2024-2 오픈소스 개발 프로젝트01

SW중심대학사업단 강재구

(학연산 742호, 010-2278-8192, kangjk@cbnu.ac.kr)

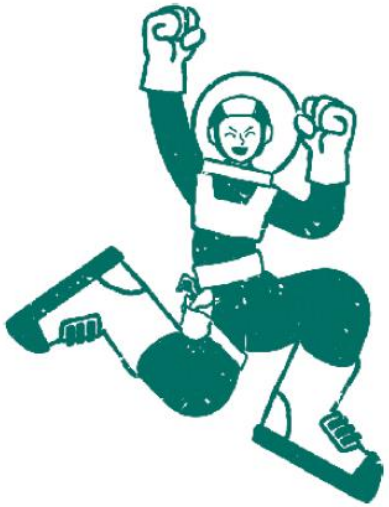
04

충북대학교
SW중심대학사업단

02. 스프링 부트의 기본 기능 익히기 (3)

02

스프링 부트의 기본 기능 익히기



2-13 웹 페이지 디자인하기

- 지금까지 질문 목록 화면과 질문 상세 화면을 만들었음
- 이제는 좀 더 그럴싸한 화면을 만들려면 화면을 디자인해야 함.
  즉, 뼈대만 있는 화면에 옷을 입혀 보는 것임

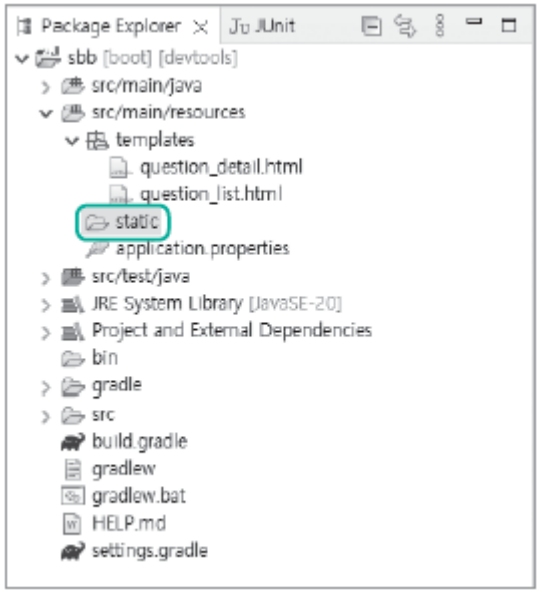
- 웹 개발에서는 색상이나 크기 등의 디자인을 적용할 때
  스타일시트(stylesheet, CSS)를 사용함.
  이번에는 SBB에 스타일시트를 적용해 보자

- 질문 상세 화면에 스타일시트를 적용하여 답변을 입력받는 텍스트 창의 크기를
  지금보다 넓히고 버튼 상단에 여백도 추가해 볼 것임

2-13 웹 페이지 디자인하기

▪ 스태틱 디렉터리와 스타일시트 이해하기

- 스타일시트 파일, 즉 CSS 파일은 HTML 파일과 달리 스태틱(static) 디렉터리에 저장해야 함



- 스프링 부트의 스태틱 디렉터리는 오른쪽과 같이 src/main/resources 디렉터리 안에 있음

- 스태틱 디렉터리를 확인했으니 앞으로 CSS 파일은 스태틱 디렉터리에 저장함

2-13 웹 페이지 디자인하기

▪ 스태틱 디렉터리와 스타일시트 이해하기

- 화면을 본격적으로 디자인하기에 앞서
  먼저 스타일시트 파일(style.css)을 만들어 보자

- static 디렉터리를 선택한 후,
  마우스 오른쪽 버튼을 누르고 [New → File]을 클릭함.
  파일 이름으로 style.css를 입력하여 스타일시트 파일을 만듦

- 그리고 다음 내용을 입력해 보자

• /static/style.css

```
textarea {
    width:100%;
}

input[type=submit] {
    margin-top:10px;
}
```

텍스트 창의 넓이를 설정한다.

버튼 상단의 마진을 설정한다.

2-13 웹 페이지 디자인하기

- **스태틱 디렉터리와 스타일시트 이해하기**
  - style.css 파일에 질문 상세 화면의 디자인 요소들을 작성함
  - 답변 등록 시 사용하는 텍스트 창의 넓이를 100%로 하고 [답변 등록] 버튼 상단에 마진을 10픽셀로 설정함

2-13 웹 페이지 디자인하기

▪ 템플릿에 스타일 적용하기

1. 이제 작성한 스타일시트 파일(style.css 파일)을 질문 상세 페이지 템플릿에 적용해 보자.

• /templates/question_detail.html

```
<link rel="stylesheet" type="text/css" th:href="@{/style.css}">
<h1 th:text="${question.subject}"></h1>
<div th:text="${question.content}"></div>
<h5 th:text="¦${#lists.size(question.answerList)}개의 답변이 있습니다.¦"></h5>
<div>
    <ul>
        <li th:each="answer : ${question.answerList}"
        th:text="${answer.content}"></li>
    </ul>
</div>
<form th:action="@{¦/answer/create/${question.id}¦}" method="post">
    <textarea name="content" id="content" rows="15"></textarea>
    <input type="submit" value="답변 등록">
</form>
```

이와 같이 question_detail.html 파일 상단에 style.css를 사용할 수 있는 링크를 추가하여 스타일시트 파일을 상세 페이지 템플릿에 적용함

2-13 웹 페이지 디자인하기

▪ 템플릿에 스타일 적용하기

2. 브라우저에 http://localhost:8080/question/detail/2를 입력해 질문 상세 화면이 어떻게 변경되었는지 확인해 보자.

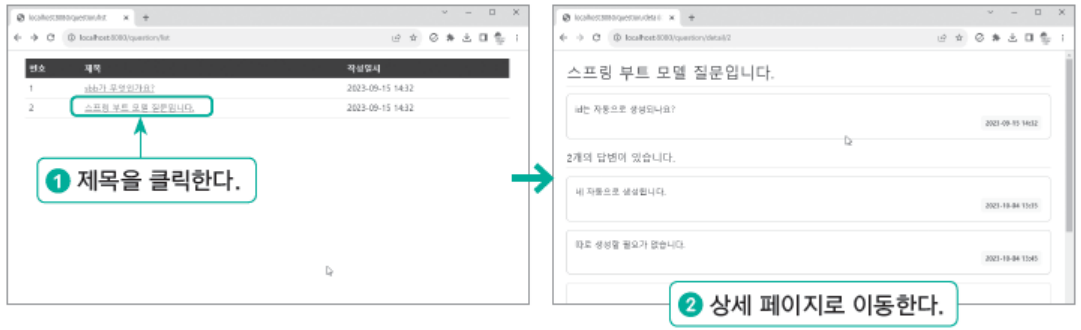
   다음처럼 스타일이 적용된 화면을 볼 수 있음

2-14 부트스트랩으로 화면 꾸미기

▪ 템플릿에 스타일 적용하기

- 부트스트랩(bootstrap)은 트위터(Twitter)를 개발하면서 만들어졌으며
  현재 지속적으로 관리되고 있는 오픈소스 프로젝트로,
  웹 디자이너의 도움 없이도 개발자 혼자서 상당히 괜찮은 웹페이지를 만들 수 있게
  도와주는 프레임워크임

- 질문 목록 페이지와 상세 페이지에 부트스트랩을 적용하여
  SBB 게시판 서비스를 좀 더 멋지게 만들어 보자

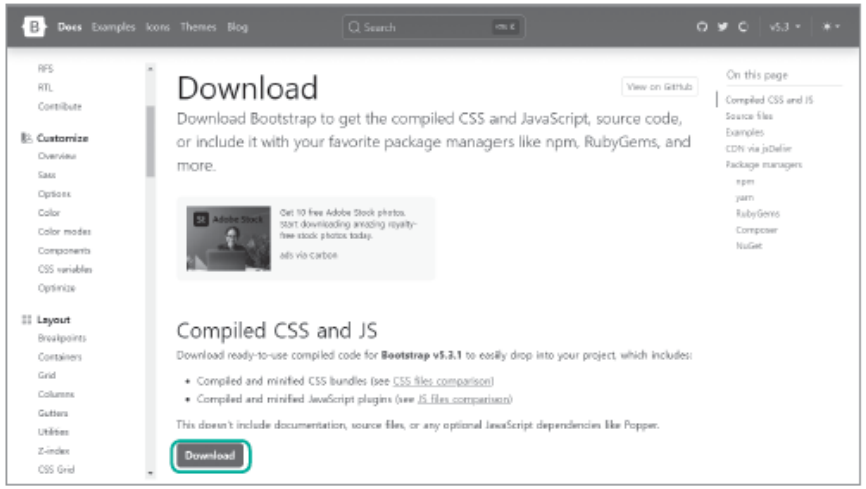


2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 설치하기

1. 우선 다음 URL에서 부트스트랩을 내려받자

```
https://getbootstrap.com/docs/5.3/getting-started/download/
```



2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 설치하기

2. 부트스트랩 다운로드 페이지에 접속한 후
[Download] 버튼을 클릭해 다음과 같은 이름의 파일을 내려받자

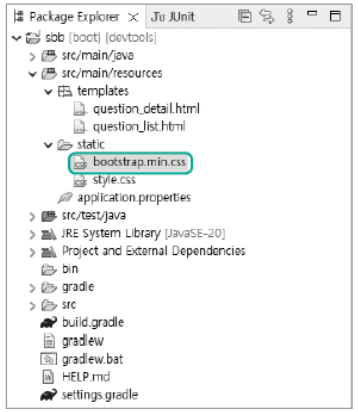
```
bootstrap-5.3.1-dist.zip
```

2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 설치하기

3. 압축 파일 안에 많은 파일들이 있는데 이 중에서 bootstrap.min.css 파일을 복사하여 스태틱 디렉터리에 저장하자

구분	파일 위치
압축 파일 내 경로	bootstrap-5.3.1-dist.zip/bootstrap-5.3.1-dist/css/bootstrap.min.css
붙여넣기할 폴더 경로	workspace/sbb/src/main/resources/static/bootstrap.min.css



2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 적용하기

1. 먼저 질문 목록 템플릿에 부트스트랩을 적용해 보자

• /templates/question_list.html

```
<link rel="stylesheet" type="text/css" th:href="@{/bootstrap.min.css}">
<div class="container my-3">
    <table class="table">
        <thead class="table-dark">
            <tr>
                <th>번호</th>
                <th>제목</th>
                <th>작성일시</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="question, loop : ${questionList}">
                <td th:text="${loop.count}"></td>
                <td>
                    <a th:href="@{|/question/detail/${question.id}|}"
                    th:text="${question.subject}"></a>
                </td>
                <td th:text="${#temporals.format(question.createDate,
                'yyyy-MM-dd HH:mm')}"></td>
            </tr>
        </tbody>
    </table>
</div>
```

2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 적용하기

1. 테이블 항목으로 '번호'를 추가함.
   번호는 loop.count를 사용하여 표시함.
   loop.count는 questionList의 항목을 th:each로 반복할 때 현재의 순서를 나타냄.

   날짜를 보기 좋게 출력하기 위해
   타임리프의 #temporals.format 기능을 사용함.

   #temporals.format은 #temporals.format(날짜 객체, 날짜 포맷)와 같이 사용하는데,
   날짜 객체를 날짜 포맷에 맞게 변환함

2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 적용하기

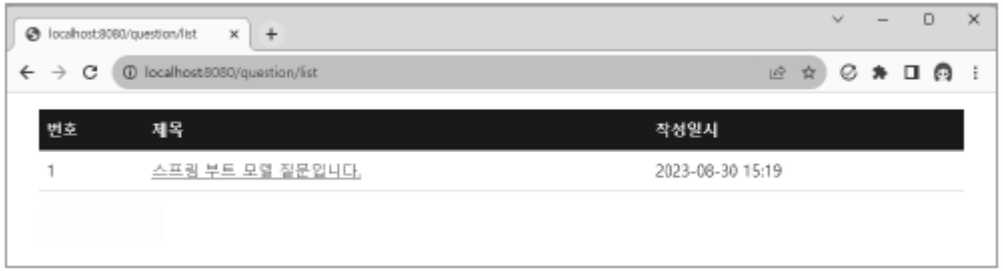
1. 가장 윗줄에 bootstrap.min.css를 사용할 수 있도록 링크를 추가함.

   그리고 위에서 사용한 class="container my-3", class="table", class="table-dark 등은 bootstrap.min.css에 이미 정의되어 있는 클래스들로 간격을 조정하고 테이블에 스타일을 지정하는 용도로 사용함

2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 적용하기

2. 부트스트랩을 적용한 질문 목록 페이지를 볼 수 있을 것임



2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 적용하기

3. 이어서 질문 상세 템플릿에도 다음처럼 부트스트랩을 적용하자

• /templates/question_detail.html

```
<link rel="stylesheet" type="text/css" th:href="@{/bootstrap.min.css}">
<div class="container my-3">
    <!-- 질문 -->
    <h2 class="border-bottom py-2" th:text="${question.subject}"></h2>
    <div class="card my-3">
        <div class="card-body">
            <div class="card-text" style="white-space: pre-line;"
            th:text="${question.content}"></div>
            <div class="d-flex justify-content-end">
                <div class="badge bg-light text-dark p-2 text-start">
                    <div th:text="${#temporals.format(question.createDate,
                    'yyyy-MM-dd HH:mm')}"></div>
```

HTML에서는 이와 같이 주석을 나타낸다.

```
            </div>
        </div>
    </div>
</div>
<!-- 답변 개수 표시 -->
<h5 class="border-bottom my-3 py-2"
    th:text="¦${#lists.size(question.answerList)}개의 답변이 있습니다.¦"></h5>
<!-- 답변 반복 시작 -->
<div class="card my-3" th:each="answer : ${question.answerList}">
    <div class="card-body">
        <div class="card-text" style="white-space: pre-line;"
        th:text="${answer.content}"></div>
        <div class="d-flex justify-content-end">
            <div class="badge bg-light text-dark p-2 text-start">
                <div th:text="${#temporals.format(answer.createDate,
                'yyyy-MM-dd HH:mm')}"></div>
```

2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 적용하기

3.

```
                </div>
            </div>
        </div>
    </div>
    <!-- 답변 반복 끝  -->
    <!-- 답변 작성 -->
    <form th:action="@{|/answer/create/${question.id}|}" method="post"
class="my-3">
        <textarea name="content" id="content" rows="10" class="form-control"></
textarea>
        <input type="submit" value="답변 등록" class="btn btn-primary my-2">
    </form>
</div>
```

부트스트랩으로 화면을 구성하다 보면 가끔 HTML 코드를 이렇게 많이 작성해야 함.

질문이나 답변은 각각 하나의 덩어리이므로 부트스트랩의 card 컴포넌트를 사용함.
부트스트랩의 card 컴포넌트는 어떤 내용을 그룹화하여 보여 줄 때 사용함

2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 적용하기

3. card 컴포넌트를 비롯하여 질문 상세 템플릿에서 부트스트랩 클래스를 많이 사용함. 다음 표를 통해 살펴보자

부트스트랩 클래스	설명
card, card-body, card-text	card 컴포넌트를 적용하는 클래스들이다.
badge	badge 컴포넌트를 적용하는 클래스이다.
form-control	텍스트 창에 form 컴포넌트를 적용하는 클래스이다.
border-bottom	아래 방향 테두리 선을 만드는 클래스이다.
my-3	상하 마진값으로 3을 지정하는 클래스이다.
py-2	상하 패딩값으로 2를 지정하는 클래스이다.
p-2	상하좌우 패딩값으로 2를 지정하는 클래스이다.
d-flex justify-content-end	HTML 요소를 오른쪽으로 정렬하는 클래스이다.
bg-light	연회색으로 배경을 지정하는 클래스이다.
text-dark	글자색을 검은색으로 지정하는 클래스이다.
text-start	글자를 왼쪽으로 정렬하는 클래스이다.
btn btn-primary	버튼 컴포넌트를 적용하는 클래스이다.

2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 적용하기

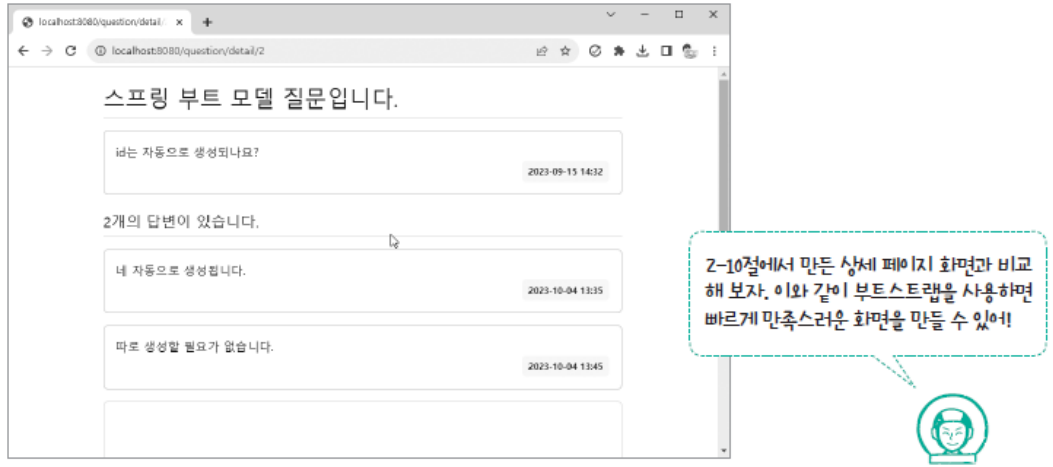
3. 질문과 답변 덩어리를 살펴보면
   style="white-space: pre-line;"과 같은 스타일을 지정해 줌.

   style="white-space: pre-line;"은 CSS 스타일 속성으로,
   사용자가 입력한대로 줄 바꿈이 적용되도록 만들어 줌

2-14 부트스트랩으로 화면 꾸미기

▪ 부트스트랩 적용하기

4. 부트스트랩을 적용한 질문 상세 화면을 완성함



2-15 표준 HTML 구조로 변경하기

- 지금까지 작성한 질문 목록(question_list.html),
  질문 상세(question_detail.html) 템플릿은 표준 HTML 구조로 작성하지 않았음

- 어떤 웹 브라우저를 사용하더라도 웹 페이지가 동일하게 보이고
  정상적으로 작동하게 하려면 반드시 웹 표준을 지키는 HTML 문서로 작성해야 함

2-15 표준 HTML 구조로 변경하기

▪ 표준 HTML 구조 살펴보기

- 다음 예로 표준 HTML 문서의 구조를 살펴보자

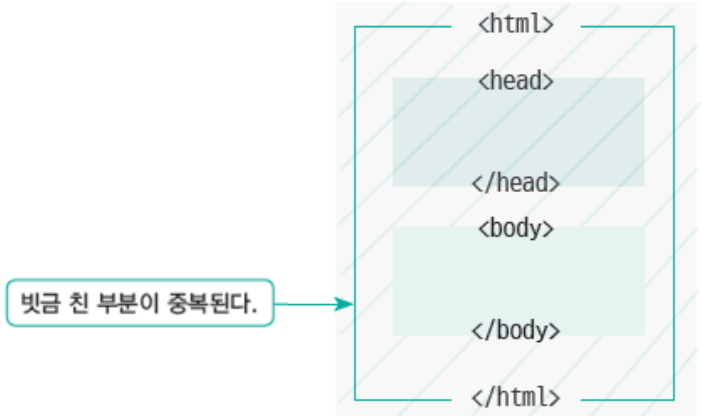
- 표준 HTML 문서의 구조는 앞의 예처럼 html, head, body 요소가 있어야 하며, CSS 파일은 <head> 태그 안에 링크되어야 함
- 또한 <head> 태그 안에는 meta, title 요소 등이 포함되어야 함

2-15 표준 HTML 구조로 변경하기

▪ 템플릿 상속하기

- 앞에서 작성한 질문 목록과 질문 상세 템플릿이
  표준 HTML 구조로 구성되도록 수정해 보자

- 그런데 이 템플릿 파일들을 모두 표준 HTML 구조로 변경하면
  body 요소를 제외한 바깥 부분은 모두 같은 내용으로 중복됨



2-15 표준 HTML 구조로 변경하기

- **템플릿 상속하기**
  - 그렇게 되면 CSS 파일 이름이 변경되거나 새로운 CSS 파일을 추가할 때마다 모든 템플릿 파일을 일일이 수정해야 함
  - 타임리프는 이런 중복의 불편함을 해소하기 위해 템플릿 상속 기능을 제공함
  - 템플릿 상속은 기본 틀이 되는 템플릿을 먼저 작성하고 다른 템플릿에서 그 템플릿을 상속해 사용하는 방법임
  - 템플릿 상속에 대해서 자세히 알아보자

2-15 표준 HTML 구조로 변경하기

▪ 템플릿 상속하기

▪ layout.html로 기본 틀 만들기

템플릿을 상속하려면 각 템플릿 파일에서 반복되는 내용을 담아
기본 틀이 되는 템플릿을 만들어야 함.
그러기 위해 templates에 layout.html 파일을 만들어 다음 내용을 작성해 보자.

• /templates/layout.html

```
<!doctype html>
<html lang="ko">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" type="text/css" th:href="@{/bootstrap.min.css}">
    <!-- sbb CSS -->
    <link rel="stylesheet" type="text/css" th:href="@{/style.css}">
    <title>Hello, sbb!</title>
</head>
<body>
<!-- 기본 템플릿  안에 삽입될 내용 Start -->
<th:block layout:fragment="content"></th:block>
<!-- 기본 템플릿 안에 삽입될 내용 End -->
</body>
</html>
```

2-15 표준 HTML 구조로 변경하기

▪ 템플릿 상속하기

▪ layout.html로 기본 틀 만들기

layout.html은 모든 템플릿이 상속해야 하는 템플릿으로,
표준 HTML 문서 구조로 정리된 기본 틀이 됨.

body 요소 안의 <th:block layout:fragment="content"></th:block>은
layout.html을 상속한 템플릿에서 개별적으로 구현해야 하는 영역이 됨.

즉, layout.html 템플릿을 상속하면 <th:block layout:fragment="content"></th:block> 영역만
수정해도 표준 HTML 문서로 작성됨

2-15 표준 HTML 구조로 변경하기

▪ 템플릿 상속하기

▪ question_list.html에 템플릿 상속하기

question_list.html 템플릿을 다음과 같이 변경하여 layout.html을 상속해 보자

• /templates/question_list.html

```
<link rel="stylesheet" type="text/css" th:href="@{/bootstrap.min.css}">
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container my-3">
    <table class="table">
        (... 생략 ...)
    </table>
</div>
</html>
```

layout.html로부터 부트스트랩 스타일시트를 상속받으므로 이 코드를 삭제한 후 아래 내용을 입력한다.

2-15 표준 HTML 구조로 변경하기

▪ 템플릿 상속하기

▪ question_list.html에 템플릿 상속하기

layout.html 템플릿을 상속하려고 <html layout:decorate="~{layout}">을 사용함.

타임리프의 layout:decorate 속성은 템플릿의
레이아웃(부모 템플릿, 여기서는 layout.html)으로 사용할 템플릿을 설정함.
속성값인 ~{layout}이 바로 layout.html 파일을 의미함.

부모 템플릿인 layout.html에는 다음과 같은 내용이 있었음

```
<!-- 기본 템플릿 안에 삽입될 내용 Start -->
<th:block layout:fragment="content"></th:block>
<!-- 기본 템플릿 안에 삽입될 내용 End -->
```

2-15 표준 HTML 구조로 변경하기

▪ 템플릿 상속하기

▪ question_list.html에 템플릿 상속하기

부모 템플릿에 작성된 이 부분을 자식 템플릿의 내용으로 적용될 수 있도록 다음과 같이 사용함.

```
<div layout:fragment="content" class="container my-3">
    (... 생략 ...)
</div>
```

이렇게 하면 부모 템플릿의 th:block 요소의 내용이
자식 템플릿의 div 요소의 내용으로 교체됨

2-15 표준 HTML 구조로 변경하기

▪ 템플릿 상속하기

▪ question_detail.html에 템플릿 상속하기

question_deatail.html도 마찬가지 방법으로 layout.html을 상속해 보자

• /templates/question_detail.html

```
<link rel="stylesheet" type="text/css" th:href="@{/bootstrap.min.css}">
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container my-3">
    <!-- 질문 -->
    <h2 class="border-bottom py-2" th:text="${question.subject}"></h2>
    (... 생략 ...)
    </form>
</div>
</html>
```

layout.html로부터 부트스트랩 스타일시트를 상속받으므로 이 코드를 삭제한 후 아래 내용을 입력한다.

2-15 표준 HTML 구조로 변경하기

▪ 템플릿 상속하기

▪ question_detail.html에 템플릿 상속하기

question_list.html 템플릿과 동일한 방법으로 layout.html 템플릿을 상속하려고
<htmllayout:decorate="~{layout}">을 사용함.

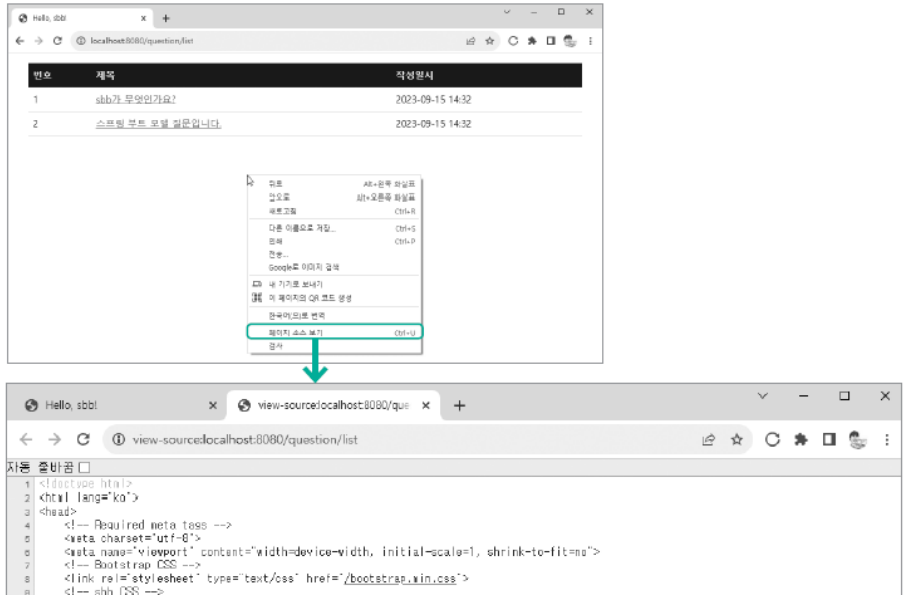
템플릿을 상속한 후, 질문 목록 또는 질문 상세 페이지를 확인해 보자.
화면에 보여 지는 것은 동일하지만 표준 HTML 구조로 변경됨.

크롬 브라우저에서 마우스 오른쪽 버튼을 클릭하고
[페이지 소스 보기]를 클릭하면 HTML 코드를 확인할 수 있음

2-15 표준 HTML 구조로 변경하기

▪ 템플릿 상속하기

▪ question_detail.html에 템플릿 상속하기



2-16 질문 등록 기능 추가하기

- 이번에는 질문을 등록하는 기능을 만들어 보자.

- 질문 목록에 질문 등록을 위한 버튼을 추가하고
  질문을 등록할 수 있는 화면을 만들어 질문 등록 기능을 완성해 보자.

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

- 질문 등록을 할 수 있도록 먼저 질문 목록 페이지에 [질문 등록하기] 버튼을 만들어야 함

- question_list.html 파일을 열고 다음과 같이 한 줄의 코드를 추가하여 질문 목록 아래에 버튼을 생성하자

• /templates/question_list.html

```
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container my-3">
    <table class="table">
        (... 생략 ...)
    </table>
    <a th:href="@{/question/create}" class="btn btn-primary">질문 등록하기</a>
</div>
</html>
```

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

- <a> ... 요소를 추가하여 부트스트랩의 btn btn-primary 클래스를 적용하면 다음과 같이 화면에 버튼 형태로 보임

- [질문 등록하기] 버튼을 한번 클릭해 보자
- /question/create URL이 호출되지만 현 상태에서는 404 오류가 발생함

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

▪ URL 매핑하기

이제 404 오류가 발생하면 무엇을 해야 하는지 잘 알 것임.
QuestionController에 /question/create에 해당하는 URL 매핑을 추가하자

• /question/QuestionController.java

```
(... 생략 ...)

public class QuestionController {

    private final QuestionService questionService;

    @GetMapping("/list") {
    (... 생략 ...)
    }

    @GetMapping(value = "/detail/{id}") {
    (... 생략 ...)
    }

    @GetMapping("/create")
    public String questionCreate() {
        return "question_form";
    }
}
```

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

▪ URL 매핑하기

[질문 등록하기] 버튼을 통한 /question/create 요청은 GET 요청에 해당하므로
@GetMapping 애너테이션을 사용함.
questionCreate 메서드는 question_form 템플릿을 출력함

▪ 템플릿 만들기

1. 질문 등록 화면을 만들기 위해 templates에
   question_form.html 파일을 생성하고 다음 내용을 작성해 보자

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

▪ 템플릿 만들기

1. • /templates/question_form.html

```
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container">
    <h5 class="my-3 border-bottom pb-2">질문 등록 </h5>
    <form th:action="@{/question/create}" method="post">
        <div class="mb-3">
            <label for="subject" class="form-label">제목</label>
            <input type="text" name="subject" id="subject" class="form-control">
        </div>
        <div class="mb-3">
            <label for="content" class="form-label">내용</label>
            <textarea name="content" id="content" class="form-control"
            rows="10"></textarea>
        </div>
        <input type="submit" value="저장하기" class="btn btn-primary my-2">
    </form>
</div>
</html>
```

이와 같이 제목과 내용을 입력하여
질문을 등록할 수 있는 템플릿을 작성함.

템플릿에는 제목과 내용을
입력할 수 있는 텍스트 창을 추가함.

제목은 일반적인 input 텍스트 창을
사용하고 내용은 글자 수에
제한이 없는 textarea 창을 사용함.

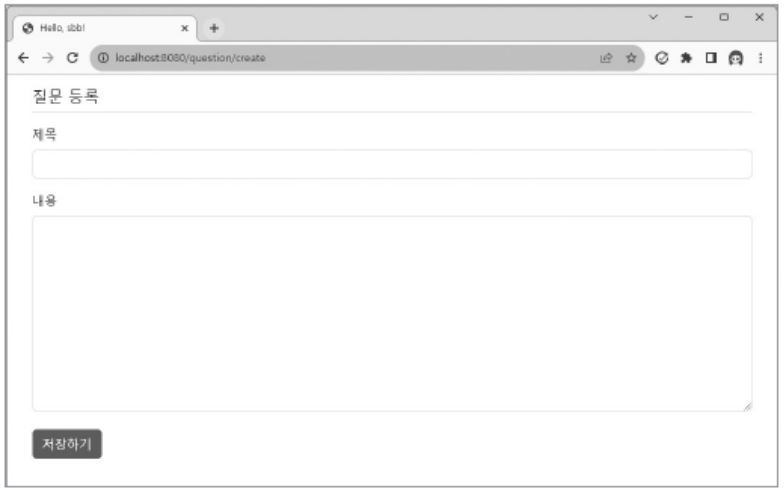
그리고 입력한 내용을 /question/create
URL로 post 방식을 이용해 전송할 수 있
도록 form과 버튼을 추가함

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

▪ 템플릿 만들기

2. 이제 질문 목록 화면에서 [질문 등록하기] 버튼을 클릭하면 다음 화면이 나타날 것임.



하지만 이 화면에서 질문과 내용을 입력하고
[저장하기] 버튼을 누르면 405 오류가 발생함.

405 오류는 'Method Not Allowed'라는 의미로,
/question/create URL을 POST 방식으로는 처리할 수 없음을 나타냄

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

▪ 템플릿 만들기

3. POST 요청을 처리할 수 있도록 다음과 같이 QuestionController를 수정해 보자

• /question/QuestionController.java

```
package com.mysite.sbb.question;

import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import lombok.RequiredArgsConstructor;
(... 생략 ...)
```

```
public class QuestionController {

    (... 생략 ...)

    @GetMapping("/create")
    public String questionCreate() {
        return "question_form";
    }

    @PostMapping("/create")
    public String questionCreate(@RequestParam(value="subject") String subject,
    @RequestParam(value="content") String content) {
        // TODO: 질문을 저장한다.
        return "redirect:/question/list"; // 질문 저장 후 질문 목록으로 이동
    }
}
```

2-16 질문 등록 기능 추가하기

- **질문 등록 버튼과 화면 만들기**
  - **템플릿 만들기**
    - POST 방식으로 요청한 /question/create URL을 처리하도록 @PostMapping 애너테이션을 지정한 questionCreate 메서드를 추가함
    - 메서드명은 @GetMapping에서 사용한 questionCreate 메서드명과 동일하게 사용할 수 있음 (단, 매개변수의 형태가 다른 경우에 가능하다.).

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

▪ 템플릿 만들기

- questionCreate 메서드는 화면에서 입력한 제목(subject)과 내용(content)을 매개변수로 받음
- 이때 질문 등록 템플릿(question_form.html)에서 입력 항목으로 사용한 subject, content의 이름과 RequestParam의 value값이 동일해야 함을 기억하자.
  그래야 입력 항목의 값을 제대로 얻을 수 있음.
- 그런데 여기서는 일단 질문 데이터를 저장하는 작업은 잠시 뒤로 미루고
  (해야 할 일을 TODO 주석으로 작성했다.)
  [저장하기] 버튼을 클릭해 질문이 저장되면
  질문 목록 페이지로 이동하는 것까지 완성해 보자

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

▪ 서비스 수정하기

1. 앞서 잠시 미룬 작업을 진행해 보자. 질문 데이터를 저장하기 위해 QuestionService.java를 다음과 같이 수정해 보자.

• /question/QuestionService.java

```
package com.mysite.sbb.question;

import java.util.List;
import java.util.Optional;
import java.time.LocalDateTime;

import com.mysite.sbb.DataNotFoundException;
import org.springframework.stereotype.Service;

import lombok.RequiredArgsConstructor;
```

```
(... 생략 ...)
public class QuestionService {
    private final QuestionRepository questionRepository;

    (... 생략 ...)

    public void create(String subject, String content) {
        Question q = new Question();
        q.setSubject(subject);
        q.setContent(content);
        q.setCreateDate(LocalDateTime.now());
        this.questionRepository.save(q);
    }
}
```

제목(subject)과 내용(content)을 입력받아 이를 질문으로 저장하는 create 메서드를 만듦

2-16 질문 등록 기능 추가하기

▪ 질문 등록 버튼과 화면 만들기

▪ 서비스 수정하기

2. 다시 QuestionController.java로 돌아가 이 서비스를 사용할 수 있도록 다음과 같이 수정해 보자

• /question/QuestionController.java

```
(... 생략 ...)
public class QuestionController {

    (... 생략 ...)

    @PostMapping("/create")
    public String questionCreate(@RequestParam(value="subject") String subject,
    @RequestParam(value="content") String content) {
        this.questionService.create(subject, content);
        return "redirect:/question/list";
    }
}
```

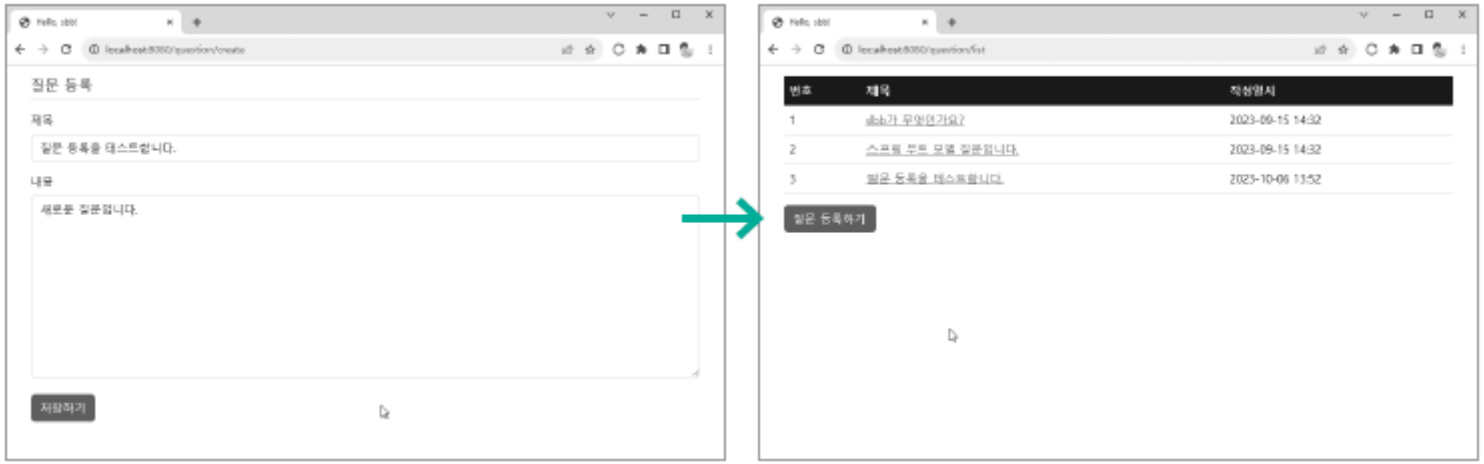
TODO 주석문을 삭제하고 그 자리에 코드를 입력

2-16 질문 등록 기능 추가하기

- **질문 등록 버튼과 화면 만들기**
  - **서비스 수정하기**
    - TODO 주석문 대신 QuestionService의 create 메서드를 호출하여 질문 데이터(subject, content)를 저장하는 코드를 작성함
    - 이렇게 수정하고 질문을 작성하고 저장하면 잘 동작하는 것을 확인할 수 있음
    - 질문 등록 화면에서 다음과 같이 질문과 내용을 입력한 후에 [저장하기] 버튼을 클릭하면 질문이 저장되고 질문 목록 화면으로 이동하는 것을 확인할 수 있음

2-16 질문 등록 기능 추가하기

- **질문 등록 버튼과 화면 만들기**
  - **서비스 수정하기**



2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

- 질문을 등록하는 기능을 구현했지만
  질문을 등록할 때 비어 있는 값으로도 등록할 수 있다는 점을 간과함

- 아무것도 입력하지 않은 상태에서 질문이 등록될 수 없게 하기 위해
  폼 클래스를 사용하여 입력값을 체크하는 방법을 사용해보자

- 폼(form) 클래스 또한 컨트롤러, 서비스와 같이
  웹 프로그램을 개발하는 주요 구성 요소 중 하나로,
  웹 프로그램에서 사용자가 입력한 데이터를 검증하는 데 사용함

2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ Spring Boot Validation 라이브러리 설치하기

- 폼 클래스를 사용해 사용자로부터 입력받은 값을 검증하려면
  먼저 Spring Boot Validation 라이브러리가 필요함
- 이 라이브러리를 설치하기 위해서 다음과 같이 build.gradle 파일을 수정해 보자.

• build.gradle

```
(... 생략 ...)

dependencies {
    (... 생략 ...)
    implementation 'nz.net.ultraq.thymeleaf:thymeleaf-layout-dialect'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
}

(... 생략 ...)
```

2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ Spring Boot Validation 라이브러리 설치하기

- build.gradle 파일을 선택한 후 마우스 오른쪽 버튼을 눌러 [Gradle → Refresh Gradle Project]를
  클릭하여 변경 사항을 적용하면 Spring Boot Valiation 라이브러리가 설치됨

- Spring Boot Validation 라이브러리를 설치하면 다음과 같은 애너테이션을 사용하여 사용

항목	설명
@Size	문자 길이를 제한한다.
@NotNull	Null을 허용하지 않는다.
@NotEmpty	Null 또는 빈 문자열("")을 허용하지 않는다.
@Past	과거 날짜만 입력할 수 있다.
@Future	미래 날짜만 입력할 수 있다.
@FutureOrPresent	미래 또는 오늘 날짜만 입력할 수 있다

@Max	최댓값 이하의 값만 입력할 수 있도록 제한한다.
@Min	최솟값 이상의 값만 입력할 수 있도록 제한한다.
@Pattern	입력값을 정규식 패턴으로 검증한다.

2-16 질문 등록 기능 추가하기

- **폼 활용하기**
  - **폼 클래스 만들기**
    - 질문 등록 페이지에서 사용자로부터 입력받은 값을 검증하는데
      필요한 폼 클래스를 만들어 보자
    - 먼저, com.mysite.sbb.question 패키지에 QuestionForm.java 파일을 만들어
      입력 항목인 subject, content에 대응하는 QuestionForm 클래스를 다음과 같이 작성해 보자

2-16 질문 등록 기능 추가하기

- **폼 활용하기**
  - **폼 클래스 만들기**

• /question/QuestionForm.java

```
package com.mysite.sbb.question;

import jakarta.validation.constraints.NotEmpty;
import jakarta.validation.constraints.Size;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class QuestionForm {
    @NotEmpty(message="제목은 필수 항목입니다.")
    @Size(max=200)
    private String subject;

    @NotEmpty(message="내용은 필수 항목입니다.")
    private String content;
}
```

- subject 속성에는 @NotEmpty와 @Size 애너테이션이 적용됨
- @NotEmpty는 해당 값이 Null 또는 빈 문자열("")을 허용하지 않음을 의미함
- 여기에 사용한 message는 검증이 실패할 경우 화면에 표시할 오류 메시지임
- @Size(max=200)은 최대 길이가 200 바이트byte를 넘으면 안 된다는 의미로, 이와 같이 설정하면 길이가 200 바이트보다 큰 제목이 입력되면 오류가 발생함
- content 속성 역시 @NotEmpty 애너테이션을 적용하여 빈 값을 허용하지 않도록 함

2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ 컨트롤러에 전송하기

- QuestionForm을 컨트롤러에서 사용할 수 있도록
  다음과 같이 컨트롤러(Question Controller)를 수정해 보자

• /question/QuestionController.java

```
package com.mysite.sbb.question;

import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping
```

```
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;

@RequestMapping("/question")
@RequiredArgsConstructor
@Controller
public class QuestionController {

    (... 생략 ...)
```

2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ 컨트롤러에 전송하기

```
    @GetMapping("/create")
    public String questionCreate() {
        return "question_form";
    }

    @PostMapping("/create")
    public String questionCreate(@Valid QuestionForm questionForm, BindingResult
bindingResult) {
        if (bindingResult.hasErrors()) {
            return "question_form";
        }
        this.questionService.create(questionForm.getSubject(), questionForm.get
Content());
        return "redirect:/question/list";
    }
}
```

- questionCreate 메서드의 매개변수를 subject, content 대신 QuestionForm 객체로 변경함
- subject, content 항목을 지닌 폼이 전송되면 QuestionForm의 subject, content 속성이 자동으로 바인딩됨
- 이렇게 이름이 동일하면 함께 연결되어 묶이는 것이 바로 폼의 바인딩 기능임

2-16 질문 등록 기능 추가하기

- **폼 활용하기**
  - **컨트롤러에 전송하기**
    - 여기서 QuestionForm 매개변수 앞에 @Valid 애너테이션을 적용함
    - @Valid 애너테이션을 적용하면 QuestionForm의 @NotEmpty, @Size 등으로 설정한 검증 기능이 동작함
    - 그리고 이어지는 BindingResult 매개변수는 @Valid 애너테이션으로 검증이 수행된 결과를 의미하는 객체임

2-16 질문 등록 기능 추가하기

- **폼 활용하기**
  - **컨트롤러에 전송하기**
    - 따라서 questionCreate 메서드는 bindResult.hasErrors( )를 호출하여 오류가 있는 경우에는 다시 제목과 내용을 작성하는 화면으로 돌아가도록 했고, 오류가 없을 경우에만 질문이 등록되도록 만듦
    - 여기까지 수정했다면 질문 등록 화면에서 아무런 값도 입력하지 말고 [저장하기] 버튼을 클릭해 보자

2-16 질문 등록 기능 추가하기

- **폼 활용하기**
  - **컨트롤러에 전송하기**
    - 아무런 입력값도 입력하지 않았으므로
      QuestionForm의 @NotEmpty에 의해 Validation이 실패하여
      다시 질문 등록 화면에 머물러 있을 것임
    - 하지만 QuestionForm에 설정한 '제목은 필수 항목입니다.'와 같은
      오류 메시지는 보이지 않음
    - 오류 메시지가 보이지 않는다면 어떤 항목에서 검증이 실패했는지 알 수가 없음.
      어떻게 해야 할까?

2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ 템플릿 수정하기

1. 검증에 실패했다는 오류 메시지를 보여 주기 위해 question_form 템플릿을 다음과 같이 수정해 보자

• /templates/question_form.html

```
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container">
    <h5 class="my-3 border-bottom pb-2">질문 등록</h5>
    <form th:action="@{/question/create}" th:object="${questionForm}" method="post">
        <div class="alert alert-danger" role="alert" th:if="${#fields.hasAnyErrors()}">
            <div th:each="err : ${#fields.allErrors()}" th:text="${err}" />
        </div>
        <div class="mb-3">
            <label for="subject" class="form-label">제목</label>
            <input type="text" name="subject" id="subject" class="form-control">
        </div>
        <div class="mb-3">
            <label for="content" class="form-label">내용</label>
            <textarea name="content" id="content" class="form-control"
            rows="10"></textarea>
        </div>
        <input type="submit" value="저장하기" class="btn btn-primary my-2">
    </form>
</div>
</html>
```

2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ 템플릿 수정하기

1. 검증에 실패할 경우 오류 메시지를 출력할 수 있도록 수정함.
   #fields.hasAnyErrors가 true라면 QuestionForm 검증이 실패한 것임.

   QuestionForm 검증이 실패한 이유는 #fields.allErrors( )로 확인할 수 있음.
   #fields.allErrors( )에는 오류의 내용이 담겨 있음.

   그리고 부트스트랩의 alert alert-danger 클래스를 사용하여
   오류 메시지가 붉은 색으로 표시되도록 함.

   이렇게 오류를 표시하려면 타임리프의 th:object 속성이 반드시 필요한데,
   th:object는 <form>의 입력 항목들이 QuestionForm과 연결된다는 점을
   타임리프에 알려주는 역할을 함

2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ 템플릿 수정하기

2. 그런데 여기까지 수정하고 테스트하기 위해
[질문 등록하기] 버튼을 클릭하면 오류가 발생할 것임.

   템플릿의 form 태그에 th:object 속성을 추가했으므로
QuestionController의 GetMapping으로 매핑한 메서드도
다음과 같이 변경해야 오류가 발생하지 않음.

   왜냐하면 question_form.html은 [질문 등록하기] 버튼을 통해
GET 방식으로 URL이 요청되더라도 th:object에 의해 QuestionForm 객체가 필요하기 때문

2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ 템플릿 수정하기

2.

• /question/QuestionController.java

```
(... 생략 ...)

public class QuestionController {

    (... 생략 ...)

    @GetMapping("/create")
    public String questionCreate(QuestionForm questionForm) {
        return "question_form";
    }
    @PostMapping("/create")
    public String questionCreate(@Valid QuestionForm questionForm, BindingResult
bindingResult) {
        if (bindingResult.hasErrors()) {
            return "question_form";
        }
        this.questionService.create(questionForm.getSubject(), questionForm.get
Content());
          return "redirect:/question/list";
      }
}
```

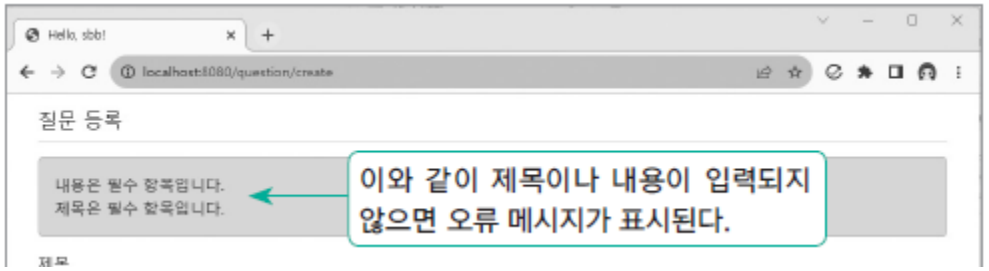
2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ 템플릿 수정하기

2. @GetMapping으로 매핑한 questionCreate 메서드에 매개변수로 QuestionForm 객체를 추가함.
   이렇게 하면 이제 GET 방식에서도
   question_form 템플릿에 QuestionForm 객체가 전달됨

2. 이렇게 수정하고 제목 또는 내용에 값을 채우지 않은 상태로 질문 등록을 진행하면([저장하기]
   버튼을 클릭하면



2-16 질문 등록 기능 추가하기

- **폼 활용하기**
  - **오류 처리하기**
    - 테스트를 진행하다 보니 또 다른 문제를 발견함
    - 예를 들어 제목을 입력하고 내용을 비워 둔 채로 [저장하기] 버튼을 누르면 오류 메시지가 나타남과 동시에 이미 입력한 제목도 사라진다는 점임
    - 입력한 제목은 남아 있어야 하지 않겠는가?

2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ 오류 처리하기

1. 이러한 문제를 해결하기 위해 이미 입력한 값이 유지되도록 다음과 같이 템플릿을 수정해 보자

• /templates/question_form.html

```
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container">
    <h5 class="my-3 border-bottom pb-2">질문 등록</h5>
    <form th:action="@{/question/create}" th:object="${questionForm}"
method="post">
        <div class="alert alert-danger" role="alert" th:if="${#fields.hasAnyEr
rors()}">
            <div th:each="err : ${#fields.allErrors()}" th:text="${err}" />
        </div>
        <div class="mb-3">
            <label for="subject" class="form-label">제목</label>
            <input type="text" th:field="*{subject}" class="form-control">
        </div>
        <div class="mb-3">
            <label for="content" class="form-label">내용</label>
            <textarea th:field="*{content}" class="form-control" rows="10"></textarea>
        </div>
        <input type="submit" value="저장하기" class="btn btn-primary my-2">
    </form>
</div>
</html>
```

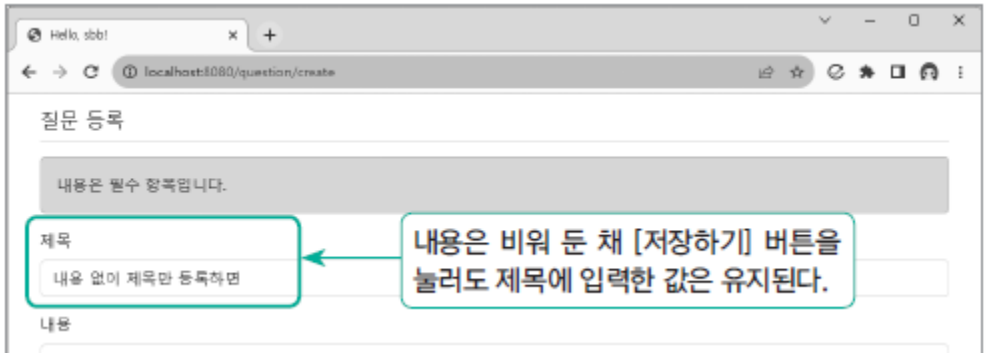
2-16 질문 등록 기능 추가하기

▪ 폼 활용하기

▪ 오류 처리하기

1. name="subject", name="content" 대신 th:field 속성을 사용하도록 변경함.
   이렇게 하면 해당 태그의 id, name, value 속성이 모두 자동으로 생성되고
   타임리프가 value 속성에 기존에 입력된 값을 채워 넣어
   오류가 발생하더라도 기존에 입력한 값이 유지됨

1. 이제 코드를 수정하고 제목만 입력한 후
   [저장하기] 버튼을 클릭해 질문 등록을
   진행해 보자. 이전에 입력했던 값(제목)(
   유지되는 것을 확인할 수 있을 것임



2-16 질문 등록 기능 추가하기

▪ 답변 등록 기능에 폼 적용하기

- 질문 등록 기능에 폼을 적용한 것처럼 답변 등록 기능에도 폼을 적용해 보자.
  질문 등록 기능을 만들 때와 동일한 방법이므로 조금 빠르게 만들어 보자

1. 먼저 답변을 등록하기 위해 필요한 폼인 AnswerForm을 다음과 같이 작성해 보자

• /answer/AnswerForm.java

```
package com.mysite.sbb.answer;

import jakarta.validation.constraints.NotEmpty;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class AnswerForm {
    @NotEmpty(message = "내용은 필수 항목입니다.")
    private String content;
}
```

2-16 질문 등록 기능 추가하기

▪ 답변 등록 기능에 폼 적용하기

2. 이어서 AnswerController를 다음과 같이 수정하자

• /answer/AnswerController.java

```
package com.mysite.sbb.answer;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import com.mysite.sbb.question.Question;
import com.mysite.sbb.question.QuestionService;

import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;

(... 생략 ...)
public class AnswerController {

    (... 생략 ...)

    @PostMapping("/create/{id}")
    public String createAnswer(Model model, @PathVariable("id") Integer id, @Valid
AnswerForm answerForm, BindingResult bindingResult) {
        Question question = this.questionService.getQuestion(id);
        if (bindingResult.hasErrors()) {
            model.addAttribute("question", question);
            return "question_detail";
        }
        this.answerService.create(question, answerForm.getContent());
        return String.format("redirect:/question/detail/%s", id);
    }
}
```

2-16 질문 등록 기능 추가하기

▪ 답변 등록 기능에 폼 적용하기

2. AnswerForm을 사용하도록 AnswerController를 변경함.

   QuestionForm을 사용했던 방법과 마찬가지로
   @Valid와 BindingResult를 사용하여 검증을 진행함.

   검증에 실패할 경우에는 다시 답변을 등록할 수 있는
   question_detail 템플릿을 출력하게 함.

   이때 question_detail 템플릿은 Question 객체가 필요하므로
   model 객체에 Question 객체를 저장한 후에 question_detail 템플릿을 출력해야 함

2-16 질문 등록 기능 추가하기

▪ 답변 등록 기능에 폼 적용하기

3. 템플릿 question_detail.html을 다음과 같이 수정하자

• /templates/question_detail.html

```
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container my-3">
    <!--질문-->
    (... 생략 ...)
    <!-- 답변 작성 -->
    <form th:action="@{/answer/create/${question.id}}" th:object="${answerForm}"
    method="post" class="my-3">
        <div class="alert alert-danger" role="alert" th:if="${#fields.hasAnyEr
rors()}">
            <div th:each="err : ${#fields.allErrors()}" th:text="${err}" />
        </div>

        <textarea th:field="*{content}" rows="10" class="form-control"></textarea>
        <input type="submit" value="답변 등록" class="btn btn-primary my-2">
    </form>
</div>
</html>
```

답변 등록 form의 입력 항목과 AnswerForm을 타임리프에 연결하기 위해 th:object 속성을 추가함.

검증이 실패할 경우 #fields.hasAnyErrors( )와 #fields.allErrors( )를 사용하여 오류 메시지를 표시하도록 함.

그리고 답변 등록 기능의 content 항목도 th:field 속성을 사용하도록 변경함

2-16 질문 등록 기능 추가하기

▪ 답변 등록 기능에 폼 적용하기

4. AnswerForm을 사용하기 위해 question_detail 템플릿을 수정하였으므로 Question Controller의 detail 메서드도 다음과 같이 수정해야 함

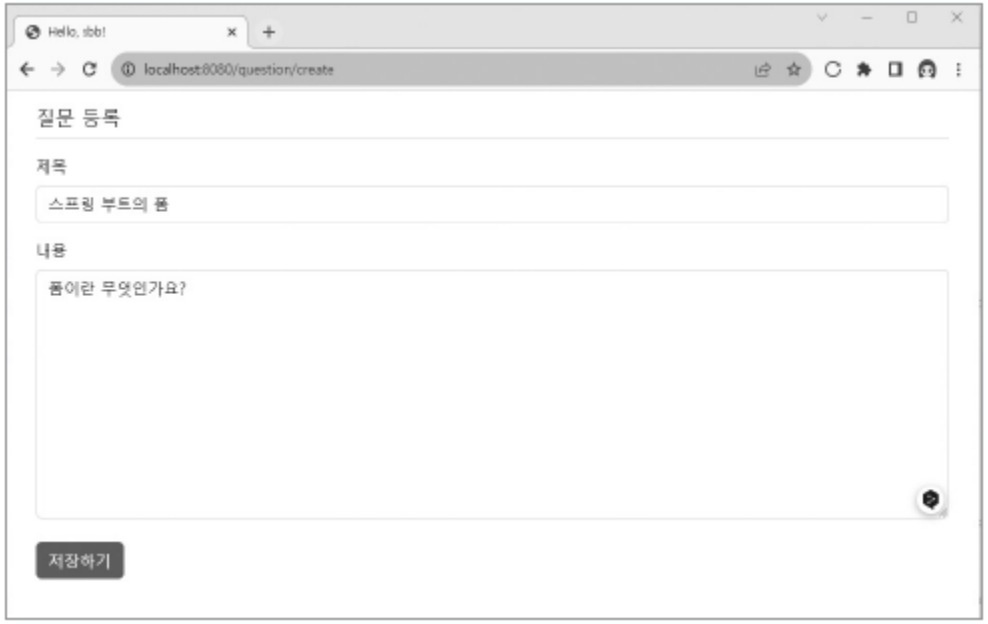
• /question/QuestionController.java

```
package com.mysite.sbb.question;

import java.util.List;
import jakarta.validation.Valid;

import com.mysite.sbb.answer.AnswerForm;

(... 생략 ...)

public class QuestionController {

    (... 생략 ...)

    @GetMapping(value = "/detail/{id}")
    public String detail(Model model, @PathVariable("id") Integer id, AnswerForm
answerForm) {
        (... 생략 ...)
    }
    @GetMapping("/create")
    public String questionCreate(QuestionForm questionForm) {
        return "question_form";
    }
   (... 생략 ...)
}
```

2-16 질문 등록 기능 추가하기

▪ 답변 등록 기능에 폼 적용하기

5. 수정을 완료한 후, 답변 등록 기능이 제대로 동작하는지 확인해 보자.
   먼저, 질문 등록 페이지에서 제목과 내용을 입력하고 저장함

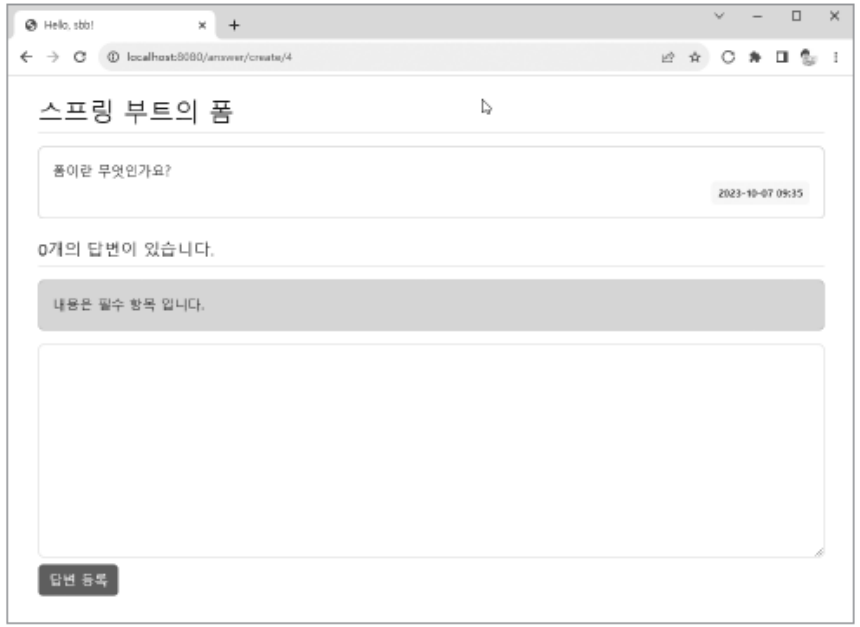


2-16 질문 등록 기능 추가하기

▪ 답변 등록 기능에 폼 적용하기

6. 질문 목록 페이지에서 제목을 클릭하면
오른쪽과 같이 질문 제목과 내용이 등장하고,
그 아래에 답변을 입력할 수 있는 공간이 있음.

   만약 내용 없이 답변을 등록하려고 시도하면
검증 오류가 발생할 것임



2-16 질문 등록 기능 추가하기

▪ 공통 템플릿 만들기

- 오류 메시지를 출력하는 HTML 코드는
  질문 등록과 답변 등록 페이지에서 모두 반복해서 사용함
- 이렇게 반복적으로 사용하는 코드를 공통 템플릿으로 만들어 사용해 보자
- 앞서 우리는 질문 등록과 답변 등록 기능을 만들 때
  입력값이 없어 오류가 발생하면 다음과 같이 오류를 표시하도록 코드를 작성함

```
<div class="alert alert-danger" role="alert" th:if="${#fields.hasAnyErrors()}">
    <div th:each="err : ${#fields.allErrors()}" th:text="${err}" />
</div>
```

2-16 질문 등록 기능 추가하기

- **공통 템플릿 만들기**
  - 앞으로 추가로 만들 템플릿에도 이와 같이 오류를 표시하는 부분이 필요할 것임
  - 이렇게 반복해서 사용하는 문장은 공통 템플릿으로 만들고
    필요한 부분에 삽입하여 쓸 수 있다면 편리하지 않을까?
  - 오류 메시지를 출력하는 부분을 공통 템플릿으로 만들어
    필요한 곳에 삽입할 수 있도록 해보자

2-16 질문 등록 기능 추가하기

▪ 공통 템플릿 만들기

▪ 오류 메시지 템플릿 만들기

오류 메시지를 표시하는 공통 템플릿을 form_errors란 이름으로 다음과 같이 작성하자.

• /templates/form_errors.html

```
<div th:fragment="formErrorsFragment" class="alert alert-danger"
    role="alert" th:if="${#fields.hasAnyErrors()}">
    <div th:each="err : ${#fields.allErrors()}" th:text="${err}" />
</div>
```

출력할 오류 메시지 부분에 th:fragment="formErrorsFragment" 속성을 추가함.
th:fragment ="formErrorsFragment"는 다른 템플릿에서 이 div 태그의 영역을
사용할 수 있도록 이름을 설정한 것임

2-16 질문 등록 기능 추가하기

▪ 공통 템플릿 만들기

▪ 기존 템플릿에 적용하기

1. 이제 위에서 작성한 오류 메시지 관련 내용이 담긴 공통 템플릿을 사용해 보자.
   먼저 질문 등록 기능을 위한 question_form.html 파일에 적용해 보자

• /templates/question_form.html

```
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container">
    <h5 class="my-3 border-bottom pb-2">질문 등록</h5>
    <form th:action="@{/question/create}" th:object="${questionForm}"
method="post">
        <div th:replace="~{form_errors :: formErrorsFragment}"></div>
```

기존에 3줄로 구성되어 있던 코드가 한 줄로 줄었다!

2-16 질문 등록 기능 추가하기

▪ 공통 템플릿 만들기

▪ 기존 템플릿에 적용하기

1.

```
        <div class="mb-3">
            <label for="subject" class="form-label">제목</label>
            <input type="text" th:field="*{subject}" class="form-control">
        </div>
        <div class="mb-3">
            <label for="content" class="form-label">내용</label>
            <textarea th:field="*{content}" class="form-control" rows="10"></textarea>
        </div>
        <input type="submit" value="저장하기" class="btn btn-primary my-2">
    </form>
</div>
</html>
```

2-16 질문 등록 기능 추가하기

▪ 공통 템플릿 만들기

▪ 기존 템플릿에 적용하기

1. th:replace 속성을 사용하면 템플릿 내에 공통 템플릿을 삽입할 수 있음.

   는
   th:replace 속성에 의해 div 요소의 내용을 form_errors 템플릿으로 대체하라는 의미임.

   여기서 formErrorsFragment는 앞서 form_errors 템플릿에서 작성한 내용 일부를 가리키는 것임

2-16 질문 등록 기능 추가하기

▪ 공통 템플릿 만들기

▪ 기존 템플릿에 적용하기

2. 답변을 등록하는 question_detail.html 파일도 다음과 같이 수정하자

• /templates/question_detail.html

```
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container my-3">
    (... 생략 ...)
    <!-- 답변 작성 -->
    <form th:action="@{|/answer/create/${question.id}|}" th:object="${answerForm}"
method="post" class="my-3">
        <div th:replace="~{form_errors :: formErrorsFragment}"></div>
        <textarea th:field="*{content}" rows="10" class="form-control"></textar
ea>
        <input type="submit" value="답변 등록" class="btn btn-primary my-2">
    </form>
</div>
</html>
```

여기도 기존에 3줄로 구성되어 있던 코드가 한 줄로 줄었다!

2-16 질문 등록 기능 추가하기

- **공통 템플릿 만들기**
  - **기존 템플릿에 적용하기**

3. 이렇게 변경을 완료한 후,
   질문 등록과 답변 등록 기능이 제대로 동작하는지 확인해 보자.

   이전과 동일하게 동작하는 것을 확인할 수 있음

되 / 새 / 김 / 문 / 제

P.172~173

Q1. H2 데이터베이스 설정하기

Q2. URL 매핑하기