

동시성 제어



INDEX

목차

1. 동시성 제어의 종류
2. 사례 : 하나의 축제에 두 번의 알람



01

동시성 제어의 종류



동시성 제어

여러 작업이 동시에 동일한 자원에 접근할 때 발생하는
충돌이나 불일치를 방지하기 위한 제어 기법





```
1  // 1번
2  public synchronized void increment() {
3      count++;
4  }
5
6  // 2번
7  private final Object lock = new Object();
8
9  public void increment() {
10     synchronized (lock) {
11         count++;
12     }
13 }
14
15 // 3번
16 @Synchronized
17 public void increment() {
18     count++;
19 }
```



Synchronized

- JVM 수준 락
- 하나의 스레드만 해당 코드 블록 실행

@Synchronized

- Lombok 라이브러리
- 현재 인스턴스가 아닌 private Lock 사용



```
1 private final ReentrantLock lock = new ReentrantLock();
2
3 public void increment() {
4     lock.lock();
5     try {
6         count++;
7     } finally {
8         lock.unlock();
9     }
10 }
```



```
1 @Service
2 public class CouponService {
3
4     @Lockable(key = "#couponId")
5     public void useCoupon(Long couponId) {
6         ...
7     }
8 }
```



ReentrantLock

- Lock 인터페이스 구현체
- Synchronized 보다 세밀한 제어 가능
- 타임아웃, 인터럽트 제어 가능
- 대기 큐로 정교한 제어 가능

AOP 활용 Lock

- Lock 잠금 과정의 중복되는 로직(공통 관심사) 분리



```
1 @Transactional(isolation = Isolation.REPEATABLE_READ)
2 public void checkBalanceTwice(Long accountId) {...}
```



```
1 CREATE TABLE member_coupon (
2     id BIGINT AUTO_INCREMENT PRIMARY KEY,
3     member_id BIGINT NOT NULL,
4     coupon_id BIGINT NOT NULL,
5     UNIQUE (member_id, coupon_id)
6 );
```



트랜잭션 격리 수준

- 하나의 트랜잭션이 다른 트랜잭션의 중간 작업에 접근 정도 제한

UNIQUE 제약 조건

- 동일 값이 중복 삽입되는 것을 원천적 차단 제약 설정



```
1  // 비관적 락
2  @Lock(LockModeType.PESSIMISTIC_WRITE)
3  Optional<Product> findById(Long id);
4
5  // 낙관적 락
6  @Entity
7  public class Product {
8
9      @Id
10     private Long id;
11
12     private int stock;
13
14     @Version
15     private Long version;
16 }
```



비관적 락

- 데이터 SELECT 순간 접근 제한
- 다른 트랜잭션이 읽거나 쓰기 불가
- 배타 락(Exclusive Lock), 공유 락(Shared Lock)

낙관적 락

- 커밋 시점 충돌 감지
- 재시도 로직 설정 가능



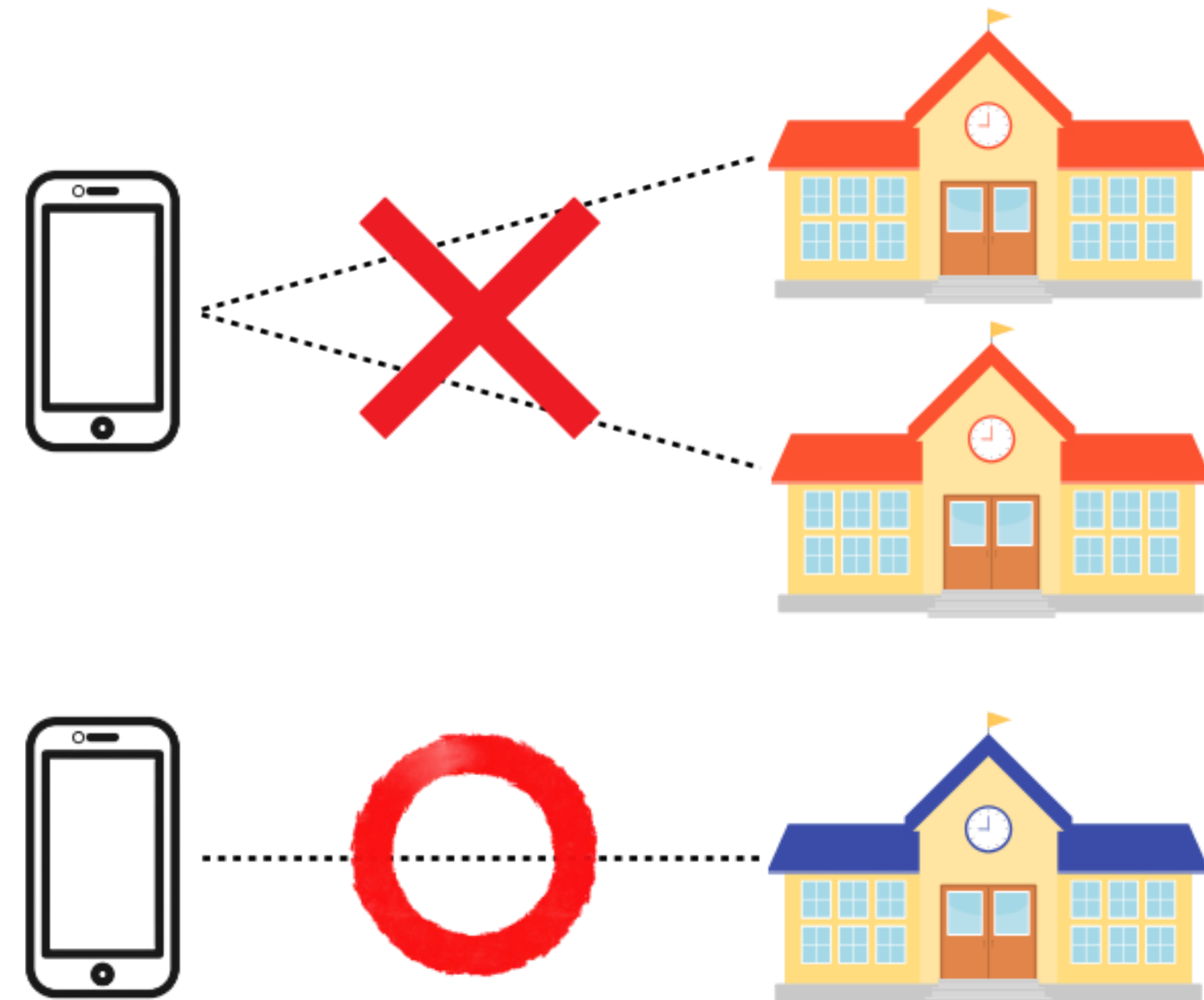
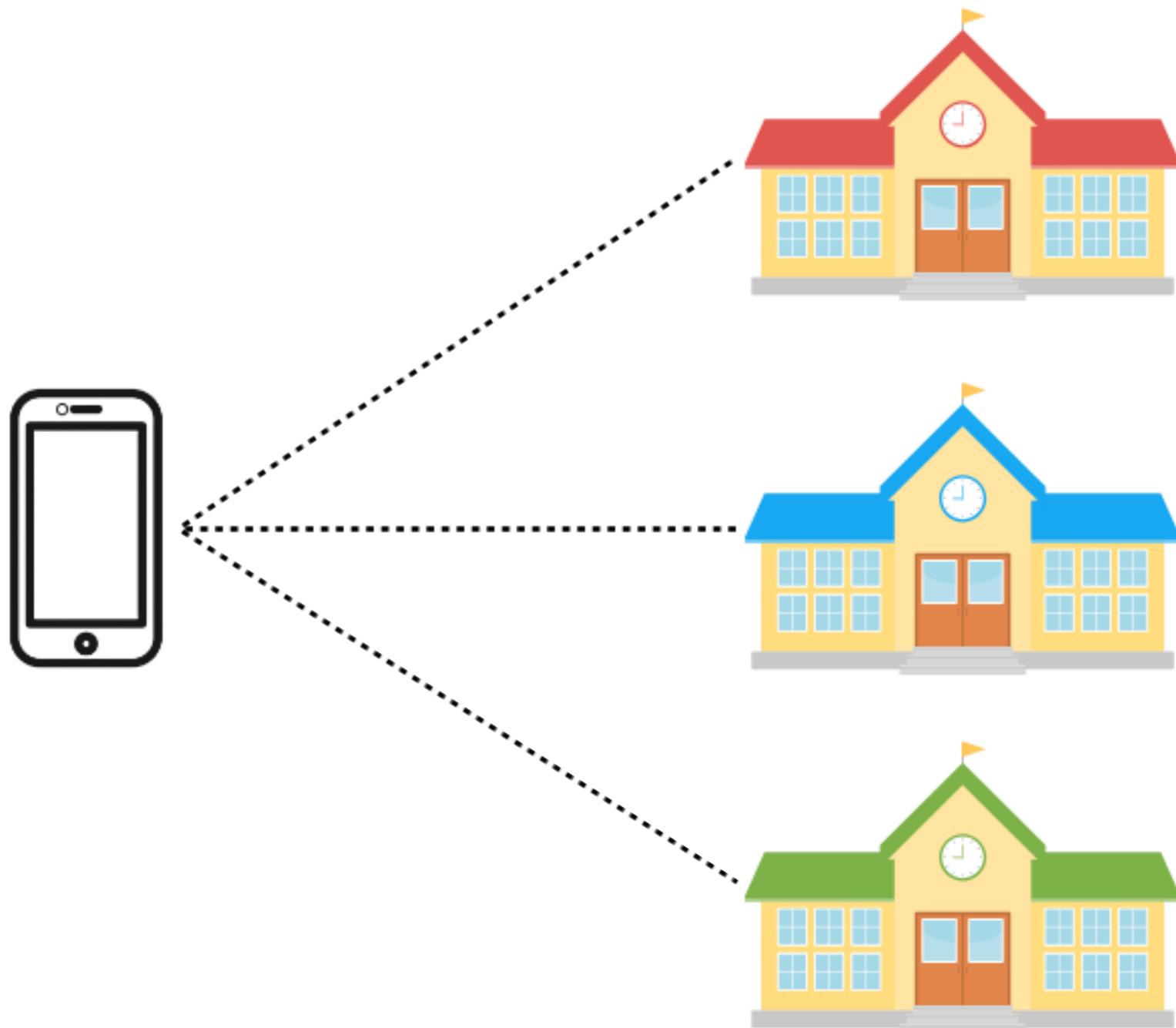
02

사례 : 하나의 축제에 두 번의 알람



비즈니스 조건

1. 하나의 기기는 여러 축제의 알람을 등록할 수 있다.
2. 하나의 축제에 하나의 알림만 등록할 수 있다.



알림

현재 접속 중인 대학
가천대학교 글로벌 캠퍼스



```
1 @Transactional
2 public FestivalNotificationResponse subscribeAndroidFestivalNotification(...) {
3     validateDuplicatedFestivalNotification(festivalId, request.deviceId());
4     ...
5     FestivalNotification saved = festivalNotificationJpaRepository.save(festivalNotification);
6     return FestivalNotificationResponse.from(saved);
7 }
```




device_id	festival_id	deleted	count
22	1	0x00	2
24	3	0x00	3

애플리케이션 VS DB 영역

제어 영역	제어 방식
애플리케이션	Synchronized
	ReentrantLock
DB	트랜잭션 격리 수준
	UNIQUE 제약 조건
	비관적 락
	낙관적 락



```
1  @Test
2  void 동시성_100개_요청시_중복_구독_방지된다() {
3      // given
4      Festival festival = FestivalFixture.create();
5      festivalJpaRepository.save(festival);
6
7      Device device = DeviceFixture.create();
8      deviceJpaRepository.save(device);
9
10     FestivalNotificationRequest request = FestivalNotificationRequestFixture.create(device.getId());
11
12     int requestCount = 100;
13     Runnable httpRequest = () -> {
14         RestAssured
15             .given()
16             .contentType(ContentType.JSON)
17             .body(request)
18             .when()
19             .post("test/festivals/{festivalId}/notifications", festival.getId());
20     };
21
22     int expectedNotificationCount = 1;
23
24     // when
25     ConcurrencyTestHelper.test(requestCount, httpRequest);
26
27     // then
28     Long result = festivalNotificationJpaRepository.countByFestivalIdAndDeviceId(festival.getId(),
29         device.getId());
30     assertThat(result).isEqualTo(expectedNotificationCount);
31 }
```

필요: 1L
실제 : 10L

Synchronized

```
1 @Transactional
2 public synchronized FestivalNotificationResponse subscribeAndroidFestivalNotification(...) {
3     validateDuplicatedFestivalNotification(festivalId, request.deviceId());
4     ...
5     FestivalNotification saved = festivalNotificationJpaRepository.save(festivalNotification);
6     return FestivalNotificationResponse.from(saved);
7 }
```

```
1 @Transactional
2 @Synchronized
3 public FestivalNotificationResponse subscribeAndroidFestivalNotification(...) {
4     validateDuplicatedFestivalNotification(festivalId, request.deviceId());
5     ...
6     FestivalNotification saved = festivalNotificationJpaRepository.save(festivalNotification);
7     return FestivalNotificationResponse.from(saved);
8 }
```

Synchronized

```
1 @Transactional
2 public synchronized FestivalNotificationResponse subscribeAndroidFestivalNotification(...) {
3     validateDuplicatedFestivalNotification(request.requestId, request.deviceId());
4     ...
5     FestivalNotification saved = festivalNotificationJpaRepository.save(festivalNotification);
6     return FestivalNotificationResponse.from(saved);
7 }
```

```
1 @Transactional
2 @Synchronized
3 public FestivalNotificationResponse subscribeAndroidFestivalNotification(...) {
4     validateDuplicatedFestivalNotification(request.requestId, request.deviceId());
5     ...
6     FestivalNotification saved = festivalNotificationJpaRepository.save(festivalNotification);
7     return FestivalNotificationResponse.from(saved);
8 }
```

저장 시점 오차 발생

필요: 1L

실제 : 2L

ReentrantLock

```
1  @PostMapping
2  @ResponseStatus(HttpStatus.CREATED)
3  public FestivalNotificationResponse subscribeFestivalNotification(
4      @PathVariable Long festivalId,
5      @RequestBody FestivalNotificationRequest request
6  ) {
7      lock.lock();
8      try {
9          return testFestivalNotificationService.subscribeFestivalNotification(festivalId, request);
10     } finally {
11         lock.unlock();
12     }
13 }
```


ReentrantLock



```
1  @PostMapping
2  @ResponseStatus(HttpStatus.CREATED)
3  public FestivalNotificationResponse subscribeFestivalNotification(
4      @PathVariable Long festivalId,
5      @RequestBody FestivalNotificationRequest request
6  ) {
7      lock.lock();
8      try {
9          return testFestivalNotificationService.subscribeFestivalNotification(festivalId, request);
10     } finally {
11         lock.unlock();
12     }
13 }
```

ReentrantLock

```
1 @PostMapping
2 @ResponseStatus(HttpStatus.CREATED)
3 public FestivalNotificationResponse subscribeFestivalNotification(
4     @PathVariable Long festivalId,
5     @RequestBody FestivalNotificationRequest request
6 ) {
7     lock.lock();
8     try {
9         return testFestivalNotificationService.subscribeFestivalNotification(festivalId, request);
10    } finally {
11        lock.unlock();
12    }
13 }
```

1. 성능 병목
2. 관심사 위배

AOP Lock (1), (2) 단점 해결 가능
단, 진입점 개발자 실수 가능성 존재

중복 알림 구독 검증 비즈니스 로직

```
1 private void validateDuplicatedFestivalNotification(Long festivalId, Long deviceId) {  
2     if (festivalNotificationJpaRepository.getExistsFlagByFestivalIdAndDeviceId(festivalId, deviceId) > 0) {  
3         throw new BusinessException("이미 알림을 구독한 축제입니다.", HttpStatus.BAD_REQUEST);  
4     }  
5 }
```


트랜잭션 격리 수준

팬텀 리드(Phantom Read) 방지 위한 SERIALIZABLE 수준 적용

```
1 @Transactional(isolation = Isolation.SERIALIZABLE)
2 public FestivalNotificationResponse subscribeAndroidFestivalNotification(...) {
3     validateDuplicatedFestivalNotification(festivalId, request.deviceId());
4     ...
5     FestivalNotification saved = festivalNotificationJpaRepository.save(festivalNotification);
6     return FestivalNotificationResponse.from(saved);
7 }
```


트랜잭션 격리 수준

트랜잭션 범위내 조회된 모든 행 잠금, 성능 저하 발생




```
1 @Transactional(isolation = Isolation.SERIALIZABLE)
2 public FestivalNotificationResponse subscribeAndroidFestivalNotification(...) {
3     validateDuplicatedFestivalNotification(festivalId, request.deviceId());
4     ...
5     FestivalNotification saved = festivalNotificationJpaRepository.save(festivalNotification);
6     return FestivalNotificationResponse.from(saved);
7 }
```


비관적 락, 공유 락(Shared Lock)

```
1  @Query(value = ""  
2      SELECT EXISTS(  
3          SELECT 1  
4          FROM festival_notification fn  
5          WHERE fn.festival_id = :festivalId  
6          AND fn.device_id = :deviceId  
7          AND fn.deleted = 0  
8          FOR SHARE  
9      )  
10     "", nativeQuery = true)  
11  int getExistsFlagByFestivalIdAndDeviceId(  
12      @Param("festivalId") Long festivalId,  
13      @Param("deviceId") Long deviceId  
14  );
```

비관적 락, 공유 락(Shared Lock)



```
1  @Query(value = ""
2      SELECT EXISTS(
3          SELECT 1
4          FROM festival_notification fn
5          WHERE fn.festival_id = :festivalId
6          AND fn.device_id = :deviceId
7          AND fn.deleted = 0
8          FOR SHARE
9      )
10     "", nativeQuery = true)
11  int getExistsFlagByFestivalIdAndDeviceId(
12      @Param("festivalId") Long festivalId,
13      @Param("deviceId") Long deviceId
14  );
```

UNIQUE 제약 조건

device_id	festival_id	deleted
1	100	0
1	100	1
1	100	0





동시성 제어

