Spring Boot 08

[API 컬렉션 조회 최적화 1]

실전! 스프링 부트와 JPA 활용 2 - API 개발과 성능 최적화

김영한

2022.08.15

API 컬렉션 조회 최적화

OneToMany 관계에서 조회할때

V1 엔티티 직접 노출

```
   @GetMapping("/api/v1/orders")
   public List<Order> ordersV1(){
       List<Order> all = orderRepository.findAllByString(new OrderSearch());
       for (Order order : all) {
           order.getMember().getName(); //강제 초기화
           order.getDelivery().getAddress();
           List<OrderItem> orderItems = order.getOrderItems();
           orderItems.stream().forEach(o -> o.getItem().getName());
           → Order내부 orderItems 도 각자 강제 초기화 해야한다.
//            orderItems.stream().map(o->o.getItem().getName());
       }
       return all;
   }
```

Hibernate5Module 를 사용하기 때문에 LAZY는 프록시 로딩 → 강제 초기화 해준다.

+ 양방향 관계는 한쪽에 @JSONIgnore를 사용해서 한쪽을 끊어줘야한다.

V2 엔티티를 DTO로 반환

```
@Getter
static class OrderDto {

   private Long orderId;
   private String name;
   private LocalDateTime orderDate;
   private OrderStatus orderStatus;
   private Address address;
   private List<OrderItem> orderItems;

   public OrderDto(Order order){
       orderId = order.getId();
       name = order.getMember().getName();
       orderDate = order.getOrderDate();
       orderStatus = order.getStatus();
       address = order.getDelivery().getAddress();
       order.getOrderItems().stream().forEach(o -> o.getItem().getName());
       orderItems = order.getOrderItems();
   }
}
```

기본적인 DTO초기화 하는 방식으로 동작시킨다면? 문제가 생긴다.

문제점 & 해결방안

DTO안에서 엔티티를 초기화 해서 랩핑하면 안된다. → 엔티티를 반환하는 결과와 같은 문제가 생기는 것. 아예 컬렉션도 다른 **DTO**로 만들어서 의존성을 없애줘야한다.

```
@GetMapping("api/v2/orders")
public List<OrderDto> ordersV2(){
   List<Order> orders = orderRepository.findAllByString(new OrderSearch());
   List<OrderDto> collect = orders.stream()
           .map(o -> new OrderDto(o))
           .collect(Collectors.toList());
   return collect;
}
-------------------------------------------------------------------------------
@Getter
static class OrderDto {

   private Long orderId;
   private String name;
   private LocalDateTime orderDate;
   private OrderStatus orderStatus;
   private Address address;
```

```
    private List<OrderItemDto> orderItems;

    public OrderDto(Order order){
        orderId = order.getId();
        name = order.getMember().getName();
        orderDate = order.getOrderDate();
        orderStatus = order.getStatus();
        address = order.getDelivery().getAddress();
        orderItems = order.getOrderItems().stream()
                .map(o -> new OrderItemDto(o))
                .collect(Collectors.toList());
    }
}
--------------------------------------------------------------------------------
@Getter
static class OrderItemDto{
    private String itemName;
    private int orderPrice;
    private int count;

    public OrderItemDto(OrderItem orderItem){
        itemName = orderItem.getItem().getName();
        orderPrice = orderItem.getOrderPrice();
        count = orderItem.getCount();
    }
}
```

내부 엔티티 컬렉션

내부 엔티티 컬렉션도 DTO를 생성해 (OrderItemDto) 반환을 해줘야 한다.

쿼리의 수

1 (오더 조회) + 멤버 2 + delivery 2 + orderItems 2 + item 4 => 쿼리가 엄청나게 나간다.

영속성 컨텍스트에 등록되면 따로 조회안해서 중복되는 데이터에서는 쿼리가 줄어들지만 해결책은 아니다.

V3 Fetch Join 사용 최적화

그냥 Join 시?

```
public List<Order> findAllWithItem() {
   return em.createQuery(
           "select o from Order o" +
                   " join fetch o.member m" +
                   " join fetch o.delivery d" +
                   " join fetch o.orderItems oi" +
                   " join fetch oi.item i", Order.class)
           .getResultList();
   // 오더가 2개 orderItem4개 --> Join 하면 Order가 4개가 된다.
}
```

확인 !

데이터 뻥튀기

select * from orders o

join order_item oi on o.order_id = oi.order_id;

ORDER_ID	ORDER_DATE	STATUS	DELIVERY_ID	MEMBER_ID	ORDER_ITEM_ID	COUNT	ORDER_PRICE	ITEM_ID	ORDER_ID
4	2022-08-15 10:26:04.98269	ORDER	5	1	6	1	10000	2	4
4	2022-08-15 10:26:04.98269	ORDER	5	1	7	2	20000	3	4
11	2022-08-15 10:26:05.076444	ORDER	12	8	13	3	20000	9	11
11	2022-08-15 10:26:05.076444	ORDER	12	8	14	4	40000	10	11

2개의 order에 아이템에 따라 4개의 데이터가 보이게된다.

일대다의 관계에서 다의 숫자로 조인이 되는것.

OneToMany의 중복제거 필요

결과를 row가 늘어나지만 같은결과가 중복 출력되게 된다.

해결 방법 **Distinct**

"select distinct o from Order o" +

" join fetch o.member m" + ... 를 해주면 order가 같으면 리스트에 담아서 반환해준다.

JPA에 Distinct는 DB에 DIstinct 키워드를 날려주고 중복을 제거해서 지정한 엔티티가 중복인 경우에 컬렉션(다른 값을)에 담아준다.

쿼리는 단 한번 나가게 된다.

어마어마한 단점! 페이징 불가

페이징 쿼리가 불가능해 진다.

```
em.createQuery(
        "select distinct o from Order o" +
                " join fetch o.member m" +
                " join fetch o.delivery d" +
                " join fetch o.orderItems oi" +
                " join fetch oi.item i", Order.class)
        .setFirstResult(1)      ← 추가하게 된다면?
        .setMaxResults(1000)
        .getResultList();
```

2022-08-15 11:07:44.856 WARN 17852 --- [nio-9090-exec-2] o.h.h.internal.ast.QueryTranslatorImpl : HHH000104: firstResult/maxResults specified with collection fetch; applying in memory!

경고가 발생한다. 그러므로 전체를 조회해서 데이터를 메모리에서 쪼갠다. [정말 위험하다]

오더의 기준자체가 틀어지게 되므로 **hibernate**가 경고를 뱉고 메모리에 올려서 갯수를 짤라오는 식으로 적용되는 것이다. 즉! 일대다 조인에서 페이징이 불가능해진다.

또 하나의 주의점

컬렉션 페치 조인은 1개만 사용할 수 있다. 만약 둘 이상으로 사용하게 된다면, 뻥튀기에 뻥튀기가 되면서 데이터가 부정확하게 조회 될 수가 있기 때문에 사용하지않는다.

컬렉션 페치조인 페이징

페이징 + 컬렉션 엔티티를 함께 조회하려면 어떻게 해야할까?

방법 순서

1. ToOne (OneToOne ManyToOne) 관계들은 전부 Fetch join 걸어도 된다. (걸어준다.)
2. 컬렉션 관계는 지연 로딩으로 해결한다. ???
3. 지연 로딩을 최적화 하기위해 hibernate.default_batch_fetch_size, @BatchSize 를 적용한다.

적용 **1**번 **+ 2**번

Order + member(ManyToOne) , Order + Delivery (OneToOne) => 쿼리 1번

order 속 orderItems 2개 - > 각 다른 아이템 2개 1+2 , 1+2 => 6번

총 6번의 쿼리가 발생한다.

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

우선 1번 적용하기 위한 페이징

application.yml 설정추가

```
jpa:
  hibernate:
    ddl-auto: create
  properties:
    hibernate:
      format_sql: true
      default_batch_fetch_size: 100  → 추가
```

를 설정하면

```
where
       orderitems0_.order_id in (
           ?, ?
       )
```

오더아이템을 IN쿼리로 미리 가져온다.

`default_batch_fetch_size: 100` 100개 까지 가져오게 되는것.

어노테이션 @BatchSize로 개별설정도 가능하다.

```
where
       item0_.item_id in (
           ?, ?, ?, ?
       )
```

아이템도 마찬가지로 orderItem에 있는것으로 찾는다. 쿼리가 3번 날아가게 된다.

사실, 1번을 생략해도 가능하긴하지만 그렇게되면 각 각 지연로딩으로 네트워크를 많이 타게 된다는 단점이 생기므로 쿼리가 한번 날아갈 수 있게하는 fetch join을 사용.

정리

쿼리 호출 수 N + 1 ⇒ 1 + 1

컬렉션 페치조인은 페이징이 불가능 하지만 이 방법은 페이징이 가능하다.

`default_batch_fetch_size` 적당한 사이즈는 100 ~ 1000 사이가 좋다. SQL IN 절을 사용해서 가져오게된다. 사이즈가 커지면 순간 부하가 증가할 수 있기는 하다. WAS 메모리 사용량은 비슷하다.