



# Spring/Data.JPA OSIV(Open Session In View)

개요

비활성화된 경우



예시

활성화된 경우


예시

## 개요

---

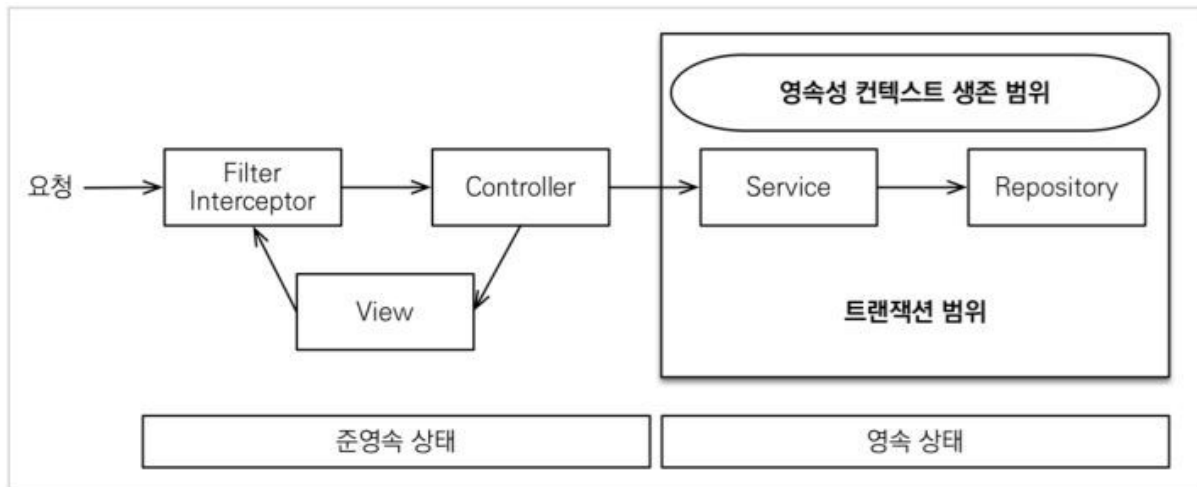
OSIV(Open Session In View) 는  Java(EE)/Persistence 영속성 컨텍스트의 생존 범위를 `Controller`, `View` 레이어까지 확장시켜서  Java(EE)/Persistence 연관관계 페치전략/지연로딩을 뷰 레이어에서 까지 가능하도록 해주는 기능이다.

이 설정은

 Spring/Data.JPA spring.jpa.open-in-view 프로퍼티를 통해 제어할 수 있다.

## 비활성화된 경우

---



**트랜잭션** 이 종료되면 Java(EE)/Persistence 영속성 컨텍스트도 같이 종료되면서 데이터베이스 커넥션도 바로 반환된다. 하지만 그렇기 때문에 Java(EE)/Persistence 연관관계 페치전략/지연로딩을 모두 트랜잭션 안에서 처리해야한다는 단점이 있다.

## 예시

▼ (OSIV를 비활성화하여) **영속성 컨텍스트**의 생존 범위가 아니게 된 **컨트롤러**에서 **지연 로딩**을 시도하는 경우 `org.hibernate.LazyInitializationException` 예외가 발생한다.

```

@Entity
class Member(
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long = 0L,
    var username: String,
    @ManyToOne(fetch = FetchType.LAZY) // 지연 로딩
    var team: Team,
)

@Entity
class Team(
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long = 0L,
    var name: String,
)
  
```

```

@Service
class MemberService(
    private val memberRepository: MemberRepository,
) {

    fun getFirstMember(): Member? {
        val members = memberRepository.findAll()
        return members[0]
    }
}

```

```

@RequestMapping("/api/members")
@RestController
class MemberController(
    private val memberService: MemberService
) {

    @RequestMapping("/first")
    fun getFirstMember(): String {
        val member = memberService.getFirstMember()!!

        // 이 시점에 member.team을 로딩하려고 시도한다.
        return "${member.id} ${member.username} ${member.team.name}"
    }
}

```

이러면 아래와 같이 `org.hibernate.LazyInitializationException` 예외가 발생한다. OSIV가 비활성화된 경우, `영속성 컨텍스트`의 생존 범위는 `서비스` 까지이지만, `컨트롤러`에서 `영속성 컨텍스트`를 통해 팀을 로딩하려고 시도했기 때문이다.

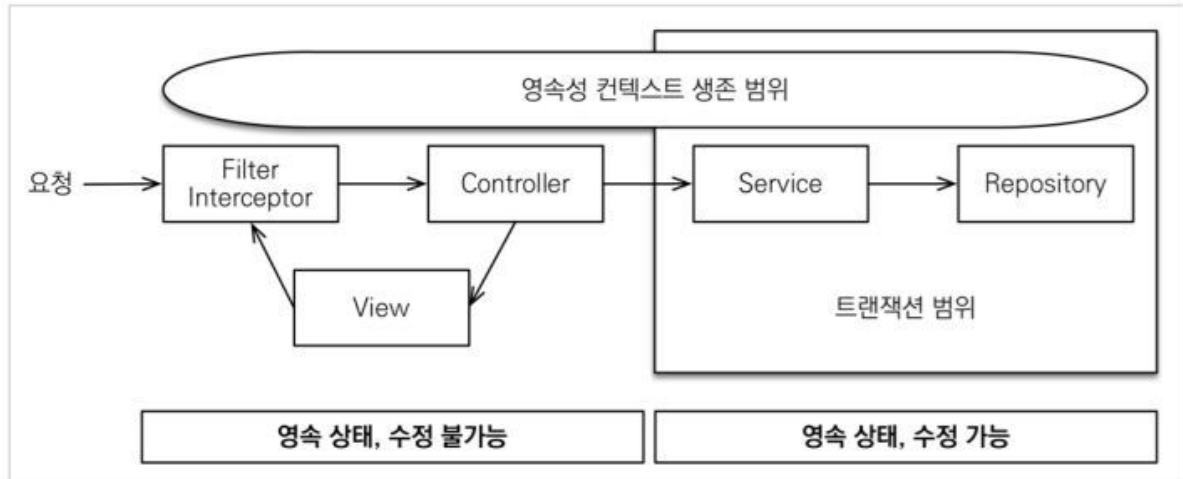
```

org.hibernate.LazyInitializationException: could not initiate lazy load for property 'team'
    at org.hibernate.proxy.AbstractLazyInitializer.initial
    at org.hibernate.proxy.AbstractLazyInitializer.getImpl
    at org.hibernate.proxy.pojo.bytebuddy.ByteBuddyInterce
    at org.hibernate.proxy.ProxyConfiguration$InterceptorD
    at com.sre.emptycanj11.api.team.repository.Team$Hibern

```

```
at java.base/java.lang.String.valueOf(String.java:2951
...
```

## 활성화된 경우



☕ Java(EE)/Persistence 영속성 컨텍스트는 **컨트롤러** 에서 최종적으로 Response를 내보낼 때까지 유지된다. 이는 ☕ Java(EE)/Persistence 연관관계 페치전략/지연로딩이 **컨트롤러** 단까지 가능하다는 점과 데이터베이스 커넥션을 획득~종료하는데까지 유지해야하는 시간이 길어짐을 의미하게 된다.

하지만 트랜잭션 범위는 **Service**, **Repository** 범위로 고정되어있으므로 **Controller**, **View** 레이어에서는 여전히 **엔티티** 를 수정할 수 없다. 즉, ☕ Java(EE)/Persistence 영속성 컨텍스트 더티 체크킹(Dirty Checking)이 동작하지 않는다. **더티 체크킹** 은 트랜잭션 내에서 발생한 변경사항에 대해서만 동작하기 때문이다.

## 예시