3. API 개발 고급 - 지연 로딩과 조회 성능 최적화

#1.인강/jpa활용편/활용편2/강의

- /간단한 주문 조회 V1: 엔티티를 직접 노출
- /간단한 주문 조회 V2: 엔티티를 DTO로 변환
- /간단한 주문 조회 V3: 엔티티를 DTO로 변환 - 페치 조인 최적화
- /간단한 주문 조회 V4: JPA에서 DTO로 바로 조회

주문 + 배송정보 + 회원을 조회하는 API를 만들자
지연 로딩 때문에 발생하는 성능 문제를 단계적으로 해결해보자.

> 참고: 지금부터 설명하는 내용은 정말 중요합니다. 실무에서 JPA를 사용하려면 100% 이해해야 합니다.
> 안그러면 엄청난 시간을 날리고 강사를 원망하면서 인생을 허비하게 됩니다.

간단한 주문 조회 V1: 엔티티를 직접 노출

OrderSimpleApiController

```
package jpabook.jpashop.api;

import jpabook.jpashop.domain.Address;
import jpabook.jpashop.domain.Order;
import jpabook.jpashop.domain.OrderStatus;
import jpabook.jpashop.repository.*;
import lombok.Data;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.time.LocalDateTime;
import java.util.List;

import static java.util.stream.Collectors.toList;

/**
 *
 * xToOne(ManyToOne, OneToOne) 관계 최적화
 * Order
```

```
 * Order -> Member
 * Order -> Delivery
 *
 */
@RestController
@RequiredArgsConstructor
public class OrderSimpleApiController {

    private final OrderRepository orderRepository;

    /**
     * V1. 엔티티 직접 노출
     * - Hibernate5Module 모듈 등록, LAZY=null 처리
     * - 양방향 관계 문제 발생 -> @JsonIgnore
     */
    @GetMapping("/api/v1/simple-orders")
    public List<Order> ordersV1() {
        List<Order> all = orderRepository.findAllByString(new OrderSearch());
        for (Order order : all) {
            order.getMember().getName(); //Lazy 강제 초기화
            order.getDelivery().getAddress(); //Lazy 강제 초기환
        }
        return all;
    }
}
```

- 엔티티를 직접 노출하는 것은 좋지 않다. (앞장에서 이미 설명)
- `order` ➜ `member` 와 `order` ➜ `delivery` 는 지연 로딩이다. 따라서 실제 엔티티 대신에 프록시 존재
- jackson 라이브러리는 기본적으로 이 프록시 객체를 json으로 어떻게 생성해야 하는지 모름 ➜ 예외 발생
- `Hibernate5Module` 을 스프링 빈으로 등록하면 해결(스프링 부트 사용중)

하이버네이트 모듈 등록

스프링 부트 버전에 따라서 모듈 등록 방법이 다르다. 스프링 부트 3.0 부터는 `javax -> jakarta` 로 변경되어서 지원 모듈도 다른 모듈을 등록해야 한다.

스프링 부트 3.0 미만: Hibernate5Module 등록

`build.gradle`에 다음 라이브러리를 추가하자

```
implementation 'com.fasterxml.jackson.datatype:jackson-datatype-hibernate5'
```

`JpashopApplication`에 다음 코드를 추가하자

```
@Bean
Hibernate5Module hibernate5Module() {
    return new Hibernate5Module();
}
```

- 기본적으로 초기화 된 프록시 객체만 노출, 초기화 되지 않은 프록시 객체는 노출 안함

만약 스프링 부트 3.0 이상을 사용하면 다음을 참고해서 모듈을 변경해야 한다. 그렇지 않으면 다음과 같은 예외가 발생한다.

```
java.lang.ClassNotFoundException: javax.persistence.Transient
```

스프링 부트 3.0 이상: Hibernate5JakartaModule 등록

`build.gradle`에 다음 라이브러리를 추가하자

```
implementation 'com.fasterxml.jackson.datatype:jackson-datatype-hibernate5-jakarta'
```

`JpashopApplication`에 다음 코드를 추가하자

```
@Bean
Hibernate5JakartaModule hibernate5Module() {
    return new Hibernate5JakartaModule();
}
```

- 기본적으로 초기화 된 프록시 객체만 노출, 초기화 되지 않은 프록시 객체는 노출 안함

다음과 같이 설정하면 강제로 지연 로딩 가능

```
@Bean
Hibernate5Module hibernate5Module() {
    Hibernate5Module hibernate5Module = new Hibernate5Module();
    //강제 지연 로딩 설정
    hibernate5Module.configure(Hibernate5Module.Feature.FORCE_LAZY_LOADING, true);
    return hibernate5Module;
}
```

- 이 옵션을 키면 `order -> member`, `member -> orders` 양방향 연관관계를 계속 로딩하게 된다. 따라서 `@JsonIgnore` 옵션을 한곳에 주어야 한다.

> 주의: 스프링 부트 3.0 이상이면 `Hibernate5Module` 대신에 `Hibernate5JakartaModule`을 사용해야 한다.

> 주의: 엔티티를 직접 노출할 때는 양방향 연관관계가 걸린 곳은 꼭! 한곳을 `@JsonIgnore` 처리 해야 한다. 안그러면 양쪽을 서로 호출하면서 무한 루프가 걸린다.

> 참고: 앞에서 계속 강조했듯이 정말 간단한 애플리케이션이 아니면 엔티티를 API 응답으로 외부로 노출하는 것은 좋지 않다. 따라서 `Hibernate5Module`를 사용하기 보다는 DTO로 변환해서 반환하는 것이 더 좋은 방법이다.

> 주의: 지연 로딩(LAZY)을 피하기 위해 즉시 로딩(EARGR)으로 설정하면 안된다! 즉시 로딩 때문에 연관관계가 필요 없는 경우에도 데이터를 항상 조회해서 성능 문제가 발생할 수 있다. 즉시 로딩으로 설정하면 성능 튜닝이 매우 어려워 진다.
> 항상 지연 로딩을 기본으로 하고, 성능 최적화가 필요한 경우에는 페치 조인(fetch join)을 사용해라!(V3에서 설명)

간단한 주문 조회 V2: 엔티티를 DTO로 변환

OrderSimpleApiController - 추가

```
/**
 * V2. 엔티티를 조회해서 DTO로 변환(fetch join 사용X)
 * - 단점: 지연로딩으로 쿼리 N번 호출
 */
@GetMapping("/api/v2/simple-orders")
public List<SimpleOrderDto> ordersV2() {
    List<Order> orders = orderRepository.findAllByString(new OrderSearch());
    List<SimpleOrderDto> result = orders.stream()
            .map(o -> new SimpleOrderDto(o))
            .collect(toList());

    return result;
}
```

```
@Data
static class SimpleOrderDto {

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

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

- 엔티티를 DTO로 변환하는 일반적인 방법이다.
- 쿼리가 총 1 + N + N번 실행된다. (v1과 쿼리수 결과는 같다.)
  - `order` 조회 1번(order 조회 결과 수가 N이 된다.)
  - `order -> member` 지연 로딩 조회 N 번
  - `order -> delivery` 지연 로딩 조회 N 번
  - 예) order의 결과가 4개면 최악의 경우 1 + 4 + 4번 실행된다.(최악의 경우)
    - 지연로딩은 영속성 컨텍스트에서 조회하므로, 이미 조회된 경우 쿼리를 생략한다.

간단한 주문 조회 V3: 엔티티를 DTO로 변환 - 페치 조인 최적화

OrderSimpleApiController - 추가

```
/**
 * V3. 엔티티를 조회해서 DTO로 변환(fetch join 사용O)
 * - fetch join으로 쿼리 1번 호출
 * 참고: fetch join에 대한 자세한 내용은 JPA 기본편 참고(정말 중요함)
 */
@GetMapping("/api/v3/simple-orders")
public List<SimpleOrderDto> ordersV3() {
    List<Order> orders = orderRepository.findAllWithMemberDelivery();
    List<SimpleOrderDto> result = orders.stream()
            .map(o -> new SimpleOrderDto(o))
            .collect(toList());
```

```
        return result;
    }
```

OrderRepository - 추가 코드

```
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();
}
```

- 엔티티를 페치 조인(fetch join)을 사용해서 쿼리 1번에 조회
- 페치 조인으로 `order -> member`, `order -> delivery`는 이미 조회 된 상태 이므로 지연로딩X

간단한 주문 조회 V4: JPA에서 DTO로 바로 조회

OrderSimpleApiController - 추가

```
private final OrderSimpleQueryRepository orderSimpleQueryRepository; //의존관계 주입

/**
 * V4. JPA에서 DTO로 바로 조회
 * - 쿼리 1번 호출
 * - select 절에서 원하는 데이터만 선택해서 조회
 */
@GetMapping("/api/v4/simple-orders")
public List<OrderSimpleQueryDto> ordersV4() {
    return orderSimpleQueryRepository.findOrderDtos();
}
```

OrderSimpleQueryRepository 조회 전용 리포지토리

```
package jpabook.jpashop.repository.order.simplequery;

import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Repository;

import javax.persistence.EntityManager;
import java.util.List;
```

```
@Repository
@RequiredArgsConstructor
public class OrderSimpleQueryRepository {

    private final EntityManager em;

    public List<OrderSimpleQueryDto> findOrderDtos() {
        return em.createQuery(
                "select new
jpabook.jpashop.repository.order.simplequery.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();
    }
}
```

OrderSimpleQueryDto 리포지토리에서 DTO 직접 조회

```
package jpabook.jpashop.repository.order.simplequery;

import jpabook.jpashop.domain.Address;
import jpabook.jpashop.domain.OrderStatus;
import lombok.Data;

import java.time.LocalDateTime;

@Data
public class OrderSimpleQueryDto {

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

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

```
            this.address = address;
        }
    }
```

- 일반적인 SQL을 사용할 때 처럼 원하는 값을 선택해서 조회
- `new` 명령어를 사용해서 JPQL의 결과를 DTO로 즉시 변환
- SELECT 절에서 원하는 데이터를 직접 선택하므로 DB ➜ 애플리케이션 네트웍 용량 최적화(생각보다 미비)
- 리포지토리 재사용성 떨어짐, API 스펙에 맞춘 코드가 리포지토리에 들어가는 단점

정리

엔티티를 DTO로 변환하거나, DTO로 바로 조회하는 두가지 방법은 각각 장단점이 있다. 둘중 상황에 따라서 더 나은 방법을 선택하면 된다. 엔티티로 조회하면 리포지토리 재사용성도 좋고, 개발도 단순해진다. 따라서 권장하는 방법은 다음과 같다.

쿼리 방식 선택 권장 순서

1. 우선 엔티티를 DTO로 변환하는 방법을 선택한다.
2. 필요하면 페치 조인으로 성능을 최적화 한다. ➜ 대부분의 성능 이슈가 해결된다.
3. 그래도 안되면 DTO로 직접 조회하는 방법을 사용한다.
4. 최후의 방법은 JPA가 제공하는 네이티브 SQL이나 스프링 JDBC Template을 사용해서 SQL을 직접 사용한다.