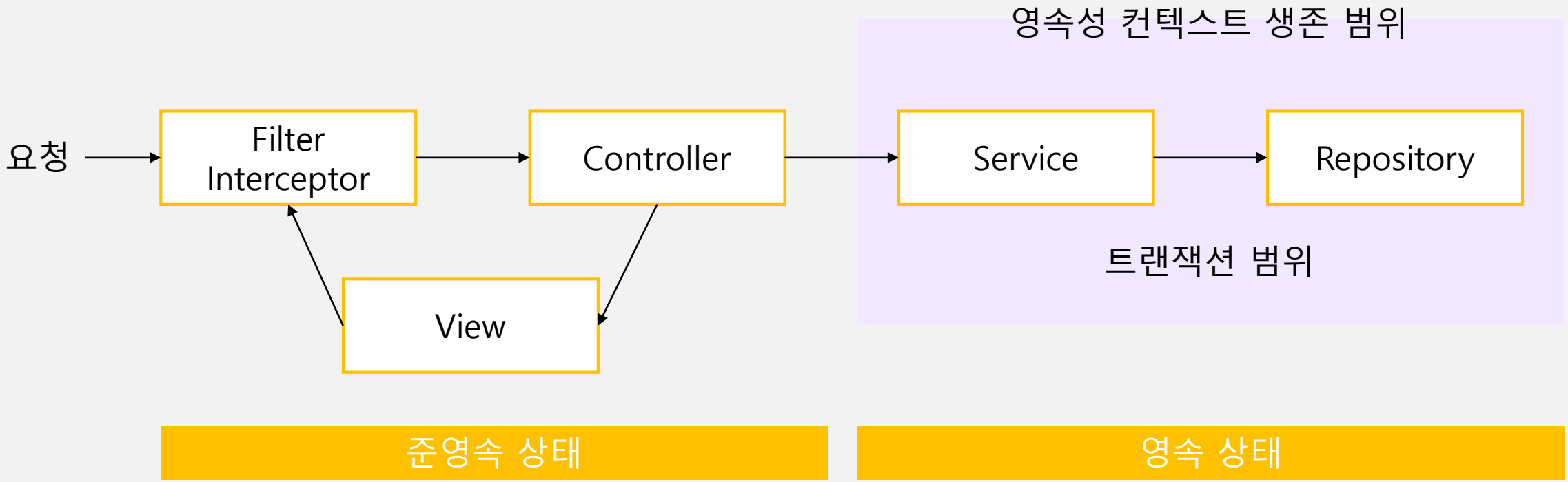
13. 웹 애플리케이션과 영속성 관리

1. 스프링 컨테이너 전략 : 트랜잭션 범위의 영속성 컨텍스트



2. 준영속 상태와 지연 로딩

트랜잭션 없는 프리젠테이션 계층에서 엔티티는 준영속 상태이다.

지연로딩 시점에 예외가 발생한다.

```java
@Entity
@Getter
@Setter
@NoArgsConstructor
public class Order1 {

    @Id @GeneratedValue
    private String id;

    @ManyToOne (fetch = FetchType.LAZY)
    @JoinColumn(name="MEMBER_ID")
    private Member1 member1;
```

```java
@GetMapping
public void view()
{
    String orderId = "test";
    Order1 order1 = orderService.findOne(orderId);
    Member1 member1 = order1.getMember1();
    System.out.println(member1.getUsername());
}
```

```
spring.datasource.url=jdbc:oracle:thin:@developerdb.spectra.co.kr:1521:orcl
spring.datasource.username=shjeon
spring.datasource.password=shjeon
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.jpa.open-in-view=false
```

```
Debug: Example1Main
Debugger  Console  Endpoints
2020-11-18 07:06:10.028  INFO 8148 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet
2020-11-18 07:06:10.028  INFO 8148 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-11-18 07:06:10.033  INFO 8148 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 5 ms
2020-11-18 07:06:10.228 ERROR 8148 --- [nio-8080-exec-1] o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servlet.service() for servlet [dispatcherServlet] in co

org.hibernate.LazyInitializationException: could not initialize proxy [spectra.jpa.Member1#shjeon] - no Session <4 internal calls>
    at spectra.jpa.Member1$HibernateProxy$RxAcqcht.getUsername(Unknown Source) ~[classes/:na]
    at spectra.jpa.OrderController.view(OrderController.java:23) ~[classes/:na] <14 internal calls>
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:634) ~[tomcat-embed-core-9.0.21.jar:9.0.21] <1 internal call>
```

3. 준영속 상태와 지연 로딩 - 해결방법

- ✓ 뷰가 필요한 엔티티를 미리 로딩해두는 방법
- ✓ OSIV를 사용해서 엔티티를 항상 영속 상태로 유지하는 방법

4. 뷰가 필요한 엔티티를 미리 로딩 해두는 방법

글로벌 페치 전략 수정 | JPQL 페치 조인 | 강제로 초기화

@ManyToOne(fetch = FetchType.**LAZY**)

↓

@ManyToOne(fetch = FetchType.**EAGER**)

→ 사용하지 않는 엔티티를 로딩한다
N+1 문제가 발생한다.

4. 뷰가 필요한 엔티티를 미리 로딩 해두는 방법

글로벌 페치 전략 수정 | JPQL 페치 조인 | 강제로 초기화

```
select o
from Order o
join fetch o.member
```

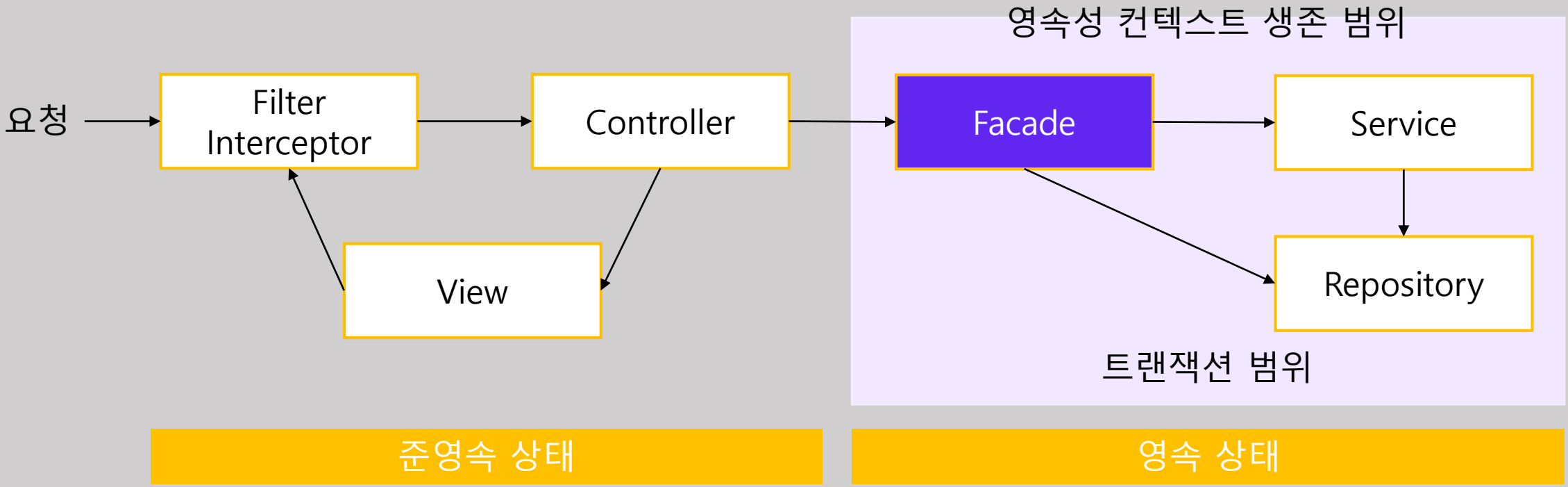
화면에 맞춘 리포지토리 메소드 증가

4. 뷰가 필요한 엔티티를 미리 로딩 해두는 방법

글로벌 페치 전략 수정 | JPQL 페치 조인 | 강제로 초기화

사용 전 Service 호출하여 @Transactional 내부에서 프록시 객체 강제로 초기화

FAÇADE 계층 추가하여 뷰를 위한 프록시 초기화



5. OSIV (Open Session In View) , OEIV (Open EntityManager In View)

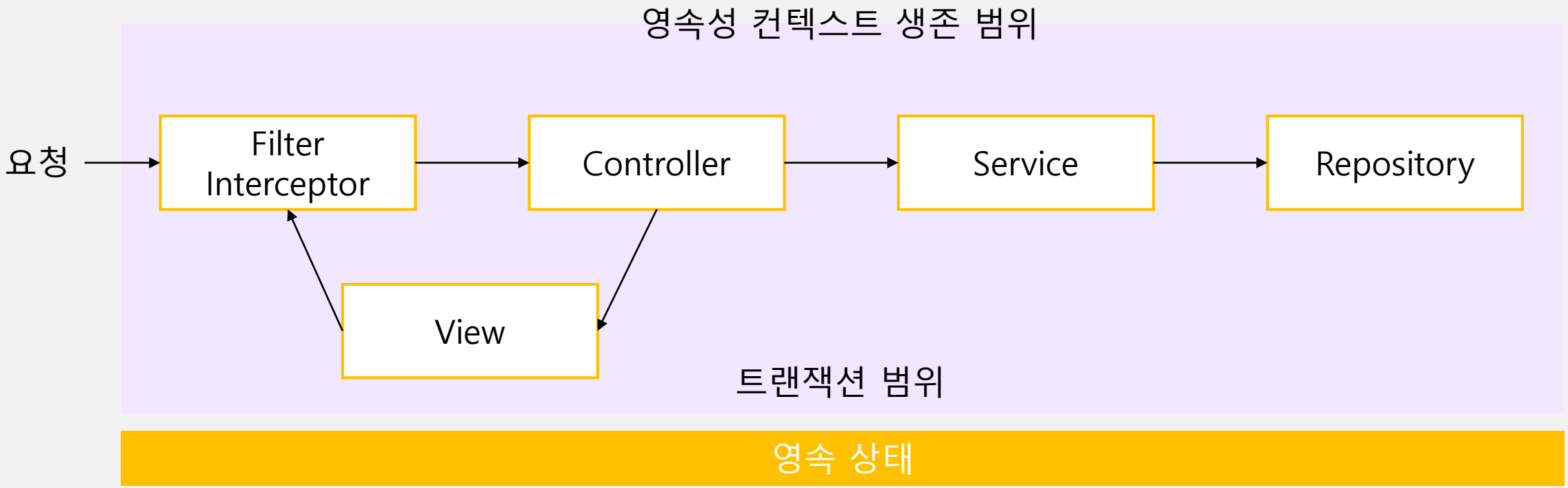
영속성 컨텍스트를 뷰까지 열어둔다.

뷰에서도 지연 로딩을 사용할 수 있다.

```
spring.datasource.url=jdbc:oracle:thin:@developerdb.spectra.co.kr:1521:orcl
spring.datasource.username=shjeon
spring.datasource.password=shjeon
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.jpa.open-in-view=true
```

6. OSIV - 과거

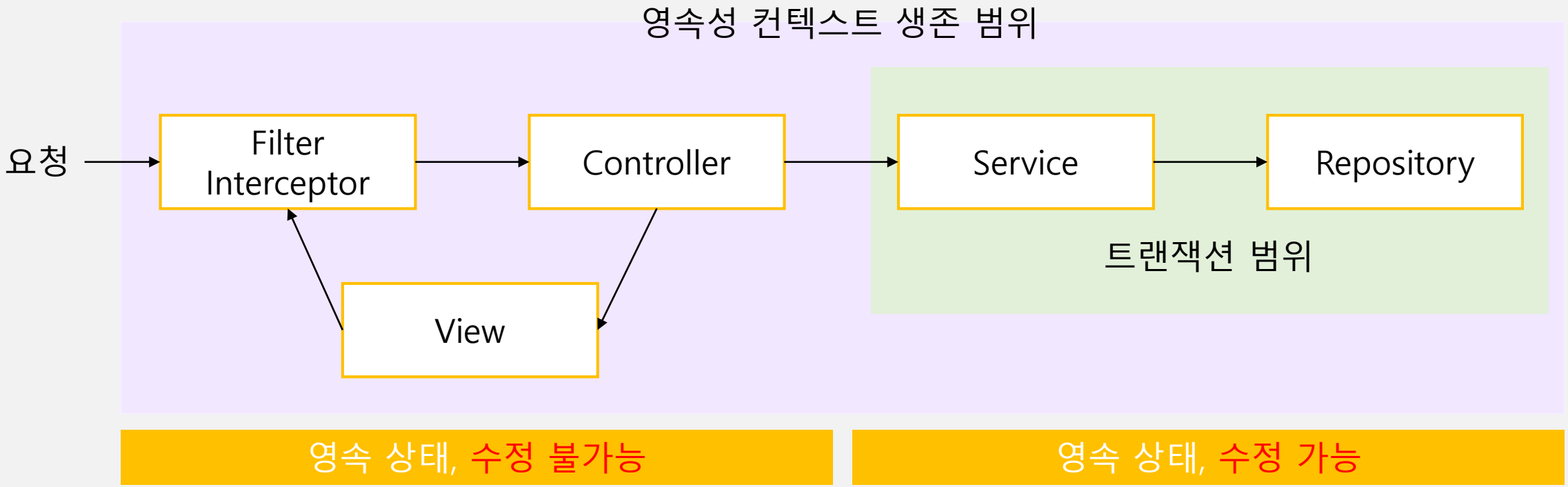
요청 당 트랜잭션 방식의 OSIV



문제점) 프리젠테이션 계층이 엔티티를 변경할 수 있다.

5. 스프링 OSIV

하이버네이트 OSIV 서블릿 필터, OSIV 스프링 인터셉터
JPA OEIV 서블릿 필터, OEIV 스프링 인터셉터



Q&A

1. findOrder(), findOrderWithMember() 처럼 조회하는 데이터에 따라 다른 메소드를 만드는 것이 좋은 방식인가? 아니면 항상 member를 로딩하는 방식이 좋은 방식인가?

⇒ 데이터양에 따라 구현 방법이 달라질 것 같습니다.

2. JPQL 페치조인을 spring data에서는 어떻게 사용해야 하나요? (controller -> service -> repository 구성)

controller

```java
@GetMapping("/fetchTest")
public void fetchTest()
{
    List<Order1> order1 = orderService.fetchTest();

    System.out.println(order1.size());
}
```

service

```java
@Service
@RequiredArgsConstructor
public class OrderService {

    private final OrderRepository orderRepository;
    private final JpaTestRepository jpaTestRepository;

    @Transactional
    public Order1 findOne(String orderId)
    {
        return orderRepository.findOne(orderId);
    }

    @Transactional
    public List<Order1> fetchTest()
    {
        return jpaTestRepository.findOrders();
    }
}
```

repository

```java
@Repository
public interface JpaTestRepository extends JpaRepository<Order1, String> {

    @Query(value="select o from Order1 o join fetch o.member1")
    List<Order1> findOrders();

}
```

3. attic에서는 OSIV를 사용해야 할까요? 아니면 사용하지 않아야 할까요?

=> 사용하지 않는 것이 좋을 것 같습니다.
OSIV 사용 시 프리젠테이션 계층에서 엔티티 수정 후 서비스 호출 시 엔티티가 수정되어 오류를 초래할 수 있고,
유지보수 측면에서도 엔티티 수정과 조회는 서비스 계층에서 호출하는 것이 좋을 것 같습니다.

4. 페치 조인의 단점인 프리젠테이션 계층과 데이터 접근 계층의 의존이 생기면 어떤 점이 문제일까요?

=> 페치 조인을 무분별하게 사용하면 화면에 맞춘 리포지토리 메소드가 증가할 수 있습니다.
예를들면 특정 화면 전용에서만 사용될 수 있는 리포지토리 메소드가 증가하고, 이 메소드는 재사용할 수 없습니다.

5. attic 에서 이런 facade 계층을 둔다면 어디에 위치하는 것이 적합할까요?

=> XXXFlowService에서 Façade 역할을 수행할 수 있을 것 같습니다.

6. OSIV 사용상의 문제를 보완할 수 방법은 어떤 것이 있을까요?

OSIV 사용상의 문제
: 컨트롤러에서 엔티티 변경 후 Transaction service 호출 시 Controller에서 변경한 내용으로 수정된다.

=> 트랜잭션이 있는 비즈니스 로직을 모두 호출하고 나서 엔티티를 변경하자!

```
@GetMapping("/fetchTest2")
public void test()
{
    Optional<Order1> order1 = orderService.test();
    order1.get().setName("test");
    orderService.biz();
}
```