FCM 대량 알림 처리 최적화:

JPA에서 JDBC Batch Insert까지

우아한테크코스 7기 BE 칼리

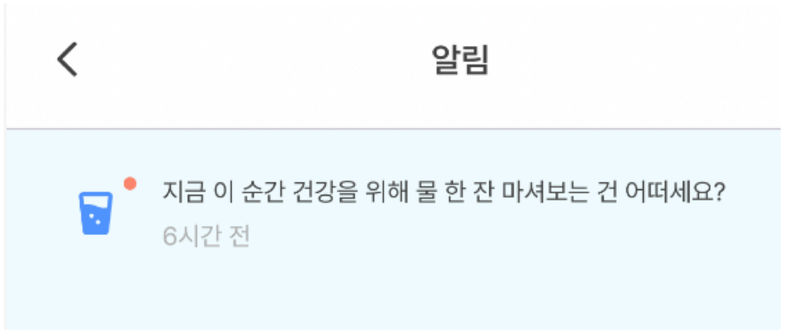
목차

1. 변경된 요구사항: 리마인드 알림

- AS-IS:
  - 대상: 모든 유저
  - 시간: 14시, 19시

1. 변경된 요구사항: 리마인드 알림

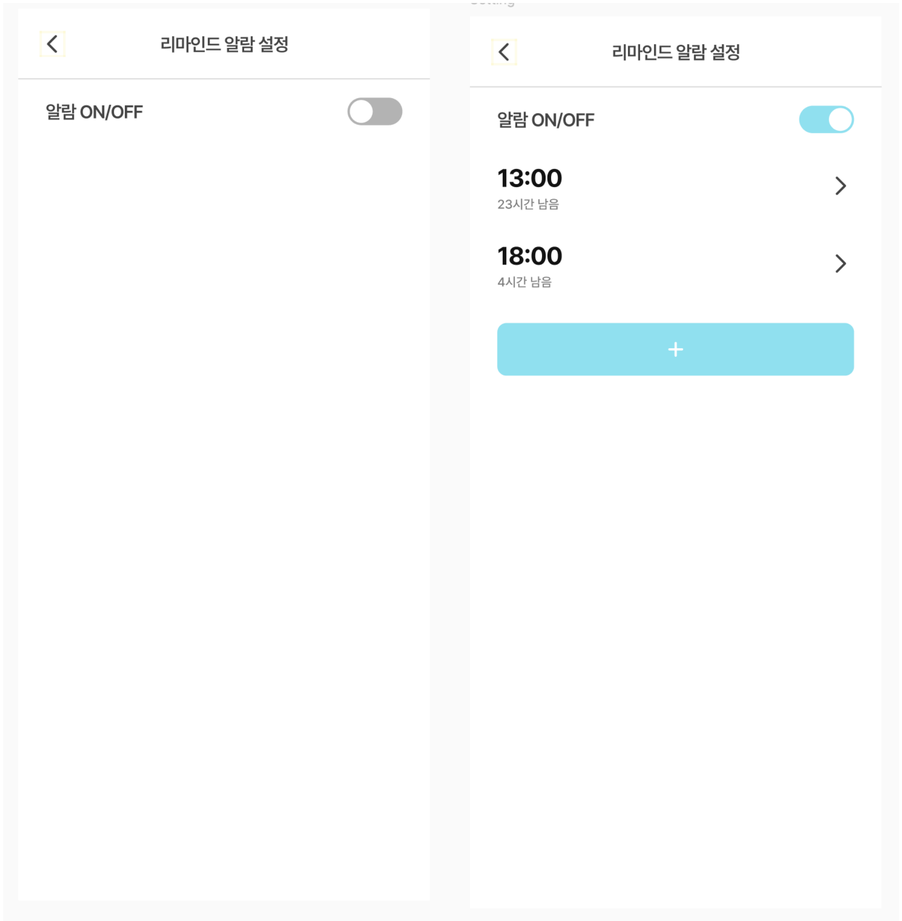
- AS-IS:
  - 대상: 모든 유저
  - 시간: 14시, 19시



1. 변경된 요구사항: 리마인드 알림

- AS-IS:
  - 대상: 모든 유저
  - 시간: 14시, 19시

- TO-BE:
  - 대상: 리마인드 알림 킨 사람
  - 시간: 1분 단위 개인화



2. 기존 방식과 한계

2.1 ERD 구조

2. 기존 방식과 한계

2.2. Service 계층 로직 (AS-IS)

```java
// [ 기존 NotificationService.java - AS-IS ]

@Transactional
// - 정해진 시간대 스케줄링 : 14시, 19시 고정
@Scheduled(cron = DAILY_2PM_CRON)
@Scheduled(cron = DAILY_7PM_CRON)
public void notifyRemindNotification() {

    // - 알림 템플릿 준비 : 리마인드 알림 메시지 선택
    NotificationMessageTemplate template = RemindNotificationMessageTemplateProvi
            .getRandomMessageTemplate();

    // - 모든 멤버 조회 : findAll로 전체 조회 (⚠️ 메모리 부담)
    List<Member> members = memberRepository.findAll();

    // - 알림 데이터 생성 : 멤버 목록을 Notification 엔티티로 변환
    List<Notification> notifications = members.stream()
            .map(member -> template.toNotification(member, LocalDateTime.now()))
            .toList();

    // - 알림 데이터 저장 : JPA saveAll 사용 (⚠️ 개별 INSERT 쿼리 발생)
    notificationRepository.saveAll(notifications);

    // - FCM 알림 발송 : Topic 기반 일괄 발송
    publisher.publishEvent(template.toSendMessageByFcmTopicRequest());
}
```

2. 기존 방식과 한계

2.3. 한계

- 1. JPA의 saveAll + IDENTITY 전략의 한계
- 2. topic 기반 발송의 한계

2. 기존 방식과 한계

2.3. 한계

- **1. JPA의 saveAll + IDENTITY 전략의 한계**

```
// 과거 코드 – 반복적인 save 호출
for (Notification notification : notifications) {
    notificationRepository.save(notification);  // ❌ DB에 여러 번 접근
}

// 개선 시도 – saveAll 사용
notificationRepository.saveAll(notifications);  // ⚠️ 여전히 개별 쿼리 발생
```

2. 기존 방식과 한계

2.3. 한계

- **1. JPA의 saveAll + IDENTITY 전략의 한계**

```
// 과거 코드 – 반복적인 save 호출
for (Notification notification : notifications) {
    notificationRepository.save(notification);  // ❌ DB에 여러 번 접근
}

// 개선 시도 – saveAll 사용
notificationRepository.saveAll(notifications);  // ⚠️ 여전히 개별 쿼리 발생
```

```
// [ Notification.java ]

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)  // IDENTITY 전략 사용
@Column(name = "notification_id")
private Long id;
```

2. 기존 방식과 한계

2.3. 한계

- **1. JPA의 saveAll + IDENTITY 전략의 한계**

```java
// 과거 코드 - 반복적인 save 호출
for (Notification notification : notifications) {
    notificationRepository.save(notification);   // ❌ DB에 여러 번 접근
}

// 개선 시도 - saveAll 사용
notificationRepository.saveAll(notifications);   // ⚠️ 여전히 개별 쿼리 발생
```

```java
// [ Notification.java ]

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)  // IDENTITY 전략 사용
@Column(name = "notification_id")
private Long id;
```

```yaml
jpa:
  show-sql: true
  properties:
    hibernate:
      order_inserts: true
      order_updates: true
      jdbc:
        batch_size: 100
```

```
[2025-10-19T13:28:33] DEBUG [5458e734271446a1] - insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
Hibernate: insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
[2025-10-19T13:28:33] DEBUG [5458e734271446a1] - insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
Hibernate: insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
[2025-10-19T13:28:33] DEBUG [5458e734271446a1] - insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
Hibernate: insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
[2025-10-19T13:28:33] DEBUG [5458e734271446a1] - insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
Hibernate: insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
[2025-10-19T13:28:33] DEBUG [5458e734271446a1] - insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
Hibernate: insert into notification (content,created_at,deleted_at,is_read,member_id,notification_type) values (?,?,?,?,?,?)
```

2. 기존 방식과 한계

2.3. 한계

- **1. JPA의 saveAll + IDENTITY 전략의 한계**

```java
// 과거 코드 – 반복적인 save 호출
for (Notification notification : notifications) {
    notificationRepository.save(notification);  // ❌ DB에 여러 번 접근
}

// 개선 시도 – saveAll 사용
notificationRepository.saveAll(notifications);  // ⚠️ 여전히 개별 쿼리 발생
```

```java
// [ Notification.java ]

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)  // IDENTITY 전략 사용
@Column(name = "notification_id")
private Long id;
```

실행 흐름

1. IDENTITY 전략이 적용된 Notification 엔티티들에 대해 JPA saveAll() 실행
2. JPA는 각 Notification 엔티티에 대해 개별적으로 INSERT 쿼리를 데이터베이스에 전송
3. 데이터베이스는 각 INSERT 쿼리를 실행하여 데이터를 저장
4. 각 INSERT 쿼리에 대해 고유한 기본키 값을 생성하고, 이를 JPA에 반환
5. JPA는 반환된 기본키 값을 각 Notification 엔티티에 설정

2. 기존 방식과 한계

2.3. 한계

- 1. JPA의 saveAll + IDENTITY 전략의 한계
- 2. topic 기반 발송의 한계

```
// [ FcmEventListener.java ]
@EventListener
public void onTopic(SendMessageByFcmTopicRequest sendMessageByFcmTopicRequest) {
    fcmClient.sendMessageByTopic(sendMessageByFcmTopicRequest);
}


// [ FcmClient.java ]

public void sendMessageByTopic(SendMessageByFcmTopicRequest sendFcmTokenMessageRe
    try {
        firebaseMessaging.send(Message.builder()
                .setNotification(Notification.builder()
                        .setTitle(sendFcmTokenMessageRequest.title())
                        .setBody(sendFcmTokenMessageRequest.body())
                        .build())
                .setTopic(sendFcmTokenMessageRequest.topic())
                .putData(ACTION, sendFcmTokenMessageRequest.action().name())
                .build());
    } catch (FirebaseMessagingException e) {
        throw new AlarmException(e);
    }
}
```

3. 개선 사안

3.1. JDBC Batch Insert

```java
// [ NotificationBatchRepository.java ]

private final JdbcTemplate jdbcTemplate;

public void batchInsert(List<NotificationInsertDto> notificationInsertDtos, int batchSize) {
    // - SQL 템플릿 준비 : INSERT 쿼리 정의
    String sql = "INSERT INTO notification (notification_type, is_read, created_at, member_id, content, deleted_at) values (?, ?, ?, ?, ?, ?)";

    // - 생성 시각 통일 : 모든 알림에 동일한 생성 시각 적용
    Timestamp currentTimestamp = Timestamp.valueOf(LocalDateTime.now());

    // - 배치 단위 처리 : batchSize만큼 나누어 처리
    for (int i = 0; i < notificationInsertDtos.size(); i += batchSize) {
        // - 배치 데이터 추출 : 현재 처리할 청크 데이터 분리
        List<NotificationInsertDto> batchNotificationInsertDtos = notificationInsertDtos.subList(i,
                Math.min(i + batchSize, notificationInsertDtos.size()));

        // - Bulk Insert 실행 : 배치 단위로 한 번에 INSERT
        jdbcTemplate.batchUpdate(sql, new BatchPreparedStatementSetter() {
            @Override
            public void setValues(PreparedStatement ps, int i) throws SQLException {
                // - PreparedStatement 값 설정 : 각 필드에 데이터 바인딩
                NotificationInsertDto dto = batchNotificationInsertDtos.get(i);
                ps.setString(1, dto.notificationType().name());
                ps.setBoolean(2, Boolean.FALSE);
                ps.setTimestamp(3, currentTimestamp);
                ps.setLong(4, dto.memberId());
                ps.setString(5, dto.content());
                ps.setTimestamp(6, null);
            }

            @Override
            public int getBatchSize() {
                // - 배치 크기 반환 : 현재 처리할 데이터 개수
                return batchNotificationInsertDtos.size();
            }
        });
    }
}
```

3. 개선 사안

3.2. 배치 Read 최적화

- 청크 단위 조회: 데이터를 나누어 조회하여 DB 부하 분산
- ID만 저장: 전체 엔티티가 아닌 ID만 메모리에 보관하여 메모리 사용량 최소화

```
private List<Long> getMemberIdsForSendingNotification(LocalDateTime now, Long lastId) {
        return ChunkReader.readChunk(
                (id, pageable) -> reminderScheduleRepository
                        .findAllActiveMemberIdsBySchedule(
                                now.toLocalTime(),
                                id,
                                pageable
                        ),
                lastId,
                CHUNK_SIZE
        );
    }
```

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

3. 개선 사안

3.3. FCM MultiCast 적용

- **3.3.1. FCM MultiCast란?**
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

Send one message to multiple devices

The Admin FCM SDKs allow you to multicast a message to a list of device registration tokens. You can use this feature when you need to send the same message to a large number of devices. You can specify up to 500 device registration tokens per invocation.

The return value includes a list of tokens that corresponds to the order of the input tokens. This is useful when you want to check which tokens resulted in errors and then handle them appropriately.

Node.js | Java | Python | Go | C#

```java
// These registration tokens come from the client FCM SDKs.
List<String> registrationTokens = Arrays.asList(
    "YOUR_REGISTRATION_TOKEN_1",
    // ...
    "YOUR_REGISTRATION_TOKEN_n"
);

MulticastMessage message = MulticastMessage.builder()
    .putData("score", "850")
    .putData("time", "2:45")
    .addAllTokens(registrationTokens)
    .build();
BatchResponse response = FirebaseMessaging.getInstance().sendEachForMulticast(message);
if (response.getFailureCount() > 0) {
  List<SendResponse> responses = response.getResponses();
  List<String> failedTokens = new ArrayList<>();
  for (int i = 0; i < responses.size(); i++) {
    if (!responses.get(i).isSuccessful()) {
      // The order of responses corresponds to the order of the registration tokens.
      failedTokens.add(registrationTokens.get(i));
    }
  }

  System.out.println("List of tokens that caused failures: " + failedTokens);
}
```

FirebaseMessagingSnippets.java

3. 개선 사안

3.3. FCM MultiCast 적용

- **3.3.1. FCM MultiCast란?**
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

하나의 메시지를 여러 기기에 전송

Admin FCM SDK를 사용하면 기기 등록 토큰 목록에 메시지를 멀티캐스트할 수 있습니다. 이 기능은 동일한 메시지를 여러 기기에 전송해야 하는 경우에 사용할 수 있습니다. 호출당 최대 500개의 기기 등록 토큰을 지정할 수 있습니다.

Node.js | Java | Python | Go | C#

```java
// These registration tokens come from the client FCM SDKs.
List<String> registrationTokens = Arrays.asList(
    "YOUR_REGISTRATION_TOKEN_1",
    // ...
    "YOUR_REGISTRATION_TOKEN_n"
);

MulticastMessage message = MulticastMessage.builder()
    .putData("score", "850")
    .putData("time", "2:45")
    .addAllTokens(registrationTokens)
    .build();
BatchResponse response = FirebaseMessaging.getInstance().sendEachForMulticast(message);
if (response.getFailureCount() > 0) {
  List<SendResponse> responses = response.getResponses();
  List<String> failedTokens = new ArrayList<>();
  for (int i = 0; i < responses.size(); i++) {
    if (!responses.get(i).isSuccessful()) {
      // The order of responses corresponds to the order of the registration tokens.
      failedTokens.add(registrationTokens.get(i));
    }
  }

  System.out.println("List of tokens that caused failures: " + failedTokens);
}
```

FirebaseMessagingSnippets.java

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

```
예시:
- 14:00 → A, B, C 사용자에게 발송
- 14:01 → D, E 사용자에게 발송
- 14:02 → A, F, G 사용자에게 발송
```

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

MultiCast:

- 구독 설정 없이 즉시 발송 가능
- 매번 다른 사용자 목록에 유연하게 대응

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

```java
// [ FcmClient.java ]

public void sendMulticast(SendMessageByFcmTokensRequest sendMessageByFcmTokensRequest) {
    try {
      firebaseMessaging.sendMulticast(MulticastMessage.builder()
              .setNotification(Notification.builder()
                      .setTitle(sendMessageByFcmTokensRequest.title())
                      .setBody(sendMessageByFcmTokensRequest.body())
                      .build())
              .addAllTokens(sendMessageByFcmTokensRequest.tokens())
              .putData(ACTION, sendMessageByFcmTokensRequest.action().name())
              .build());
  } catch (FirebaseMessagingException e) {
      throw new AlarmException(e);
  }
}
```

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기

You're using APIs that are no longer part of recent versions of firebase-admin since version 13. You will need to upgrade your code.

According to the examples in the documentation, firebase-admin doesn't export a function called `messaging`. The examples all show use of `getMessaging`.

Also, sendMulticast has been removed from recent versions of firebase-admin. The replacement is `sendEachForMulticast`.

Read the release notes for the Firebase Messaging in the Admin SDK version 13:

> Removed deprecated sendAll(), sendMulticast(), sendToDevice(), sendToDeviceGroup(), sendToTopic(), and sendToCondition() APIs.

This is confirmed in the API documentation for getMessaging. It returns a Messaging object which has a method sendForEachMulticast.

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

3. 개선 사안

3.3. FCM MultiCast 적용

- 3.3.1. FCM MultiCast란?
- 3.3.2. Topic 방식의 한계
- 3.3.3. MultiCast 방식의 장점
- 3.3.4. 구현 과정 및 트러블슈팅
- 3.3.5. Token 기반 vs Topic 기반

구분	Token 기반 (MultiCast)	Topic 기반
대상	특정 사용자 선택 가능	구독한 모든 사용자
유연성	높음 (개인화 가능)	낮음 (일괄 발송만 가능)
사용 사례	개인화된 알림, 특정 그룹 알림	공지사항, 전체 알림

3. 개선 사안

3.4. 비동기 & 트랜잭션 분리

- 3.4.1. 문제 상황
- 3.4.2. 해결 방안

3. 개선 사안

3.4. 비동기 & 트랜잭션 분리

- 3.4.1. 문제 상황
- 3.4.2. 해결 방안

3. 개선 사안

3.4. 비동기 & 트랜잭션 분리

- 3.4.1. 문제 상황
- 3.4.2. 해결 방안

FCM 알림 전송 실패 시, DB에 저장된 알림 데이터까지 롤백

3. 개선 사안

3.4. 비동기 & 트랜잭션 분리

- 3.4.1. 문제 상황
- 3.4.2. 해결 방안

3. 개선 사안

3.4. 비동기 & 트랜잭션 분리

- 3.4.1. 문제 상황
- 3.4.2. 해결 방안

@TransactionalEventListener + @Async 적용

```java
// [ FcmEventListener.java ]

@Async
@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
public void onTokens(SendMessageByFcmTokensRequest sendMessageByFcmTokensRequest) {
    Lists.partition(sendMessageByFcmTokensRequest.allTokens(), FCM_BATCH_SIZE)
            .forEach(tokens -> {
                fcmClient.sendMulticast(sendMessageByFcmTokensRequest.withTokens(tokens));
            });
}
```

4. 성능 측정

AS-IS vs TO-BE 성능 측정

- 처리 속도

batch 부분 10만명 기준 1m 26s -> 229m

```
[2025-10-15T13:50:54] INFO  [] - [MOCK FCM] topic=mulkkam, title=물 마실 시간!, body=지금 이 순간 건강을 위해 물 한 잔 마셔보는 건 어떠세요?, action=GO_HOME
[2025-10-15T13:50:54] WARN  [adb506252eab4199] - [API Performance] accountId = null, GET /notifications/test - 86639ms [Status: 200]
[2025-10-15T13:50:54] INFO  [adb506252eab4199] - [RESPONSE] /notifications/test accountId = null, (200 OK) token = null, responseBody: NONE
```

```
[2025-10-16T13:36:08] INFO  [2ed1247751b74306] - [REQUEST] GET /notifications/test token = null
[2025-10-16T13:36:08] INFO  [2ed1247751b74306] - [API Performance]: accountId = null, GET /notifications/test - 229ms [Status: 200]
[2025-10-16T13:36:09] INFO  [2ed1247751b74306] - [RESPONSE] /notifications/test accountId = null, (200 OK) token = null, responseBody: NONE
```

5. 향후 개선 방향

- 1. Multicast 응답 처리
- 2. 스케줄링 최적화

5. 향후 개선 방향

- 1. Multicast 응답 처리
- 2. 스케줄링 최적화

5. 향후 개선 방향

- 1. Multicast 응답 처리
- 2. 스케줄링 최적화

반환 값은 입력 토큰 순서에 해당하는 토큰 목록을 포함합니다. 이는 오류가 발생한 토큰을 확인한 후 적절하게 처리하려는 경우에 유용합니다.

Node.js 자바 Python Go C#

```java
// These registration tokens come from the client FCM SDKs.
List<String> registrationTokens = Arrays.asList(
    "YOUR_REGISTRATION_TOKEN_1",
    // ...
    "YOUR_REGISTRATION_TOKEN_n"
);

MulticastMessage message = MulticastMessage.builder()
    .putData("score", "850")
    .putData("time", "2:45")
    .addAllTokens(registrationTokens)
    .build();
BatchResponse response = FirebaseMessaging.getInstance().sendEachForMulticast(message);
if (response.getFailureCount() > 0) {
  List<SendResponse> responses = response.getResponses();
  List<String> failedTokens = new ArrayList<>();
  for (int i = 0; i < responses.size(); i++) {
    if (!responses.get(i).isSuccessful()) {
      // The order of responses corresponds to the order of the registration tokens.
      failedTokens.add(registrationTokens.get(i));
    }
  }

  System.out.println("List of tokens that caused failures: " + failedTokens);
}
```

FirebaseMessagingSnippets.java

5. 향후 개선 방향

- 1. Multicast 응답 처리
- 2. 스케줄링 최적화

끗