6장. 레디스를 메시지 브로커로 사용하기

레디스로 당장 메시지를 처리하지 못해도 쌓아 둔 뒤 나중에 처리할 수 있는 채널을 만들어주는 메시지 브로커의 핵심 역할을 구현하는 방법을 알아봅시다.

메시지 브로커

메시지 큐 vs 이벤트 스트림

- 방향성: 메시지 큐는 소비자의 큐로 데이터를 직접 푸시, 이벤트 스트림은 특정 저장소에 하나의 메시지를 보내 소비자가 스트림에서 같은 메시지를 풀할 수 있음.
- 영속성: 메시지 큐는 데이터를 읽어갈 때 큐에서 삭제, 이벤트 스트림은 특정 기간동안 저장할 수 있음.

→ 메시지 큐는 1:1 상황에서, 스트림은 n:m 상황에서 유리할 수 있겠다 -!

메시지 브로커로서의 레디스

- list 자료 구조와 레디스가 제공하는 pub/sub으로 빠르고 간단하게 구현 가능
- 한 번 채널 전체에 전파된 뒤 삭제되는 일회성의 특징
- 메시지가 잘 전달됐는지 등의 정보는 보장하지 않음

  `fire-and-forget` **패턴**

  - 비동기 프로그래밍에서 사용되는 디자인 패턴
  - 결과에 대한 응답을 기다리지 않고 바로 다음 코드를 실행하는 것
- 카프카에서 영감을 받아 만들어진 자료구조인 `stream`은 append-only로 동작함

레디스의 pub/sub

최소한의 메시지 전달 기능만 제공하는 가벼운 서비스

```
# 데이터 전파
# PUBLISH 채널명 메시지 -> 메시지를 수신한 구독자 수 반환
> PUBLISH hello world
(integer 1)

# 채널 구독
# SUBSCRIBE 채널명1 (채널명2 ...)
# 클라이언트가 구독자로 동작하면 pub/sub 관련 외 다른 커맨드를 수행할 수
> SUBSCRIBE event1 event2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "event1"
3) (integer) 1
1) "subscribe"
2) "event2"
3) (integer) 2

# 특정 패턴에 해당하는 채널을 한 번에 구독 (glob-style 패턴 지원)
> PSUBSCRIBE mail-*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"
2) "mail-*"
3) (integer) 1
```

클러스터 구조에서의 pub/sub

메시지를 발행하면 해당 메시지는 **클러스터에 속한 모든 노드에 자동으로 전달**된다.

이는 클러스터 환경의 핵심 목표와 부합하지 않고, 불필요한 리소스 사용과 네트워크 부하를 일으킨다.

shared pub/sub

- 레디스 7.0에서 도입되어 각 채널은 슬롯에 매핑되고 같은 슬롯을 가지고 있는 노드 간에만 pub/sub 메시지를 전파한다.
- 불필요한 복제를 줄여 자원이 절약된다.

```
# 노드의 복제본에만 메시지 전달
10.0.0.1:6379> SPUBLISH apple a
-> Redirected to slot [7092] located at 10.0.0.2:6379
(integer) 1
10.0.0.2:6379>
```

list로 메시징 큐 만들기

1️⃣ 트위터 타임라인 관리

```
# RPUSHX: list가 이미 존재할 때에만 아이템을 추가
# UserA가 data3 피드를 게시한 경우 팔로워들의 타임라인을 업데이트함
# 트위터에 자주 들어오지 않는 유저(userD)는 타임라인 캐시를 관리하지 않아도
> RPUSHX Timelinecache:UserB data3
(integer) 26
> RPUSHX Timelinecache:UserC data3
(integer) 5
> RPUSHX Timelinecache:UserD data3
(integer) 0
```

2️⃣ 원형 큐

특정 아이템을 계속해서 반복 접근해야 하거나 여러 개 클라이언트가 병렬적으로 같은 아이템에 접근하는 경우

```
# 데이터 등록
> LPUSH client A
(integer) 1
> LPUSH client B
(integer) 2
> LPUSH client C
(integer) 3

# 데이터 확인
LRANGE client 0 -1
1) "C"
```

```
2) "B"
3) "A"

# 원형 큐 사용
> RPOPLPUSH client client
"A"

# 데이터 확인
LRANGE client 0 -1
1) "A"
2) "C"
3) "B"
```

Stream

레디스 5.0에서 새로 추가된 자료구조, append-only

1) 대량의 데이터를 효율적으로 처리하는 플랫폼

2) 생성한 데이터를 다양한 소비자가 처리할 수 있게 지원하는 데이터 저장소 or 중간 큐잉 시스템

특징

- 각 메시지는 시간과 관련된 유니크한 ID를 가지며, 중복되지 않는다.
  - **밀리세컨드 파트**: 실제 아이템이 저장될 시점에 레디스 노드 로컬 시간
  - **시퀀스 파트**: 같은 밀리세컨드에 저장된 데이터의 순서
- ID를 직접 지정할 수 있으나 `최소 ID값은 0-1`이며 이후에 저장되는 ID가 이전에 저장된 ID보다 작을 수 없다.

데이터 읽기

실시간 리스닝

```
XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...
```

ID를 이용한 검색

```
XRANGE key start end [COUNT count]
XREVRANGE key end start [COUNT count]
```

원하는 시간대의 데이터를 조회할 수 있다.

ACK와 보류 리스트

예상치 못한 장애로 시스템이 종료된 경우 이를 인지하고 재처리할 수 있는 기능이다.

소비자 그룹에 속한 소비자가 메시지를 읽어가면 각 소비자별로 읽어간 메시지에 대한 리스트를 새로 생성하고 마지막으로 읽어간 데이터의 ID로 `last_delivered_id` 값을 업데이트 한다.

last_delivered_id

- 소비자 그룹에 마지막으로 전달한 ID가 무엇인지 파악한다.
- 동일한 메시지를 중복으로 전달하지 않기 위해 사용한다.

```
# 소비자 그룹에서 보류 중인 리스트가 있는지 확인
# XPENDING <key> <groupname> [<start-id> <end-id> <count> [<c
> XPENDING Email EmailServiceGroup
1) (integer) 9 # 현재 소비자 그룹에서 ACK를 받지 못해 보류중인 메시지의
2) "165911354522-0" # 보류 중인 ID의 최소값
3) "165918730382-0" # 보류 중인 ID의 최대값
4) 1) 1) "es1"  # 소비자 별 보류 중인 리스트 개수
      2) "1"
   2) 1) "es2"
      2) "1"
   3) 1) "es3"
      2) "7"

# 데이터가 처리되었음을 알림
> XACK Email EmailServiceGroup 165911354522-0
(integer) 1
```

메시지 보증 전략

- **at most once**: 최소 한 번 메시지를 보냄, 소비자가 메시지를 받자마자 ACK를 보냄
- **at least once**: 소비자가 받은 메시지를 모두 처리한 뒤 ACK를 보냄
- **exactly once**: 무조건 한 번씩만 전송됨을 보장. 레디스에서는 추가 자료 구조를 이용해서 이미 처리된 메시지인지를 확인하는 과정이 필요

메시지 재할당

보류 리스트를 만들었으나 소비자 서버에 장애가 발생해 복구되지 않는다면?

```
# 메시지 소유권을 다른 소비자에게 재할당
# XCLAIM <key> <group> <consumer> <min-idle-time> <ID-1> <ID-
EmailService 1: XCLAIM email EGroup EService3 3600000 162646
EmailService 2: XCLAIM email EGroup EService3 3600000 162646
```

EmailService1의 커맨드가 먼저 실행되면 메시지 보류 시간이 즉시 0으로 재설정되기 때문에 이후 커맨드는 무시되어 중복 메시지 할당을 방지한다.

```
# 보류했던 메시지 중 하나를 자동으로 가져와서 처리
# XAUTOCLAIM <key> <group> <consumer> <min-idle-time> <start>
> XAUTOCLAIM email EGroup es1 360000 0-0 count 1
1) "1659848396477-0" # 다음 대기중인 보류 메시지 ID (없을 경우 0-0)
2) 1) 1) "1659848396477-0" # 소유권이 이전된 메시지의 정보 제공
        2) 1) "subject"
                2) "second"
                3) "body"
                4) "hihi"
```

dead letter: 재할당되는 stream 내 메시지는 counter 값을 1씩 증가시킨다. 여러 소비자에게 재할당을 반복해도 처리되지 못한 메시지는 counter가 특정 값에 도달하면 특수한 다른 stream으로 보내 관리자가 추후에 처리될 수 있도록 해야 한다.

stream 상태 확인

```
# 특정 소비자 그룹에 속한 소비자의 정보 조회
# XINFO consumers <stream key><소비자 그룹 이름>
> xinfo consumers email emailsservicegroup
1) 1) "name"
```

```
        2) "es1"
        3) "pending"
        4) (ingteger) 1
        5) "idle"
        6) (integer) 650129
...

# stream에 속한 전체 소비자 그룹 목록 조회
# XINFO GROUPS <stream key>
> xinfo groups email
1) 1) "name"
        2) "bigroup"
        3) "consumers"
        4) (integer) 1
        5) "pending"
        6) (integer) 6
        7) "last-delivered-id"
        8) "16435947933470-0"
        9) "entries-read"
        10) (integer) 6
        11) "lag"
        12) (integer) 4
...

# stream 자체의 정보 조회
# XINFO STREAM <stream key>
> XINFO STREAM email
1) "length"
2) (integer) 10
3) "radix-tree-keys"
4) (integer) 1
...
# 내부적으로 어떻게 인코딩 되는지, 첫 번째 메시지 ID, 마지막 메시지 ID 등
```