Spring Boot 07 [API 지연로딩 최적화]

실전! 스프링 부트와 JPA 활용 2 - API 개발과 성능 최적화

김영한

2022.08.14

API 개발 고급

조회용 샘플 데이터 입력

- 주로 문제가 조회에서 발생한다. 입력, 수정은 한 건을 가지고와서 사용하기 때문에 문제가 생기지 않는다.

```
@Component          → 스프링의 컴포넌트 스캔의 대상이 된다.
@RequiredArgsConstructor
public class InitDB {

    private final InitService initService;

    @PostConstruct
    public void init(){ → 스프링의 init메소드.
       initService.dbInit1();
    }

   @Component
   @Transactional
   @RequiredArgsConstructor
   static class InitService{


       private final EntityManager em;
       public void dbInit1(){
           Member member = new Member();
           member.setName("userA");
           member.setAddress(new Address("서울", "1", "1111"));
           em.persist(member);
```

지연 로딩과 조회 성능 최적화

V1엔티티 그대로 반환 **[** 에러 발생 **]**

```
@GetMapping("/api/v1/simple-orders")
public List<Order> ordersV1(){
    List<Order> all = orderRepository.findAllByString(new OrderSearch());
    return all;
}
```

에러 **:** 무한 루프

→**JSON** 생성시 **Member-> Orders -> Member...** 무한루프에 빠지게 된다**.**

연관관계 둘중하나에 **@JSONIgnore** 로 끊어줘야 한다**.**

에러 **:** 지연 로딩

—--그 후에도 에러 발생 org.springframework.http.converter.HttpMessageConversionException

com.fasterxml.jackson.databind.exc.InvalidDefinitionException 발생한다.

이는 지연로딩이기 때문에 Order를 불러올때 Order객체속에 Member인스턴스가

Member 객체 대신에 ByteBuddyInterCepter 프록시 객체가 들어가 있어서 발생하는 에러이다.

값을 표출해야하지만 프록시 이기 때문에 JSON이 제대로 변환을 하지 못한다.

```
@Bean   →jackson-datatype-hibernate5 라이브러리를 넣어서 사용
Hibernate5Module hibernate5Module() {
  return new Hibernate5Module();
}
```

지연로딩을 무시하게 해준다. – JSON 형태 반환

```
{
    "id": 4,
    "member": null,
    "orderItems": null,
    "delivery": null,
    "orderDate": "2022-08-14T15:58:00.61252",
    "status": "ORDER",
    "totalPrice": 50000
```

```
    },
```

강제 **LAZY LOADING** 시키면

```
@Bean
Hibernate5Module hibernate5Module() {
  Hibernate5Module hibernate5Module = new Hibernate5Module();
  hibernate5Module.configure(Hibernate5Module.Feature.FORCE_LAZY_LOADING,true);
  return hibernate5Module;
}
```

필요하지 않는 정보까지 전부 가져온다. + 엔티티 그대로 노출

```
 {
        "id": 4,
        "member": {
            "id": 1,
. . .}
        },
        "orderItems": [
            {
                "id": 6,
            . . .
            },
            {
                "id": 7,
            . . .                }
        ],
        "delivery": {
            "id": 5,
            . . .        },
        "orderDate": "2022-08-14T16:01:05.732663",
        "status": "ORDER",
        "totalPrice": 50000
    },
```

`hibernate5Module.configure(Hibernate5Module.Feature.FORCE_LAZY_LOADING,true);`
를 사용하지 않고 그냥 **iter**루프를 돌려서 강제 초기화를 시키면 필요한 부분을 볼 수가 있다.

결론

엔티티를 그대로 노출하면 양방향 연관관계의 문제, **LAZY LOADING**의 문제, 엔티티에 반환시 보상항의 문제등 모든 문제가 다중으로 발생할 수 있기 때문에 **DTO**로 변환해서 반환하는

방법이 좋다. 지연로딩을 피하고자 즉시로딩을 사용하는것은 해결책이 될 수 없다.**(N+1**문제 발생**)**

V2 DTO 반환

```
@GetMapping("/api/v2/simple-orders")
public List<SimpleOrderDto> ordersV2(){
   List<Order> order = orderRepository.findAllByString(new OrderSearch());
   List<SimpleOrderDto> result = order.stream()
           .map(SimpleOrderDto::new)
           .collect(Collectors.toList());
   return result;
}
```

```
@Data
static class SimpleOrderDto{
   private Long orderId;
   private String name;
   private LocalDateTime orderDate;
   private OrderStatus orderStatus;
   private Address address;

   public SimpleOrderDto(Order order) {
       System.out.println("============LAZY===============");
       orderId = order.getId();
       name = order.getMember().getName();
       orderDate = order.getOrderDate();
       orderStatus = order.getStatus();
       address = order.getDelivery().getAddress();
   }
}
```

→ 이런 식으로 DTO 형식으로 반환하는데 배열 형식을 그대로 반환하면 좋지않기 때문에

형식을 `static class Result<T> { }` DTO를 변형해서 반환해주면 여러모로 사용성이 올라간다.

V1, V2 같은 문제점 (지연 로딩)

ORDER -> 주문 2개 -> 결과 주문수 2개

오더를 1개 조회할때 Member, Delivery를 조회해야한다. (조인이 되어있지 않다)

(Order조회 1번 ⇒ Member 조회 1번 + Delivery 조회 1번) * 2 = 5 [최악의 경우]

N+1 문제가 발생하게 된다**. EAGER**를 사용해도 문제가 생긴다**.**

V3 Fetch Join 사용

```
public List<Order> findAllWithMemberDelivery() {
   return em.createQuery(
           "select o from Order o" +
                   " join fetch o.member m" +
                   " join fetch o.delivery d", Order.class
   ).getResultList();
}
```

JPQL에 fetch join을 사용

```
@GetMapping("/api/v3/simple-orders")
public List<SimpleOrderDto> ordersV3() {
   List<Order> orders = orderRepository.findAllWithMemberDelivery();
   List<SimpleOrderDto> orderResult = orders.stream().map(o -> new
SimpleOrderDto(o))
           .collect(Collectors.toList());
   return orderResult;
}
```

Version2 와 반환모양이나 처리는 똑같지만

쿼리가 한번 나가게된다**.**

V4 쿼리 DTO바로 조회

쿼리를 바로 DTO로 받기위한 DTO 생성

```
@Data
public class OrderSimpleQueryDto {
   private Long orderId;
   private String name;
   private LocalDateTime orderDate;
   private OrderStatus status;
   private Address address;

   public OrderSimpleQueryDto(Long orderId, String name, LocalDateTime orderDate,
OrderStatus status, Address address) {
       this.orderId = orderId;
       this.name = name;
```

```
        this.orderDate = orderDate;
        this.status = status;
        this.address = address;
    }
}
```

```
public List<OrderSimpleQueryDto> findOrderDtos() {

    return em.createQuery(
            "select new jpabook.jpashop.repository.OrderSimpleQueryDto(o.id, m.name,
o.orderDate, o.status, d.address)" +
                    " from Order o" +
                    " join o.member m" +
                    " join o.delivery d", OrderSimpleQueryDto.class)
            .getResultList();
}
```

JPQL 쿼리 생성 ⇒ 생성한 DTO 패키지까지 전부 작성, new 를 통해 생성해주고 반환해준다.

address는 값 타입이 때문에 가능하다.

```
@GetMapping("/api/v4/simple-orders")
public List<OrderSimpleQueryDto> ordersV4() {
    return orderRepository.findOrderDtos();
}
```

```
select
        order0_.order_id as col_0_0_,
        member1_.name as col_1_0_,
        order0_.order_date as col_2_0_,
        order0_.status as col_3_0_,
        delivery2_.city as col_4_0_,
        delivery2_.street as col_4_1_,
        delivery2_.zipcode as col_4_2_
    from
        orders order0_
    inner join
        member member1_
            on order0_.member_id=member1_.member_id
    inner join
        delivery delivery2_
            on order0_.delivery_id=delivery2_.delivery_id
```

생성된 쿼리 – 원하는 정보만 가지고 있는 것들을 가져온다.

→ 리포지토리는 순수한 엔티티를 조회하는데 쓰기 위해 V4 형식의 DTO를 조회하는 용으로 사용하는 리포지토리를 따로 빼서 만들어서 관리하는 것이 좋다.

결론

재사용성이 V4같은 경우에는 떨어지게 된다. V3 같은 경우 여러곳에서 사용가능하고 원하는 DTO로 변환해서 사용도 가능하게 된다. 성능면에서 V4에서 조금 더 좋긴하지만 DTO로 조회하였기 때문에 영속성 엔티티가 아니기 때문에 수정 등의 비즈니스 로직을 수행할 수 없다. 서로의 장단점이 있기 때문에 필요한 곳에서 적용해서 사용하는 것이 필요하다.

---

1. 엔티티를 DTO로 변환하는 방법을 선택한다.
2. 필요하면 페치조인으로 성능을 최적화 한다.
3. DTO로 직접 조회를 사용한다.
4. 최후로 JPA가 제공하는 네이티브 SQL이나 JDBCTemplate을 사용해 SQL직접작성