Redis Basic

설치

- Redis(REmote DIctionary Server)
  - ✓ 설치: https://redis.io/docs/latest/operate/oss_and_stack/install/install-stack/
    - Public Cloud 서비스 이용
    - Linux에 설치
      - 소스 파일을 이용해서 설치: https://download.redis.io/redis-stable.tar.gz
      - 패키지를 이용한 설치
    - Mac에 설치
      - brew tap redis/redis
      - brew install --cask redis
    - Windows에 설치
      - PowerShell에 wsl을 이용해서 설치
      - https://github.com/microsoftarchive/redis/releases 에서 다운로드 받아서 설치

설치

- 설치
  - ✓ Docker를 이용한 설치
    - 이미지 다운로드
      - docker pull redis
    - 이미지 확인
      - docker images
    - 컨테이너 생성 및 실행
      - docker run --name myredis -d -p 6379:6379 redis

설치

- 설정
  - ✓ 설정 파일 위치
    - Linux: /etc/redis/redis.conf
    - Mac: /opt/homebrew/etc/redis.conf
    - Windows: 설치된 디렉토리의 redis.windows-service.conf

설치

❖ 설정

✓ 설정 항목

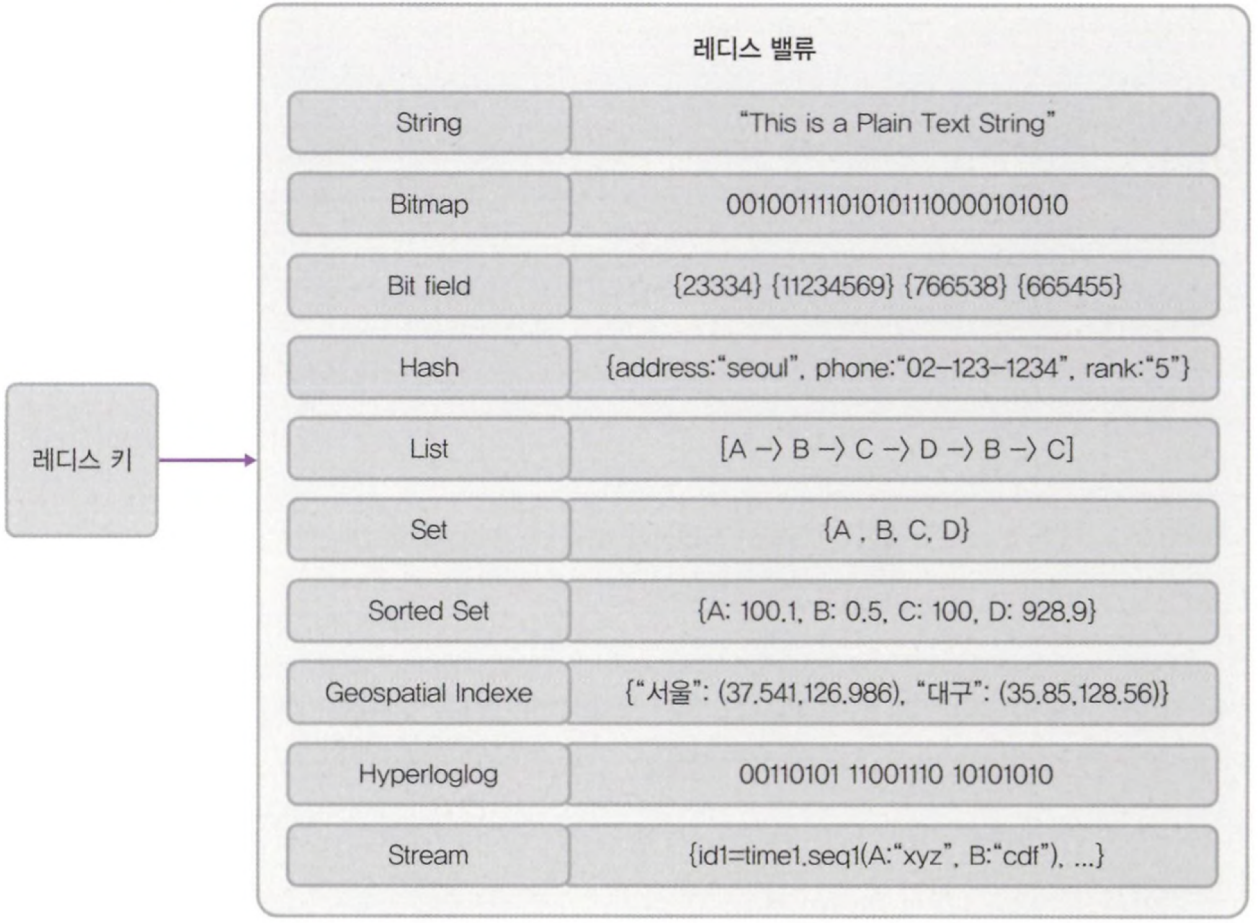
- bind: redis가 bind할 interface를 설정하는 것으로 default는 127.0.0.1
- port: Client, Slave 서버가 통신할 포트 설정으로 default는 6379 번이며 Cluster 모드로 운영 시 서버들 간 통신은 기본 포트에 10000을 더한 16379 포트로 통신
- protected-mode: 패스워드를 설정해야만 redis에 접근할 수 있도록 하는 설정으로 default : yes
- requirepass / masterauth: requirepass 파라미터는 서버에 접속하기 위한 패스워드 값을 의미하며 masterauth 파라미터는 복제 구조를 사용할 때 필요한데 연결될 마스터의 패스워드 값을 의미하는데 만약 복제 연결을 사용할 예정이라면 이 두 값은 같은 값으로 설정하는 것이 바람직한데 기본값은 없음
- daemonize: redis 프로세스를 데몬으로 실행시키고자 할 때 사용하는 것으로 기본값은 no
- dir: redis의 작업 디렉토리 설정으로 기본값은 ./
- tcp-backlog: redis 서버의 초당 클라이언트 연결 개수로default : 511
- timeout: 클라이언트에서 서버 접속 시의 timeout 값으로 default : 0 (항상 connection 열어두기)
- maxclients : 최대 접속 가능한 클라이언트 수로 default : 10000
- tcp-keepalive : 클라이언트가 죽었을 때 서버가 확인하여 클라이언트와의 접속을 제거하는 시간으로 default : 300

설치

- 접속
  - ✓ 설치 시 제공된 redis-cli를 이용
  - ✓ 외부 접속 시에는 redis-cli -h <IP주소> -p <포트번호> -a <패스워드>
  - ✓ IP 주소를 생략하면 127.0.0.1 이고 포트번호를 생략하면 6379
  - ✓ 도커에서 접속: docker run -it --link myredis:redis --rm redis redis-cli -h redis -p 6379

Redis

- Collection(자료구조)
  - ✓ 종류



Redis

- Collection(자료구조)
  - ✓ 종류
    - String: 문자열 데이터를 저장 및 조회할 수 있는 기본 자료구조
    - Hash
      - 해시 자료 구조로 해시 필드(field) 와 해시 밸류(value)로 구성
      - 해시 데이터는 Redis 키와 매핑되어 있으므로 해시 밸류를 생성 및 조회하려면 Redis 키와 해시 필드를 동시에 사용해야 암
    - List: 리스트 데이터로 리스트 아이템은 링크드 리스트 형태로 서로 연결되어 있음
    - Set: 리스트와 비슷한 집합 데이터이며 리스트와 다른 점은 아이템의 중복을 허용하지 않는 자료 구조
    - Sorted Set
      - Set과 비슷한 집합 데이터이지만 정렬 기능을 제공하기 때문에 중복되지 않는 아이템을 저장할 수 있으며 이때 아이템은 스코어와 함께 저장할 수 있음
      - 스코어 값을 사용하여 정렬하고 스코어 값이 중복되면 아이템 값을 사용하여 정렬
    - BitMap: 비트 연산을 사용할 수 있는 자료 구조

Redis

- ❖ Collection(자료구조)
  - ✓ 종류
    - Hyperloglog
      - ◆ Hyperloglogs는 집합의 아이템 개수를 추정할 수 있는 알고리즘 이름이자 이를 사용할 수 있는 Redis 자료 구조로 이 알고리즘은 비트 패턴을 분석하여 비교적 정확한 추정 값을 계산할 수 있음
      - ◆ 특정 상품의 조회 수를 1만 239회라고 정확하게 계산할 때는 시스템 부하가 발생할 수 있으므로 추정 값을 계산하는데 최적화되어 1만회 같은 근사 값을 조회할 수 있는데 중복된 값을 제거할 수 있고 저장 공간이 작으므로 카운트에 적합
    - Geospatial Indexe: 지점과 위도와 경도를 사용할 수 있는 자료 구조로 위도와 경도를 계산하여 두 지점의 거리를 구할 수 있는 명령어를 제공
    - Stream
      - ◆ Redis 5.0 버전부터 제공하는 기능으로 이벤트성 로그를 처리
      - ◆ 메시지 서비스 기능으로 스트림 키 이름과 값, 필드를 사용할 수 있는 자료 구조 형태

Redis

- ❖ Collection(자료구조)
  - ✓ String
    - ● 개요
      - ◆ string은 redis에서 데이터를 저장할 수 있는 가장 간단한 자료 구조
      - ◆ 최대 512MB의 문자열 데이터를 저장할 수 있으며 이진 데이터를 포함하는 모든 종류의 문자열이 binary-safe하게 처리되기 때문에 JPEG 이미지와 같은 바이트 값 및 HTTP 응답값 등의 다양한 데이터를 저장하는 것도 가능
      - ◆ string은 키와 실제 저장되는 아이템이 일대일로 연결되는 유일한 자료 구조이며 string이 아닌 다른 자료 구조에서는 하나의 키에 여러 개의 아이템이 저장됨

Redis

- ❖ Collection(자료구조)
  - ✓ String
    - ● 데이터 저장
      - ◆ SET hello world

      OK
      - ◆ GET hello

      "world"

Redis

- ❖ Collection(자료구조)
  - ✓ String
    - ● 데이터 저장
      - ◆ hello라는 키에 다른 값이 연결돼 있었다면 기존 값은 새로 입력된 값으로 대체되며 저장돼 있던 값이 같은 string 형태가 아닌 다른 형태의 자료 구조라도 동일하게 동작
      - ◆ SET과 함께 NX 옵션을 사용하면 지정한 키가 없을 때에만 새로운 키를 저장
      - ◆ XX 옵션을 사용하면 키가 이미 있을 때에만 새로운 값으로 덮어 쓰며 새로운 키를 생성하지 않도록 동작
      - ◆ redis:6379> SET hello newval NX

      (nil)
      - ◆ redis:6379> SET hello newval XX

      OK
      - ◆ redis:6379> GET hello

      "newval"

Redis

- ❖ Collection(자료구조)
  - ✓ String
    - ● 데이터 저장
      - ◆ string 자료 구조에는 숫자 형태의 데이터를 저장하는 것도 가능
      - ◆ NCR, INCRBY와 같은 명령어를 이용하면 string 자료 구조에 저장된 숫자를 원자적으로atomic 조작할 수 있음
      - ◆ SET counter 100

      OK
      - ◆ INCR counter

      (integer) 101
      - ◆ INCR counter

      (integer) 102
      - ◆ INCRBY counter 50

      (integer) 152

Redis

- ❖ Collection(자료구조)
  - ✓ String
    - ● 데이터 저장
      - ◆ 여러 개의 데이터 저장 및 조회
      - ◆ MSET a 10 b 20 c 30

      OK
      - ◆ MGET a b c

      1) "10"

      2) "20"

      3) "30"

Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 개요
      - ◆ List Collection은 하나의 key에 여러 value들을 list로 저장하는 구조
      - ◆ 하나의 list에는 최대 42억여 개의 아이템을 저장할 수 있음
      - ◆ 인덱스를 이용해 데이터에 직접 접근할 수 있음
      - ◆ 일반적으로 list는 서비스에서 스택과 큐로서 사용
    - ● 양쪽 끝에서 데이터 삽입: LPUSH, RPUSH
      - ◆ LPUSH mylist E

      (integer) 1
      - ◆ RPUSH mylist B

      (integer) 2
      - ◆ LPUSH mylist D A C B A

      (integer) 7

Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 범위 내의 데이터 조회: LRANE
      - ◆ 형식: LRANGE key start stop
      - ◆ 인덱스 규칙
        - ▪ 0 : 첫 번째 요소
        - ▪ 1 : 두 번째 요소
        - ▪ -1 : 마지막 요소
        - ▪ -2 : 뒤에서 두 번째 요소

Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 범위 내의 데이터 조회: LRANE
      - ◆ LRANGE mylist 0 -1

1) "A"
2) "B"
3) "C"
4) "A"
5) "D"
6) "E"
7) "B"

      - ◆ LRANGE mylist 0 3

1) "A"
2) "B"
3) "C"
4) "A"

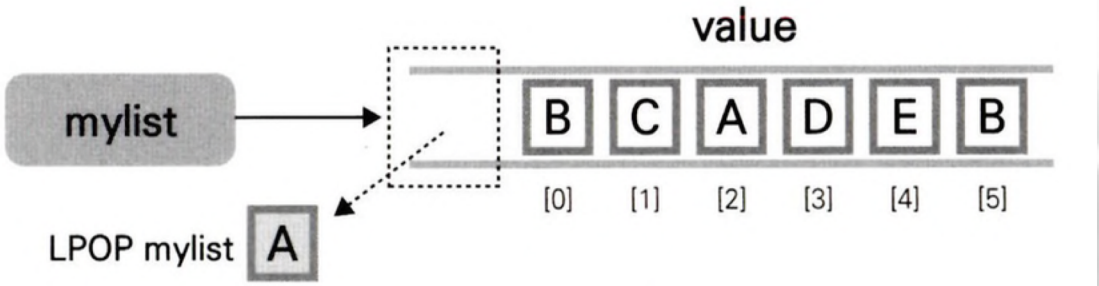
Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 왼쪽 과 오른쪽 끝의 데이터를 삭제하면서 조회: LPOP, RPOP
      - ◆ 횟수 설정 가능
      - ◆ LPOP mylist

      "A"
      - ◆ LPOP mylist 2

      1) "B"

      2) "C"



Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 범위를 제외하고 모든 데이터 삭제: LTRIM
      - ◆ LTRIM key start stop
      - ◆ LRANGE mylist 0 -1

      1) "A"

      2) "D"

      3) "E"

      4) "B"

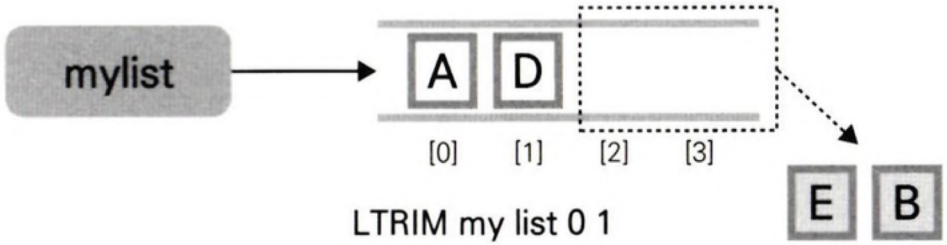
      - ◆ LTRIM mylist 0 1

      OK

      - ◆ LRANGE mylist 0 -1

      1) "A"

      2) "D"

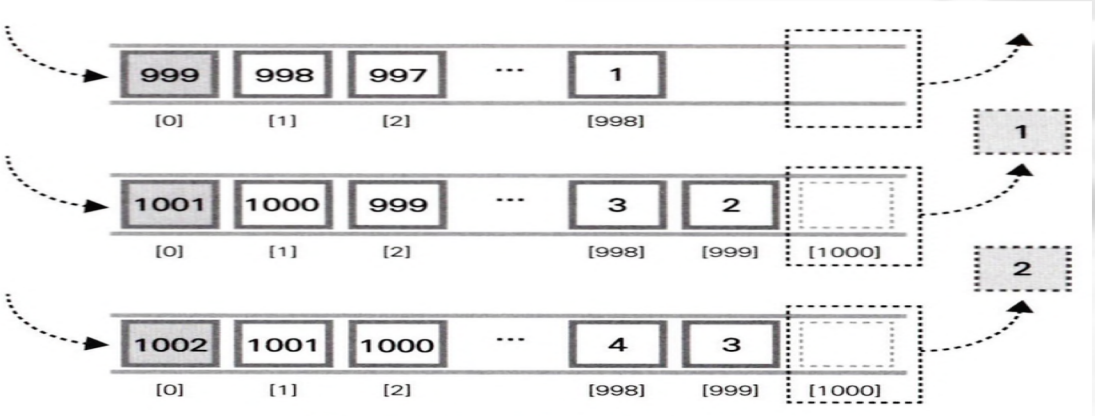


Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 범위를 제외하고 모든 데이터 삭제: LTRIM
      - ◆ 활용
        - ▪ LPUSH와 LTRIM 명령어를 함께 사용하면 고정된 길이의 큐를 쉽게 유지할 수 있음
        - ▪ list에 로그를 저장하는 경우 로그는 계속해서 쌓이는 데이터이기 때문에 주기적으로 로그 데이터를 삭제해 저장 공간을 확보하는 것이 일반적
        - ▪ redis의 list에 최대 1,000 개의 로그 데이터를 보관하고 싶다면 데이터를 저장할 때 LPUSH와 LTRIM 명령을 함께 사용할 수 있음

          LPUSH logdata <data>

          LTRIM logdata 0 999



Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 중간에 데이터 삽입: LINSERT
      - ◆ 형식

        LINSERT mylist BEFORE 기준값 newValue

        LINSERT mylist AFTER 기준값 newValue
        - ▪ LINSERT 명령어는 원하는 데이터의 앞이나 뒤에 데이터를 추가할 수 있음
        - ▪ 앞에 추가하려면 BEFORE 옵션을 뒤에 추가하려면 AFTER 옵션을 추가
        - ▪ 지정한 데이터가 없으면 오류를 반환

Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 중간에 데이터 삽입: LINSERT
      - ◆ LINSERT mylist BEFORE D C

      (integer) 3
      - ◆ LINSERT mylist BEFORE C B

      (integer) 4
      - ◆ LINSERT mylist BEFORE B E

      (integer) 5
      - ◆ LRANGE mylist 0 -1

      1) "A"

      2) "E"

      3) "B"

      4) "C"

      5) "D"

Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 데이터 수정: LSET
      - ◆ 형식: LSET key index data
      - ◆ LSET mylist 2 F

      OK
      - ◆ LRANGE mylist 0 -1

      1) "A"

      2) "E"

      3) "B"

      4) "C"

      5) "D"

Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 특정 인덱스의 데이터 조회: LINDEX
      - ◆ 형식: LINDEX key index
      - ◆ LINDEX mylist 3

      "C"

Redis

- ❖ Collection(자료구조)
  - ✓ List
    - ● 특정 데이터 삭제: LREM
      - ◆ 형식: LREM key count value
      - ◆ LREM mylist 1 C

      (integer) 1
    - ● LLEN: 데이터의 총 갯수를 조회
      - ◆ LLEN mylist

      (integer) 4

Redis

- ❖ Collection(자료구조)
  - ✓ Set
    - Set Collection은 List Collection과 같이 하나의 key에 여러 value를 저장할 수 있으며 다른 점은 데이터가 입력된 순서와 상관없이 저장되고 중복되지 않음
    - 교집합, 합집합, 차집합 등의 집합 연산과 관련한 명령어를 제공
    - Sets에서는 value를 member 라고 부르며 데이터가 있는지 없는지 체크하는 용도로 사용

Redis

- ❖ Collection(자료구조)
  - ✓ Set
    - ● 데이터 추가
      - ◆ 형식 sadd (key1) (value1 value2 value3 ...)
      - ◆ SADD myset A

      (integer) 1
      - ◆ SADD myset A A A C B D D E F F F F G

      (integer) 6

Redis

- ❖ Collection(자료구조)
  - ✓ Set
    - ● 모든 데이터 조회
      - ◆ SMEMBERS (key1)
      - ◆ SMEMBERS myset

      1) "A"
      2) "C"
      3) "B"
      4) "D"
      5) "E"
      6) "F"
      7) "G"

Redis

- ❖ Collection(자료구조)
  - ✓ Set
    - ● 집합에 value가 있는지 확인
      - ◆ 형식: SISMEMBER (key1) (value1)
      - ◆ sismember myset A

      (integer) 1
      - ◆ sismember myset Z

      (integer) 0
    - ● 집합에서 무작위로 member를 조회(해당 member 삭제 안됨)
      - ◆ SRANDMEMBER (key1) (count)
      - ◆ SRANDMEMBER myset

      "A"
      - ◆ SRANDMEMBER myset 2

      1) "A"

      2) "D"

Redis

- ❖ Collection(자료구조)
  - ✓ Set
    - ● 집합에서 member를 삭제
      - ◆ 형식
        - ▪ SREM (key1) (value1)
        - ▪ SPOP (key1)
      - ◆ SREM myset B

        (integer) 1
      - ◆ SPOP myset

        "G"

Redis

- ❖ Collection(자료구조)
  - ✓ Set
    - ● 집합 연산
      - ◆ SADD set:111 A B C D E

      (integer) 5
      - ◆ SADD set:222 D E F G H

      (integer) 5

      - ◆ SINTER set:111 set:222

      1) "D"

      2) "E"

Redis

- ❖ Collection(자료구조)
  - ✓ Set
    - ● 집합 연산
      - ◆ SUNION set:111 set:222

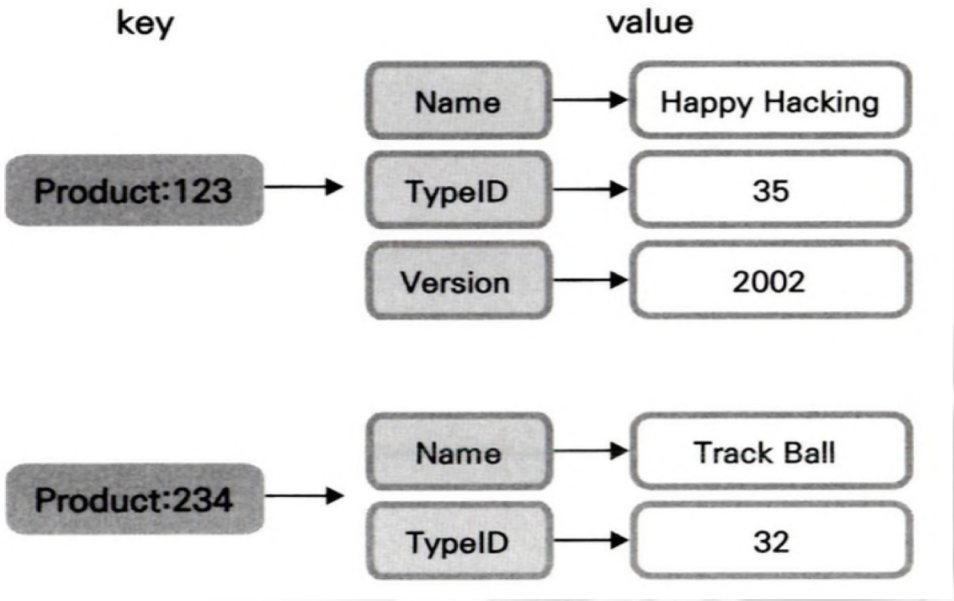
1) "G"
2) "C"
3) "B"
4) "A"
5) "E"
6) "D"
7) "F"
8) "H"

- ◆ SDIFF set:111 set:222

1) "C"
2) "B"
3) "A"

Redis

- Collection(자료구조)
  - ✓ Hash
    - Hash collection은 key 하나에 여러 개의 field와 value로 구성
    - 필드는 하나의 hash 내에서 유일하며 필드와 값 모두 문자열 데이터로 저장됨
    - hash에서는 각 아이템마다 다른 필드를 가질 수 있으며 동적으로 다양한 필드를 추가할 수 있다는 특징이 있음



Redis

- ❖ Collection(자료구조)
  - ✓ Hash
    - ● 아이템 저장: HSET key field value
      - ◆ HSET Product:123 Name "Happy Hacking"

      (integer) 1
      - ◆ HSET Product:123 TypeID 35

      (integer) 1
      - ◆ HSET Product:123 Version 2002

      (integer) 1
      - ◆ HSET Product:123 Name "Track Ball" TypeID 32

      (integer) 0

Redis

- ❖ Collection(자료구조)
  - ✓ Hash
    - ● 아이템 가져오기
      - ◆ 하나의 필드 가져오기: HGET key field
      - ◆ 여러 개의 필드 가져오기: HMGET key field [field]
      - ◆ 모든 필드 가져오기: HGETALL key
      - ◆ HGET Product:123 TypeID

      "32"
      - ◆ HMGET Product:123 Name TypeID

      1) "Track Ball"

      2) "32"
      - ◆ HGETALL Product:123

      1) "Name"

      2) "Track Ball"

      3) "TypeID"

      4) "32"

      5) "Version"

      6) "2002"

Redis

- ❖ Collection(자료구조)
  - ✓ Hash
    - ● key에 속한 모든 field/value 조회: HKEYS/HVALS
      - ◆ hkeys Product:123

      1) "Name"

      2) "TypeID"

      3) "Version"

      - ◆ hvals Product:123

      1) "Track Ball"

      2) "32"

      3) "2002"

Redis

- ❖ Collection(자료구조)
  - ✓ Hash
    - ● 아이템 삭제: HDEL <key> <field1> [field2 ...]
      - ◆ HDEL Product:123 Name

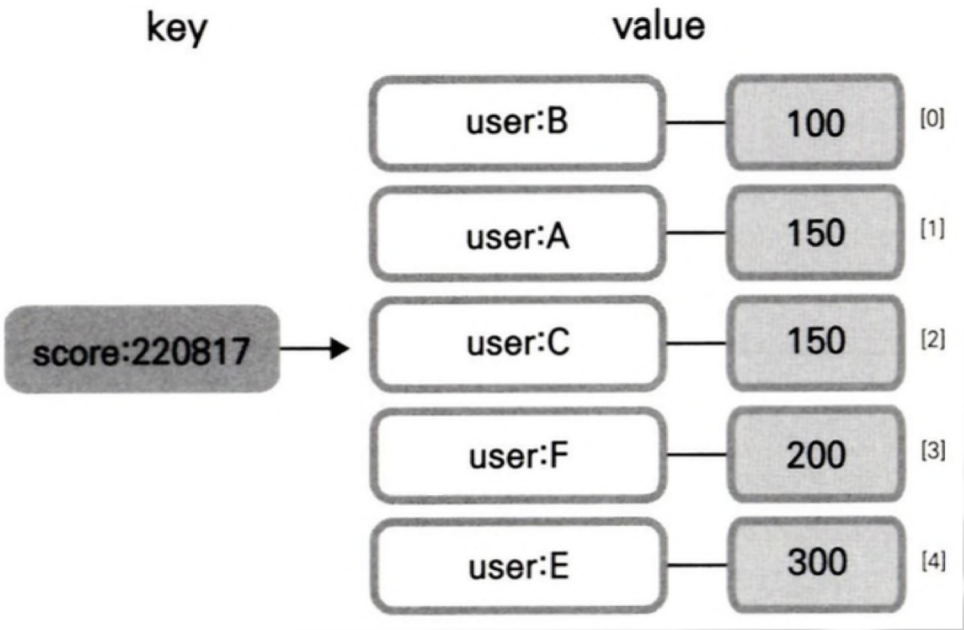
      (integer) 1
      - ◆ HDEL Product:123 TypeID Version

      (integer) 2
      - ◆ hkeys Product:123

      (empty array)

Redis

- Collection(자료구조)
  - ✓ Sorted Set(ZSet)



Redis

- Collection(자료구조)
  - ✓ Sorted Set(ZSet)
    - Sorted Set collection은 스코어5co값에 따라 정렬되는 고유한 문자열의 집합
    - 모든 아이템은 스코어-값 쌍을 가지며 저장될 때부터 스코어 값으로 정렬돼 저장
    - 같은 스코어를 가진 아이템은 데이터의 사전 순으로 정렬돼 저장
    - 데이터는 중복없이 유일하게 저장되므로 set과 유사하다고 볼 수 있으며 각 아이템은 스코어라는 데이터에 연결돼 있어 이 점에서 hash와 유사
    - 모든 아이템은 스코어 순으로 정렬돼 있어 list처럼 인덱스를 이용해 각 아이템에 접근할 수 있음
    - list와 sorted set 모두 순서를 갖는 자료 구조이므로 인덱스를 이용해 아이템에 접근할 수 있음
    - 배열에서 인덱스를 사용하는 것이 더 일반적이기 때문에 redis에서도 list에서 인덱스를 다루는 것이 더 빠를 것이라고 생각 할 수 있지만 인덱스를 이용해 아이템에 접근할 일이 많다면 list가 아닌 sorted set을 사용하는 것이 더 효율적
    - list에서 인덱스를 이용해 데이터를 접근하는 것은 O(n)으로 처리되지만 sorted set에서는 O(log(n))

Redis

- ❖ Collection(자료구조)
  - ✓ Sorted Set(ZSet)
    - ● 아이템 저장: ZADD key score value [score value]
      - ◆ 저장하고자 하는 데이터가 이미 sorted set에 속해 있다면 스코어만 업데이트 되고 업데이트된 스코어에 의해 아이템이 재정렬
      - ◆ 지정한 키가 존재하지 않을 때에는 sorted set 자료 구조를 새로 생성하며 키가 이미 존재하지만 sorted set이 아닐 경우에는 오류를 리턴
      - ◆ 스코어는 배정 밀도 부동소수점 숫자(double precision floating) point number를 문자열로 표현한 값이어야 함
      - ◆ 옵션
        - ▪ XX: 아이템이 이미 존재할 때에만 스코어를 업데이트
        - ▪ NX: 아이템이 존재하지 않을 때에만 신규 삽입하며 기존 아이템의 스코어를 업데이트하지 않음
        - ▪ LT: 업데이트하고자 하는 스코어가 기존 아이템의 스코어보다 작을 때에만 업데이트 하고 기존에 아이템이 존재하지 않을 때에는 새로운 데이터를 삽입
        - ▪ GT: 업데이트하고자 하는 스코어가 기존 아이템의 스코어보다 클 때에만 업데이트 하고 기존에 아이템이 존재하지 않을 때에는 새로운 데이터를 삽입

Redis

- ❖ Collection(자료구조)
  - ✓ Sorted Set(ZSet)
    - ● 아이템 저장: ZADD key score value [score value]
      - ◆ ZADD score:220817 100 user:B

      (integer) 1
      - ◆ ZADD score:220817 150 user:A 150 user:C 200 user:F 300 user:E

      (integer) 4

Redis

- ❖ Collection(자료구조)
  - ✓ Sorted Set(ZSet)
    - ● 아이템 조회: ZRANGE key start stop [BYSCORE | BYLEX] [REV] [LIMIT offset count] [WITHSCORES]
      - ◆ 인덱스로 조회: ZRANGE score:220817 1 3 WITHSCORES

1) "user:A"
2) "150"
3) "user:C"
4) "150"
5) "user:F"
6) "200”

- ◆ 인덱스로 조회: ZRANGE score:220817 1 3 WITHSCORES REV

1) "user:F"
2) "200"
3) "user:C"
4) "150"
5) "user:A"
6) "150"

Redis

- ❖ Collection(자료구조)
  - ✓ Sorted Set(ZSet)
    - ● 아이템 조회: ZRANGE key start stop [BYSCORE | BYLEX] [REV] [LIMIT offset count] [WITHSCORES]
      - ◆ 스코어로 조회: ZRANGE score:220817 100 150 BYSCORE WITHSCORES

1) "user:B"
2) "100"
3) "user:A"
4) "150"
5) "user:C"
6) "150"

- ◆ 스코어로 조회: ZRANGE score:220817 (100 150 BYSCORE WITHSCORES

1) "user:A"
2) "150"
3) "user:C"
4) "150"

Redis

- ❖ Collection(자료구조)
  - ✓ Sorted Set(ZSet)
    - ● 아이템 조회: ZRANGE key start stop [BYSCORE | BYLEX] [REV] [LIMIT offset count] [WITHSCORES]
      - ◆ 스코어로 조회: ZRANGE score:220817 100 +inf BYSCORE WITHSCORES

1) "user:B"
2) "100"
3) "user:A"
4) "150"
5) "user:C"
6) "150"
7) "user:F"
8) "200"
9) "user:E"
10) "300"

Redis

- ❖ Collection(자료구조)
  - ✓ Sorted Set(ZSet)
    - ZRANGEBYSCORE: score로 범위를 지정해서 조회 - zrangebyscore (key) (min) (max) (withscores) (limit offset count)
      - ◆ zrangebyscore score:220817 100 200

      1) "user:B"

      2) "user:A"

      3) "user:C"

      4) "user:F"
    - ZREVRANGEBYSCORE: score로 범위를 지정해서 큰 것부터 조회 - zrevrangebyscore (key) (max) (min) (withscores) (limit offset count)
      - ◆ zrevrangebyscore score:220817 200 100

      1) "user:F"

      2) "user:C"

      3) "user:A"

      4) "user:B"

Redis

- ❖ Collection(자료구조)
  - ✓ Sorted Set(ZSet)
    - ZSCORE: member를 지정해서 score를 조회 - zscore (key) (member)
      - ◆ zscore score:220817 user:F

      "200"
    - ZREM: 집합에서 member 삭제 - zrem (key) (member)
      - ◆ zrem score:220817 user:F

      (integer) 1
    - ZREMRANGEBYSCORE: score로 범위를 지정해서 member를 삭제 - zremrangebyscore (key) (min) (max)
      - ◆ zremrangebyscore score:220817 150 200

      (integer) 2

Redis

- Collection(자료구조)
  - ✓ Bitmap
    - 비트맵은 독자적인 자료 구조는 아니며 string 자료 구조에 bit 연산을 수행할 수 있도록 확장한 형태
    - string 자료 구조가 binary safe하고 최대 512MB의 값을 저장할 수 있는 구조이기 때문에 2^{32}의 비트를 가지고 있는 비트맵 형태라고 볼 수 있는 것
    - 비트맵을 사용할 때의 가장 큰 장점은 저장 공간을 획기적으로 줄일 수 있다는 것인데 각각의 유저가 정수 형태의 ID로 구분되고 전체 유저가 40억이 넘는다고 해도 각 유저에 대한 y/n 데이터는 512MB 안에 충분히 저장할 수 있음
    - SETBIT로 비트를 저장할 수 있으며 GETBIT 명령어로 저장된 비트를 조회할 수 있음
    - 한 번에 여러 비트를 SET하려면 BITFIELD 명령어를 사용

Redis

- ❖ Collection(자료구조)
  - ✓ Bitmap
    - ● 저장 과 1의 개수 세기
      - ◆ SETBIT mybitmap 2 1

      (integer) 0
      - ◆ GETBIT mybitmap 2

      (integer) 1
      - ◆ BITFIELD mybitmap SET u1 6 1 SET u1 10 1 SET u1 14 1

      1) (integer) 0

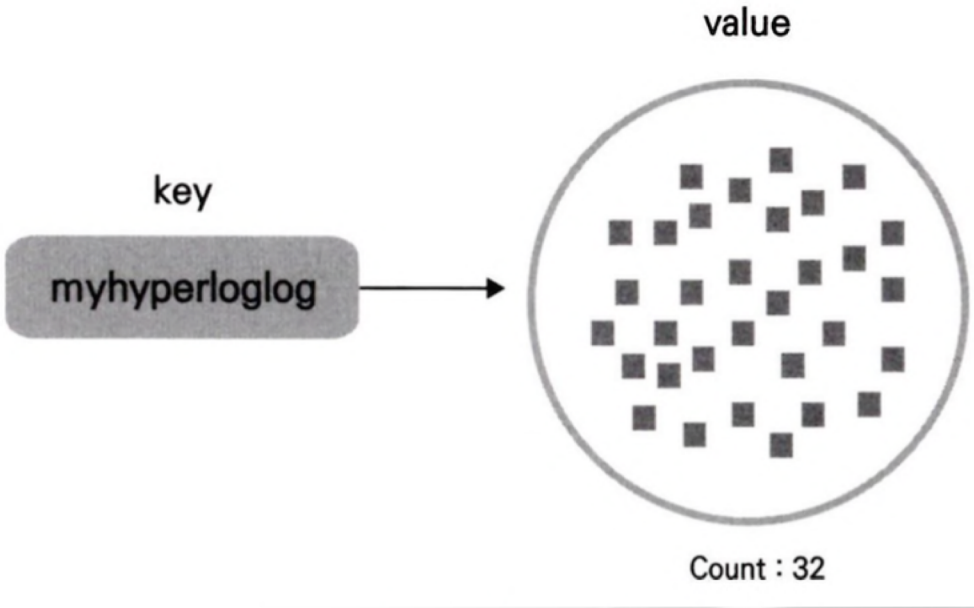
      2) (integer) 0

      3) (integer) 0
      - ◆ BITCOUNT mybitmap

      (integer) 4

Redis

- Collection(자료구조)
  - ✓ Hyperloglog



Redis

- Collection(자료구조)
  - ✓ Hyperloglog
    - hyperloglog는 집합의 원소 개수인 카디널리티를 추정할 수 있는 자료 구조
    - 대량의 데이터에서 중복되지 않는 고유한 값을 집계할 때 유용하게 사용할 수 있는 데이터 구조
    - 일반적으로 set과 같은 데이터 구조에서는 중복을 피하기 위해 저장된 데이터를 모두 기억하고 있기 때문에 저장되는 데이터가 많아질수록 그만큼 많은 메모리를 사용
    - hyperLoglog는 입력되는 데이터 그 자체를 저장하지 않고 자체적인 방법으로 데이터를 변경해 처리하기 때문에 hyperloglog 자료 구조는 저장되는 데이터 개수에 구애받지 않고 계속 일정한 메모리를 유지할 수 있으며 중복되지 않는 유일한 원소의 개수를 계산할 수 있음
    - 하나의 hyperloglog 자료 구조는 최대 12KB의 크기를 가지며 redis에서 카디널리티 추정의 오차는 0.81%로 비교적 정확하게 데이터를 추정
    - hyperloglog에는 최대 18,446,744,073,709,551,616(2^)개의 아이템을 저장할 수 있음
    - PFADD 명령어로 hyperloglog에 아이 템을 저장할 수 있으며 PFCOUNT 명령어로 아이템의 개수 즉 카디널리티를 추정

Redis

- ❖ Collection(자료구조)
  - ✓ Hyperloglog
    - PFADD members 123

    (integer) 1
    - PFADD members 500

    (integer) 1
    - PFADD members 12

    (integer) 1
    - PFCOUNT members

    (integer) 3

Redis

- Collection(자료구조)
  - ✓ Geospatial

Redis

- ❖ Collection(자료구조)
  - ✓ Geospatial
    - Geospatial 자료 구조는 경도 와 위도 데이터 쌍의 집합으로 간편하게 지리 데이터를 저장할 수 있는 방법
    - 내부적으로 데이터는 sorted set 으로 하나의 자료 구조 안에 키는 중복돼 저장되지 않음
    - 데이터 추가
      - ◆ GEOADD travel 14.399698913595286 50.09924276349484 prague

      (integer) 1

      - ◆ GEOADD travel 127.0016985 37.5642135 Seoul -122.43454762275572 37.78530362582044 SanFrancisco

      (integer) 2
    - 데이터 조회
      - ◆ GEOPOS travel prague

      1) 1) "14.39969927072525"
         2) "50.0992415092729"

Redis

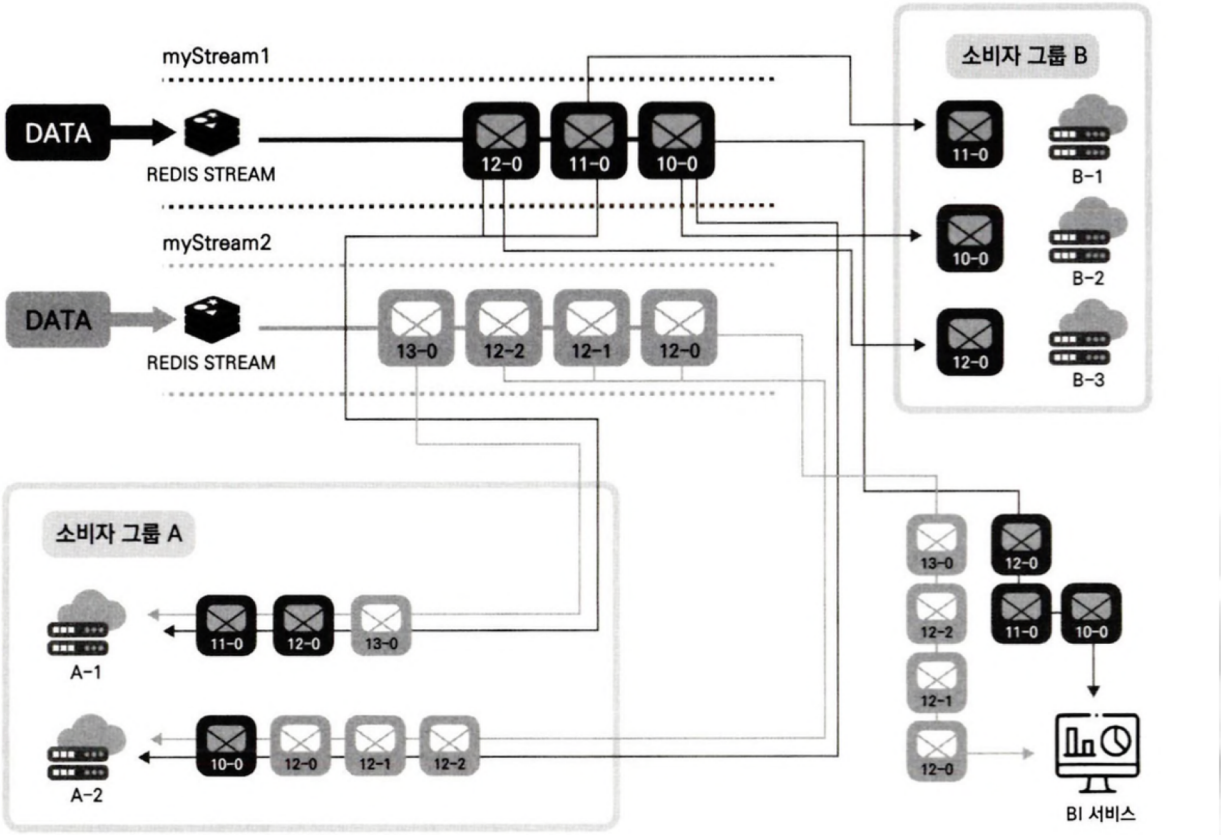
- ❖ Collection(자료구조)
  - ✓ Geospatial
    - ● 도시 사이의 거리
      - ◆ GEODIST travel Seoul prague KM

      "8252.9957"

      - ◆ GEOSEARCH 명령어를 이용하면 특정 위치를 기준으로 원하는 거리 내에 있는 아이템을 검색할 수 있는데 BYRADIUS 옵션을 사용하면 반경 거리를 기준으로 BYBOX 옵션을 사용하면 직사각형 거리를 기준으로 데이터를 조회할 수 있음

Redis

- Collection(자료구조)
  - ✓ Stream



Redis

- Collection(자료구조)
  - ✓ Stream
    - stream은 redis를 메시지 브로커로서 사용할 수 있게 하는 자료 구조
    - 전체적인 구조는 카프카에서 영향을 받아 만들어졌으며 카프카에서처럼 소비자 그룹 개념을 도입해 데이터를 분산 처리할 수 있는 시스템
    - stream 자료 구조는 데이터를 계속해서 추가하는 방식(append-only)으로 저장하므 실시간 이벤트 혹은 로그성 데이터의 저장을 위해 사용할 수 있음

Redis

- redis에서 키를 관리하는 방법
  - ✓ 키의 자동 생성 과 삭제
    - 키가 존재하지 않을 때 아이템을 넣으면 아이템을 삽입하기 전에 빈 자료 구조를 생성
    - 모든 아이템을 삭제하면 키도 자동으로 삭제
    - 키가 없는 상태에서 키 삭제, 아이템 삭제, 자료 구조 크기 조회 같은 읽기 전용 명령어를 수행하면 에러를 반환하는 대신 키가 있으나 아이템이 없는 것처럼 동작
      - ◆ DEL mylist

      (integer) 1

      - ◆ LLEN mylist

      (integer) 0

      - ◆ LPOP mylist

      (nil)

Redis

- ❖ redis에서 키를 관리하는 방법
  - ✓ 키와 관련된 명령
    - ● 키의 조회: EXISTS key
      - ◆ 키가 존재하면 1 존재하지 않으면 0
      - ◆ SET hello world

      OK
      - ◆ EXISTS hello

      (integer) 1
      - ◆ EXISTS world

      (integer) 0

Redis

- redis에서 키를 관리하는 방법
  - ✓ 키와 관련된 명령
    - 특정 패턴을 이용한 키의 조회: KEYS pattern
      - KEYS *: 모든 키 조회
      - h?llo는 hello, hallo가 매칭될 수 있음 – ?는 글자 수 1개
      - H*llo는 hllo, heeeello가 매칭될 수 있음 - *은 글자 수 제한 없음
      - h[ae]llo는 hello, hallo가 매칭될 수 있지만 hillo는 매칭되지 않음
      - h[^e]llo는 hallo, hbllo가 매칭될 수 있지만 hello는 매칭되지 않음
      - h[a-b]llo는 hallo, hbllo만 매칭될 수 있음

Redis

- redis에서 키를 관리하는 방법
  - ✓ 키와 관련된 명령
    - 특정 패턴을 이용한 키의 일정 부분 조회: SCAN cursor [MATCH pattern] [COUNT count] [TYPE type]
      - 첫번째 보여지는 데이터는 다음 커서이며 0이 리턴되면 더 이상 검색할 키 값이 없다는 것을 의미
      - SCAN 0

```
1) "0"
2) 1) "set:222"
   2) "hello"
   3) "members"
   4) "score:220817"
   5) "myset"
   6) "mybitmap"
   7) "travel"
   8) "set:111"
```

Redis

- redis에서 키를 관리하는 방법
  - ✓ 키와 관련된 명령
    - 데이터 정렬: SORT key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC I DESC] [ALPHA] [STORE destination]
      - LPUSH mylist c

      (integer) 1
      - LPUSH mylist a

      (integer) 2
      - LPUSH mylist b

      (integer) 3
      - LPUSH mylist Hello

      (integer) 4
      - SORT mylist

      (error) ERR One or more scores can't be converted into double
      - SORT mylist ALPHA

      1) "Hello"

      2) "a"

      3) "b"

      4) "c"

Redis

- redis에서 키를 관리하는 방법
  - ✓ 키와 관련된 명령
    - 이름 변경
      - RENAME key newkey
      - RENAMENX key newkey: 변경할 키가 존재하지 않을 때에만 동작
    - 복제
      - COPY source destination [DB destination-db] [REPLACE]
      - Source에 지정된 키를 destination 키에 복사하는데 이미 있는 경우 에러가 반환되는데 REPLACE 옵션을 사용하면 destination 키를 삭제한 뒤 값을 복사하기 때문에 에러가 발생하지 않음
    - 자료 구조 확인
      - Type key
    - 모든 키 삭제
      - FLUSHALL [ASYNC | SYNC]
    - 하나 이상의 키 삭제
      - DEL key [key]
      - UNLINK key [key]: 백그라운드에서 다른 스레드에 의해 처리

Redis

- Redis 유효 기간 설정
  - ✓ Redis에 저장되는 모든 데이터는 유효 기간을 설정할 수 있음
  - ✓ 유효 기간이 지난 데이터는 Redis가 해당 데이터를 메모리에서 삭제하므로 사용자의 별도 개입없이 메모리를 효율적으로 사용할 수 있음
  - ✓ 유효 기간 설정
    - EXPIRE 명령어를 사용하여 이미 생성된 데이터에 유효 기간을 설정하는 것으로 EXPIRE 명령어와 Redis 키를 인자로 입력
    - EXPIRE key seconds [NX | XX | GT | LT]
      - NX: 해당 키에 만료 시간이 정의돼 있지 않을 경우에만 명령어 수행
      - XX: 해당 키에 만료 시간이 정의돼 있을 때에만 명령어 수행
      - GT: 현재 키가 가지고 있는 만료 시간보다 새로 입력한 초가 더 클 때에만 수행
      - LT: 현재 키가 가지고 있는 만료 시간보다 새로 입력한 초가 더 작을 때에만 수행
    - EXPIREAT key unix-time-seconds [NX | XX | GT | LT]
      - 유닉스 타임스탬프에 만료될 수 있도록 설정

Redis

- ❖ Redis 유효 기간 설정
  - ✓ 유효 기간 설정
    - EXPIRETIME key
      - ◆ 키가 삭제되는 유닉스 타임스탬프를 초 단위로 반환
      - ◆ 만료 시간이 설정되어 있지 않으면 -1
      - ◆ 키가 없을 때는 -2
    - TTL
      - ◆ 키가 몇 초 뒤에 만료되는지 반환
      - ◆ 만료 시간이 설정되어 있지 않으면 -1
      - ◆ 키가 없을 때는 -2

Redis

- Redis 유효 기간 설정
  - ✓ 설정 시 고려 사항
    - Redis는 메모리에 데이터를 저장하므로 저장 공간이 한정적이기 때문에 Redis에 데이터를 저장할 때는 데이터의 유효 기간을 설정하는 것을 권장
    - 유효 기간이 만료된 데이터는 Redis가 직접 정리
    - 유효 기간을 설정하지 않는다면 개발자가 직접 데이터를 삭제할 때까지 영원히 유지되므로 애플리케이션에서 오래된 데이터를 마킹하고 직접 삭제하는 별도의 프로세스가 필요
    - 애플리케이션에서 오래된 데이터를 직접 삭제해야 한다면 다음 상황을 고려
      - Redis는 싱글 스레드로 사용자의 명령어를 처리하므로 짧은 시간에 Redis의 DEL 명령어를 사용하여 많은 데이터를 삭제한다면 서비스에서 실행한 명령어들은 메시지 큐에서 대기하는 시간이 길어질 수 있음
      - 데이터를 삭제하기 위해 Redis 키를 스캔하는 명령어(SCAN, HSCAN 등)는 고민해 보고 사용해야 하는데 실행 시간이 긴 명령어도 메시지 큐에 대기하고 있는 다른 명령어에 영향을 미침
      - 삭제 기능을 포함한 애플리케이션이 장애 상황이거나 배포가 데이터를 삭제하지 못한다면 Redis 데이터 사용량은 점점 증가