2024-2 오픈소스 개발 프로젝트01

SW중심대학사업단 강재구

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

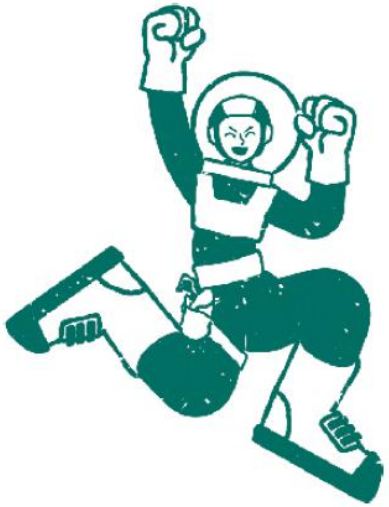
03

충북대학교
SW중심대학사업단

02. 스프링 부트의 기본 기능 익히기 (2)

02

스프링 부트의 기본 기능 익히기



2-06 도메인별로 분류하기

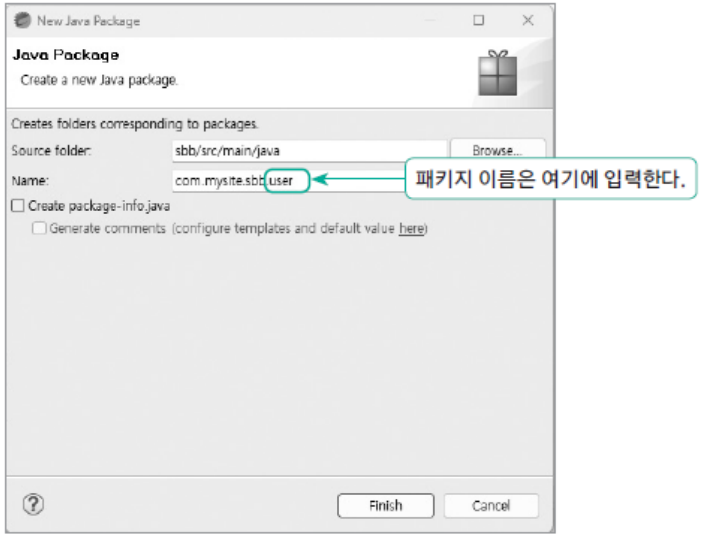
- 지금까지 만든 자바 파일은 패키지별로 정리되어 있지 않아 어수선함
- 패키지를 활용하면 자바 파일을 원하는 대로 분류할 수 있음
- 지금까지 작성한 파일은 com.mysite.sbb 패키지 안에 모두 있음
- 이렇게 하나의 패키지 안에 모든 자바 파일을 넣고 관리하는 것은 바람직하지 않음

- 그러므로 SBB의 도메인별로 패키지를 나누어 자바 파일을 관리해야 함
- 다음 표와 같이 SBB 프로젝트의 도메인별로 패키지를 생성하고 각 패키지에 맞도록 해당 파일을 이동하자

도메인 이름	패키지 이름	설명
question	com.mysite.sbb.question	게시판의 질문과 관련된 자바 파일 모음
answer	com.mysite.sbb.answer	게시판의 답변과 관련된 자바 파일 모음
user	com.mysite.sbb.user	사용자와 관련된 자바 파일 모음

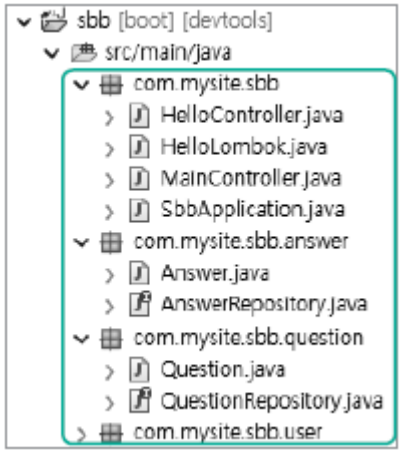
2-06 도메인별로 분류하기

1. 패키지를 생성하기 위해 com.mysite.sbb 패키지를 선택한 후 마우스 오른쪽 버튼을 눌러 [New → Package]를 클릭함. 그다음, New Java Package 창에서 com.mysite.sbb 다음에 패키지 이름을 입력하여 패키지를 만들어 보자



2-06 도메인별로 분류하기

2. 패키지를 생성했다면 각 패키지로 파일을 이동해 보자.



com.mysite.sbb.question 패키지에 Question.java,
Question Repository.java 파일을 이동시킴.
이어서 com.mysite.sbb.answer 패키지를 생성하고
Answer.java, AnswerRepository. java 파일을 이동시킴.

이때 Answer.java에서 Question 클래스를 import하는
위치가 변경되므로 import com.mysite.sbb.question.Question; 문장을 추가해야 함.
Ctrl + Shift + 0 키를 누르면 필요한 import 문을 쉽게 추가할 수 있음

2-06 도메인별로 분류하기

2. 그리고 나머지 파일들은 특정 도메인에 속하지 않았으므로
com.mysite.sbb 패키지에 그대로 놔둠.

이와 같이 자바 파일을 도메인에 따라 패키지로 나누어 관리하면
비슷한 기능이나 관련된 개념을 함께 묶어 코드들을 구조화하여 정리하게 되므로
코드를 쓰거나 읽을 때 혹은 유지 보수를 할 때 편리함

2-07 질문 목록 만들기

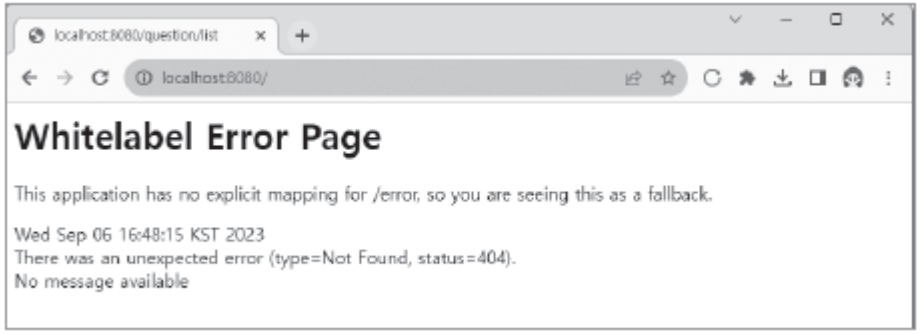
- 이번에는 SBB의 핵심 기능인 질문 목록이 담긴 페이지를 만들려고 함

- 구상하고 있는 질문 목록은 다음 주소에 접속할 때 등장해야 함

```
http://localhost:8080/question/list
```

- 로컬 서버를 실행한 뒤, 웹 브라우저에서 해당 주소를 입력하여 접속해 보자

- 지금은 404 오류 페이지가 나타남



2-07 질문 목록 만들기

▪ 질문 목록 URL 매핑하기

1. 앞서 등장한 404 오류를 해결하려면
   /question/list URL을 매핑하기 위한 컨트롤러가 필요함.

   먼저, QuestionController.java 파일을 생성해 다음과 같이 작성해 보자.
   이때 아까 만든 question 패키지 안에 QuestionController.java 파일을 생성하자

• /question/QuestionController.java

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

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class QuestionController {
    @GetMapping("/question/list")
    @ResponseBody
    public String list() {
        return "question list";
    }
}
```

2-07 질문 목록 만들기

▪ 질문 목록 URL 매핑하기

2. 이렇게 입력하고 로컬 서버를 실행한 뒤,
   다시 http://localhost:8080/question/list에 접속해 보자.
   그럼 화면에 'question list'라는 문자열이 출력됨

이와 같이 문자열이 화면에 출력되었다면
일단 질문 목록을 노출할 페이지의 /question/list URL에 매핑을 성공한 것임

2-07 질문 목록 만들기

▪ 템플릿 설정하기

- 앞서 문자열 'question list'를 직접 자바 코드로 작성하여 브라우저에 리턴함
- 하지만 보통 브라우저에 응답하는 문자열은
  이 예처럼 자바 코드에서 직접 만들지 않음.
  일반적으로 많이 사용하는 방식은 템플릿 방식임
- 템플릿(template)은 자바 코드를 삽입할 수 있는 HTML 형식의 파일을 말함
  이러한 템플릿을 사용하기 위해 스프링 부트에서는 템플릿 엔진을 지원함
- 템플릿 엔진에는 Thymeleaf, Mustache, Groovy, Freemarker, Velocity 등이 있는데,
  여기서는 스프링 진영에서 추천하는 타임리프Thymleaf 템플릿 엔진을 사용함

2-07 질문 목록 만들기

▪ 템플릿 설정하기

- 타임리프를 사용하려면 먼저 설치가 필요함.
  다음과 같이 build.gradle 파일을 수정하여 타임리프를 설치해 보자

• build.gradle

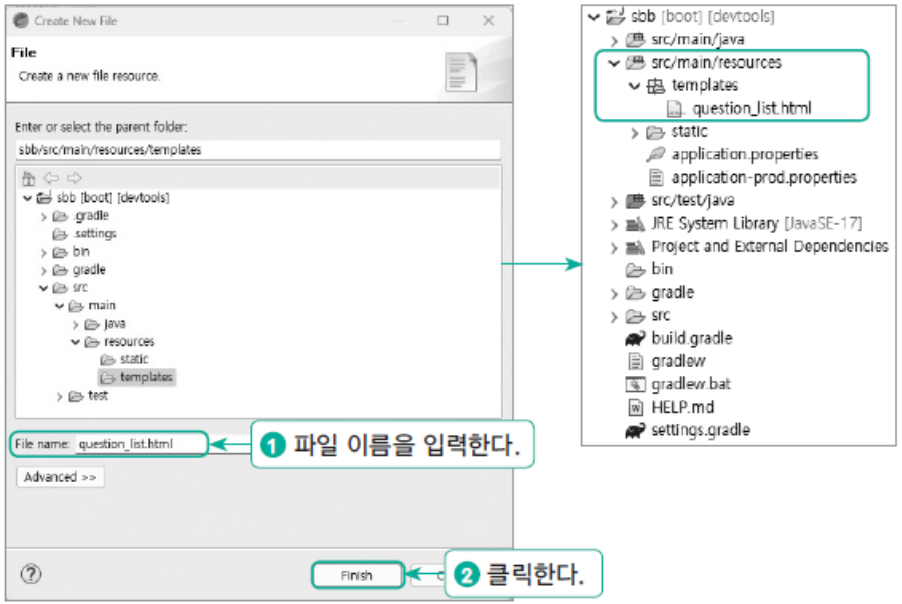
```
(... 생략 ...)

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

(... 생략 ...)
```

- 이전과 마찬가지로 build.gradle 파일을 선택한 후 마우스 오른쪽 버튼을 눌러 [Gradle → Refresh Gradle Project]를 클릭하여 변경 사항을 적용하면 타임리프가 설치됨

2-07 질문 목록 만들기

▪ 템플릿 사용하기

1. src/main/resources 디렉터리에서 templates를 선택한 후,
   마우스 오른쪽 버튼을 누르고 [New → File]을 클릭함.
   파일 이름으로 question_list.html를 입력하여 템플릿을 생성함



2-07 질문 목록 만들기

▪ 템플릿 사용하기

2. question_list.html 파일의 내용은 다음과 같이 작성해 보자

• /templates/question_list.html

```
<h2>Hello Template</h2>
```

2-07 질문 목록 만들기

▪ 템플릿 사용하기

3. 다시 QuestionController.java 파일로 돌아가 다음과 같이 수정해 보자

• /question/questionController.java

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

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;   ← 기존 코드를 삭제한다.

@Controller
public class QuestionController {

    @GetMapping("/question/list")
    @ResponseBody   ← 기존 코드를 삭제한다.
    public String list() {
        return "question_list";   ← 문자열 'question list'가 아니라 파일명을 의미하는 'question_list'임에 주의하자.
    }
}
```

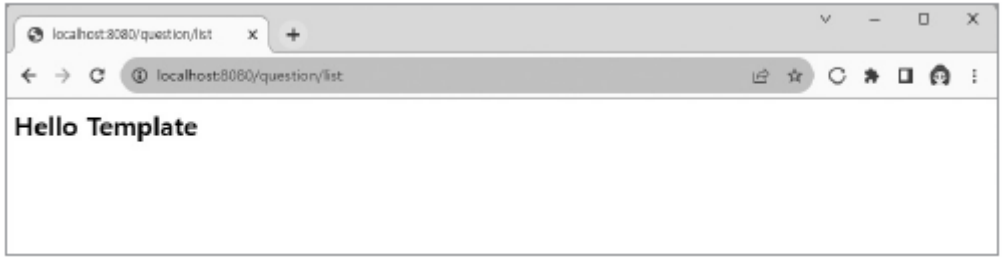
이제 템플릿을 사용하기 때문에 기존에 사용하던
@ResponseBody 애너테이션은 필요 없으므로 삭제함.
그리고 list 메서드에서 question_list.html 템플릿 파일 이름인 'question_list'를 리턴함

2-07 질문 목록 만들기

▪ 템플릿 사용하기

4. 그리고 다시 로컬 서버를 실행한 뒤,
   http://localhost:8080/question/list에 접속해 보자.

   question_list.html 파일에 작성한 <h2>Hello Template</h2> 내용이
   브라우저에 출력되는 것을 확인할 수 있음



2-07 질문 목록 만들기

▪ 데이터를 템플릿에 전달하기

- 앞선 실습을 통해 템플릿에 저장된 내용을 화면에 전달하는 것은 성공함.
  이제 질문 목록이 담긴 데이터를 조회하여 이를 템플릿을 통해 화면에 전달해 보려고 함

- 질문 목록과 관련된 데이터를 조회하려면 QuestionRepository를 사용해야 함
- QuestionRepository로 조회한 질문 목록 데이터는
  Model 클래스를 사용하여 템플릿에 전달할 수 있음

- 코드를 작성하면서 더 자세히 알아보자.
  먼저, 다음과 같이 QuestionController.java를 수정해 보자

2-07 질문 목록 만들기

▪ 데이터를 템플릿에 전달하기

• /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 lombok.RequiredArgsConstructor;

@RequiredArgsConstructor
@Controller
public class QuestionController {

    private final QuestionRepository questionRepository;

    @GetMapping("/question/list")
    public String list(Model model) {
        List<Question> questionList = this.questionRepository.findAll();
        model.addAttribute("questionList", questionList);
        return "question_list";
    }
}
```

매개변수로 Model을 지정하면 객체가 자동으로 생성된다.

2-07 질문 목록 만들기

- **데이터를 템플릿에 전달하기**
  - @RequiredArgsConstructor 애너테이션의 생성자 방식으로 questionRepository 객체를 주입함
  - @RequiredArgsConstructor는 롬복이 제공하는 애너테이션으로, final이 붙은 속성을 포함하는 생성자를 자동으로 만들어 주는 역할을 함
  - 따라서 스프링 부트가 내부적으로 QuestionController를 생성할 때 롬복으로 만들어진 생성자에 의해 questionRepository 객체가 자동으로 주입됨

2-07 질문 목록 만들기

▪ 데이터를 템플릿에 전달하기

- 그리고 QuestionRepository의 findAll 메서드를 사용하여
  질문 목록 데이터인 question List를 생성하고
  Model 객체에 'questionList'라는 이름으로 저장함

- 여기서 Model 객체는 자바 클래스와 템플릿 간의 연결 고리 역할을 함

- Model 객체에 값을 담아 두면 템플릿에서 그 값을 사용할 수 있음.
  Model 객체는 따로 생성할 필요 없이 컨트롤러의 메서드에
  매개변수로 지정하기만 하면 스프링 부트가 자동으로 Model 객체를 생성함

2-07 질문 목록 만들기

▪ 데이터를 화면에 출력하기

- DB로부터 데이터를 조회하여 Model 객체에 저장하였고,
  이제 Model 객체를 통해 전달받은 데이터들을 템플릿에서 활용할 준비를 마침.
  질문 목록 데이터를 화면에 노출해 보자.

1. 이전에 입력한 내용을 지우고
   다음과 같이 question_list.html 템플릿을
   수정해 보자

• /templates/question_list.html

```
<table>
    <thead>
        <tr>
            <th>제목</th>
            <th>작성일시</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="question : ${questionList}">
            <td th:text="${question.subject}"></td>
            <td th:text="${question.createDate}"></td>
        </tr>
    </tbody>
</table>
```

2-07 질문 목록 만들기

▪ 데이터를 화면에 출력하기

1. 질문 목록을 HTML의 테이블 구조로 표시함.
   여기서 눈여겨볼 코드는 th:each="question: ${questionList}"임.

   여기서 th:는 타임리프에서 사용하는 속성임을 나타내는데,
   바로 이 부분이 자바 코드와 연결됨.

   question_list.html 파일에 사용한 타임리프 속성들을 잠시 살펴보자.

```
<tr th:each="question : ${questionList}">
```

2-07 질문 목록 만들기

▪ 데이터를 화면에 출력하기

1. QuestionController의 list 메서드에서 조회한 질문 목록 데이터를 'questionList'라는 이름으로 Model 객체에 저장함.

   타임리프는 Model 객체에 저장한 questionList를 ${questionList}로 읽을 수 있음.

   위의 코드는 questionList에 저장된 데이터를 하나씩 꺼내 question 변수에 대입한 후 questionList의 개수만큼 반복하며 <tr> ... </tr> 문장을 출력하라는 의미.

   자바의 for each 문을 떠올리면 쉽게 이해할 수 있을 것.

2-07 질문 목록 만들기

▪ 데이터를 화면에 출력하기

1. 다음 코드는 question 객체의 subject를 <td> 태그로 출력함.

```
<td th:text="${question.subject}"></td>
```

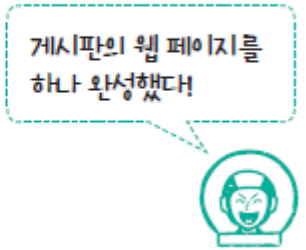
다음 코드도 question 객체의 createDate를 출력함

```
<td th:text="${question.createDate}"></td>
```

2-07 질문 목록 만들기

▪ 데이터를 화면에 출력하기

2. 브라우저에서 다시 http://localhost:8080/question/list에 접속해 보자



2-07 질문 목록 만들기

▪ 데이터를 화면에 출력하기

- 앞서 우리가 2-05절에서 테스트로 등록한
  질문 목록 데이터가 조회되어 이와 같이 출력됨
- 만약 질문 엔티티에 데이터를 더 추가했다면 더 많은 데이터가 화면에 노출될 것임

- 이제 URL을 매핑하는 것부터 HTML 템플릿을 활용해
  웹 페이지에 DB를 읽어 출력하는 것까지 모두 익혔고,
  SBB 게시판의 웹 페이지 하나를 완성하게 됨

2-08 루트 URL 사용하기

- 서버의 URL을 요청할 때 도메인명 뒤에
  아무런 주소도 덧붙이지 않는 URL을 루트 URL이라고 함.
  예를 들어 구글의 루트 URL은 google.com임

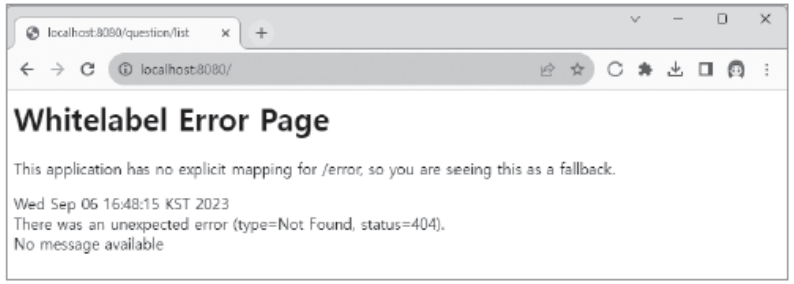
- 루트 URL을 요청했을 때 보여 지는 페이지를 메인 페이지라고 함

- SBB 서비스도 질문 목록을 메인 페이지로 정하고,
  루트 URL을 요청했을 때 질문 목록 화면으로 이동되도록 만들어 보자

2-08 루트 URL 사용하기

- 즉, 웹 브라우저에서 http://localhost:8080/question/list 대신 루트 URL인 http://localhost:8080로 접속해도 질문 목록 화면을 출력하도록 해보자

- 현재 루트 URL를 매핑하지 않아서 브라우저에서 루트 URL에 접속하면 다음과 같은 404 오류 페이지가 나타남



2-08 루트 URL 사용하기

- 루트 URL 호출 시 404 오류 페이지 대신 질문 목록 화면을 출력하기 위해 다음과 같이 MainController.java를 수정해 보자

• MainController.java

```
package com.mysite.sbb;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MainController {

    @GetMapping("/sbb")
    @ResponseBody
    public String index() {
        return "안녕하세요 sbb에 오신것을 환영합니다.";
    }

    @GetMapping("/")
    public String root() {
        return "redirect:/question/list";
    }
}
```

이 부분에 리다이렉트할 URL을 입력한다.

2-08 루트 URL 사용하기

- 이와 같이 root 메서드를 추가하고 / URL을 매핑함
- 리턴 문자열 'redirect:/question/list'는 /question/list URL로 페이지를 리다이렉트하라는 명령어임
- 여기서 리다이렉트란 클라이언트가 요청하면 새로운 URL로 전송하는 것을 의미함
- 이제 http://localhost:8080 페이지에 접속하면 root 메서드가 실행되어 질문 목록이 표시되는 것을 확인할 수 있음

2-09 서비스 활용하기

- 질문 목록에서 질문의 제목을 클릭하면
  해당 질문과 관련된 상세 내용이 담긴 화면으로 넘어가게끔 만들자
- 먼저 서비스란 무엇인지 알고 넘어가자
- 그동안 QuestionController에서 QuestionRepository를 직접 접근해
  질문 목록 데이터를 조회함
- 하지만 대부분의 규모 있는 스프링 부트 프로젝트는
  컨트롤러에서 리포지터리를 직접 호출하지 않고
  중간에 서비스를 두어 데이터를 처리함
- 이러한 서비스를 사용하여 SBB 프로그램을 개선해 보자

2-09 서비스 활용하기

- **서비스가 필요한 이유**
  - 서비스란 스프링에서 데이터 처리를 위해 작성하는 클래스
  - 서비스가 필요한 이유를 좀 더 자세히 알아보자

  - **복잡한 코드를 모듈화할 수 있다**

    예를 들어 A라는 컨트롤러가 어떤 기능을 수행하기 위해
    C라는 리포지터리의 메서드 a, b, c를 순서대로 실행해야 한다고 가정해 보자.

    B라는 컨트롤러도 A 컨트롤러와 동일한 기능을 수행해야 한다면
    A, B 컨트롤러가 C 리포지터리의 메서드 a, b, c를 호출해 사용하는 중복된 코드를 가지게 됨.
    이런 경우 C 리포지터리의 a, b, c 메서드를 호출하는 기능을 서비스로 만들고
    컨트롤러에서 이 서비스를 호출하여 사용할 수 있음.
    즉, 서비스를 사용하면 이와 같은 모듈화가 가능함

2-09 서비스 활용하기

▪ 서비스가 필요한 이유

▪ 엔티티 객체를 DTO 객체로 변환할 수 있다

우리가 앞에서 작성한 Question, Answer 클래스는 모두 엔티티 클래스임.
엔티티 클래스는 데이터베이스와 직접 맞닿아 있는 클래스이므로
컨트롤러 또는 타임리프와 같은 템플릿 엔진에 전달해 사용하는 것은 좋지 않음.

왜냐하면 엔티티 객체에는 민감한 데이터가 포함될 수 있는데,
타임리프에서 엔티티 객체를 직접 사용하면 민감한 데이터가 노출될 위험이 있기 때문.

이러한 이유로 Question, Answer 같은 엔티티 클래스는
컨트롤러에서 사용하지 않도록 설계하는 것이 좋음

2-09 서비스 활용하기

▪ 서비스가 필요한 이유

▪ 엔티티 객체를 DTO 객체로 변환할 수 있다

그래서 Question, Answer를 대신해 사용할 DTOData Transfer Object 클래스가 필요함.
Question, Answer 등의 엔티티 객체를 DTO 객체로 변환하는 작업도 필요함.

그러면 엔티티 객체를 DTO 객체로 변환하는 일은 어디서 처리해야 할까?
이때도 서비스가 필요함.

서비스는 컨트롤러와 리포지터리의 중간에서
엔티티 객체와 DTO 객체를 서로 변환하여 양방향에 전달하는 역할을 함

2-09 서비스 활용하기

▪ 서비스 만들기

- 컨트롤러에서 리포지터리 대신 사용할 서비스를 만들어 보자
- 먼저 src/main/java 디렉터리의 com.mysite.sbb.question 패키지에 QuestionService.java 파일을 만들어 다음과 같은 내용을 작성해 보자

• /question/QuestionService.java

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

import java.util.List;
import org.springframework.stereotype.Service;
import lombok.RequiredArgsConstructor;

@RequiredArgsConstructor
@Service
public class QuestionService {

    private final QuestionRepository questionRepository;

    public List<Question> getList() {
        return this.questionRepository.findAll();
    }
}
```

2-09 서비스 활용하기

▪ 서비스 만들기

- 생성한 클래스를 서비스로 만들기 위해서는
  이와 같이 클래스명 위에 @Service 애너테이션을 붙이면 됨
- @Controller, @Entity 등과 마찬가지로 스프링 부트는
  @Service 애너테이션이 붙은 클래스는 서비스로 인식하므로
  서비스를 쉽게 생성할 수 있음

- 이 코드에서는 질문 목록 데이터를 조회하여 리턴하는 getList 메서드를 추가함
- getList 메서드의 내용을 살펴보면 컨트롤러(QuestionController)에서
  리포지터리를 사용했던 부분을 그대로 옮긴 것을 알 수 있음

2-09 서비스 활용하기

▪ 컨트롤러에서 서비스 사용하기

- QuestionController.java 파일로 돌아가 QuestionController가 리포지터리 대신 서비스를 사용하도록 다음과 같이 수정해 보자

• /question/QuestionController.java

```
(... 생략 ...)

@RequiredArgsConstructor
@Controller
public class QuestionController {

    private final QuestionService questionService;

    @GetMapping("/question/list")
    public String list(Model model) {
        List<Question> questionList = this.questionService.getList();
        model.addAttribute("questionList", questionList);
        return "question_list";
    }
}
```

2-09 서비스 활용하기

▪ 컨트롤러에서 서비스 사용하기

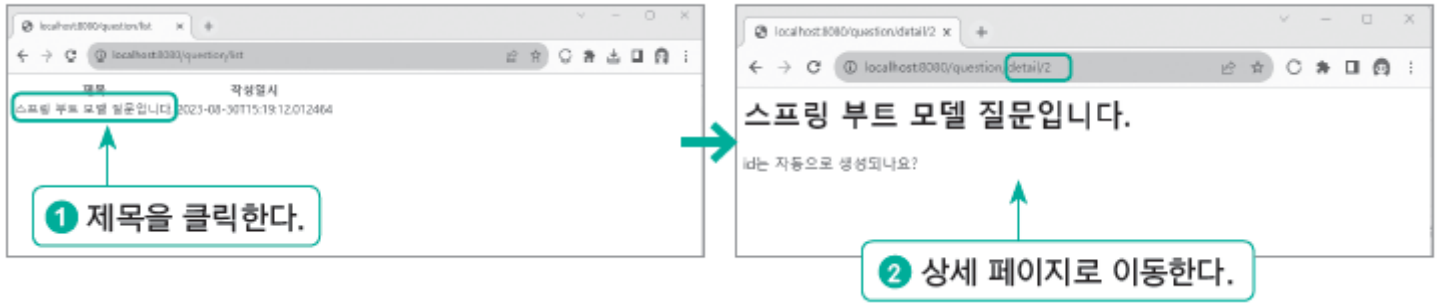
- 브라우저로 http://localhost:8080/question/list 페이지에 접속하면 리포지터리를 사용했을 때와 동일한 화면을 볼 수 있음

- 앞으로 작성할 다른 컨트롤러들도 이와 같이 리포지터리를 직접 사용하지 않고 컨트롤러 → 서비스 → 리포지터리 순서로 접근하는 과정을 거쳐 데이터를 처리할 것임

2-10 상세 페이지 만들기

▪ 컨트롤러에서 서비스 사용하기

- 이번에는 질문 목록에서 질문의 제목을 클릭하면
  해당 질문과 관련된 상세 내용이 담긴 페이지로 넘어가게끔 기능을 추가해 보자



2-10 상세 페이지 만들기

▪ 질문 목록에 링크 추가하기

- 먼저, 질문 목록의 제목을 클릭하면 상세 화면이 호출되도록 제목에 링크를 추가하자
- 질문 목록 템플릿인 question_list.html을 수정해 보자

• /templates/question_list.html

```
<table>
    <thead>
        <tr>
            <th>제목</th>
            <th>작성일시</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="question, index : ${questionList}">
            <td>
                <a th:href="@{|/question/detail/${question.id}|}"
                th:text="${question.subject}"></a>
            </td>
            <td th:text="${question.createDate}"></td>
        </tr>
    </tbody>
</table>
```

2-10 상세 페이지 만들기

▪ 질문 목록에 링크 추가하기

- <td> 태그를 통해 질문 목록의 제목을 텍스트로 출력하던 것에서
  질문의 상세 내용이 담긴 웹페이지로 이동할 수 있는 링크로 변경함
- 제목에 상세 페이지 URL을 연결하기 위해 타임리프의 th:href 속성을 사용함
- 이때 URL은 반드시 @{와 } 문자 사이에 입력해야 함
- 여기서는 문자열 /question/detail/과 ${question.id} 값이 조합되어
  /question/detail/${question.id}로 작성함

2-10 상세 페이지 만들기

▪ 질문 목록에 링크 추가하기

- 만약 좌우에 | 없이 다음과 같이 사용하면 오류가 발생함

```
<a th:href="@{/question/detail/${question.id}}" th:text="${question.subject}"></a>
```

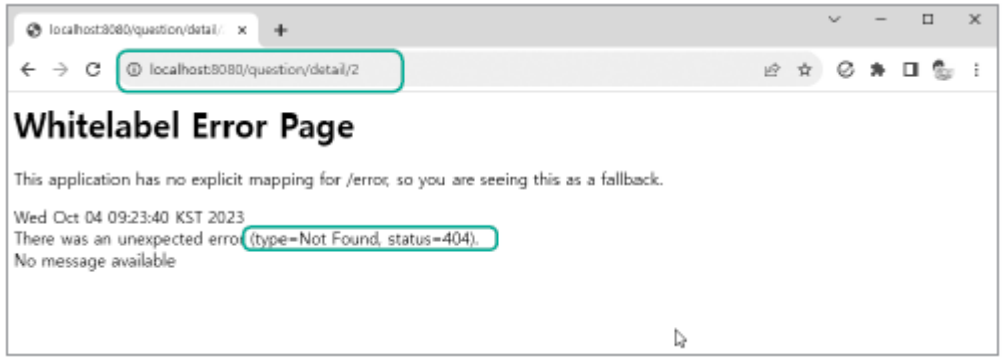
- 타임리프에서는 /question/detail/과 같은 문자열과 ${question.id}와 같은 자바 객체의 값을 더할 때는 반드시 다음처럼 |로 좌우를 감싸 주어야 함

```
<a th:href="@{|/question/detail/${question.id}|}" th:text="${question.subject"></a>
```

2-10 상세 페이지 만들기

▪ 상세 페이지 컨트롤러 만들기

1. 브라우저를 통해 질문 목록 페이지에 접속하여 링크를 클릭해 보자.
   다음과 같은 오류가 발생함



아직 http://localhost:8080/question/detail/2를 매핑하지 않았기 때문에
404 오류가 발생함

2-10 상세 페이지 만들기

▪ 상세 페이지 컨트롤러 만들기

2. 오류를 해결하기 위해 QuestionController에 질문 상세 페이지 URL을 매핑해 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 lombok.RequiredArgsConstructor;

@RequiredArgsConstructor
@Controller
public class QuestionController {

    private final QuestionService questionService;

    @GetMapping("/question/list")
    public String list(Model model) {
        List<Question> questionList = this.questionService.getList();
        model.addAttribute("questionList", questionList);
        return "question_list";
    }

    @GetMapping(value = "/question/detail/{id}")
    public String detail(Model model, @PathVariable("id") Integer id) {
        return "question_detail";
    }
}
```

2-10 상세 페이지 만들기

▪ 상세 페이지 컨트롤러 만들기

2. 요청한 URL인 http://localhost:8080/question/detail/2의 숫자 2처럼 변하는 id값을 얻을 때에는 @PathVariable 애너테이션을 사용함.

   이때 @GetMapping(value = "/question/detail/{id}")에서 사용한 id와 @PathVariable("id")의 매개변수 이름이 이와 같이 동일해야 함.

   다시 로컬 서버를 실행한 후, 브라우저에서 URL을 입력해 보자

2-10 상세 페이지 만들기

▪ 상세 페이지 컨트롤러 만들기

3. 그런데 코드를 수정하고 다시 URL을 호출하면
이번에는 404 대신 500 오류가 발생할 것임.

왜냐하면 응답으로 리턴한 question_detail 템플릿이 없기 때문.
Templates에 question_detail.html 파일을 새로 만들어 다음 내용을 작성해 보자

• /templates/question_detail.html

```
<h1>제목</h1>
<div>내용</div>
```

2-10 상세 페이지 만들기

▪ 상세 페이지 컨트롤러 만들기

4. 다시 로컬 서버를 재시작한 뒤, URL을 요청하면
   오류 없이 다음과 같은 화면이 나타날 것임

2-10 상세 페이지 만들기

▪ 상세 페이지에 서비스 사용하기

이제 화면에 출력한 '제목'과 '내용' 문자열 대신
질문 데이터의 제목(subject)과 내용(content)을 출력해 보자.
먼저, 제목과 내용에 들어갈 질문 데이터를 조회해 보자.

1. 질문 데이터를 조회하기 위해서 앞서 만든 QuestionService.java를 수정해 보자

• /question/QuestionService.java

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

import java.util.List;
import java.util.Optional;

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

import lombok.RequiredArgsConstructor;

@RequiredArgsConstructor
@Service
public class QuestionService {
    private final QuestionRepository questionRepository;

    public List<Question> getList() {
        return this.questionRepository.findAll();
    }

    public Question getQuestion(Integer id) {
        Optional<Question> question = this.questionRepository.findById(id);
        if (question.isPresent()) {
            return question.get();
        } else {
            throw new DataNotFoundException("question not found");
        }
    }
}
```

2-10 상세 페이지 만들기

▪ 상세 페이지에 서비스 사용하기

1. id값으로 질문 데이터를 조회하기 위해 getQuestion 메서드를 추가함.

   리포지터리로 얻은 Question 객체는 Optional 객체이므로
   if~else 문을 통해 isPresent 메서드로 해당 데이터(여기서는 id값)가
   존재하는지 검사하는 과정이 필요함.

   만약 id값에 해당하는 질문 데이터가 없을 경우에는
   예외 클래스인 DataNotFoundException이 실행되도록 함

2-10 상세 페이지 만들기

▪ 상세 페이지에 서비스 사용하기

2. 사실 DataNotFoundException 클래스는 아직 존재하지 않아 컴파일 오류가 발생함. com.mysite.sbb 패키지에 자바 파일을 추가로 만들어 DataNotFoundException 클래스를 정의해 보자

• DataNotFoundException.java

```
package com.mysite.sbb;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "entity not found")
public class DataNotFoundException extends RuntimeException {
    private static final long serialVersionUID = 1L;
    public DataNotFoundException(String message) {
        super(message);
    }
}
```

2-10 상세 페이지 만들기

▪ 상세 페이지에 서비스 사용하기

2. DataNotFoundException은 데이터베이스에서
   특정 엔티티 또는 데이터를 찾을 수 없을 때 발생시키는 예외 클래스로 만듦.

   이 예외가 발생하면 스프링 부트는 설정된 HTTP 상태 코드(HttpStatus.NOT_FOUND)와
   이유("entity not found")를 포함한 응답을 생성하여 클라이언트에게 반환하게 됨.

   여기서는 404 오류를 반환하도록 작성함

2-10 상세 페이지 만들기

▪ 상세 페이지에 서비스 사용하기

3. 그리고 QuestionController.java로 돌아가 QuestionService의 getQuestion 메서드를 호출하여 Question 객체를 템플릿에 전달할 수 있도록 다음과 같이 수정하자

• /question/QuestionController.java

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

    (... 생략 ...)

    @GetMapping(value = "/question/detail/{id}")
    public String detail(Model model, @PathVariable("id") Integer id) {
        Question question = this.questionService.getQuestion(id);
        model.addAttribute("question", question);
        return "question_detail";
    }
}
```

2-10 상세 페이지 만들기

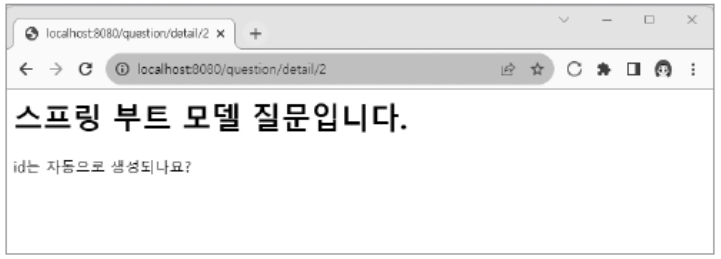
▪ 상세 페이지 출력하기

1. 상세 페이지 템플릿인 question_detail 파일로 돌아가 다음과 같이 수정해 보자.
   이때 QuestionController의 detail 메서드에서 Model 객체에
   'question'이라는 이름으로 Question 객체를 저장했으므로 다음과 같이 작성할 수 있음

• /templates/question_detail.html

```
<h1 th:text="${question.subject}"></h1>
<div th:text="${question.content}"></div>
```

1. 이제 다시 상세 페이지를 요청해 보자. 다음과 같은 화면이 나타날 것임



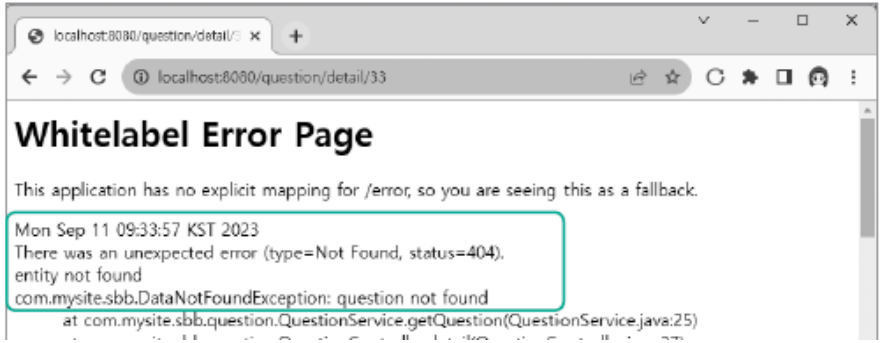
2-10 상세 페이지 만들기

▪ 상세 페이지 출력하기

3. 이번에는 33과 같은 존재하지 않는 id값을 입력해 페이지를 요청해 보자.

```
http://localhost:8080/question/detail/33
```

이와 같이 존재하지 않는 데이터를 조회하려고 할 경우에는
DataNotFound Exception이라는 예외 클래스가 실행되어
404 Not found 오류가 발생하는 것을 확인할 수 있음



2-11 URL 프리픽스 알아 두기

- 이제 질문 상세 페이지에서 답변을
  입력할 수 있도록 프로그램을 만들어 보자

- 이와 같은 내용을 배우기 전에
  QuestionController.java의 URL 매핑을 잠시 살펴보자

- 현재 QuestionController.java에는 다음 2개의 URL이 매핑되어 있음

```
1 @GetMapping("/question/list")
2 @GetMapping(value = "/question/detail/{id}")
```

URL 매핑 시 value 매개변수는 생략할 수 있다.

2-11 URL 프리픽스 알아 두기

- URL의 프리픽스가 모두 /question으로 시작한다는 것을 알 수 있음
- 프리픽스prefix란 URL의 접두사 또는 시작 부분을 가리키는 말로,
  QuestionController에 속하는 URL 매핑은 항상
  /question 프리픽스로 시작하도록 설정할 수 있음
- QustionController 클래스명 위에
  @RequestMapping("/question") 애너테이션을 추가하고,
  메서드 단위에서는 /question을 생략하고 그 뒷부분만을 적으면 됨

2-11 URL 프리픽스 알아 두기

- 이 내용을 바탕으로 QuestionController.java를 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 lombok.RequiredArgsConstructor;
```

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

    private final QuestionService questionService;

    @GetMapping("/list")
    public String list(Model model) {
        (... 생략 ...)
    }

    @GetMapping(value = "/detail/{id}")
    public String detail(Model model, @PathVariable("id") Integer id) {
        (... 생략 ...)
    }
}
```

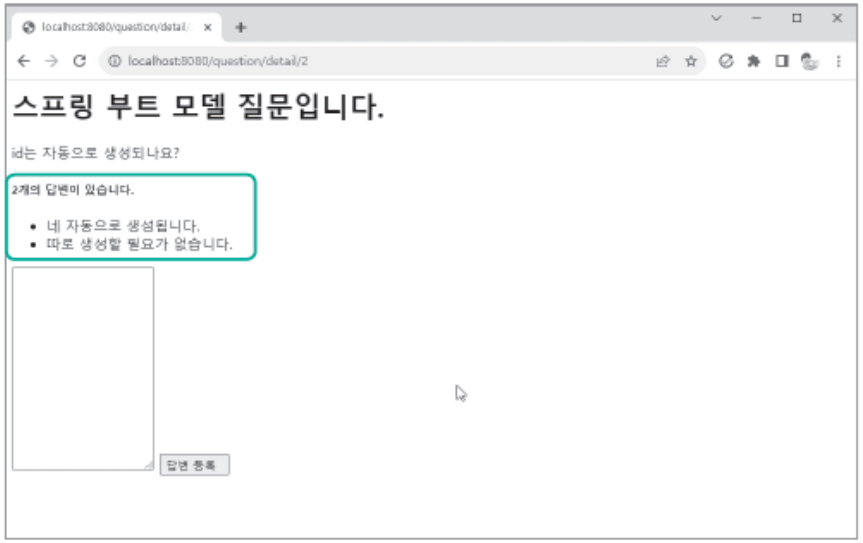
2-11 URL 프리픽스 알아 두기

- list 메서드의 URL 매핑은 /list이지만 @RequestMapping 애너테이션에서 이미 /question URL을 매핑했기 때문에 /question + /list가 되어 최종 URL 매핑은 /question/list가 됨
- 그러므로 이와 같이 수정하면 기존과 완전히 동일하게 URL 매핑이 이루어짐
- 다만, 앞으로 QuestionController.java에서 URL을 매핑할 때 반드시 /question으로 시작한다는 것을 기억해 두자

2-12 답변 기능 만들기

- 실습을 통해 우리는 질문 목록을 확인하고,
  질문 제목을 클릭하면 내용을 상세하게 볼 수 있는 페이지까지 만들어 봄

- 이번에는 질문에 답변을 입력하고,
  입력한 답변을 질문 상세페이지에서 확인할 수 있도록 구현해 보자



2-12 답변 기능 만들기

▪ 텍스트 창과 등록 버튼 만들기

1. 상세 페이지 템플릿인 question_detail.html에 답변 저장을 위한 form, textarea, input 요소를 다음과 같이 추가해 보자

• /templates/question_detail.html

```
<h1 th:text="${question.subject}"></h1>
<div th:text="${question.content}"></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>
```

2-12 답변 기능 만들기

▪ 텍스트 창과 등록 버튼 만들기

1. [답변 등록] 버튼을 누르면 전송되는 form의 action은
   타임리프의 th:action 속성으로 생성함.

   이제 텍스트 창에 답변을 작성하고, 답변 등록 버튼을 클릭하면
   /answer/create/2(여기서 '2'는 질문 데이터의 고유 번호를 의미한다.)와 같은
   URL이 post 방식으로 호출될 것임

2-12 답변 기능 만들기

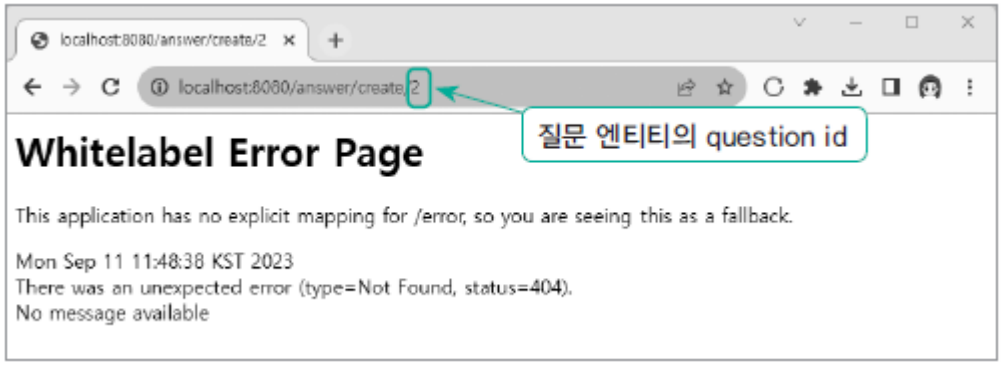
▪ 텍스트 창과 등록 버튼 만들기

2. 코드를 추가했으면 로컬 서버를 실행한 후, 질문 상세 페이지에 접속해 보자

2-12 답변 기능 만들기

▪ 텍스트 창과 등록 버튼 만들기

3. 이제 [답변 등록] 버튼을 누르면 POST 방식으로 /answer/create/<question id> URL이 호출될 것임.
하지만 아직 /answer/create/<question id> URL을 매핑하지 않았으므로 버튼을 누르면 다음과 같은 404 페이지가 나타남



이 오류를 해결하려면 답변 컨트롤러를 만들고 http://localhost:8080/answer/create/2 URL을 매핑해야 함

2-12 답변 기능 만들기

▪ 답변 컨트롤러 만들기

- 앞에서 질문 컨트롤러(QuestionController.java)를 만들었듯이
  답변 컨트롤러를 만들어 URL을 매핑해 보자

- 그러기 위해 이번에는 src/main/java 디렉터리의 com.mysite.sbb.answer 패키지에
  답변 컨트롤러로 AnswerController.java 파일을 만들어 다음과 같은 내용을 작성해 보자

• /answer/AnswerController.java

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

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

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

import lombok.RequiredArgsConstructor;
```

```
@RequestMapping("/answer")  ← 여기서는 URL 프리픽스를 /answer로 고정한다.
@RequiredArgsConstructor
@Controller
public class AnswerController {

    private final QuestionService questionService;

    @PostMapping("/create/{id}")
    public String createAnswer(Model model, @PathVariable("id") Integer id,
    @RequestParam(value="content") String content) {
        Question question = this.questionService.getQuestion(id);
        // TODO: 답변을 저장한다.  ← 답변을 저장하는 코드는 아직 작성 전이므로 이와 같이 주석으로 안내했다.
        return String.format("redirect:/question/detail/%s", id);
    }
}
```

2-12 답변 기능 만들기

▪ 답변 컨트롤러 만들기

- /answer/create/{id}와 같은 URL 요청 시
  createAnswer 메서드가 호출되도록 @PostMapping으로 매핑함
- @PostMapping 애너테이션은 @GetMapping과 동일하게
  URL 매핑을 담당하는 역할을 하지만, POST 요청을 처리하는 경우에 사용함
- 질문 컨트롤러의 detail 메서드와 달리 createAnswer 메서드의
  매개변수에는 @RequestParam(value="content") String content가 추가됨
- 이는 앞서 작성한 템플릿(question_detail.html)에서
  답변으로 입력한 내용(content)을 얻으려고 추가한 것임

2-12 답변 기능 만들기

▪ 답변 컨트롤러 만들기

- 템플릿의 답변 내용에 해당하는 <textarea>의 name 속성명이 content이므로 여기서도 변수명을 content로 사용함

- /create/{id}에서 {id}는 질문 엔티티의 id이므로 이 id값으로 질문을 조회하고 값이 없을 경우에는 404 오류가 발생할 것임

2-12 답변 기능 만들기

▪ 답변 서비스 만들기

1. 먼저, com.mysite.sbb.answer 패키지에 AnswerService.java 파일을 만들어 다음과 같은 내용을 작성해 보자

• /answer/AnswerService.java

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

import java.time.LocalDateTime;

import com.mysite.sbb.question.Question;
import org.springframework.stereotype.Service;

import lombok.RequiredArgsConstructor;

@RequiredArgsConstructor
@Service
public class AnswerService {
    private final AnswerRepository answerRepository;

    public void create(Question question, String content) {
        Answer answer = new Answer();
        answer.setContent(content);
        answer.setCreateDate(LocalDateTime.now());
        answer.setQuestion(question);
        this.answerRepository.save(answer);
    }
}
```

2-12 답변 기능 만들기

▪ 답변 서비스 만들기

1. AnswerService에는 답변(Answer)을 생성하기 위해 create 메서드를 추가함. create 메서드는 입력받은 2개의 매개변수인 question과 content를 사용하여 Answer 객체를 생성하여 저장함

1. 이제 작성한 create 메서드를 AnswerController에서 사용해 보자.

   TODO 주석문을 삭제하고 그 자리에 AnswerService의 create 메서드를 호출하여 답변을 저장할 수 있게 함

• /answer/AnswerController.java

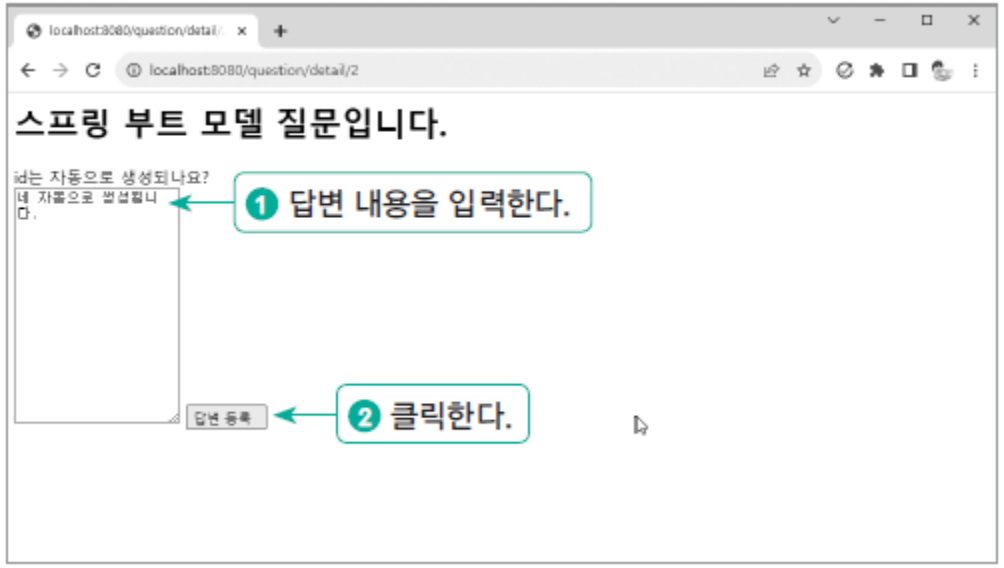
```
(... 생략 ...)

public class AnswerController {

    private final QuestionService questionService;
    private final AnswerService answerService;

    @PostMapping("/create/{id}")
    public String createAnswer(Model model, @PathVariable("id") Integer id,
    @RequestParam(value="content") String content) {
        Question question = this.questionService.getQuestion(id);
        this.answerService.create(question, content);
        return String.format("redirect:/question/detail/%s", id);
    }
}
```

TODO 주석문을 삭제하고 그 자리에 코드를 입력한다.

2-12 답변 기능 만들기

▪ 답변 서비스 만들기

3. 다시 질문 상세 페이지(http://localhost:8080/question/detail/2)에 접속하여 텍스트 창에 아무 값이나 입력하고 [답변 등록] 버튼을 클릭해 보자



2-12 답변 기능 만들기

▪ 답변 서비스 만들기

3. 현 상태에서는 확인할 수 없지만 답변은 잘 저장됨.

   그런데 화면에는 아무런 변화가 없는데,
   아직 등록한 답변 내용을 화면에 표시하도록
   템플릿에 추가하지 않았기 때문임

2-12 답변 기능 만들기

▪ 상세 페이지에 답변 표시하기

1. 답변은 질문 아래에 보이도록 상세 페이지 템플릿인 question_detail.html에 다음과 같이 추가해 보자.

• /templates/question_detail.html

```
<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>
```

기존 코드에 답변을 확인할 수 있는 영역을 추가함.
#lists.size(question.answerList)는 답변 개수를 의미함.
따라서 '1개의 답변이 있습니다.' 와 같은 문장이 화면에 표시될 것임.

 태그로 답변 리스트에 관한 내용을 묶음.
그리고 <ul> 태그를 사용하여 질문에 연결된 답변을 모두 표시함

2-12 답변 기능 만들기

▪ 상세 페이지에 답변 표시하기

2. 이제 질문 상세 페이지를 새로 고침 하면 텍스트 창에 입력한 답변이 보일 것임