Working with Redis

Exercise Workbook

Contents

Lab 1: Redis Cli 연결하기 2

Lab 2: String 활용하기 4

Lab 3: Lists 활용하기 7

Lab 4: Sets 활용하기 9

Lab 5: Hashes 활용하기 11

Lab 6: Sorted Sets 활용하기 13

Lab 7: HyperLogLogs 활용하기 15

Lab 1: Redis Cli 연결하기

1. 현재 수행중인 docker 확인 (redis)

```
docker ps
```

```
jsjeong@JS-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED         STATUS         PORTS
               NAMES
0a1415d90e3d   redis:latest         "docker-entrypoint.s…"   9 minutes ago   Up 9 minutes   6379/tcp
               serene_mendeleev
493a9a9ac9bc   myapp-mongo:latest   "docker-entrypoint.s…"   4 hours ago     Up 4 hours     27017/tcp
               nervous_cray
c3fc8f841806   redis:latest         "docker-entrypoint.s…"   4 hours ago     Up 4 hours     6379/tcp
               interesting_heyrovsky
bf9eddd50ea8   myapp-mongo:latest   "docker-entrypoint.s…"   4 hours ago     Up 4 hours     27017/tcp
               trusting_margulis
a757dcb836c9   myapp-app:latest     "docker-entrypoint.s…"   4 hours ago     Up 4 hours
               practical_noether
6055e576a8f3   myapp-mongo          "docker-entrypoint.s…"   16 hours ago    Up 16 hours    0.0.0.0:270
17->27017/tcp  myapp-mongo-1
```

2. *Redis docker 가 안보이는 경우에는 이전 랩에서와 같이* ***docker-compose up –build*** *를 해서 생성하시기 바랍니다. (Option)*

3. Redis docker 의 컨테이너 id 를 확인해서 도커에 연결한다. (화면에 하이라이트 된 부분의 ID)

```
docker exec -it <Container ID> bash
```

```
jsjeong@JS-MacBook-Pro ~ % docker exec -it 0a1415d90e3d bash
root@0a1415d90e3d:/data#
```

4. redis-cli 명령 수행하여 Redis 서버에 연결하기

```
redis-cli
```

4.1 cli 가 연결이 되면 프롬프트가 보이고 redis 명령어를 수행할 수 있다.

```
root@0a1415d90e3d:/data# redis-cli
127.0.0.1:6379> HGETALL myhash
1) "field3"
2) "value3"
127.0.0.1:6379>
```

Lab 2: String 활용하기

1. 문자열 값 설정하기

```
SET mykey "Hello, Redis!"
```

위 명령은 키 `mykey`에 값 `"Hello, Redis!"`를 설정합니다.

2. 만료 시간이 있는 값 설정:

```
SET mykey "Hello, Redis!" EX 3600
```

위 명령은 키 `mykey`에 값 `"Hello, Redis!"`를 설정하고 만료 시간을 3600 초(1 시간)로 설정합니다.

3. 한 번에 여러 개의 키-값 설정:

```
MSET key1 "value1" key2 "value2" key3 "value3"
```

위 명령은 `key1`에 `"value1"`, `key2`에 `"value2"`, `key3`에 `"value3"` 값을 설정합니다.

4. 키가 존재하지 않을 때만 값을 설정:

```
SETNX mykey "Hello, Redis!"
```

위 명령은 키 `mykey`가 존재하지 않을 때만 값을 `Hello, Redis!"`로 설정합니다.

5. 값을 설정하고 이전 값을 반환:

```
GETSET mykey "New Value"
```

위 명령은 키 `mykey`에 값을 `"새로운 값"`으로 설정하고 이전 값(Hello Redis!)을 반환합니다.

물론, 계속해서 Redis CLI 에서 `SET` 명령어의 예시를 소개해드리겠습니다:

6. 특정 키의 값에 문자열을 추가:

```
APPEND mykey ", Redis!"
```

위 명령은 키 `mykey`에 현재 값에 ", Redis!" 문자열을 추가합니다. → “New Value, Redis!” 값을 갖는다.

7. 값의 일부를 가져오기:

```
GETRANGE mykey 0 4
```

위 명령은 키 `mykey`의 값에서 인덱스 0 부터 4 까지의 부분 문자열을 가져옵니다.

“New V” 출력

8. 특정 키의 값 존재 여부 확인:

```
EXISTS mykey
```

위 명령은 키 `mykey`의 값이 존재하는지 확인합니다. 결과로 1 이면 값이 존재하고, 0 이면 값이 존재하지 않습니다.

9. 값의 길이 확인:

```
STRLEN mykey
```

위 명령은 키 `mykey`의 값의 길이를 반환합니다.

10. 키의 값을 증가시키기:

```
INCR mykey
```

위 명령은 키 `mykey`의 값을 1 씩 증가시킵니다. 값이 정수형이어야 합니다.

Mykey 는 에러가 발생함.

```
127.0.0.1:6379> INCR mykey
(error) ERR value is not an integer or out of range
```

11. 특정 키의 값을 업데이트:

```
SET mykey "New Value" XX
```

위 명령은 키 **`mykey`** 의 값이 존재할 경우에만 값을 업데이트합니다. 값이 존재하지 않으면 업데이트하지 않습니다.

Lab 3: Lists 활용하기

1. Push a value to the head (left) of a list:

```
LPUSH mylist "value1"
```

이 명령은 "mylist" 키를 사용하여 "value1"을 목록의 헤드로 푸시합니다.

2. Push multiple values to the head (left) of a list:

```
LPUSH mylist "value2" "value3"
```

이 명령은 "value2" 및 "value3"을 키 "mylist"를 사용하여 목록의 헤드로 푸시합니다.

3. Push a value to the tail (right) of a list:

```
RPUSH mylist "value4"
```

이 명령은 "value4"를 키 "mylist"를 사용하여 목록의 끝에 푸시합니다.

4. Push multiple values to the tail (right) of a list:

```
RPUSH mylist "value5" "value6"
```

이 명령은 "value5" 및 "value6"을 키 "mylist"를 사용하여 목록의 끝에 푸시합니다.

5. Retrieve the length of a list:

```
LLEN mylist
```

이 명령은 "mylist" 키가 있는 목록의 길이를 반환합니다.

6. Retrieve elements from a list by index:

```
LRANGE mylist 0 2
```

이 명령은 "mylist" 키가 있는 목록에서 인덱스 0 부터 인덱스 2(포함)까지의 요소를 반환합니다.

7. Retrieve and remove the first element from a list:

```
LPOP mylist
```

이 명령은 "mylist" 키가 있는 목록에서 첫 번째 요소를 제거하고 반환합니다.

8. Retrieve and remove the last element from a list:

```
RPOP mylist
```

이 명령은 "mylist" 키가 있는 목록에서 마지막 요소를 제거하고 반환합니다.

9. Insert a value before or after a specific element in a list:

```
LINSERT mylist BEFORE "value3" "newvalue"
```

이 명령은 "mylist" 키가 있는 목록에서 "value3"이 처음 나타나는 앞에 "newvalue"를 삽입합니다.

10. Trim a list to keep only a specified range of elements:

```
LTRIM mylist 0 2
```

이 명령은 "mylist" 키를 사용하여 목록을 트리밍하여 인덱스 0 에서 인덱스 2(포함)까지의 요소만 유지합니다.

Lab 4: Sets 활용하기

1. Add elements to a set:

```
SADD myset "element1"
SADD myset "element2" "element3"
```

이 명령은 "myset" 키가 있는 세트에 "element1", "element2" 및 "element3"를 추가합니다.

2. Retrieve all elements of a set:

```
SMEMBERS myset
```

이 명령은 "myset" 키가 있는 집합의 모든 요소를 반환합니다.

3. Check if an element exists in a set:

```
SISMEMBER myset "element1"
```

이 명령은 "myset" 키가 있는 집합에 "element1"이 있는지 확인하고 1(true) 또는 0(false)을 반환합니다.

4. Get the cardinality (number of elements) of a set:

```
SCARD myset
```

이 명령은 "myset" 키가 있는 집합의 요소 수를 반환합니다.

5. Remove elements from a set:

```
SREM myset "element2"
SREM myset "element3" "element4"
```

이 명령은 "myset" 키가 있는 집합에서 "element2", "element3" 및 "element4"를 제거합니다.

6. Get the union of multiple sets:

```
SUNION myset1 myset2
```

이 명령은 집합 "myset1" 및 "myset2"의 합집합(즉, 각 집합에 있는 모든 고유 요소)을 반환합니다.

7. Get the intersection of multiple sets:

```
SINTER myset1 myset2
```

이 명령은 집합 "myset1"과 "myset2"의 교집합(즉, 두 집합에 있는 공통 요소)을 반환합니다.

8. Get the difference between two sets:

```
SDIFF myset1 myset2
```

이 명령은 "myset1"에는 있지만 "myset2"에는 없는 요소(즉, 집합 간의 차이)를 반환합니다.

9. Move an element from one set to another:

```
SMOVE myset1 myset2 "element1"
```

이 명령은 "element1"을 "myset1"에서 "myset2"로 이동합니다.

Lab 5: Hashes 활용하기

1. Set multiple field-value pairs in a hash:

```
HMSET myhash field1 value1 field2 value2 field3 value3
```

이 명령은 키 "myhash"를 사용하여 해시에서 필드-값 쌍 "field1:value1", "field2:value2" 및 "field3:value3"을 설정합니다.

2. Get the value of a specific field in a hash:

```
HGET myhash field1
```

이 명령은 키 "myhash"를 사용하여 해시에서 "field1"의 값을 검색합니다.

3. Get all field-value pairs in a hash:

```
HGETALL myhash
```

이 명령은 "myhash" 키가 있는 해시의 모든 필드-값 쌍을 반환합니다.

4. Check if a field exists in a hash:

```
HEXISTS myhash field1
```

이 명령은 "myhash" 키가 있는 해시에 "field1"이 있는지 확인하고 1(true) 또는 0(false)을 반환합니다.

5. Get all fields in a hash:

```
HKEYS myhash
```

이 명령은 "myhash" 키가 있는 해시의 모든 필드(키)를 반환합니다

6. Get all values in a hash:

```
HVALS myhash
```

이 명령은 "myhash" 키가 있는 해시의 모든 값을 반환합니다.

7. Increment a numeric field in a hash:

```
HINCRBY myhash field1 5
```

이 명령은 "myhash" 키가 있는 해시의 "field1" 값을 5 씩 증가시킵니다.

8. Delete one or more fields in a hash:

```
 HDEL myhash field1 field2
```

이 명령은 키가 "myhash"인 해시에서 "field1" 및 "field2"를 삭제합니다.

Lab 6: Sorted Sets 활용하기

Certainly! Here are some examples of using sorted set structures in Redis CLI:

1. Add members to a sorted set with associated scores:

```
ZADD mysortedset 10 member1 5 member2 7 member3
```

이 명령은 3 개의 멤버("member1", "member2" 및 "member3")를 각각 점수가 10, 5 및 7 인 정렬된 세트 "mysortedset"에 추가합니다.

2. Get the rank of a member in a sorted set:

```
ZRANK mysortedset member2
```

이 명령은 정렬된 집합 "mysortedset"에서 "member2"의 순위를 검색합니다. 순위는 세트가 오름차순으로 정렬되었을 때 멤버의 위치입니다.

3. Get the score of a member in a sorted set:

```
ZSCORE mysortedset member1
```

이 명령은 정렬된 집합 "mysortedset"에서 "member1"의 점수를 검색합니다.

4. Get a range of members from a sorted set by their ranks:

```
ZRANGE mysortedset 0 2
```

이 명령은 정렬된 집합 "mysortedset"에서 0 에서 2(포함) 범위의 순위를 가진 구성원을 검색합니다. 멤버는 점수에 따라 오름차순으로 반환됩니다.

5. Get a range of members from a sorted set by their scores:

```
ZRANGEBYSCORE mysortedset 5 8
```

이 명령은 정렬된 세트 "mysortedset"에서 점수 범위가 5 에서 8(포함)인 구성원을 검색합니다.

6. Increment the score of a member in a sorted set:

```
ZINCRBY mysortedset 3 member1
```

이 명령은 정렬된 집합 "mysortedset"에서 "member1"의 점수를 3 씩 증가시킵니다.

7. Remove one or more members from a sorted set:

```
ZREM mysortedset member2 member3
```

이 명령은 정렬된 집합 "mysortedset"에서 "member2" 및 "member3"을 제거합니다.

Lab 7: **HyperLogLogs** 활용하기

1. Add elements to a HyperLogLog:

```
PFADD myloglog element1 element2 element3
```

이 명령은 HyperLogLog 구조 "myloglog"에 "element1", "element2" 및 "element3"를 추가합니다.

2. Get the approximate cardinality of a HyperLogLog:

```
PFCOUNT myloglog
```

이 명령은 HyperLogLog "myloglog"에 있는 고유 요소의 대략적인 수를 반환합니다.

3. Merge multiple HyperLogLogs into a destination HyperLogLog:

```
PFMERGE destinationloglog myloglog1 myloglog2 myloglog3
```

이 명령은 HyperLogLog "myloglog1", "myloglog2" 및 "myloglog3"을 대상 HyperLogLog "destinationloglog"로 병합합니다. 결과 HyperLogLog 에는 병합된 모든 HyperLogLog 의 고유한 요소가 포함됩니다.