STREAMS Introduction

소개 Introduction

스트림 Streams

스트림(Stream)은 로그 데이터를 처리하기 위해서 5.0에서 새로 도입된 데이터 타입입니다.
여러 산업(industry)에서는 많은 경우 데이터가 연속적으로 발생합니다. 이 데이터의 특징은 사람이 아니고 기계(machine)가 발생시키며 연속적이고 대량이라는 것입니다. 또 하나의 특징은 기존 데이터를 수정하지 않고 오직 추가로 발생한다는 것입니다. 우리는 이런 종류의 데이터를 스트림(stream) 또는 로그(log) 데이터라고 합니다.
스트림 데이터의 실 예를 들어보면, 반도체, 디스플레이, 제철 같은 여러 제조공정에서는 온도, 습도, 압력, 진동, 기울기, 조도(밝기), 연기 등을 감지하는 많은 센서(sensor)가 동작하여 초 단위 이하로 데이터를 발생시킵니다. 제조공정에서는 이 데이터를 이용해서 생산품의 불량의 원인을 분석하는데 이용하며, 경우에 따라서는 불량을 사전에 예방할 수도 있습니다.
반도체 웨이퍼(wafer) 불량
가까운 예를 하나 더 들면, 여러 서버들을 모니터링한다고 합시다. 그럼 각 서버의 CPU, Memory, Disk I/O, Network I/O 등의 데이터를 수집해서 실시간 차트를 보여주고 알람을 보낼 수도 있습니다.
레디스 5.0에서는 이런 데이터를 보관하고 처리하는데 적합한 데이터 구조인 스트림(stream)을 도입했습니다.
디스크 vs 메모리 , RDB 테이블 vs 스트림
그럼 이제 스트림에 데이터를 넣고, 조회하는 방법을 하나씩 살펴보겠습니다.

데이터 넣기(추가): XADD

Sensor-1234의 온도 데이터를 넣어보겠습니다.

```
XADD sensor-1234 * temperature 98.7
1538319053569-0
```

XADD 명령문(syntax)은 다음과 같습니다.

```
XADD key ID field value [field2 value2 ...]
```

key=sensor-1234, ID=*, field=temperature, value=98.7 입니다.
여기서 눈여겨 볼 것은 ID=* 입니다. ID를 *로 입력하고 명령의 결과로 ID를 리턴합니다.
ID는 두 개의 숫자로 구성되는데
<millisecondsTime>-<sequenceNumber>
앞은 millisecond 단위의 timestamp 숫자이고, 뒤는 같은 밀리 초 내에 또 데이터가 들어오면 중복을 방지하기 위해서 숫자가 1, 2, 3 와 같이 하나씩 증가합니다. 이 sequenceNumber는 8바이트 정수이므로 실제적으로 동일한 밀리 초 내에 생성 될 수 있는 엔트리 수에는 제한이 없습니다. ID를 *로 입력하는 것은 레디스가 있는 서버 시간으로 ID를 생성하는 것입니다.
ID를 직접 입력할 수도 있습니다.

```
XADD sensor-1234 1538319053569-1 temperature 98.8
1538319053569-1
```

ID는 이전에 입력된 것보다 항상 커야 합니다. 같거나 작으면 에러가 발생합니다.

데이터 길이(항목수) 조회하기: XLEN

```
XLEN sensor-1234
(integer) 2
```

정확하게 표현하면 ID 개수를 조회하는 것입니다.

데이터 조회하기: XRANGE

```
XRANGE sensor-1234 - +
1) 1) 1538319053569-0
   2) 1) "temperature"
      2) "98.7"
2) 1) 1538319053569-1
   2) 1) "temperature"
      2) "98.8"
```

XRANGE 명령문(syntax)은 다음과 같습니다.

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

key=sensor-1234, start=-, end=+ 입니다.
Count를 사용해 조회할 데이터 개수를 지정할 수 있습니다.
Start와 end에 특정 ID를 지정할 수 있습니다.

```
XRANGE key 1538319053569 1538319053569
```

최근 데이터부터 조회하고 싶으면 XREVRANGE 명령을 사용하세요.

```
XREVRANGE key + -
```

데이터 읽어오기: XREAD

```
XREAD count 1 STREAMS sensor-1234 1538322045065-0
1) 1) "sensor-1234"
   2) 1) 1) 1538322045065-1
      2) 1) "temperature"
         2) "98.8"
```

XREAD 명령문(syntax)은 다음과 같습니다.

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

- Count는 읽어올 데이터 개수를 지정합니다. Count를 지정하지 않으면 모든 데이터를 읽어옵니다.
- ID는 지정한 ID의 다음 데이터를 읽어옵니다. 처음 데이터를 읽으려면 ID로 0을 지정합니다.
- Block은 새 데이터가 들어오기를 기다렸다 들어오면 읽어옵니다. 이 때는 ID로 특별히 $를 사용합니다. Lists의 BLPOP과 비슷합니다.

```
XREAD block 5000 STREAMS sensor-1234 $
```

데이터 삭제하기: XDEL, XTRIM

XDEL

```
XDEL sensor-1234 1538322045065-0
(integer) 1
```

XDEL 명령문(syntax)은 다음과 같습니다.

```
XDEL key ID
```

XTRIM

sensor-1234에 데이터가 100가 있다면 오래된 순으로 90개를 지우고 최신 데이터 10개를 남깁니다.

```
XTRIM sensor-1234 maxlen 10
```

XTRIM 명령문(syntax)은 다음과 같습니다.

```
XTRIM key MAXLEN [~] count
```

특별한 옵션인 ~는 약(about)입니다. Sensor-1234에 100만개의 데이터가 있다면 999,990개를 지우는데 시간이 걸릴 것입니다. 그러면 데이터를 지우는 동안 데이터를 추가(XADD)하거나 처리(XREAD) 될 수 없습니다. 대량 데이터를 신속히 처리해야하는 스트림에서는 이런 처리 지연이 발생하지 않도록해야 합니다. 그래서 짧은 시간에 처리할 수 있을 정도의 데이터를 삭제하는 기능입니다.

다른 데이터 타입과 비교

LIST

그 동안 레디스에서 이런 스트림/로그 데이터를 처리하는데 사용된 데이터 타입은 주로 LIST 였습니다.

- LPUSH, XADD: 역할은 비슷합니다. 속도 차이는 있습니다. LPUSH는 O(1)이고 XADD는 O(log(N))입니다.
  O(1), O(N), O(log(N))
- RPOP, XREAD: RPOP은 데이터를 읽고 지웁니다. XREAD는 읽기만 합니다. RPOP은 O(1)이고 XREAD는 O(log(N))입니다. RPOP은 데이터를 읽은 후 애플리케이션 문제로 데이터를 처리하지 못했을 경우 복구할 수 있는 방법이 없읍니다.
  LIST에서 데이터를 읽기만 하려면 LINDEX 명령을 사용하세요.
  STREAM에서 RPOP처럼 데이터를 읽고 지우려면 XREAD와 XDEL을 사용하세요.
- LREM, XDEL: LREM은 O(N)이고 XDEL는 O(log(N))입니다.
- LTRIM, XTRIM: LTRIM은 O(N)이고 XTRIM은 O(log(N))입니다.
- 비고 속도: LIST는 값(string)을 비교합니다. STREAM은 ID(숫자)를 비교합니다. 따라서 STREAM이 비교 속도가 더 빠릅니다.
- 데이터: LIST는 필드와 값을 애플리케이션에서 구별(파싱)해야 합니다.
  LSIT: "sensor-1234;1538319053569;temperature;98.7"
  STREAM은 레디스에서 구별됩니다.
- 중간에 데이터 넣기: LIST는 가능합니다. STREAM은 불가능합니다.

Sorted SET

Score/ID로 소트된다는 점에서 두 데이터 타입은 비슷합니다. 스트림/로그 데이터를 ZSET에 저장한다면 score로 timestamp를 사용합니다.

- ZADD, XADD: 둘 다 O(log(N)) 입니다.
- ZPOPMAX, XREAD: 마지막 1개의 데이터를 읽어온다면 둘 다 O(log(N)) 입니다.
- ZREM, XDEL: 둘 다 O(log(N)) 입니다.

- 데이터: ZSET은 값의 중복을 허용하지 않습니다. 그래서 동일한 값이 들어올 경우에 대비해서 값에 timestamp 같이 구별할 수 있는 데이터를 추가해야 합니다.
  예) Score: 1538319053569, Value: "sensor-1234;1538319053569;temperature;98.7"
- 중간에 데이터 넣기: ZSET은 가능합니다. STREAM은 불가능합니다.

HASH

필드를 갖는다는 점에서 두 데이터 타입은 비슷합니다. 그런데 HASH에서 hgetall 명령을 실행하면 입력된 필드 순서와는 무관하게 조회됩니다만, STREAM에서는 입력된 순서를 유지합니다.

다른 데이터 타입에는 없는 스트림의 특징, 장점

- **레디스 스트림에서는 소비자(Consumer)를 지정해서 데이터를 읽을 수 있고,**
- **그 소비자가 데이터를 제대로 처리했는지 확인하는 방법을 제공하며,**
- **만약 제대로 처리하지 못했다면 다른 소비자에게 할당해서 처리하도록 하는 방법을 제공합니다.**

이제 이렇게 처리하는 방법을 알아보도록 합시다.

소비자그룹 만들기: XGROUP

```
XGROUP CREATE sensor-1234 ConsumerGroup $
```

테스트 데이터 넣기

```
XADD sensor-1234 * temperature 100
XADD sensor-1234 * temperature 101
XADD sensor-1234 * temperature 102
XADD sensor-1234 * temperature 103
XADD sensor-1234 * temperature 104
XADD sensor-1234 * temperature 105
```

소비자그룹을 이용한 데이터 읽기: XREADGROUP

```
XREADGROUP GROUP ConsumerGroup Consumer-A count 1 STREAMS sensor-1234 >
1) 1) "sensor-1234"
   2) 1) 1) 1538568726521-0
         2) 1) "temperature"
            2) "100"
XREADGROUP GROUP ConsumerGroup Consumer-A COUNT 1 STREAMS sensor-1234 >
1) 1) "sensor-1234"
   2) 1) 1) 1538568728545-0
         2) 1) "temperature"
            2) "101"
```

처리 여부 확인하기: XPENDING

```
XPENDING sensor-1234 ConsumerGroup
1) (integer) 2
2) 1538568726521-0
3) 1538568728545-0
4) 1) 1) "Consumer-A"
   2) "2"
```

처리 확정하기: XACK

```
XACK sensor-1234 ConsumerGroup 1538568726521-0
(integer) 1
```

처리 여부 다시 확인하기

```
XPENDING sensor-1234 ConsumerGroup
1) (integer) 1
2) 1538568728545-0
3) 1538568728545-0
4) 1) 1) "Consumer-A"
   2) "1"
```

팬딩 시간 알아보기: XINFO CONSUMERS

```
XINFO CONSUMERS sensor-1234 ConsumerGroup
1) 1) name
   2) "Consumer-A"
   3) pending
   4) (integer) 1
   5) idle
   6) (integer) 31963
```

팬딩된 데이터를 다른 소비자에게 할당하고 처리하기: XCLAIM

```
XCLAIM sensor-1234 ConsumerGroup Consumer-B 30000 1538568728545-0
1) 1) 1538568728545-0
   2) 1) "temperature"
      2) "101"
```

이 데이터 한 건에 대해서만 다른 소비자가 처리하도록 한 것입니다.
XCLAIM으로 처리한 후에 XACK를 보내야 합니다.
XCLAIM 간단한 명령문(syntax)은 다음과 같습니다.

```
XCLAIM key group consumer min-idle-time ID
```

min-idle-time은 밀리초를 지정합니다.
이제 계속 데이터를 읽어서 처리합니다.

```
XREADGROUP GROUP ConsumerGroup Consumer-B COUNT 1 STREAMS sensor-1234 >
1) 1) "sensor-1234"
   2) 1) 1) 1538568731409-0
      2) 1) "temperature"
         2) "102"
```

스트림 하나에 여러 소비자 할당하기

하나의 스트림에 처리해야 할 데이터가 많고, 처리 시간이 오래 걸린다면, 아래에서 처럼 소비자(consumer)를 여러 개 지정할 수 있습니다.
예를 들어, 1초에 3개의 데이터(Data-1,2,3)가 들어오고 애플리케이션에서 하나의 데이터를 처리하는데 1초가 걸린다면 3개의 소비자(Consumer-A,B,C)를 지정해서 각각 다른 애플리케이션에서 처리하도록 합니다.

```
Data-1 -> Consumer-A
Data-2 -> Consumer-B
Data-3 -> Consumer-C
```

Data-4 -> Consumer-A
Data-5 -> Consumer-B
Data-6 -> Consumer-C

Streams 명령어 리스트

Commands	Version	Syntax	Description
XADD	5.0.0	key ID field string [field string ...]	
XLEN	5.0.0	key	
XRANGE	5.0.0	key start end [COUNT count]	
XREVRANGE	5.0.0	key end start [COUNT count]	
XREAD	5.0.0	[COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]	
XDEL	5.0.0	key ID [ID ...]	
XTRIM	5.0.0	key MAXLEN [~] count	
XGROUP	5.0.0	[CREATE key group id-or-$] [DESTROY key group] [DELCONSUMER key group consumer] [SETID key group id-or-$]	
XREADGROUP	5.0.0	GROUP group consumer [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]	
XACK	5.0.0	key group ID [ID ...]	
XPENDING	5.0.0	key group [start end count] [consumer]	
XCLAIM	5.0.0	key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]	
XAUTOCLAIM	6.2.0	key group consumer min-idle-time start [COUNT count] [JUSTID]	
XINFO	5.0.0	[CONSUMERS key group] [GROUPS key] [STREAM key] [HELP]	

Total : 14

<< HRANDFIELD STREAMS Introduction XADD >>

redisgate@gmail.com 02.503.2235
서울시 강남구 강남대로 342 역삼빌딩 5층 (역삼동) 우 06242