HTTP와 REST 컨트롤러

11.1 REST API의 동작 이해하기

REST API : REST 기반으로 API를 구현한 것

- **REST** : HTTP URL로 서버의 자원을 명시하고, HTTP 메서드로 해당 자원에 대해 CRUD(생성, 조회, 수정, 삭제)하는 것
- **API** : 클라이언트가 서버의 자원을 요청할 수 있도록 서버에서 제공하는 인터페이스
- REST API를 잘 구현하면 클라이언트가 기긱에 구애받지 않고 서버의 자원 이용이 가능하고, 서버가 클라이언트의 요청에 체계적으로 대응 가능 ⇒ 서버 프로그램의 재사용성과 확장성이 좋아짐

11.2 REST API의 구현 과정

- REST API의 URL 설계
  - **조회 요청** : /api/articles 또는 /api/aritlces/{id}

    → GET 메서드로 Article 목록 전체 또는 단일 Article을 조회
  - **생성 요청** : /api/articles

    → POST 메서드로 새로운 Article을 생성해 목록에 저장
  - **수정 요청** : /api/aritcles/{id}

    → PATCH 메서드로 특정 Article의 내용을 수정
  - **삭제 요청** : /api/articles/{id}

    → DELETE 메서드로 특정 Article을 삭제
- REST API로 요청과 응답을 주고받을 REST 컨트롤러 사용
- 응답할 때 적절한 상태 코드를 반환하기 위해 REsponseEntity 클래스 활용

11.3 REST API 구현하기

- com.example.firstproject에서 com.example. firstproject.api로 새 api 패키지 생성

1) REST 컨트롤러 맛보기

- api 패키지에 FirstApiController 생성
- 클래스 위에 @Controller(일반 컨트롤러) 대신 `@RestController`(REST 컨트롤러) 사용
- localhost:8080/api/hello로 URL 요청이 들어왔을 때 hello world! 출력하는 메서드 작성
- 서버 실행과 Talend API Tester 사용으로 확인

💫**REST 컨트롤러와 일반 컨트롤러의 차이**

- REST 컨트롤러 : JSON이나 텍스트 같은 데이터를 반환
- 일반 컨트롤러 : 뷰 페이지를 반환

2) REST API: GET 구현하기

모든 게시글 조회하기

- api 패키지에 ArticleApiController 생성
- @RestController 어노테이션 추가
- GET 요청 처리하는 메서드 생성
  - @GetMapping으로 "/api/articles" 주소로 오는 URL 요청 받기
  - 메서드 수행 결과로 Article 묶음을 반환하므로 반환형이 List<Article>인 index() 메서드 정의
    - return 문에는 articleRepository의 findAll() 메서드를 사용해 DB에 저장된 모든 Article을 가져와 반환
  - 클래스 내부에 articleRepository 선언, @Autowired 어노테이션 붙여 의존성 주입
  - 서버 실행 및 Talend API Tester 사용으로 확인

단일 게시글 조회하기

- index() 메서드 복사해 아래에 붙여넣고, @GetMapping의 URL을 "/api/articles/{id}로 수정
- 메서드 반환형을 Article로 수정, 메서드 이름을 show()로 수정
  - return 문에는 DB에서 id로 검색해 얻은 엔티티를 가져오도록 수정, 해당 엔티티가 없으면 null을 반환하도록 함
- DB에서 id로 검색하려면 show() 메서드의 매개변수로 id를 받아 와야 하는데, id는 요청 URL에서 가져오므로 매개변수 앞에 @PathVariable을 붙임
- 서버 실행 및 Talend API Tester 사용으로 확인

3) REST API: POST 구현하기

- @PostMapping으로 "/api/articles" 주소로 오는 URL 요청 받기
- 반환형이 Article인 create() 메서드 정의, 수정할 데이터를 dto 매개변수로 받아 옴
  - 받아온 dto는 DB에서 활용할 수 있도록 엔티티로 변환해 article 변수에 넣고, articleRepository를 통해 DB에 저장한 후 반환
- 서버 실행 및 Talend API Tester 사용으로 확인
- dto 매개변수 앞에 `@RequestBody` 어노테이션 추가

  ⇒ 요청 시 본문(BODY)에 실어 보내는 데이터를 create() 메서드의 매개변수로 받아올 수 있음

4) REST API: PATCH 구현하기

데이터 전체를 수정할 경우

- 데이터 수정 요청 받아 처리할 메서드 생성

    - @PatchMapping으로 "/api/articles/{id}" 주소로 오는 URL 요청 받기
    - 반환형이 Article인 update() 메서드 정의, 매개변수로 요청 URL의 id와 요청 메시지의 본문 데이터 받아옴
- 메서드 본문 작성
    - 수정용 엔티티 생성하기
        - 클라이언트에서 받은 수정 데이터가 담긴 dto를 엔티티로 변환해 article 변수에 저장
        - id와 article 내용 로그 찍어보기
        - @Slf4j 어노테이션 추가
    - DB에 대상 엔티티가 있는지 조회하기
        - articleRepository.findById(id)를 통해서 DB에서 해당 id를 가진 엔티티를 가져오되 없다면 널(null)을 반환 → target 변수에 저장
    - 대상 엔티티가 없거나 수정하려는 id가 잘못됐을 경우 처리하기
        - 대상 엔티티가 없거나(target == null) 수정 요청 id와 본문 id가 다를(id != article.getId) 경우 조건문 실행
        - 잘못된 요청임을 확인하도록 id와 article 내용 로그 찍기
        - Article을 ResponseEntity에 담아 반환해 반환하는 데이터에 상태 코드 실어 보낼 수 있도록 함
        - if 문의 실행 결과 REsponseEntity의 상태(status)에는 400 또는 HttpStatus.BAD_REQUEST를, 본문(body)에는 반환할 데이터가 없으므로 null 실어 반환
    - 대상 엔티티가 있으면 수정 내용으로 업데이트하고 정상 응답 보내기
        - article 엔티티에 담긴 수정용 데이터를 DB에 저장, updated 변수에 저장
        - 수정된 데이터는 ResponseEntity에 담아 보냄
            - 상태는 정상 응답이므로 200 또는 HttpStatus.OK 싣고, 본문(body)에는 반환할 데이터인 updated를 실음

일부 데이터만 수정할 경우

- target에 patch() 메서드로 article(수정할 내용만)을 붙임
- target을 DB에 저장
- patch() 메서드 생성
- 수정할 내용이 있는 경우에만 동작
    - if 문으로 article(수정 엔티티)의 title이 null이 아니면(article.title != null), this(target)의 title의 갱신, 같은 방법으로 content도 갱신

5) REST API: DELETE 구현하기

- DELETE 요청을 받아 처리할 메서드 생성
    - @DeleteMapping 으로 "/api/articles/{id}" URL 요청 받기
    - 반환형으로 ResponseEntity에 <Article> 실어 보내는 delte()메서드 정의, URL의 id를 매개변수로 받아 옴

- 메서드 작성
  - DB에서 대상 엔티티가 있는지 조회하기
    - 삭제할 대상 엔티티가 있는지 조회, 없으면 null 반환
    - 반환값은 target에 저장
  - 대상 엔티티가 없어서 요청 자체가 잘못됐을 경우 처리하기
    - target이 null이면 ResponseEntity의 상태(status)에는 BAD_REQUEST, 본문(body)에는 null을 실음
  - 대상 엔티티가 있으면 삭제하고 정상 응답(200) 반환하기
    - 찾은 대상 엔티티를 삭제, ResponseEntity의 상태(status)에는 HttpStatus.OK, 본문(body)에는 null을 실음
    - return 문에 body(null) 대신 build() 도 가능
      - build() : HTTP 응답의 body가 없는 ResponseEntity 객체를 생성