와플스튜디오 Spring Seminar

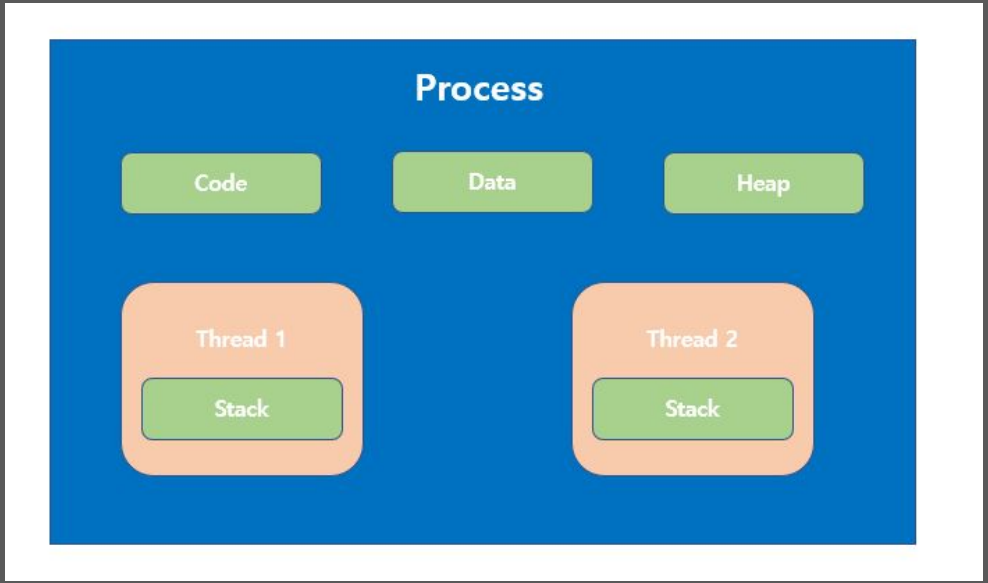
세미나장: 정원식

2023.10.25.(수) 19:00

Week3

Table of Contents

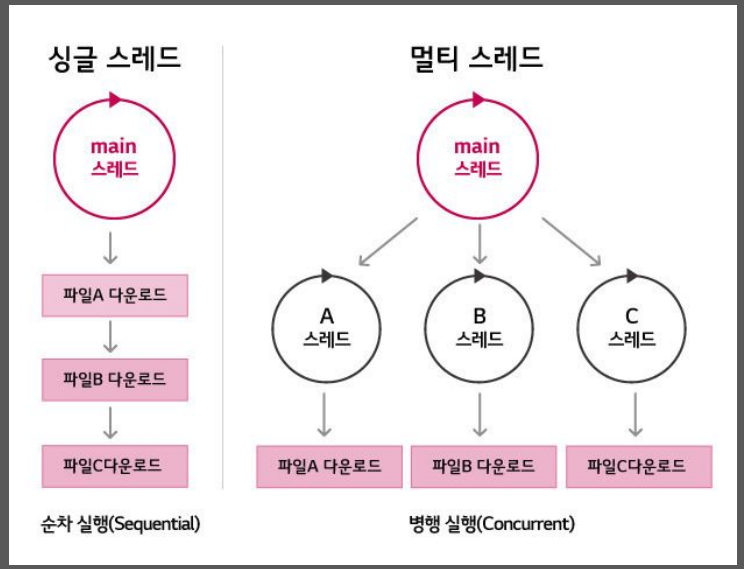
스레드



- 쓰레드(Thread)란 프로세스 내에서 실행되는 흐름의 단위 혹은 CPU 스케줄링의 기본 단위
- 프로세스 내에서 Code, Data, Heap 영역을 공유한다.

스레드

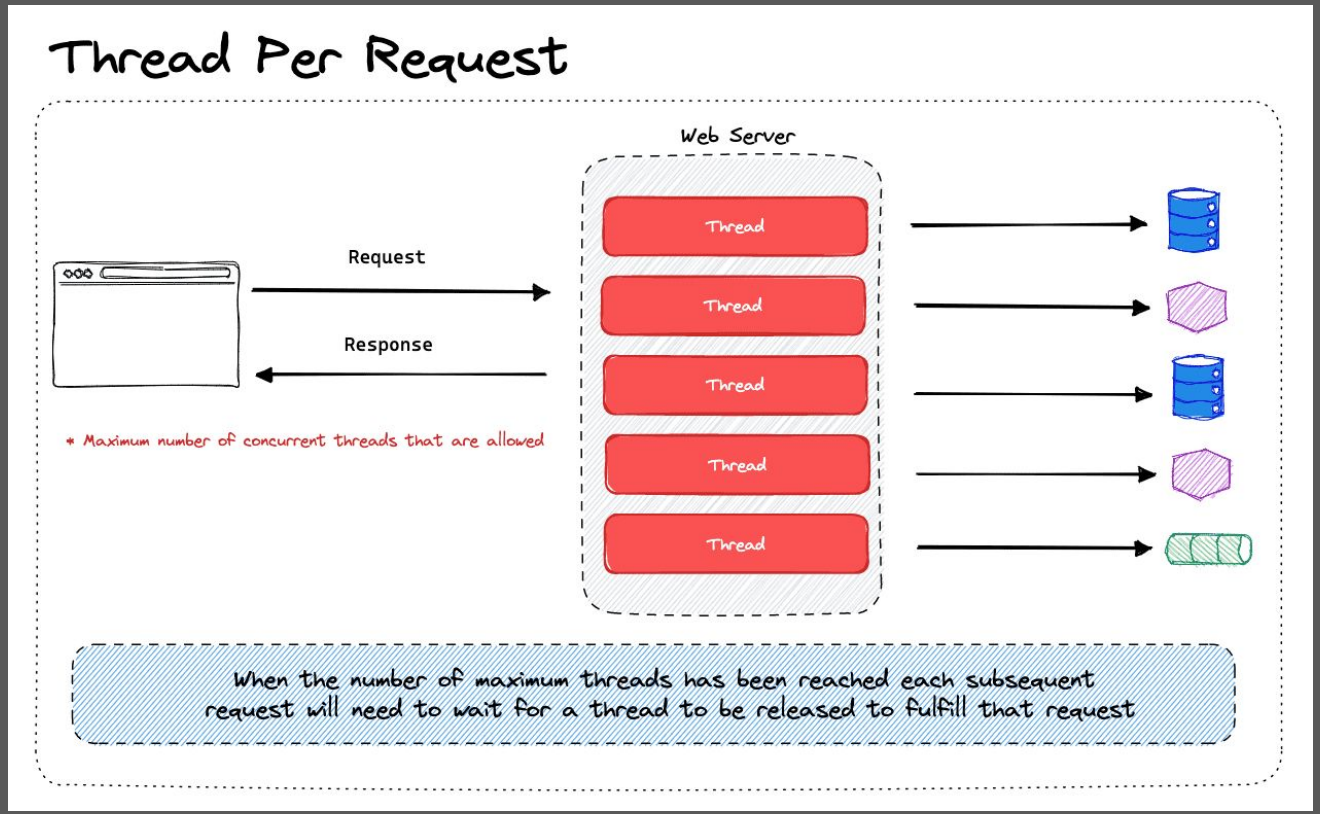
멀티 스레딩



- 하나의 프로세스를 다수의 실행 단위로 구분하여 자원을 공유하고, 자원의 생성과 관리의 중복성을 최소화하여 수행 능력을 향상시키는 것을 멀티쓰레딩이라고 한다.
- 하나의 프로그램에 동시에 여러개의 일을 수행할수 있도록 해주는 것이다.
- 시스템 자원 소모가 감소. 프로세스를 생성하는 system call이 줄어들기 때문

스레드

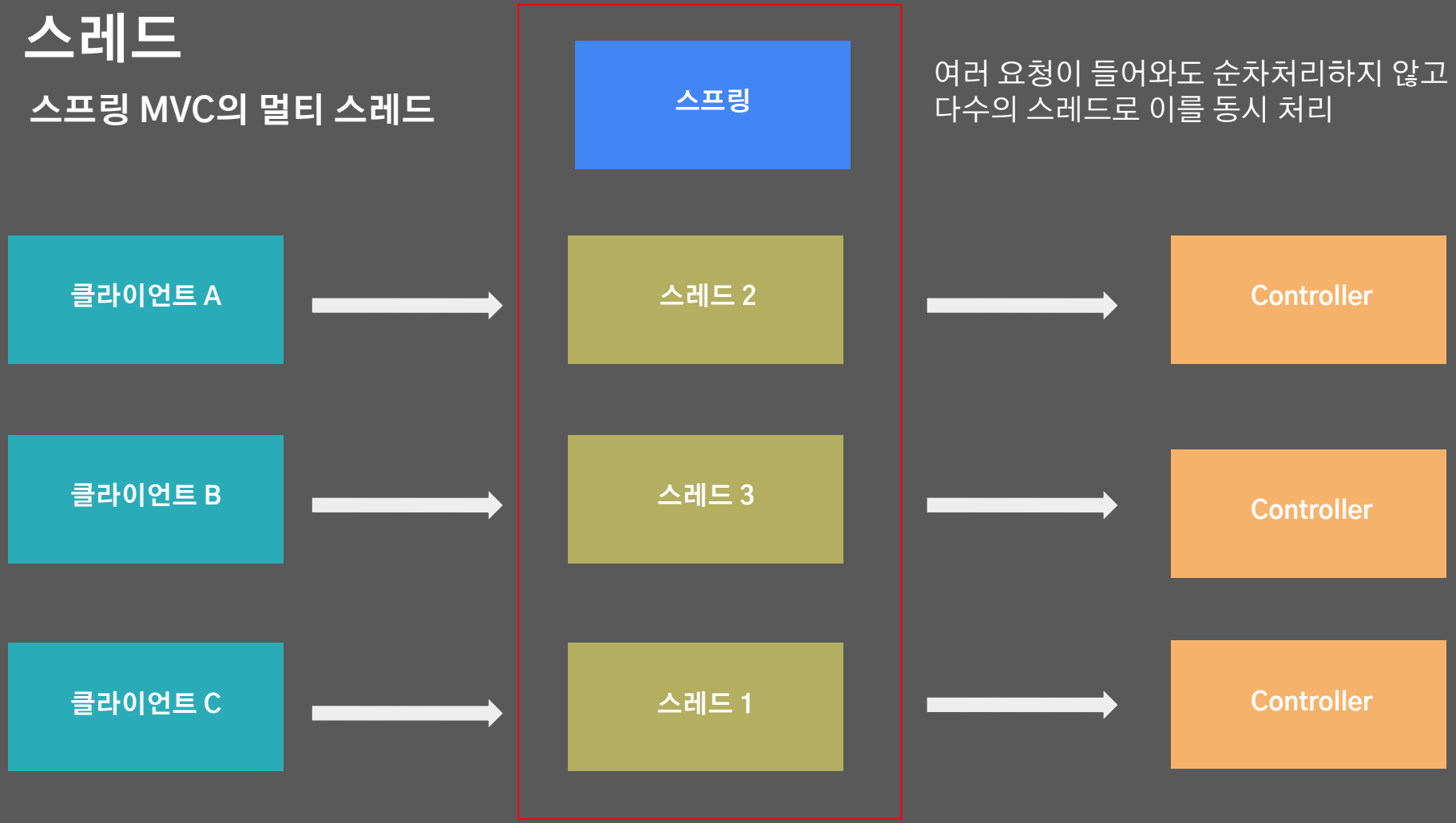
스프링 MVC의 스레드 모델



한 개의 스레드가 한 개의 클라이언트 요청을 처리한다.

스레드

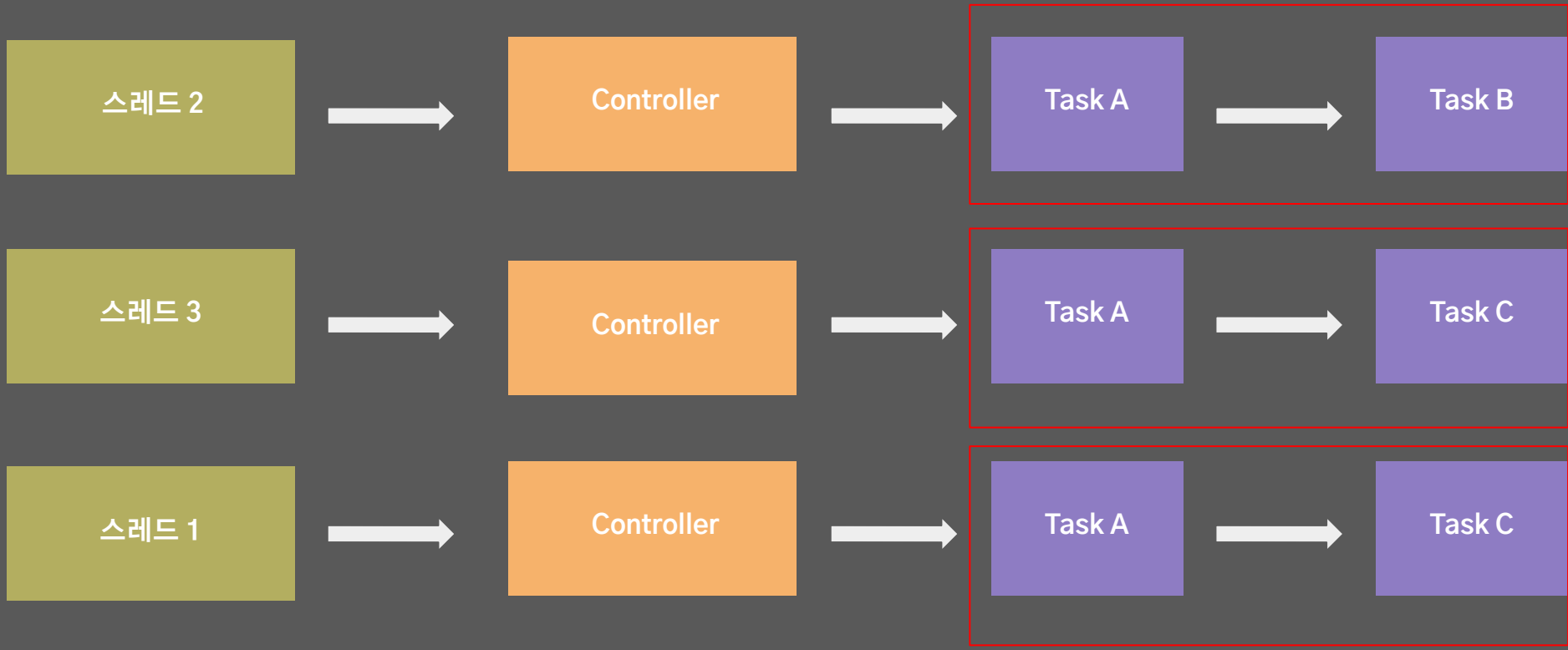
스프링 MVC의 멀티 스레드



여러 요청이 들어와도 순차처리하지 않고
다수의 스레드로 이를 동시 처리

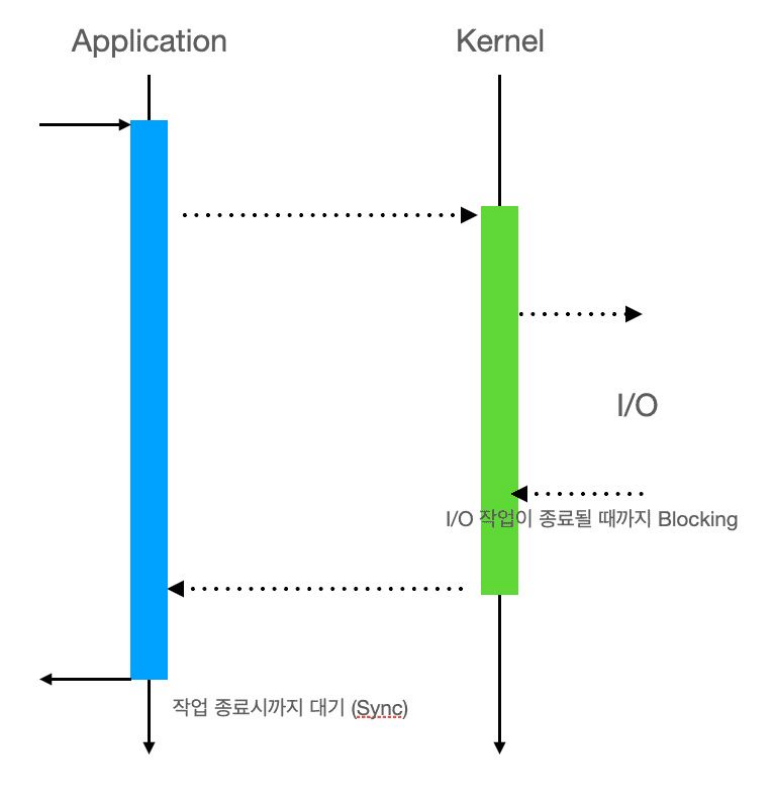
스레드

요청은 동시 처리하나 각 요청에 대한 서브 태스크는 순차 처리



스레드

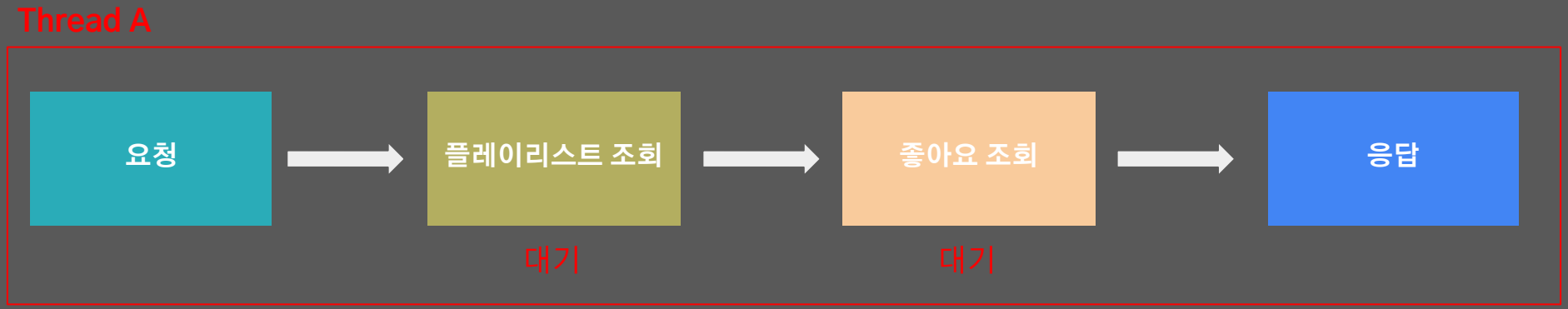
블로킹 IO



IO 작업이 완료될 때까지 스레드는 대기

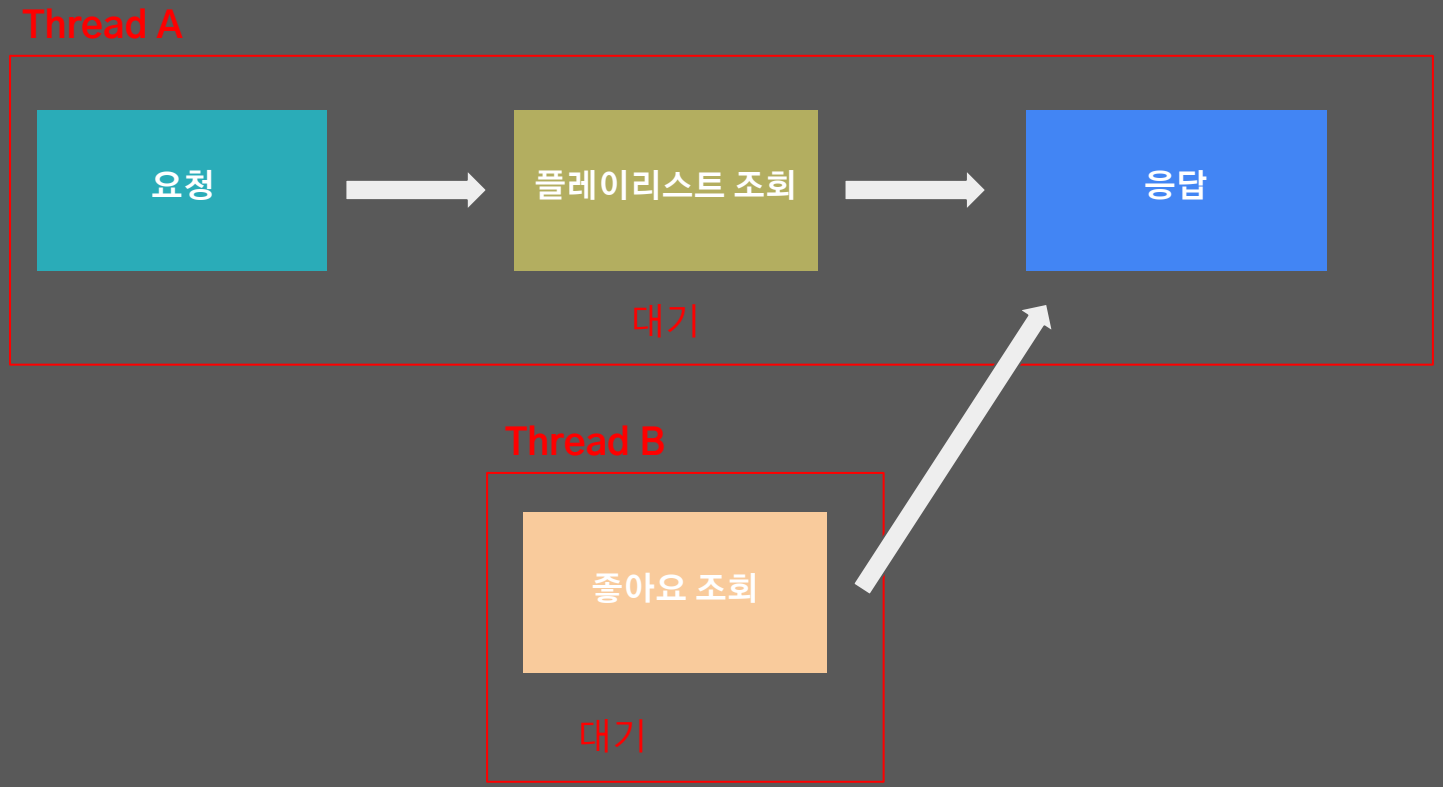
스레드

플레이리스트 조회 API 싱글 스레드 처리



스레드

플레이리스트 조회 API 멀티 스레드 처리



스레드

플레이리스트 조회 API 멀티 스레드 처리 예제 코드

```
private val threads = Executors.newFixedThreadPool( nThreads: 4)

@GetMapping("/api/v2/playlists/{id}")
fun getPlaylistV2(
    @PathVariable id: Long,
    user: User?,
): PlaylistResponse {
    val liked :Future<Boolean!> = threads.submit<Boolean> {
        if (user == null) {
            false ^submit
        } else {
            playlistViewService.create(playlistId = id, userId = user.id)
            playlistLikeService.exists(playlistId = id, userId = user.id) ^submi
        }
    }

    val playlist :Playlist = playlistService.get(id)

    return PlaylistResponse(playlist, liked.get())
}
```

서브 태스크를 병렬로 처리할 스레드 그룹을 선언

좋아요 여부를 현재 스레드가 아닌 스레드 그룹의 멤버 스레드에서 조회

플레이리스트는 현재 스레드에서 조회

스레드

싱글 스레드-멀티스레드 성능 비교

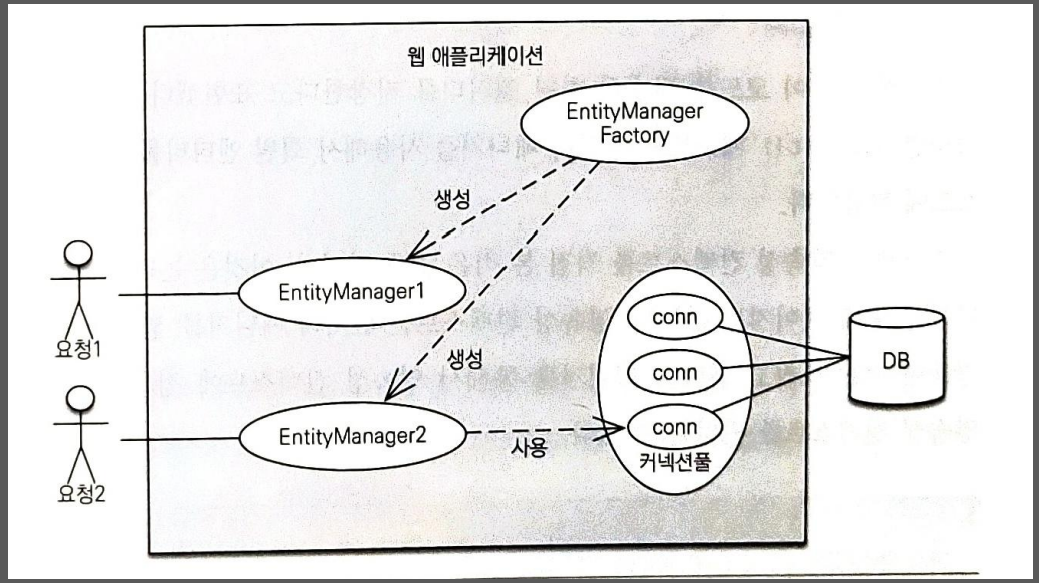
```
@Test
fun `플레이리스트 조회 싱글스레드-멀티스레드 비교`() {
    var start :Long = System.currentTimeMillis()

    for (i :Int in 0 ≤ until < 1000) {
        mvc.perform(
            MockMvcRequestBuilders
                .get( urlTemplate: "http://localhost:8080/api/v1/playlists/1") // v2는 플레이리스트 조회와 좋아요 조회를 순차 처리
                .header( name: "Authorization", ...values: "Bearer gnirps")
        )
    }

    val durationV1 :Long = System.currentTimeMillis() - start

    start = System.currentTimeMillis()

    for (i :Int in 0 ≤ until < 1000) {
        mvc.perform(
            MockMvcRequestBuilders
                .get( urlTemplate: "http://localhost:8080/api/v2/playlists/1") // v2는 플레이리스트 조회와 좋아요 조회를 병렬 처리
                .header( name: "Authorization", ...values: "Bearer gnirps")
        )
    }

    val durationV2 :Long = System.currentTimeMillis() - start

    println("v1: $durationV1, v2: $durationV2")
}
```

응답 속도가 2배 가까이 차이남

```
v1: 6914, v2: 3419
> Task :test
2023-10-24T21:17:36.0
```

스레드

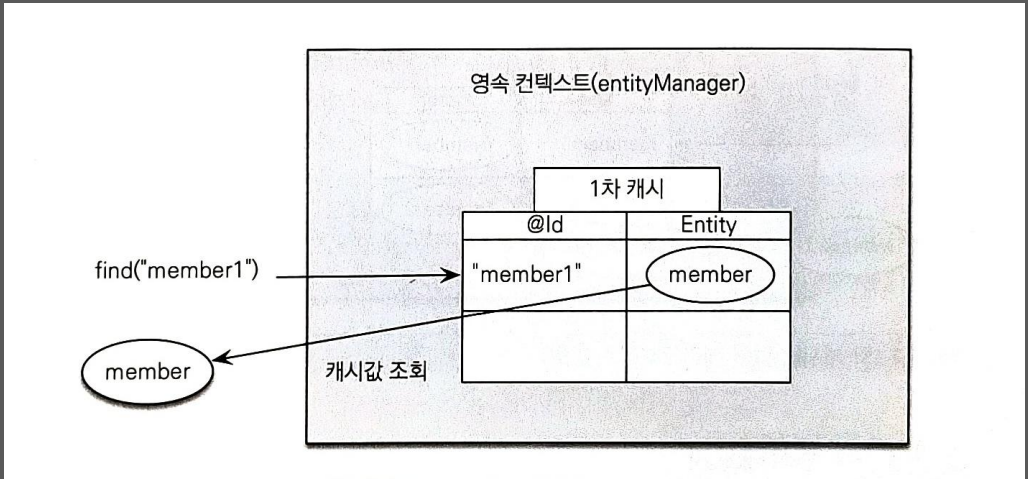
Spring JPA의 스레드 모델



- 한 개의 요청 – 한 개의 스레드
- 한 개의 스레드 – 한 개의 엔티티매니저 (영속성 컨텍스트)
- 한 개의 스레드 – 한 개의 영속성 컨텍스트

스레드

영속성 컨텍스트 복습



- 엔티티 캐싱
- 쓰기 지연
- 지연 로딩
- 변경 감지

현재까지 우리는 엔티티 캐싱, 지연 로딩을 사용했는데 모두 영속성 컨텍스트가 가능하게 해준 것.

스레드

JPA 기능은 멀티 스레드 환경에서 어떻게 동작할까?

```
private val threads = Executors.newFixedThreadPool( nThreads: 1)

@Transactional
@Test
fun `다른 스레드에서 조회한 엔티티를 가지고, 현재 스레드에서 영속성 컨텍스트 관련 기능을 사용할 수 없다`() {
    val songFromCurrentThread : SongEntity = songRepository.findById( id: 1L).get()

    assertDoesNotThrow {
        println(songFromCurrentThread.album.title) // 영속성 컨텍스트를 통해 album을 lazy load
    }

    val songFromTheOtherThread : SongEntity! = threads.submit<SongEntity> { songRepository.findById( id: 1L).get() }.get()

    assertThrows<LazyInitializationException> {
        println(songFromTheOtherThread.album.title) // 영속성 컨텍스트를 통해 album을 lazy load하려 하지만 다른 스레드(영속성 컨텍스트)에서 조회한 것이기 때문에 에러 발생
    }
}
```

다른 스레드에서 조회한 엔티티를 가지고 영속성 컨텍스트의 기능을 사용할 수 없다.
영속성 컨텍스트는 스레드마다 개별적으로 존재하기 때문.

스레드

ThreadLocal

```
@Component
class QueryCounter : StatementInspector {
    data class Result<K>(
        val value: K,
        val queryCount: Int,
    )
                                Week1에서 다룬 ThreadLocal
    private val isCounting: ThreadLocal<Boolean> = ThreadLocal.withInitial { false }
    private val queryCount: ThreadLocal<Int> = ThreadLocal.withInitial { 0 }
```

같은 객체이지만 그 값은 각 스레드가 개별적으로 갖는다.

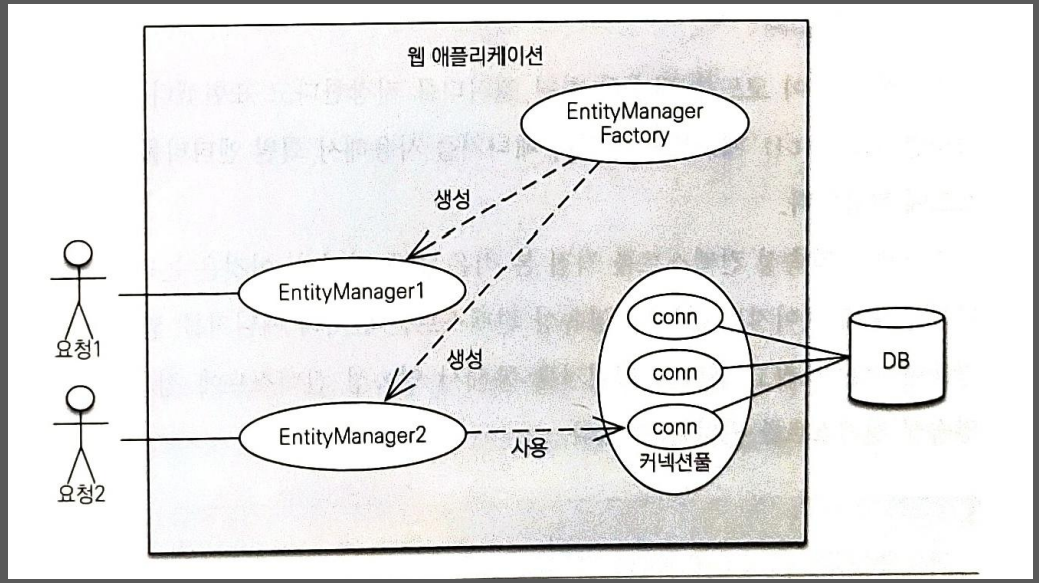
스레드

ThreadLocal 테스트

```
@Test
fun `ThreadLocal은 쓰레드별로 서로 다른 값을 본다 - 기본`() {
    val threadLocalNum : ThreadLocal<Int!>! = ThreadLocal.withInitial { 1 } // 초기값 1

    assertThat(threadLocalNum.get()).isEqualTo(1)

    threadLocalNum.set(2) // 현재 스레드에서 2로 변경

    assertThat(threadLocalNum.get()).isEqualTo(2) // 2로 변경된 값 확인

    threads.submit {
        assertThat(threadLocalNum.get()).isEqualTo(1) // 다른 스레드에서 조회했기 때문에 초기값 1

        threadLocalNum.set(3) // 다른 스레드에서 3으로 변경

        assertThat(threadLocalNum.get()).isEqualTo(3) // 3으로 변경된 값 확인
    }
        .get()

    assertThat(threadLocalNum.get()).isEqualTo(2) // 다른 스레드에서 3으로 변경했기 때문에 현재 스레드에서는 여전히 2
}
```

스레드

영속성 컨텍스트도 하나의 ThreadLocal과 같다.



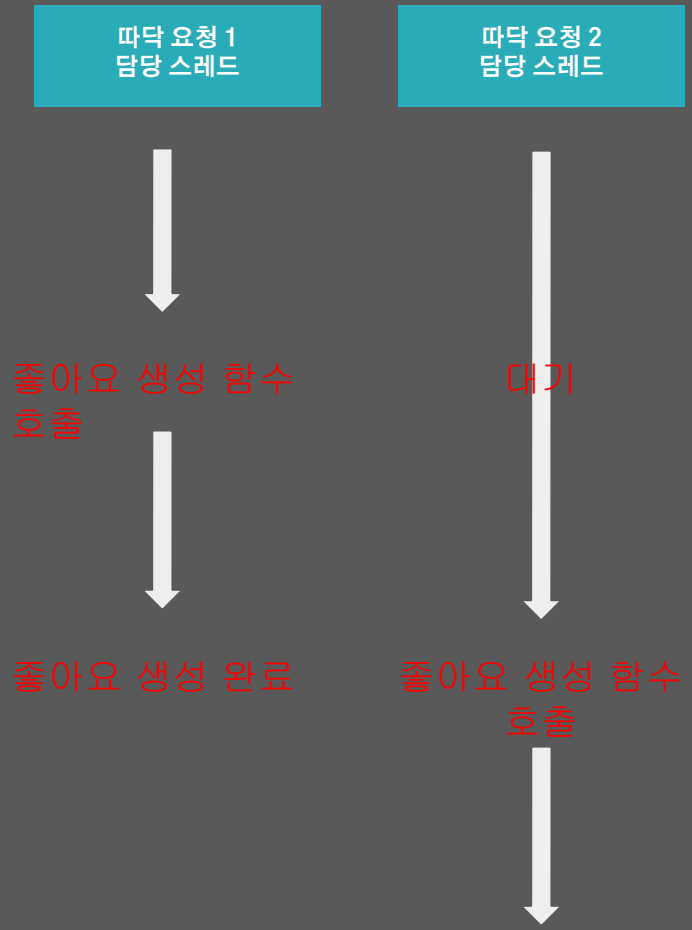
- 한 개의 요청 – 한 개의 스레드
- 한 개의 스레드 – 한 개의 엔티티매니저 (영속성 컨텍스트)
- 한 개의 스레드 – 한 개의 영속성 컨텍스트

스레드

‘동기화’를 통한 따닥 이슈 해결

```
@Synchronized
override fun createSynchronized(playlistId: Long, userId: Long) {
    create(playlistId, userId)
}
```

- 동기화: 다수의 스레드가 하나의 공유 데이터 혹은 코드 블록에 동시에 접근하지 못하도록 막는 것.



스레드

Synchronized는 여러 서버 인스턴스간 공유가 불가능

스레드

DB Unique Key

```
create table playlist_likes (
    id bigint auto_increment,
    playlist_id bigint not null,
    user_id bigint not null,
    primary key (id),
    unique (playlist_id, user_id)
);
```

중복 데이터 생성 방지를 어플리케이션이 아닌 DB에 위임

```
Caused by: org.h2.jdbc.JdbcSQLIntegrityConstraintViolationException: Unique index or primary key violation: "PUBLIC.CONSTRAINT_INDEX_9 ON
 PUBLIC.PLAYLIST_LIKES(PLAYLIST_ID NULLS FIRST, USER_ID NULLS FIRST) VALUES ( /* key:1 */ CAST(1 AS BIGINT), CAST(1 AS BIGINT))"; SQL statement:
insert into playlist_likes (playlist_id,user_id,id) values (?,?,default) [23505-214]
```

데이터베이스 락

유니크 키를 적용할 수 없는 경우에는?

```
create table playlists (
    id bigint auto_increment,
    title varchar(255),
    subtitle varchar(255),
    image varchar(200),
    group_id bigint,
    view_cnt bigint default 0,
    primary key (id)
);
```

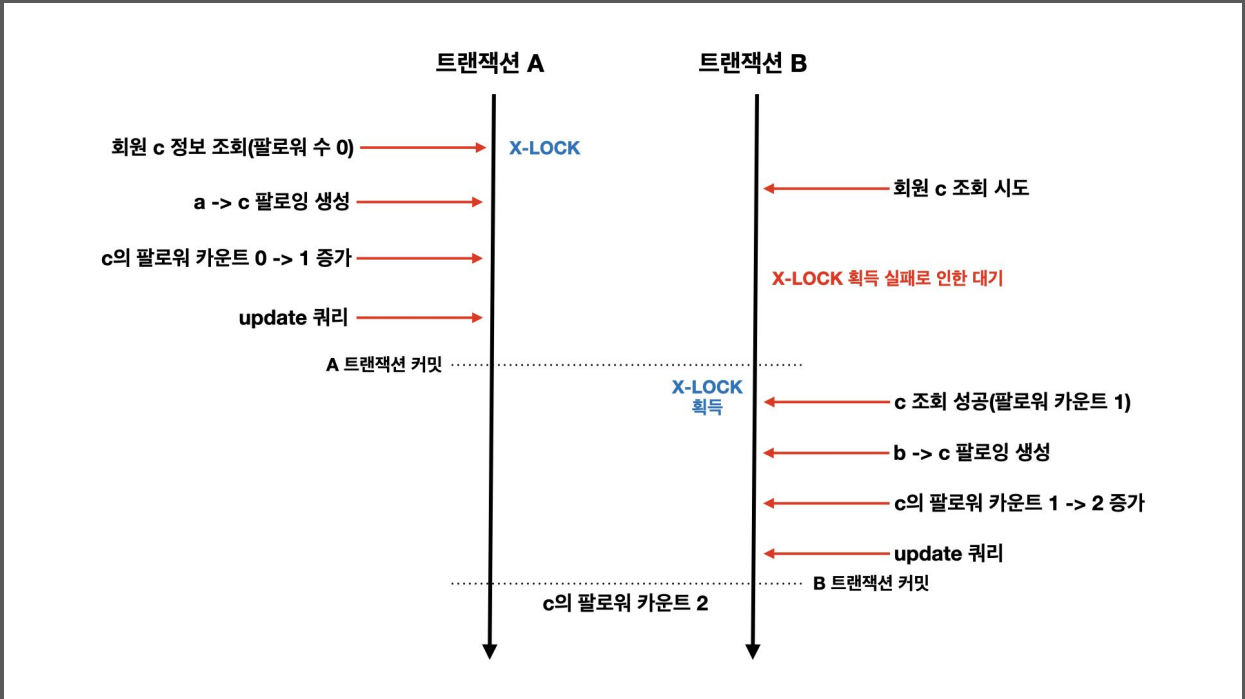
플레이리스트를 조회할 때마다 view_cnt 칼럼의 값을 +1 해줘야 한다.

이 때 동시에 요청이 2번 들어오면 들어오면, view_cnt의 값은 +2가 아닌 +1이 된다.

그런데 view_cnt는 unique key를 적용할 수 없음.

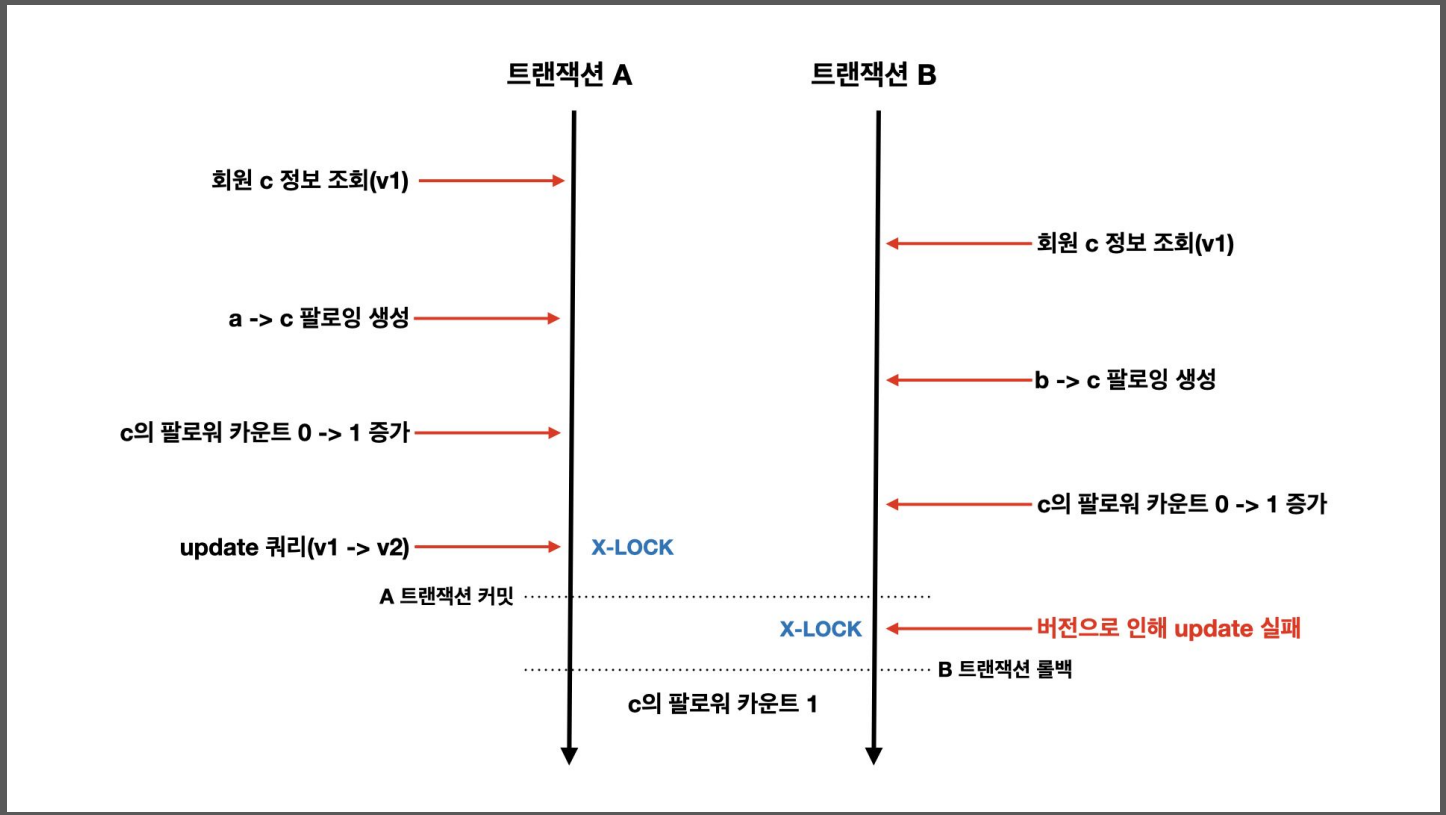
데이터베이스 락

비관적 락



데이터베이스 락

낙관적 락



데이터베이스 락

update `table` set `column` = `column` + 1

```
@Modifying(clearAutomatically = true, flushAutomatically = true)
@Query(value = "update Member m set m.followerCount = m.followerCount + 1 where m.id = :fo
void increaseFollowerCount(Long followingMemberId);
```

JPA

변경 감지

```
@SpringBootTest
class DirtyCheckTest @Autowired constructor(
    private val playlistRepository: PlaylistRepository,
    txManager: PlatformTransactionManager,
){
    private val txTemplate = TransactionTemplate(txManager)

    @Test
    fun `JPA 변경 감지`() {
        txTemplate.execute {   @Transactional을 코드로 푼 것
            val playlist : PlaylistEntity  = playlistRepository.findById( id: 1L).get()

            playlist.viewCnt++ ^execute
        }
    }
}
```

트랜잭션 커밋시 변경된 값을 체크하여 update 쿼리를 날린다.

```
Hibernate: select p1_0.id,p1_0.group_id,p1_0.image,p1_0.subtitle,p1_0.title,p1_0.view_cnt from playlists p1_0 where p1_0.id=?
Hibernate: update playlists set group_id=?,image=?,subtitle=?,title=?,view_cnt=? where id=?
```

Q & A