Spring Boot 06 [API 개발 기본]

실전! 스프링 부트와 JPA 활용 2 - API 개발과 성능 최적화

김영한

2022.08.13

Postman 설치

@Controller + @ResponceBody ⇒ @RestController

Rest API

Rest : Representational State Transfer 의 약자로 HTTP의 장점을 사용해 자원을 효과적으로 표현하고 전달해야한다. 자원(Resource) 와 행위(Method)에 대한 표현으로 이뤄진 표현이 중요하다.

RESTful API란 무엇인가요? ⇒ aws 홈페이지 설명

회원등록 **API V1 [** 문제 있음 **]**

```
@RestController
@RequiredArgsConstructor
public class MemberAPiController {

    private final MemberService memberService;

    @PostMapping("/api/v1/members")  → API 매핑
    public CreateMemberResponse saveMemberV1(@RequestBody @Valid Member member){
→ JSON 형식으로 보냈을시에 Member에 매핑된다.
        Long id = memberService.join(member);
        return new CreateMemberResponse(id);
    }
    @Data → 리턴형식으로 제이슨방식으로 리턴 { id : "값" }
    static class CreateMemberResponse {
        private Long id;
        public CreateMemberResponse(Long id) {
            this.id = id;
        }
    }
}
```

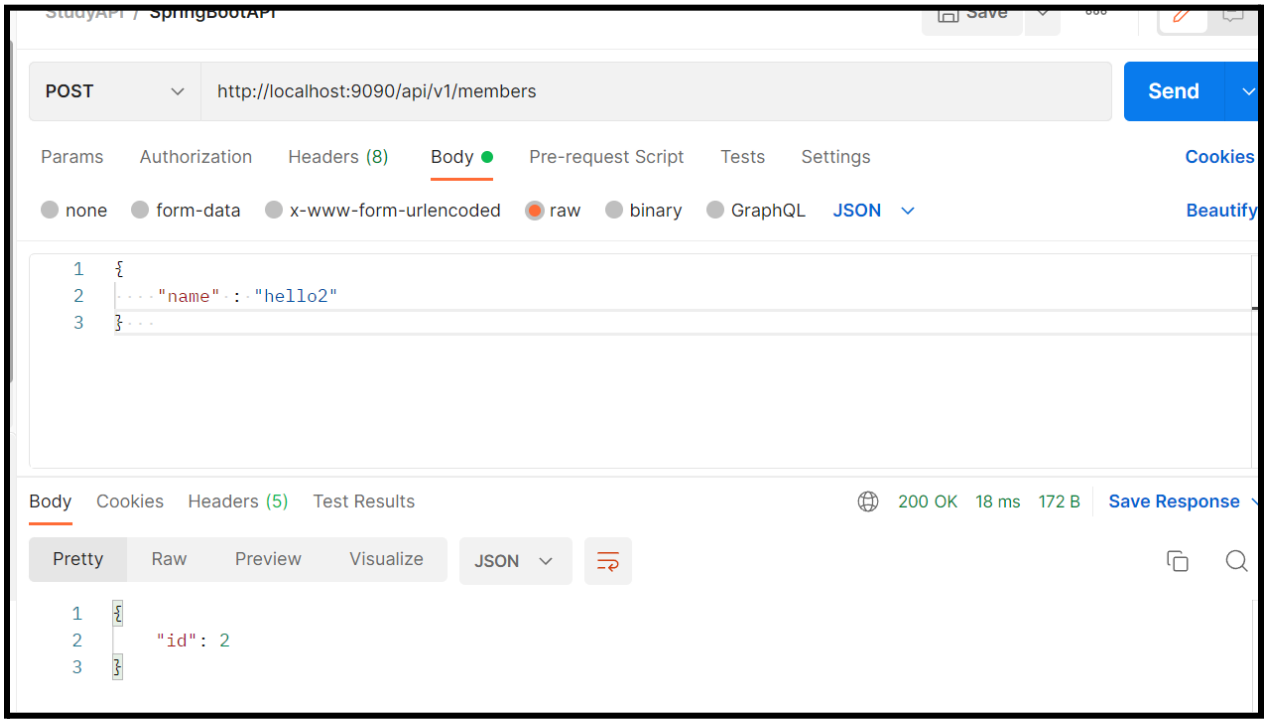
기본적으로 JSON 형식으로 보내고 Postman으로 rest API join테스트

사실 화면에 프레젠테이션 계층에 검증 로직 엔티티에 들어가 있으면 다양한 상황에서 문제가 생길 수 있다. 그리고 **Member** 엔티티 **name** 이 다른식으로 변경된다면 **(username)** 요청 정보가 변경되어야 하기 때문에 **API** 동작에 문제가 생긴다.

엔티티가 변경되었을때 **API**스펙도 변경되어야 한다는게 문제이다. 그러니 **DTO**를 만들어서 연결자를 만들어줘야한다**!**

API를 설계할 때 엔티티를 파라미터로 받으면 안된다.

postman 화면



회원등록 **API V2 [DTO** 사용 **]**

```
@PostMapping("/api/v2/members")
public CreateMemberResponse saveMemberV2(@RequestBody @Valid CreateMemberRequest
request){ → 차이점은 파라미터이다.

   Member member = new Member();
   member.setName(request.getName());
```

```
    Long id = memberService.join(member);
    return new CreateMemberResponse(id);
}
@Data
static class CreateMemberRequest{     → 프레젠테이션 계층으로 연결되는 DTO
    private String name;
}
```

→엔티티가 변경되면 무조건 컴파일 오류만 발생하게 되는것.

엔티티가 변경되더라도 **API** 스펙은 변경이 안된다.

DTO 설계의 장점

API 스펙에 대해 DTO만 보게 되어도 요구 request를 알기 쉽고 위에 발생한 문제들이 사라지고 엔티티가 변경되더라도 API스펙이 변경이 되지 않는다.

절대로 엔티티를 노출하지 말자! → 엔티티를 최대한 순수하게 가져가자

OCP! 변경에 닫혀있고 확장에 열려있는 설계를 해야한다.

회원 정보 수정

```
@PutMapping("/api/v2/members/{id}")   → PUT 매핑
public UpdateMemberResponse updateMemberResponse(
        @PathVariable("id") Long id,
        @RequestBody @Valid UpdateMemberRequest request){
    memberService.update(id, request.getName());
    Member findMember = memberService.findOnd(id);

    return new UpdateMemberResponse(findMember.getId(), findMember.getName());
}

@Data         → DTO 에서는 자유롭게 lombok사용, 어차피 데이터 전달밖에 하지않기 때문에
static class UpdateMemberRequest{
    private String name;
}

@Data
@AllArgsConstructor
static class UpdateMemberResponse{
```

```
    private Long id;
    private String name;
}
```

회원 정보 조회 [수정 필요]

→ 엔티티 그대로 노출 (엔티티 정보 그대로 전달)

```
@GetMapping("/api/v1/members")
public List<Member> membersV1(){
    return memberService.findMembers();
}
```

@JsonIgnore로 필요없는 정보를 뺄 수도 있긴하지만 → 다른 API 를 설계할때 엔티티에 추가해 놓은 어노테이션이 발목을 잡을 가능성이 크다!

또한 엔티티에 프레젠테이션 로직이 추가되기 시작하면 문제가 생긴다.

회원 정보 조회 [**DTO** 사용]

```
@GetMapping("/api/v2/members")
public Result memberV2(){
    List<Member> findMembers = memberService.findMembers();
    List<MemberDto> collect = findMembers.stream()
            .map(m -> new MemberDto(m.getName()))
            .collect(Collectors.toList());

    return new Result(collect.size(),collect);
}

@Data
@AllArgsConstructor
static class Result<T> {
    private T data; 카운트를 추가하고 싶을때 private int count;
}

@Data
@AllArgsConstructor
static class MemberDto{
    private String name;
}
```

API 공부시작 의문점

기존의 MVC 방식에 따라 웹 페이지 간의 이동시

요청페이지 → Controller → 응답 페이지 [자세한 부분 생략] 이라고 한다면

RestAPI를 사용한 RestController는 요청페이지와 응답페이지를 어떻게 연결해줄까?

예전에 프로젝트할때 오픈 API를 사용하게 되면

1. 비동기 통신 AJAX를 통해 화면에 뿌려준다.

2. 페이지가 로드 될 때 fetch를 사용해 사용한다.

대략 위 두가지 방식으로 사용을 했었다. SPA 싱글페이지어플리케이션이라면 위와 같은 방식이라고 생각이 되지만, HTTP 상태 메시지를 만들수 있기 때문에 그것으로 설계를 하는 것이 더욱 효율적이지 않을까 생각이 된다.

정답은 존재하지 않지만 **RestAPI**를 사용하는 이유가 **API**로써 자바랑 통신하고 그걸 이용하는 이용자는 **API** 스펙만을 가지고 이외의 것을 만들수 있기에 유용하지 않나 싶다**.**