Spring Boot 09

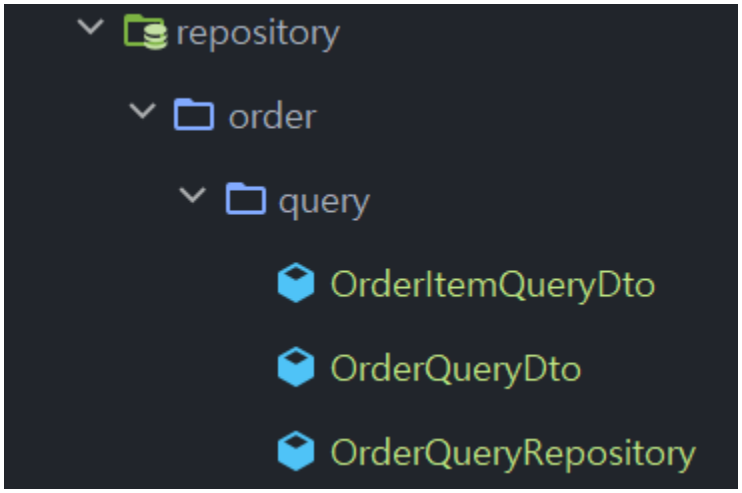
[API 컬렉션 조회 최적화 2]

실전! 스프링 부트와 JPA 활용 2 - API 개발과 성능 최적화

김영한

2022.08.15

V4 JPA에서 DTO 직접 조회



생성 후에 DTO 직접 조회할 준비

컬렉션을 루프 돌아 조회함

```
@Repository
@RequiredArgsConstructor
public class OrderQueryRepository {

    private final EntityManager em;

    public List<OrderQueryDto> findOrderQueryDtos() {
        List<OrderQueryDto> result = findOrders();

        result.forEach(o->{
            List<OrderItemQueryDto> orderItems = findOrderItems(o.getOrderId());
            o.setOrderItems(orderItems);
        });

        return result;
    }
```

```
    private List<OrderQueryDto> findOrders() {
        return em.createQuery("select new
jpabook.jpashop.repository.order.query.OrderQueryDto(o.id, m.name, o.orderDate,
o.status, d.address)" +
                        " from Order o" +
                        " join o.member m" +
                        " join o.delivery d", OrderQueryDto.class)
                .getResultList();
    }

    private List<OrderItemQueryDto> findOrderItems(Long orderId) {
        return em.createQuery("select new
jpabook.jpashop.repository.order.query.OrderItemQueryDto(oi.order.id, i.name,
oi.orderPrice, oi.count)"+
                " from OrderItem oi" +
                " join oi.item i" +
                " where oi.order.id = :orderId",OrderItemQueryDto.class)
                .setParameter("orderId",orderId)
                .getResultList();
    }
}
```

→JPQL에서 도메인을 생성하면서 조회한다.

컬렉션은 루프를 돌면서

```
result.forEach(o->{
            List<OrderItemQueryDto> orderItems = findOrderItems(o.getOrderId());
            o.setOrderItems(orderItems);
        });
```

다시 조회를 해주면서 채워 넣은것이다.

즉, 다대일 관계는 직접 조인해서 쿼리를 돌린다.

하지만 루프를 돌면서 **1 + 2 (N + 1** 문제**)** 쿼리가 발생하게 된다.

V5 N+1 해결 DTO 조회

루프를 돌면서 하나씩 조회하면 N+1 쿼리발생 문제가 나타나기 때문에

이전에 학습한 IN 쿼리를 사용한 한번에 조회하는 방식을 사용해 최적화 할 수 있게된다.

```
public List<OrderQueryDto> findAllByDto_optimization() {
   List<OrderQueryDto> result = findOrders(); → 1번
   List<Long> orderIds = result.stream()
           .map(o -> o.getOrderId())
           .collect(Collectors.toList());  → 2번

   List<OrderItemQueryDto> orderItems =
           em.createQuery("select new
jpabook.jpashop.repository.order.query.OrderItemQueryDto(oi.order.id, i.name,
oi.orderPrice, oi.count)" +
                   " from OrderItem oi" +
                   " join oi.item i" +
                   " where oi.order.id in :orderIds", OrderItemQueryDto.class)
           .setParameter("orderIds", orderIds)
           .getResultList(); → 3번

   Map<Long, List<OrderItemQueryDto>> orderItemMap = orderItems.stream()
           .collect(Collectors.groupingBy(orderItemQueryDto ->
orderItemQueryDto.getOrderId())); → 4번

   result.forEach(o->o.setOrderItems(orderItemMap.get(o.getOrderId()))); → 5번

   return result;
}
```

로직 순서

1. 기본 다대일 연관관계는 조인으로 조회한다.
2. 일대다 관계의 외래키를 컬렉션 형태로 담는다 (orderIds)
3. orderItems를 IN 쿼리를 통해 한번에 조회한다.
4. orderItems을 다시 orderId키를 가진 Map으로 표현한다 orderItempMap
5. 메모리 상에서 result에 orderId에 따라 map의 value값을 넣어준다.

⇒ 페치조인을 생각하면 생각보다. DTO를 직접조회 하는것이 이점이 있는지 생각이 또 한번 들지만 서로의 여러 방법이 존재한다.

V6 DTO 조회 플랫데이터 최적화

쿼리를 한번에 실행해서 한번에 조회한다. -즉 한번에 조회가능한 DTO를 설계한다.

우선 한번에 **Join**해서 확인

```
@Data
public class OrderFlatDto {
    private Long orderId;
    private String name;
    private LocalDateTime orderDate;
    private OrderStatus orderStatus;
    private Address address;
    private String itemName;
    private int orderPrice;
    private int count;
```

```
public List<OrderFlatDto> findAllByDto_flat() {
    return em.createQuery(
            "select new jpabook.jpashop.repository.order.query.OrderFlatDto(o.id,
m.name, o.orderDate, o.status, d.address, i.name, oi.orderPrice, oi.count)" +
                    " from Order o" +
                    " join o.member m" +
                    " join o.delivery d" +
                    " join o.orderItems oi" +
                    " join oi.item i", OrderFlatDto.class)
            .getResultList();
}
```

결과 값

```
[
    {
        "orderId": 4,
        "name": "userA",
        "orderDate": "2022-08-15T17:17:02.271621",
        "orderStatus": "ORDER",
        "address": {
            "city": "서울",
            "street": "1",
            "zipcode": "1111"
        },
        "itemName": "JPA1 BOOK",
        "orderPrice": 10000,
```

```
        "count": 1
    },
    {
        "orderId": 4,
        "name": "userA",
        "orderDate": "2022-08-15T17:17:02.271621",
        "orderStatus": "ORDER",
        "address": {
            "city": "서울",
            "street": "1",
            "zipcode": "1111"
        },
        "itemName": "JPA2 BOOK",
        "orderPrice": 20000,
        "count": 2
    },
. .  .
]
```


Order ID 4 정보가 두개이다**.** 뻥튀기 **!** 중복 제거를 안했기 때문에

일대다 **Join** 중복제거 **-** 직접 제거

JPA에서 중복을 제거하는 방법을 통해 중복을 제거해준다.

```
public List<OrderQueryDto> ordersV6(){
   List<OrderFlatDto> flats = orderQueryRepository.findAllByDto_flat();
   return flats.stream()
           .collect(Collectors.groupingBy(o -> new OrderQueryDto(o.getOrderId(),
                           o.getName(), o.getOrderDate(), o.getOrderStatus(),
o.getAddress()),
                   Collectors.mapping(o -> new OrderItemQueryDto(o.getOrderId(),
                           o.getItemName(), o.getOrderPrice(), o.getCount()),
Collectors.toList())
           )).entrySet().stream()
           .map(e -> new OrderQueryDto(e.getKey().getOrderId(),
                   e.getKey().getName(), e.getKey().getOrderDate(),
e.getKey().getOrderStatus(),
                   e.getKey().getAddress(), e.getValue()))
           .collect(Collectors.toList());
}
```

직접 어떻게든 메모리내에서 분해해서 다른형태 **DTO**로 변경한다 **.**

쿼리는 한번이지만 결국 중복데이터도 가지고 오기 때문에 느릴 수가 있고 페이징이 불가능하다.

총 정리 **- API** 컬렉션 조회

V1 : 엔티티 조회 -그대로 반환

V2 : 엔티티 조회 후 DTO로 변환

V3 : 페치 조인 후 DTO로 변환 페이징 불가 → data 뻥튀기 distinct로 제거

→ **V3.1 :** 페치 조인 페이징

ToOne 관계는 지연로딩 유지하고

컬렉션은 지연로딩 유지, IN쿼리가 발생하게 한다.

V4 : JPA에서 DTO를 직접 조회

V5 : 컬렉션 을 IN절을 사용해 미리 조회해서 최적화

V6 : 플랫 데이터 JOIN 결과를 조회 후 가공해서 사용

추천 적용 순서

우선 엔티티 조회 방식으로 **DTO** 변환 하는 방식사용 우선

엔티티 조회 방식인 코드를 거의 수정하지 않고 최적화 시도가 가능하다.

DTO를 직접 조회하는 경우는 DTO도 전부 수정해야 하는 문제가 생길 수도 있다.

1. 엔티티 조회 방식으로 우선 접근
    a. 페치 조인으로 쿼리 수를 최적화
    b. 컬렉션 최적화
        i. 페이징 필요 → batch_fetch_size로 최적화
        ii. 페이징 필요X -> 페치 조인 사용
2. 해결 안될시 DTO 조회 방식 사용
3. 해결 안될시 NativeSQL or 스프링 JdbcTemplate

질문점 **distinct**

조인시 중복 엔티티에 대해서 중복을 제거해주고 다른 부분을 Collection으로 변경해줌 (JPA기능)

new 생성자를 인한 DTO 직접 조회를 할때는 distinct가 정확히 같은 데이터에서만 작동하는것, 이것을 혹시 엔티티 중복 제거처럼 동작하게는 못하는지?