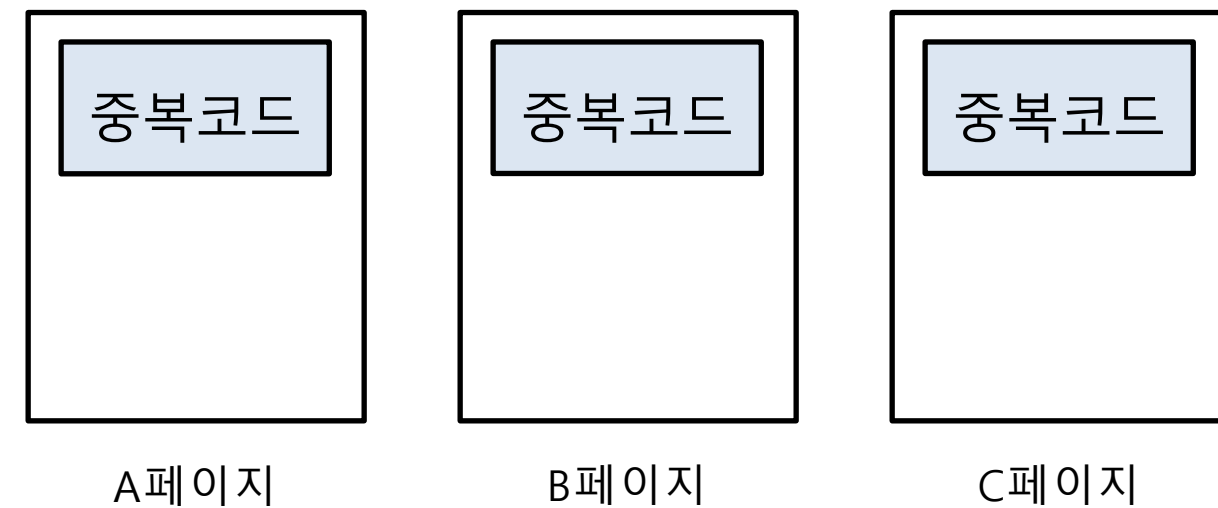
## 템플릿 레이아웃이란?

### 일반적인 사이트의 구조

- 웹페이지를 개발 할 때, 여러 페이지에서 공통으로 사용하는 부분이 있다.
- 예를 들어 헤더에는 메뉴, 검색창이 있고 푸터에는 연락처, 사이트 맵 등이 있다.
- 이러한 공통 부분은 일관성과 유지보수를 고려하여 관리해야 한다.
- 만약 공통 부분은 복사해서 사용한다면, 공통 부분을 수정할 때 각페이지에서 일일이 수정해야한다.



사이트 예시



# 템플릿 레이아웃1

## 템플릿 레이아웃이란?

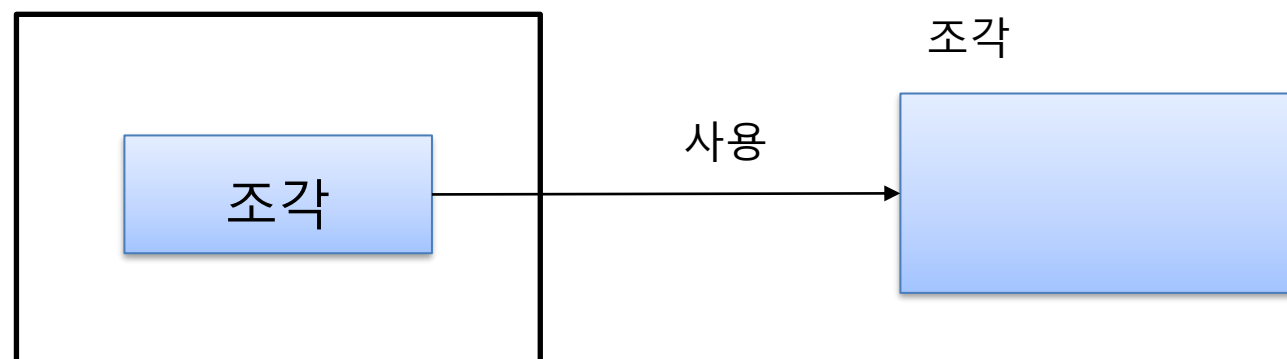
### 타임리프의 레이아웃

- 타임리프의 레이아웃 기능을 사용하면 페이지의 공통부분을 쉽게 관리할 수 있다.

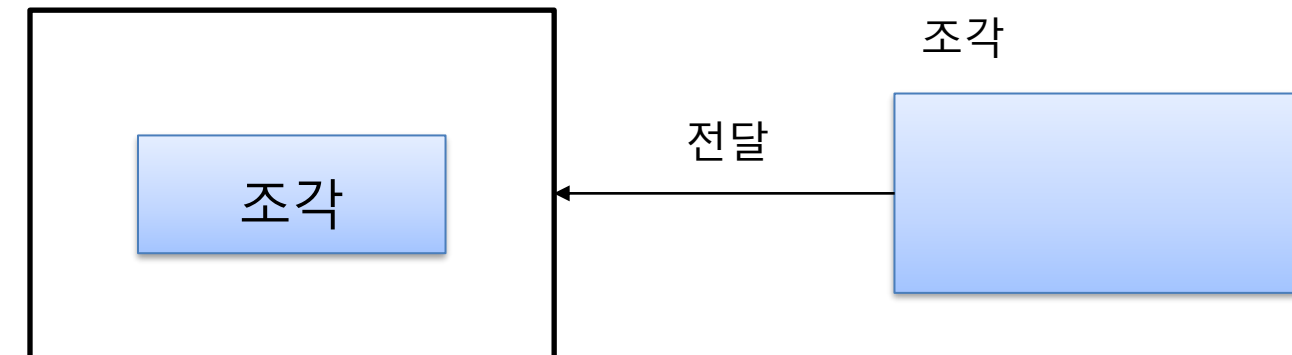
### 레이아웃 사용 방법

1. 특정부분을 가지고 와서 포함하는 방식
2. 특정부분을 파라미터로 전달하는 방식

레이아웃



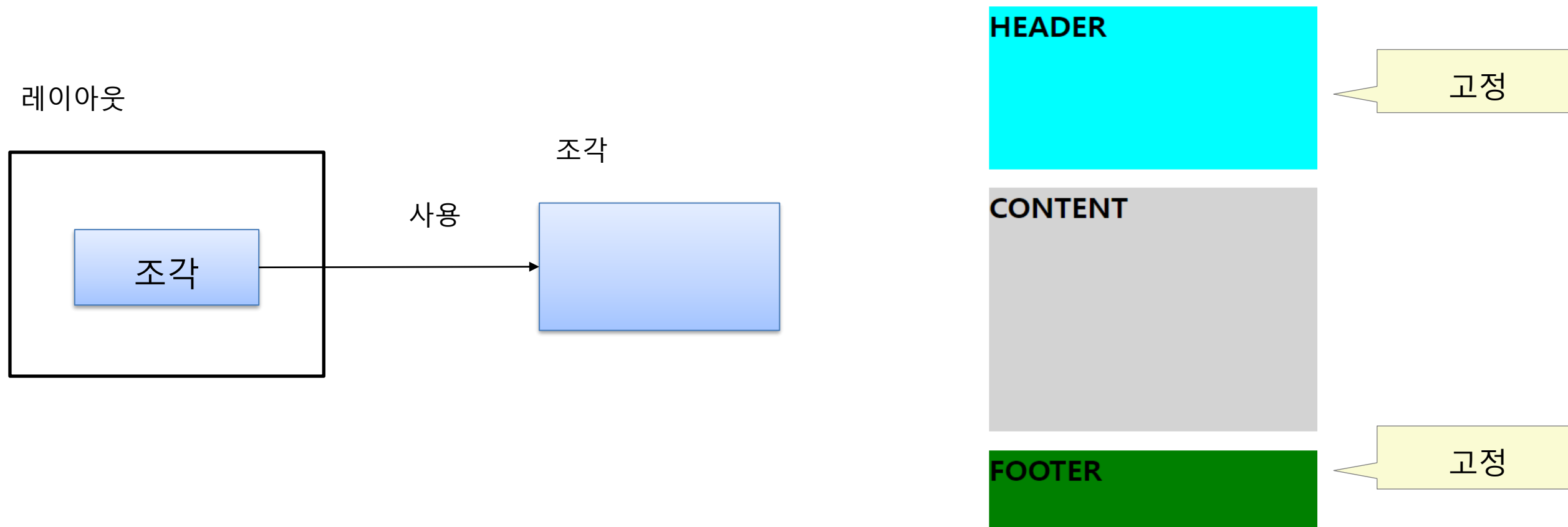
레이아웃





### 조각 가져오는 방식

- 템플릿 파일에서 외부 파일에 정의된 코드 조각을 포함시키는 방법이다.
- 타임리프를 사용하여 헤더와 푸터는 고정하고中间的 content 영역만 변경하는 레이아웃을 만들 수 있다.



## 1. 레이아웃 파일 만들기

전체 화면을 담당하는 기본 레이아웃 파일을 만든다.

헤더와 푸터는 고정하고 중간에 CONTENT 영역만 다른 내용으로 교체한다.

```
<body>
<div class="header">
<h1>HEADER</h1>
</div>
<div class="content" >
<h1>CONTENT</h1>
</div>
<div class="footer">
<h1>FOOTER</h1>
</div>
</body>
```



HEADER

CONTENT

FOOTER

## 2. 조각 파일 만들기

별도의 html 파일을 만들고, 'th:fragment' 속성을 추가하여 코드 조각으로 정의한다.

```
<h1 th:fragment="frag">새로운 내용</h1>
```

## 3. 레이아웃에서 조각 사용하기

이제 레이아웃에서 조각 파일을 가져와서 사용한다.

'th:replace' 속성을 사용하여 해당 태그를 조각으로 교체한다.

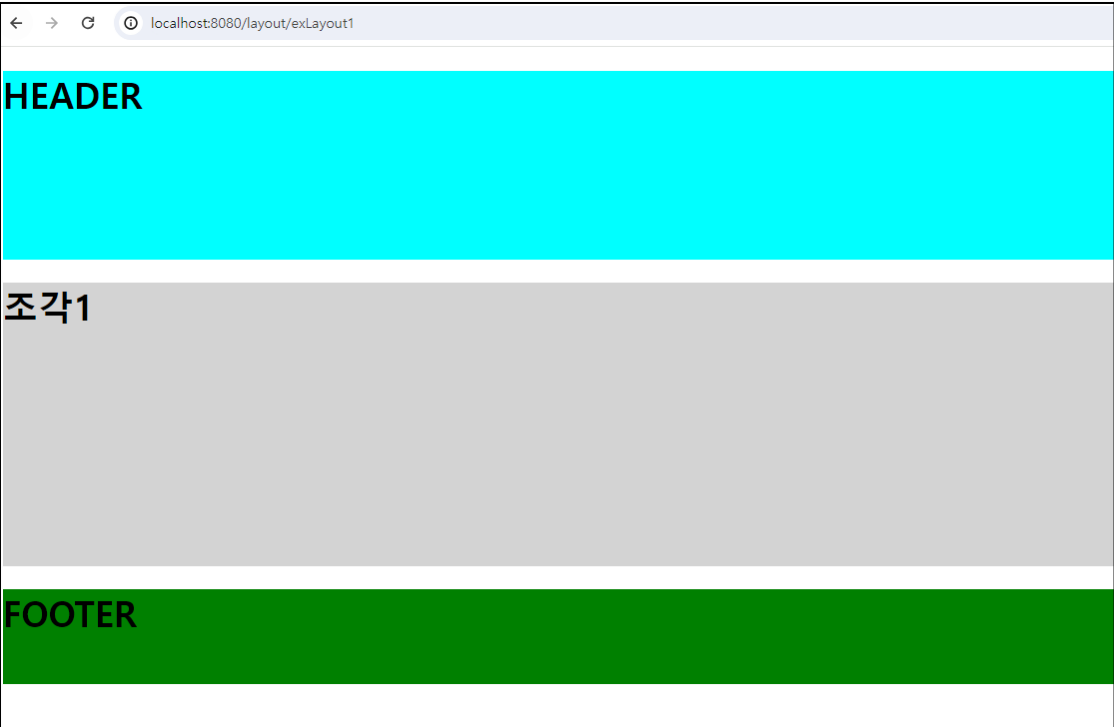
```
<div class="content" >  
    <th:block th:replace = "~{layout/fragment :: frag}"></th:block>  
</div>
```

파일경로 :: 조각이름

#### 4. 테스트

localhost:8080/layout/exLayout 주소로 접속한다.

전체 화면은 레이아웃을 사용했고, 중간 부분은 다른 페이지의 코드를 사용하였다.

	<pre>22 23 &lt;div class="header"&gt; 24   &lt;h1&gt;HEADER&lt;/h1&gt; 25 &lt;/div&gt; 26 &lt;div class="content"&gt; 27   &lt;h1&gt;조각1&lt;/h1&gt; 28 &lt;/div&gt; 29 &lt;div class="footer"&gt; 30   &lt;h1&gt;FOOTER&lt;/h1&gt; 31 &lt;/div&gt; 32</pre>
---	---

결과 화면

렌더링된 소스 코드

레이아웃과 조각을 사용하면, 같은 구조를 가지는 여러 페이지를 만들 수 있다.

그러나 공통으로 사용하는 헤더 부분을 수정할 때, 모든 페이지를 일일이 수정해야 하는 문제점이 있다.



```
<style>
  .header {
    height: 20vh;
    background-color: pink;
  }
```

수정..

첫번째 페이지

```
<style>
  .header {
    height: 20vh;
    background-color: pink;
  }
```

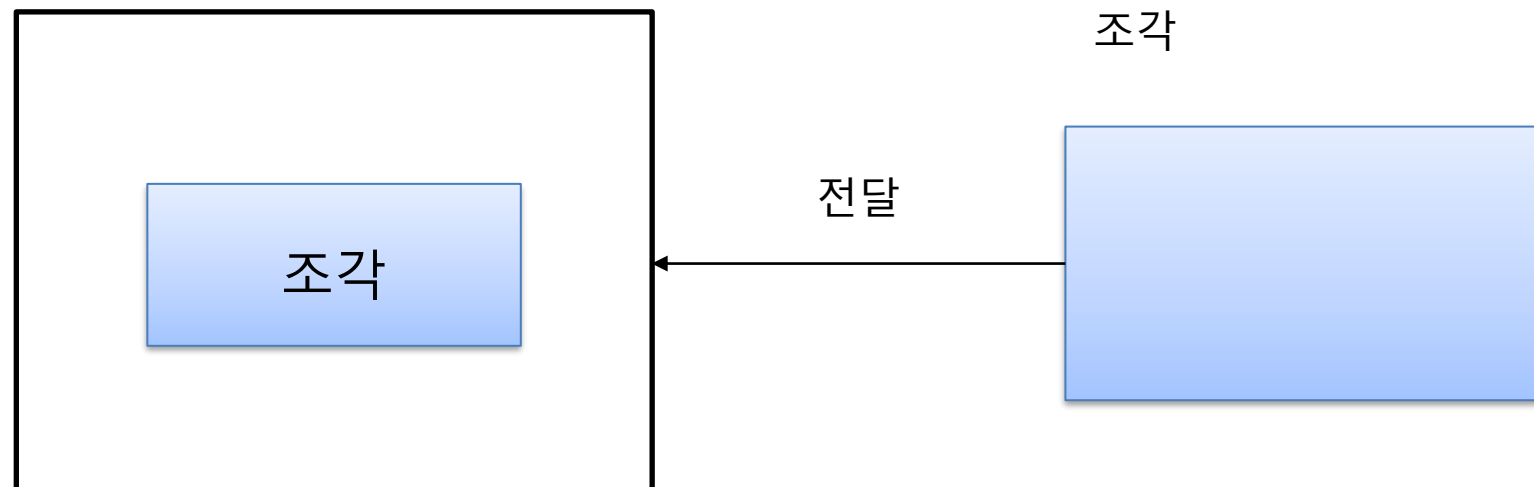
또 수정..

두번째 페이지

### 조각을 전달하는 방식

- 조각 파일에서 템플릿 파일로 코드 조각을 전달하는 방법이다.
- 공통의 레이아웃을 만들고, 여러 페이지에서 재사용 할 수 있다.

레이아웃



## 1. 레이아웃 파일 만들기

전체 화면을 담당하는 기본 레이아웃 파일을 만든다.

문서의 위쪽과 아래쪽에 <th:block> 처리를 한다.

th:fragment 속성으로 content라는 파라미터를 받을 수 있도록 처리한다.

```
<html>
<th:block th:fragment="setContent(content)">
...
</th:block>
</html>
```

## 2. 레이아웃에서 조각 사용하기

중간 부분은 th:replace 속성을 이용해서 파라미터로 전달받은 content를 출력한다.

```
<div class="content" >
    <th:block th:replace = "${content}"></th:block>
</div>
```

### 3. 조각 파일 만들기

별도의 html 파일을 만들고, 'th:fragment' 속성을 추가하여 코드 조각으로 정의한다.

'th:replace' 속성을 사용하여 exLayout3 파일에 조각을 전달한다.

this:content는 현재 파일의 content 조각을 의미한다.

```
<th:block th:replace="~/layout/exLayout3 :: setContent(~{this::content})">

    <th:block th:fragment="content">
        <h1>조각1</h1>
    </th:block>

</th:block>
```



#### 4. 테스트

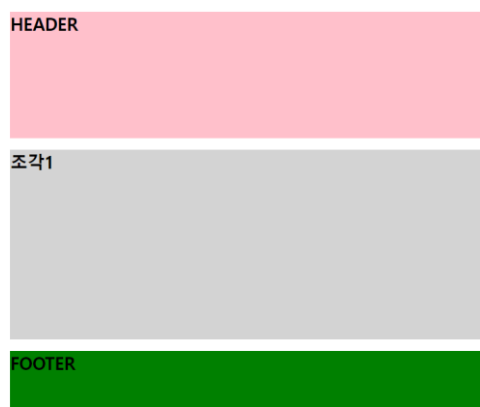
localhost:8080/layout/fragment2 주소로 접속한다.

전체 화면은 템플릿 파일을 사용했고, 중간 부분은 조각 코드를 사용하였다.

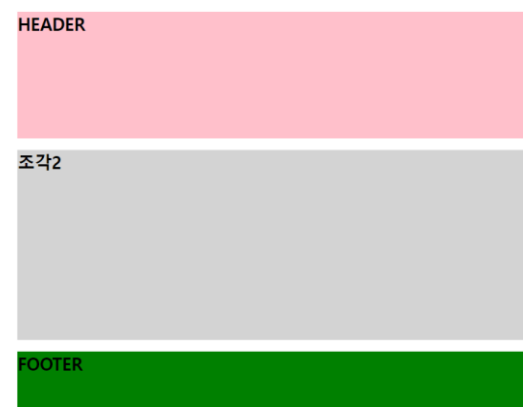


#### 공통 부분 변경하기

레이아웃만 수정하면, 모든 페이지의 헤더가 한번에 변경된다.



1번 페이지



2번 페이지

### 부트스트랩이란?

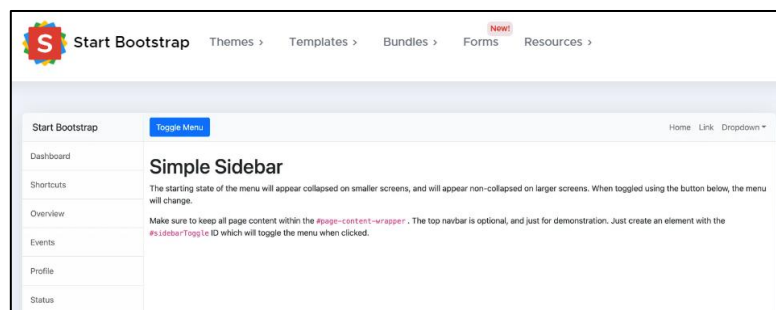
- 부트스트랩은 웹 개발에 사용되는 프론트엔드 오픈소스이다.
- 개발자들이 웹 프로그램을 빠르게 개발할 수 있도록 돕는 위해 만들어졌다.
- html, css, javascript 파일로 구성되어 있으며, 완성된 디자인을 재사용 할 수 있다.



부트스트랩 사이트

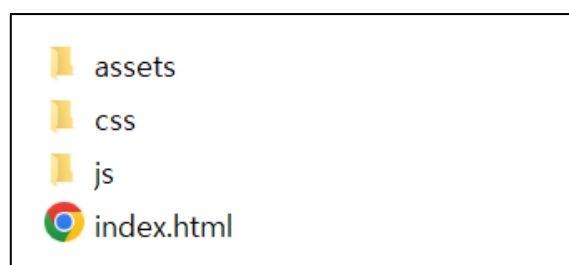
이후에 예제에서 사용할 템플릿을 만든다.

1. 부트스트랩 사이트에서 "simple-sidebar"라는 템플릿을 다운로드한다.



<https://startbootstrap.com/template/simple-sidebar>

2. 다운로드 받은 파일의 압축을 해제하고, 모든 파일을 프로젝트의 static 폴더로 복사한다.

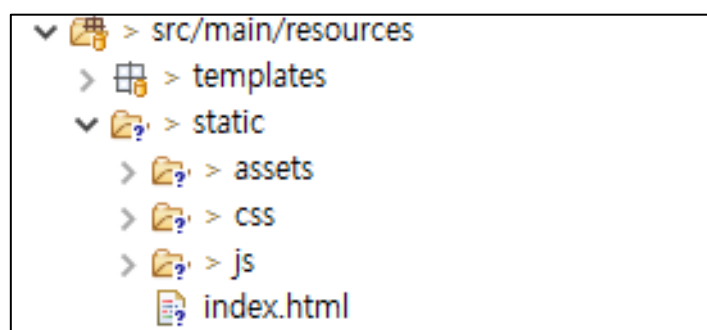


**assets:** 아이콘

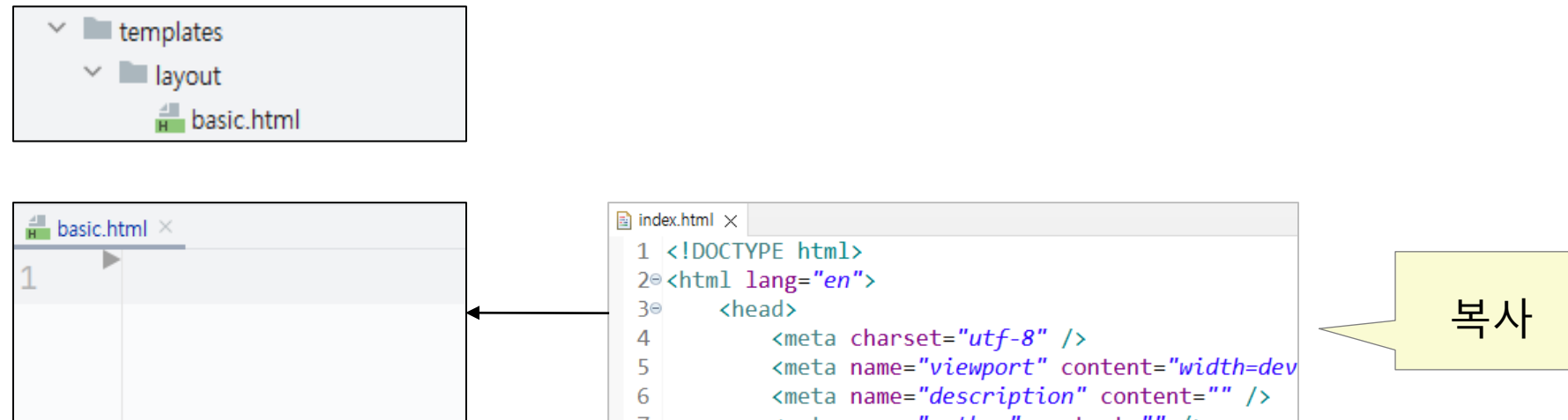
**css:** html 디자인 처리를 위한 스타일시트

**js:** html 이벤트 처리를 위한 자바스크립트

**index.html:** html 샘플 파일

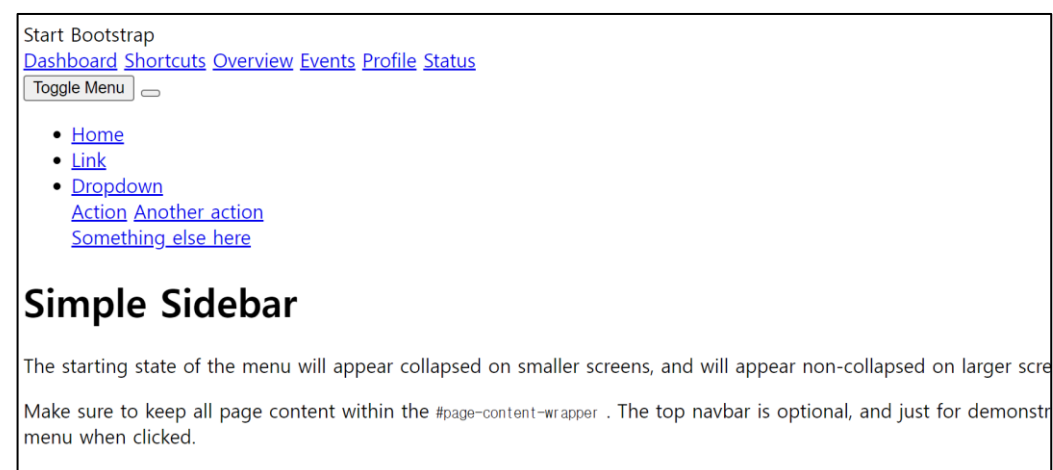


3. basic.html 파일을 생성하고, index.html의 내용을 그대로 복사한다.



이제 /layout/basic 주소로 접속한다.

리소스 파일의 경로를 찾지 못해서 화면이 깨져버리는 문제가 발생했다.



4. basic.html 파일에서 링크를 수정한다.

assets, css, js 파일을 연결하는 링크를 찾아서 앞에 "/" 를 붙인다.

```
<!-- Favicon-->  
<link rel="icon" type="image/x-icon" href="/assets/favicon.ico" />  
<!-- Core theme CSS (includes Bootstrap)-->  
<link href="/css/styles.css" rel="stylesheet" />
```

/가 없을 때 → <http://localhost:8080/layout/css/styles.css>

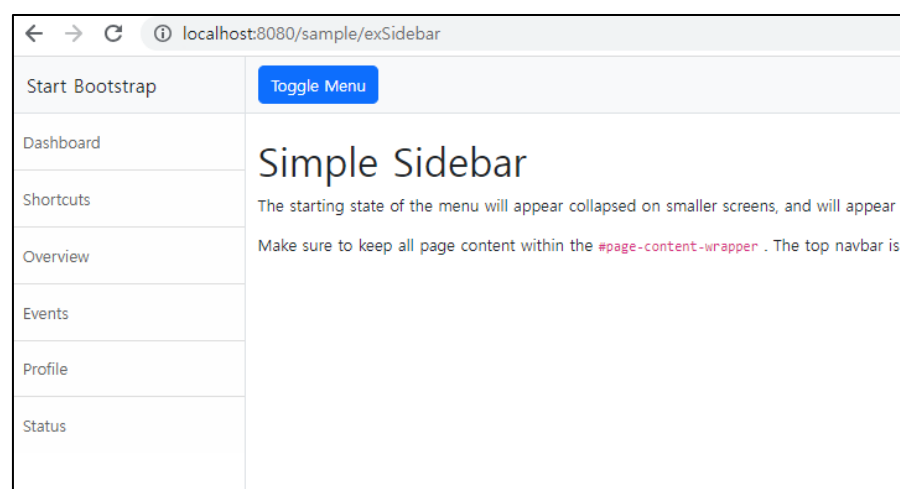
/가 있을 때 → <http://localhost:8080/css/styles.css>

하단에 있는 자바스크립트 파일의 링크를 상단으로 이동한다.

링크가 바디 태그보다 밑에 있으면 기능을 사용할 수 없다.

```
<!-- Bootstrap core JS-->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"></script>
<!-- Core theme JS-->
<script src="/js/scripts.js"></script>
```

다시 /layout/basic 주소로 접속하면 화면이 정상적으로 나타난다.



5. basic.html 파일을 레이아웃으로 만든다.

문서의 위쪽과 아래쪽에 <th:block> 처리를 한다.

th:fragment 속성으로 content라는 파라미터를 받을 수 있도록 처리한다.

```
<html>
<th:block th:fragment="setContent(content)">
...
</th:block>
</html>
```

중간에 container-fluid 부분의 내용을 삭제하고, 파라미터로 전달받은 content를 출력한다.

```
<!-- Page content-->
<div class="container-fluid">
  <h1 class="mt-4">Simple Sidebar</h1>
  <p>The starting state of the menu will appear collapsed on
  <p>
    Make sure to keep all page content within the
    <code>#page-content-wrapper</code>
    . The top navbar is optional, and just for demonstrat
    <code>#sidebarToggle</code>
    ID which will toggle the menu when clicked.
  </p>
</div>
```

```
<!-- Page content-->
<div class="container-fluid">
  <th:block th:replace = "${content}"></th:block>
</div>
```

6. 조각 파일을 만든다.

exSidebar.html 파일을 생성한다.

'th:fragment' 속성을 추가하여 코드 조각으로 정의한다.

'th:replace' 속성을 사용하여 basic 파일에 content 조각을 전달한다.

```
<th:block th:replace="~/layout/basic :: setContent(~{this::content})">

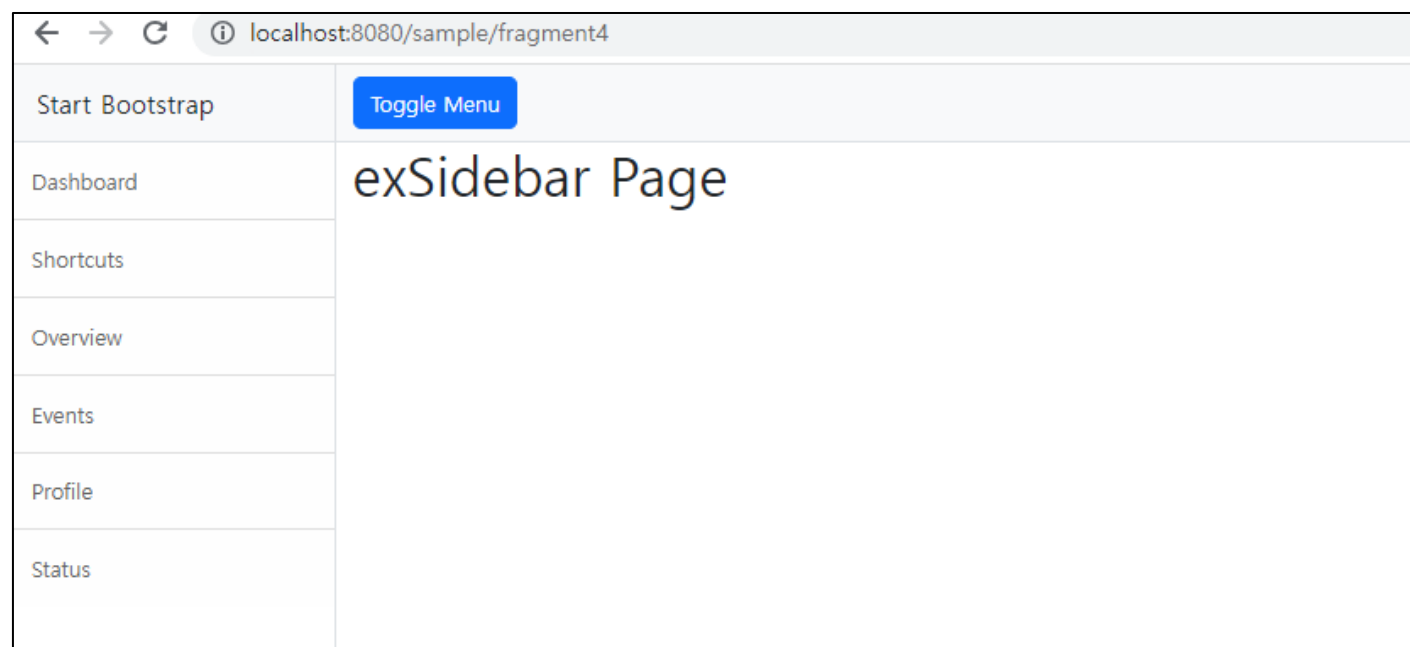
    <th:block th:fragment="content">
        <h1>exSidebar Page</h1>
    </th:block>

</th:block>
```



## 테스트

브라우저에서 `/layout/exSidebar` 주소로 접속하면 다음과 같이 화면이 나타난다.

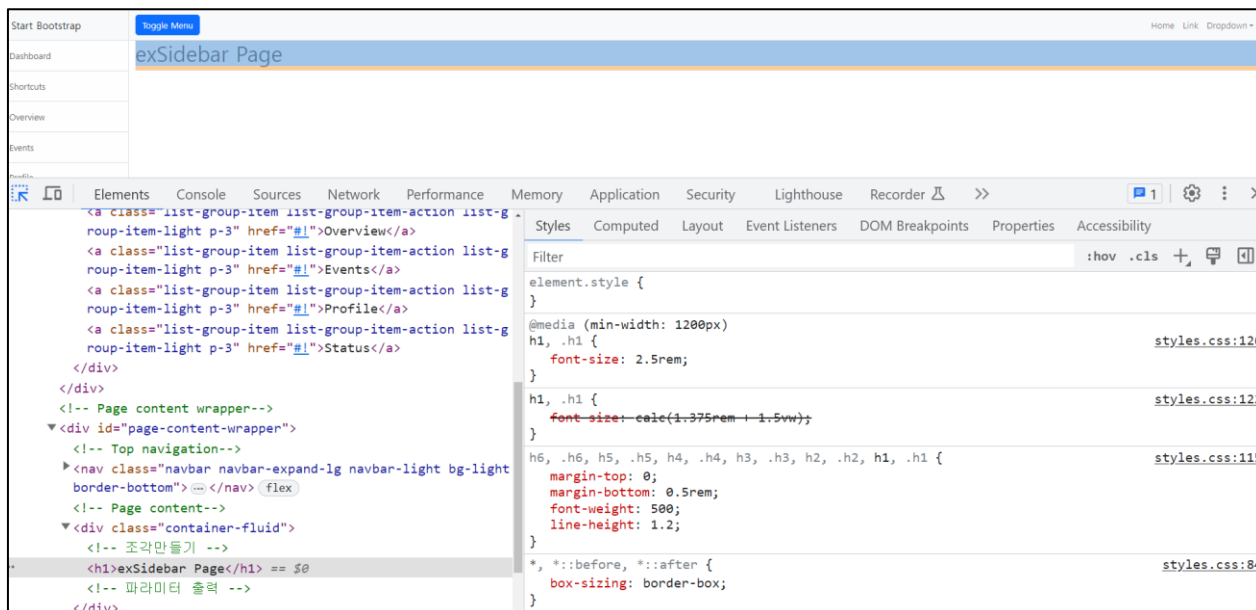


### 개발자도구로 코드 확인하기

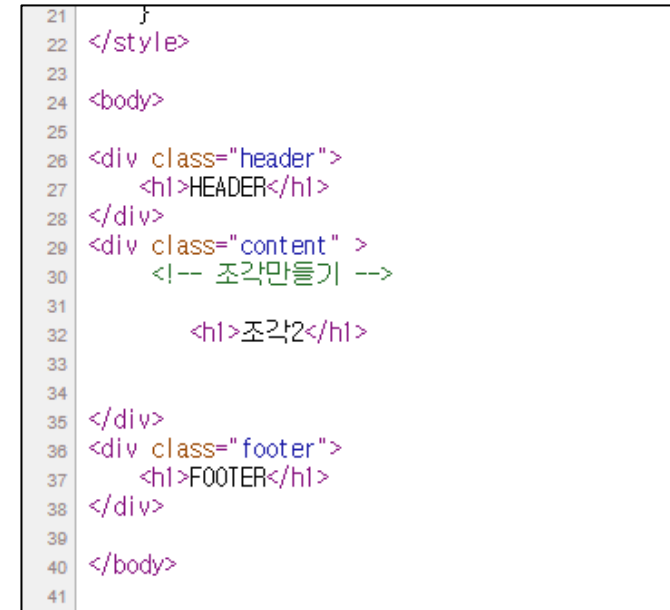
1. 브라우저 개발자도구에서 "select 메뉴"를 클릭한다.
2. 화면에서 "page content" 영역을 선택한다.

### 페이지 소스보기로 코드 확인하기

1. 마우스 우 클릭을 한다.
2. "페이지 소스보기 메뉴"를 선택한다.



개발자도구 – element  
메뉴



페이지 소스 보기 메뉴