공학
트렌드

소프트웨어공학센터 경영지원TF팀

# 대용량 서비스를 지탱하는 분산 캐시 시스템
# Part 2 : Redis 분산 캐시 시스템 소개

2015. 2. 24. [123호]

Contents

Ⅰ. Redis 캐시와 API 소개
Ⅱ. Redis 적용 사례

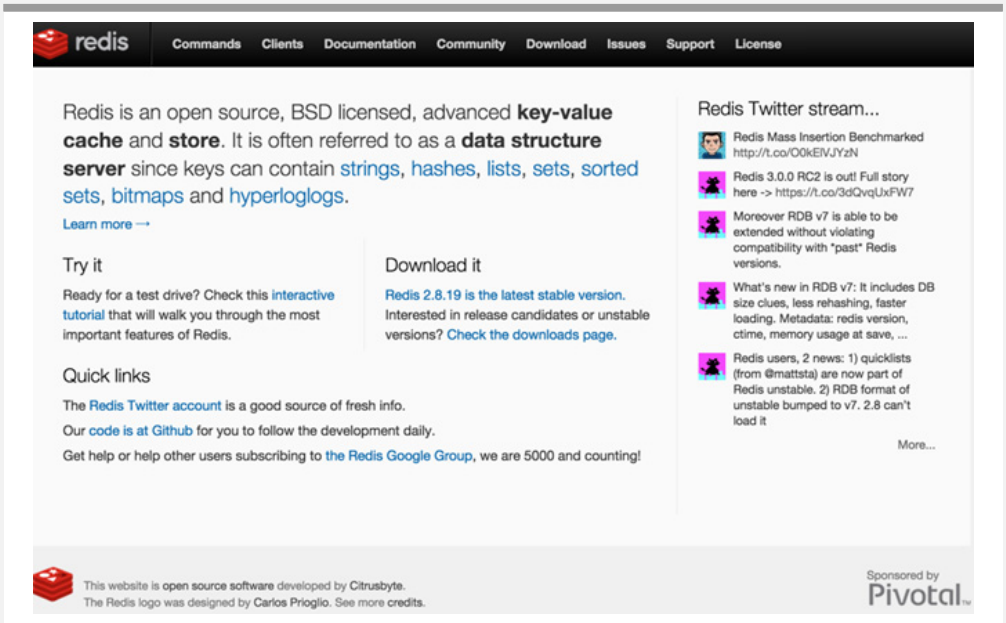
정보통신산업진흥원 부설
nipa 소프트웨어공학센터

# Ⅰ. Redis 캐시와 API 소개

## 1.1 소개

Redis는 REmote DIctionary Server의 약자이며, 웹 로그 분석 시스템[1]의 성능을 높이기 위해서 개발되었다. BSD 라이선스(Three Clause BSD License) 오픈소스 기반이다. DB-Engine 선호도 랭킹[2]에 따르면 선호도 1위를 기록하고 있다. 〈그림 1〉은 Redis 홈페이지의 첫 화면이다. 심플한 화면 구성을 가지고 있다. 문서와 다운로드 링크, 명령어, 언어별 클라이언트, 이슈 트래킹 화면, 지원 정보, 라이선스 등을 포함하고 있다.

**그림 1_Redis 홈페이지 메인 화면**




출처 : http://redis.io

Redis 서버는 BSD 라이선스 기반의 오픈소스이며, 키-값(key-value) 캐시이다. 키-값 캐시뿐 아니라 다양한 데이터 타입(strings, hashes, lists, sets, sorted sets, bitmaps, hyperloglogs)을 저장할 수 있다. 각 타입에 대해서 원자 작업(atomic operation)을 실행할

1) https://github.com/antirez/lloogg
2) http://db-engines.com/en/ranking_trend/key-value+store

수 있다. 원자 작업, 다양한 타입을 지원하기 때문에 큐(Queue), 공유 메모리(Shared memory), 사전(Dictionary)으로 활용될 수 있다. 또한, 오픈 소스인 Redis를 확장한 다양한 오픈 소스들이 있다. 특히 Redis를 활용한 유명한 큐가 있는데, restmq, redis-queue, resque[3] 등이 있다. Redis는 단순히 인메모리(in memory) 캐시이기도 하지만 캐시 데이터를 DB처럼 파일로 저장할 수 있다. 이 특징 때문에 단순 분산 캐시 시스템이 아닌 NoSQL로 분류하기도 한다. 그리고, Redis는 Ansi C로 개발되었기 때문에 다양한 운영체제에 쉽게 포팅을 할 수 있어서 이식성이 매우 뛰어나다는 장점이 있다. 1개의 thread로 동작하는 구조로 되어 있지만, 성능이 좋아 많은 개발자들이 사용하고 있다.

Redis는 회사가 아닌 커뮤니티이다. 따라서 Redis 지원을 이메일로 받고 있다. 보다 더 강한 지원이 필요하면 Pivotal에 문의할 수 있다. Redis 개발에 대해서 2013년 5월 전까지는 VMWare가 지원했으나, 현재는 Pivotal에서 지원하고 있다. 현재 Redis에서는 다양한 언어로 된 클라이언트가 많이 지원되고 있다. C, C++, Java, Python, Ruby, Erlang, PHP등 수 많은 클라이언트와 추천 오픈소스가 있으니, 홈페이지를 참조하기 바란다[4]. 모든 클라이언트가 서버의 모든 기능을 포함하지 않고 있으니, 반드시 개발자가 원하는 기능이 클라이언트에 포함되었는지 반드시 확인하는 것이 좋다. 참고로 클라이언트는 서버와 통신할 수 있는 규약[5]에 따라 구현할 수 있다.

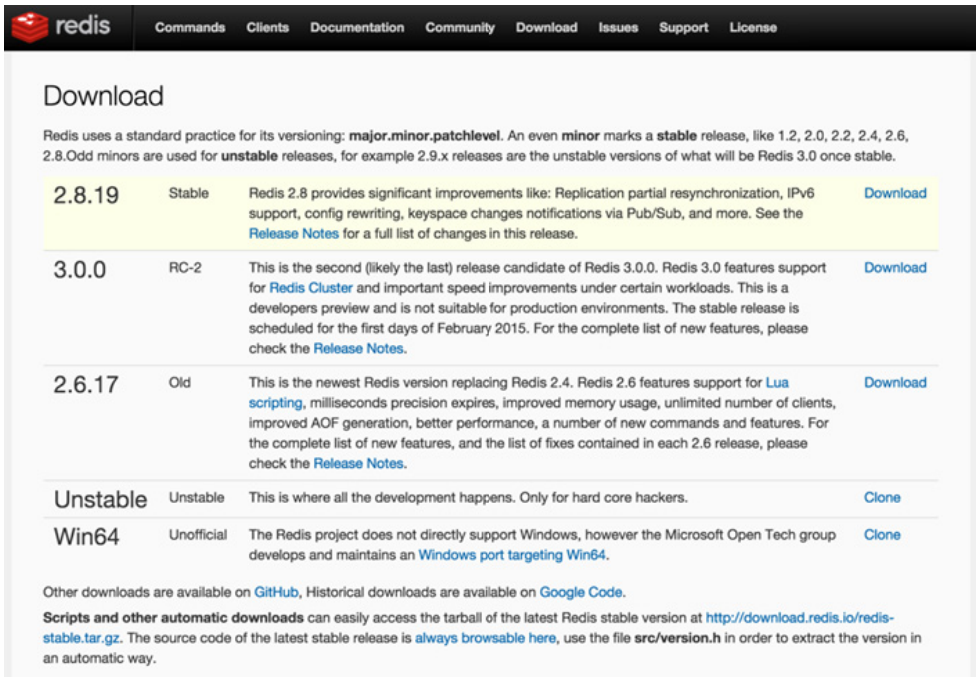
## 1.2 설치

〈그림 2〉페이지에 접속해서 버전을 확인하고 화면 우측에 있는 Download 링크를 클릭하여 Redis를 다운로드 한다. Redis 다운로드 파일은 소스로 되어 있기 때문에 압축을 풀고 make를 실행, 소스를 빌드한 후 바이너리를 설치한다. Mac이나 Linux 사용자 기준으로 설명한다.

```
$ wget http://download.redis.io/releases/redis-2.8.19.tar.gz
$ tar xzf redis-2.8.19.tar.gz
$ cd redis-2.8.19
$ make
```

3) restmq(http://restmq.com/), redis-queue(https://github.com/taganaka/redis-queue), resque(https://github.com/resque/resque)
4) http://www.redis.io/clients
5) http://redis.io/topics/protocol

그림 2_Redis 다운로드 페이지 화면




출처 : http://redis.io/download

컴파일 된 Redis 서버를 실행시킬 수 있는 redis-server 파일을 실행한다. 설정 파일이 없어서 경고(Warning) 로그가 있지만 실행을 위한 필수 조건은 아니다. 설정 파일이 없기 때문에 redis-server가 간단히 기본 설정을 지정한다. Redis 고유 마크가 출력되고 6379 포트가 오픈 된다.

```
$ src/redis-server
[55154] 26 Jan 21:14:31.590 # Warning: no config file specified, using the default config. In order to specify a config file use src/redis-server /path/to/redis.conf
[55154] 26 Jan 21:14:31.591 # You requested maxclients of 10000 requiring at least 10032 max file descriptors.
[55154] 26 Jan 21:14:31.591 # Redis can't set maximum open files to 10032 because of OS error: Operation not permitted.
[55154] 26 Jan 21:14:31.591 # Current maximum open files is 4096. maxclients has been reduced to 4064 to compensatefor low ulimit. If you need higher maxclients increase 'ulimit -n'.
                _.-``__ ''-._
           _.-``    `.  `_.  ''-._           Redis 2.8.19 (00000000/0) 64 bit
       .-`` .-```.  ```₩/    _.,_ ''-._
  (    '      ,       .-`  | `,    )     Running in stand alone mode
  |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
  |    `-._   `._    /     _.-'    |     PID: 55154
   `-._    `-._  `-./  _.-'    _.-'
  |`-._`-._    `-.__.-'    _.-'_.-'|
```

```
|    `-._`-._        _.-'_.-'    |                http://redis.io
  `-._    `-._`-.__.-'_.-'    _.-'
|`-._`-._    `-.__.-'    _.-'_.-'|
|    `-._`-._        _.-'_.-'    |
  `-._    `-._`-.__.-'_.-'    _.-'
      `-._    `-.__.-'    _.-'
          `-._        _.-'
              `-.__.-'

[55154] 26 Jan 21:14:31.592 # Server started, Redis version 2.8.19
[55154] 26 Jan 21:14:31.592 * The server is now ready to accept connections on port 6379
```

서버를 실행했으니, Redis 커맨드 클라이언트를 실행해본다. redis-cli 파일을 실행한다. 간단히 키-값 정보를 저장한다. 간단히 kimchi 라는 키를 1이라는 값으로 저장하고 kimchi 의 값을 읽어보자.

```
$ src/redis-cli
127.0.0.1:6379> set kimchi 1
OK
127.0.0.1:6379> get kimchi
"1"
```

Redis 서버를 종료하고 Redis 서버를 실행해서 다시 kimchi 키의 정보를 얻어올 수 있는지 확인한다. 좀 전에 src/redis-server를 실행한 터미널에서 Ctrl + C 를 입력해서 종료시키면 RDB 스냅샷이라는 파일에 저장되었다고 로그가 보일 것이다. Redis는 RDB 포맷으로 캐시 데이터를 파일에 저장한다. 기본 디폴트는 RDB 파일 저장 방식을 사용하고 있다.

```
^C
[55154 | signal handler] (1422274886) Received SIGINT scheduling shutdown...
[55154] 26 Jan 21:21:26.176 # User requested shutdown...
[55154] 26 Jan 21:21:26.176 * Saving the final RDB snapshot before exiting.
[55154] 26 Jan 21:21:26.177 * DB saved on disk
[55154] 26 Jan 21:21:26.177 # Redis is now ready to exit, bye bye...
```

다시 Redis 커맨드 클라이언트를 실행하면 전에 저장한 kimchi 키 정보를 읽어올 수 있다.

```
$ src/redis-cli
127.0.0.1:6379> get kimchi
"1"
```

Redis 커맨드 명령을 shell에서 실행할 수 있다. -h 옵션과 호스트명을 주면 다음과 같이 캐시 데이터를 다룰 수 있다.

```
$ src/redis-cli -h hostname set temp "x-men"
OK
```

참고로 서버 내부 구조에 대한 간략한 설명은 구글 웹캐쉬를[6] 참조한다.

6) http://webcache.googleusercontent.com/search?q=cache:http://www.enjoythearchitecture.com/redis-architecture

## 1.3 지원하는 데이터 타입

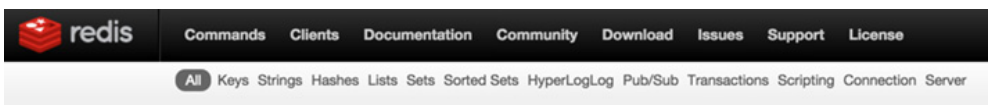
단순한 키-값(String) 뿐 아니라 Hash, List, Set, Bitmap, Hyperloglog를 지원한다. redis-cli를 통해 지원하는 데이터 타입을 확인할 수 있다. 만약 java 개발자라면 라이브러리인 jedis[7]를 사용하면 편리할 것이다. /src/test/java의 redis.clients.jedis.tests.commands 패키지를 참조하면 데이터 타입의 예제가 있을 것이다. 데이터 타입 테스트는 Redis서버를 실행한 후, redis-cli를 실행시켜서 테스트를 할 수 있다. 만약 서버 설치가 되어 있지 않더라도 〈그림 3〉처럼 웹으로 테스트해볼 수 있는 Redis 에뮬레이터를 활용할 수 있다.

**그림 3_Redis 웹 에뮬레이터**


출처 : http://try.redis.io/

명령어 집합은 페이지의 Commands를 클릭하면 명령어와 Redis의 다양한 기능을 볼 수 있다. 명령어가 상당히 많기 때문에 본 페이지에서 기술하지 않은 내용은 〈그림 4〉처럼 아래 명령어를 보면서 확인해야 한다.

**그림 4_Redis 명령어**




출처 : http://redis.io/commands

7) https://github.com/xetorthio/jedis

단순히 저장하는 것뿐 아니라 합집합(union), 교집합(inter), 집합 합치기(merge)등을 제공하는 것이 있으니 사용할 때마다 reference로 삼는 것이 좋다.

### 1.3.1 String

단순한 키-값을 저장할 수 있다. set으로 키-값을 저장하고, get으로 키에 할당된 값을 읽어올 수 있다. mget으로 여러 키의 값을 가져올 수 있다. keys 명령어를 이용해서 Redis에 저장된 key를 볼 수 있다. 참고로 redis-cli 실행 시에 ip와 포트가 127.0.0.1:6379>와 같이 디폴트로 출력된다.

```
$ redis-cli
127.0.0.1:6379> set user:phone "031-123-1234"
OK
127.0.0.1:6379> get user:phone
"031-123-1234"
127.0.0.1:6379> set user:id "nosql"
O
127.0.0.1:6379> get user:id
"nosql"
127.0.0.1:6379> mget user:id user:phone
1) "nosql"
2) "031-123-1234"
127.0.0.1:6379> keys *
1) "user:id"
2) "user:phone"
```

언어별 API를 사용하는 것도 동일하다. 예를 들어 java redis client인 jedis는 다음과 같이 사용할 수 있다.

```
JedisPool pool = new JedisPool(new JedisPoolConfig(), "localhost");

Jedis jedis = pool.getResource();

try {
        jedis.set("user:phone", "031-123-1234");
        jedis.set("user:id", "nosql");
        System.out.println(jedis.get("user:phone"));
        System.out.println(jedis.get(""user:id"));
        System.out.println(jedis.dbSize());
} finally {
    pool.returnResource(jedis);
}
pool.destroy();
```

### 1.3.2 Hash

Hash를 저장한다. hset과 hget으로 여러 개 값을 저장 또는 읽을 수 있다. hgetall을 이용해서 Hash 키에 대한 모든 값을 읽어올 수 있다. hincrby는 Hash에 키를 생성한 후, 주어진 값을 더한다.

```
127.0.0.1:6379> hset user id "nosql"
(integer) 1
127.0.0.1:6379> hset user phone "031-123-1234"
(integer) 1
127.0.0.1:6379> hgetall user
1) "id"
2) "nosql"
3) "phone"
4) "031-123-1234"
127.0.0.1:6379> hset user visitcount 0
(integer) 1
127.0.0.1:6379> hincrby user visitcount 1
(integer) 1
127.0.0.1:6379> hincrby user visitcount 30
(integer) 31
```

### 1.3.3 List

rpush로 List에 주어진 값을 저장하고 lrange로 List의 범위를 입력하여 List 값을 구할 수 있다. rpop으로 List의 내용을 뒤에서 빼낼 수 있다.

```
127.0.0.1:6379> rpush user:visitor "James"
(integer) 1
127.0.0.1:6379> rpush user:visitor "Jackson"
(integer) 2
127.0.0.1:6379> rpush user:visitor "Cony"
(integer) 3
127.0.0.1:6379> lrange user:visitor 0 -1
1) "James"
2) "Jackson"
3) "Cony"
127.0.0.1:6379> rpop user:visitor
"Cony"
127.0.0.1:6379> lrange user:visitor 0 -1
1) "James"
2) "Jackson"
```

### 1.3.4 Set

Set은 일반적인 Set과 Sorted Set 두 가지로 사용할 수 있다. 일반적인 Set의 명령어의 prefix는 s로 시작한다. sadd로 Set에 추가를 할 수 있고, spop으로 Set에서 랜덤하게 값을 끄집어내서 읽어올 수 있다. Set의 모든 값을 보려면 smembers를 이용한다.

```
127.0.0.1:6379> sadd user:friend "Jay"
(integer) 1
127.0.0.1:6379> sadd user:friend "Samuel"
(integer) 1
127.0.0.1:6379> sadd user:friend "Kyle"
(integer) 1
127.0.0.1:6379> smembers user:friend
1) "Samuel"
2) "Jay"
3) "Kyle"
127.0.0.1:6379> scard user:friend
(integer) 3
127.0.0.1:6379> spop user:friend
"Kyle"
```

Sorted Set의 명령어의 prefix는 z로 시작한다. Set에 점수(score)를 준 집합체이다. zadd 할 때 Sorted Set 이름과 점수를 주고 값을 저장한다. 그리고 zrange로 score별로 주어 Sorted Set의 순서를 얻어온다. 같은 점수일 때는 알파벳 순서대로 소팅된다.

```
127.0.0.1:6379> zadd user:ordered_friend 3 "Kyle"
(integer) 1
127.0.0.1:6379> zadd user:ordered_friend 1 "Samuel"
(integer) 1
127.0.0.1:6379> zadd user:ordered_friend 1 "Janet"
(integer) 1
127.0.0.1:6379> zrange user:ordered_friend 0 -1 withscores
1) "Janet"
2) "1"
3) "Samuel"
4) "1"
5) "Kyle"
6) "3"
```

### 1.3.5 HyperLogLog

2.8.9부터 새로 추가되었다. 하이퍼로그(HyperLog)는 추정된 값을 얻어올 수 있는 알고리즘을 사용한다. Hash나 Set를 사용하여 정확하게 저장하고 읽어올 수 있지만 속도가 느릴 수 있다. HyperLogLog를 이용하면 아주 정확하지는 않지만 적당한 추정 값으로 읽어올 수 있다. Redis에서 적용하는 HyperLogLog는 하이퍼로그 알고리즘을 기반으로 집합체로 쓸 수 있다. 자세한 내용은 참고자료 3번을 참조하라. pfadd는 HyperLogLog 집합체에 값을 추가한다. pfcount는 해당 집합체의 개수를 얻어온다.

```
127.0.0.1:6379> pfadd user:visitors "Samuel" "Jay" "Jedy" "Clark"
(integer) 1
127.0.0.1:6379> pfcount user:visitors
(integer) 4
```

### 1.3.6 TTL

지금까지 Redis에 저장한 데이터들은 모두 데이터의 유효기간(TTL, Time To Live)을 사용하지 않았다. expire를 이용하여 저장할 키의 유효기간을 지정할 수 있다. 단위는 초이다. 유효기간이 지나면 데이터는 자동으로 삭제된다. 만약 따로 expire을 주지 않으면 메모리에서 제거되지 않는다.

```
127.0.0.1:6379> set user "kim"
OK
127.0.0.1:6379> expire user 10
(integer) 1
127.0.0.1:6379> get user              # 10 초 전
"kim"
127.0.0.1:6379> get user              # 10 초 후
(nil)
```

## 1.4 관리 기능

Redis 설정과 관련된 내용을 볼 수 있는 명령어는 config 이다. set/get 방식으로 설정 값을 임의로 변경할 수 있다.

```
127.0.0.1:6379> config get dbfilename
1) "dbfilename"
2) "dump.rdb"
127.0.0.1:6379> config get *
...
```

info 명령어는 Redis status를 볼 수 있는 화면이다. 단순한 정보부터 replication 정보, 연결되어 있는 클라이언트 개수를 표현한다.

```
127.0.0.1:6379> info
# Server
redis_version:2.8.19
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:b57fd9d6bdddacbd
...
```

Redis로 요청하는 모든 명령어를 보고 싶으면 monitor 명령을 실행한다.

## 1.5 기타 기능

pub/sub 기능은 Message Queue 의 publish/subscribe 기능과 비슷하다. 한 클라이언트가

메세지를 Redis에 publish 하면 Redis에 subscribe 된 다수의 클라이언트에 메세지를 전달한다. 그 외 Redis는 transaction을 사용할 수 있고, lua 스크립트를 사용하여 코딩 내용을 넣을 수 있다.

## II. Redis 적용 사례

Redis는 운영하기 쉬운 편이라 많은 개발자들이 운영하고 있다. Redis 운영에 필요한 특징을 살펴본다.

### 2.1 저장 (Persistency)

Redis는 캐시 데이터를 파일(dataset)로 저장할 수 있다. 해당 파일을 바탕으로 Master-Slave 구조를 유지할 수 있고, Redis를 재시작 해야 하는 상황에 데이터 손실 없이 재시작 하고 데이터를 활용할 수 있다. 또한 복구를 빨리 할 수 있는 큰 장점을 가지고 있다. 이런 특징 때문에 Nosql로 분류하기도 한다. Redis에서는 파일을 저장하는 방식은 두 가지로 나눈다. 첫 번째는 메모리에 있는 캐시 데이터를 순간적으로 RDB 파일로 덤프하는 방식과 메모리에 있는 캐시 데이터를 변경할 때마다 저장하는 AOF(Append-Only File) 방식으로 나눌 수 있다.

RDB로 저장하는 방식은 Redis에서 임의의 프로세스를 fork(현재 실행 중인 프로세스를 복사해서 또 다른 프로세스를 실행하는 방식) 해서 메모리를 한 번에 많이 사용하여 캐시 데이터를 파일로 저장(snapshot)하게 하는 것이다. “save” 또는 “bgsave” 명령어를 이용해서 최소 몇 번 변경 시 매 몇 초마다 저장할 수 있게 설정할 수 있다. 완벽한 내구성은 보장하지 않는다. 따라서 메모리를 많이 사용 중일 때는 성능이 저하되기도 하지만, 백업 파일 때문에 문제가 일어날 때 바로 복구가 가능하다는 특성이 있다. AOF에 비해서 빨리 복구되는 장점이 있다.

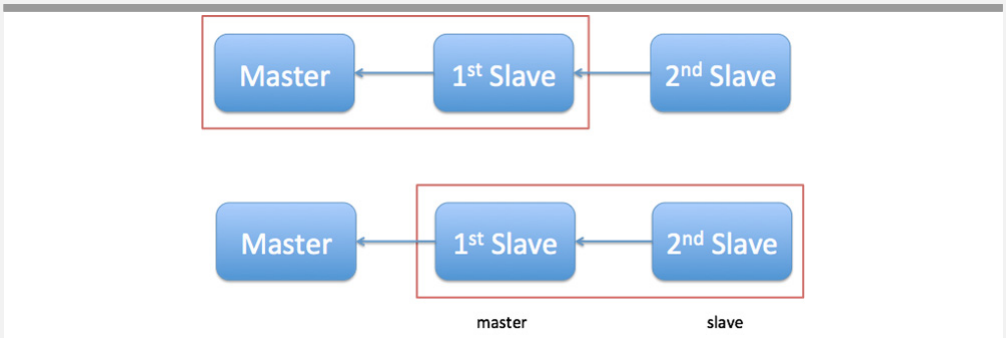
Master-Slave 구성하고 있는 구조에서 Master 또는 Slave 에 시스템 장애가 발생하여 사용할 수 없다면 RDB 파일을 만들고 복사해야 하는데, 이때 메모리가 2배 정도 늘어나도록 해야 한다. 따라서 이를 위해 메모리가 부족해서 swap 파일을 생성해야 할 수도

있는 상황이 생길 수 있다. 바로 이 때 느려질 수 있다. 또한 RDB 파일을 생성하다 실패하면 write 요청을 거절할 수 있기 때문에 "stop-writes-on-bgsave-error"의 값을 no로 하거나 RDB 저장을 하지 않도록 하면 된다. AOF 방식은 완전히 내구성을 지닐 수 있는 방안으로 Redis의 데이터가 변경된 이후 명령어의 로그가 저장된다. 설정 파일에서 appendonly라는 설정으로 확인할 수 있다. Write 요청 리퀘스트를 요청 그대로 AOF 파일로 저장하기 때문에 디스크의 한계로 파일 용량 제한이 이슈가 될 수 있다. 안정적인 특성을 가지고 있지만 요청 Write가 많은 상태에서 처리 중, 장애 발생 시 요청 리퀘스트를 저장하지 못함으로서 생기는 데이터 손실은 있을 수 있다. 그리고 fsync 정책에 따라 성능의 차이가 나는 것으로 알려져 있다.

## 2.2 복제 (Relication)

일반적인 Redis의 Replication은 Mysql DB와 비슷한 구조이다. Redis는 여러 개의 Slave를 가질 수 있다. 즉, Master-Slave 또는 Master-Slave-Salve 구조를 가질 수 있다. Non-blocking 이라 극소수 데이터는 언제나 불일치할 수 있는 부분이 있으니 데이터 일치성(Data Consistency)가 매우 중요하다면 쓸 수 없다. Redis 는 Master-Slave 구조만 가질 수 있다. Slave Redis에서 "slaveof" 명령어를 이용하여 Master Redis에 Sync를 하도록 되어 있다. 따라서 Master-Slave-Slave 구조는 〈그림 5〉와 같이 Master-Slave의 2개의 쌍으로 이루어진다.

그림 5_Redis Master-Slave-Slave 구조

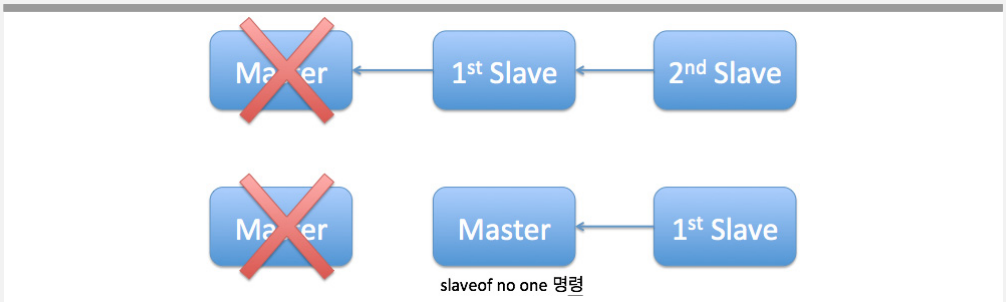



Slave의 디폴트 설정은 read only 모드이다. 따라서 write 되지 않도록 "slave-read-only" 값이 enable(yes)된다. 따라서 Slave에 write 리퀘스트 요청이 들어오면 저장이 되지 않도록 한다. 참고로 Redis의 info 명령어를 이용해서 정상적으로 replication 되었는지 확인할 수 있다.

## 2.3 장애 조치 (Failover)

중요한 부분은 Master의 데이터가 장애로 인해서 Data가 손실 될 경우, 묶여 있는 Slave의 데이터도 동시에 손실 될 수 있다. 이럴 때는 Master가 장애 시에는 첫 번째 Slave에 "slaveof no one"이라는 명령어를 사용하여 첫 번째 Slave가 새로운 Master가 되게 한다. 〈그림 6〉의 위 이미지는 Master-Slave-Slave 구조에서 Master가 장애가 난 것을 의미하고, 아래 이미지는 첫 번째 Slave에서 "slaveof no one" 명령어를 사용하여 새로운 Master가 된다는 것을 의미한다.

**그림 6_Redis Master-Slave-Slave 구조에서 Master 장애 시 장애 조치 방법**



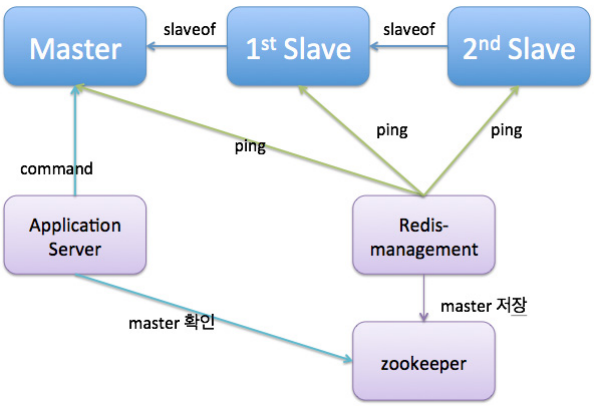

Master 장애 시 개발자가 수동으로 작업할 내용은 다음과 같다.

- 새로운 Master가 될 Redis 장비에서 slaveof no one 명령어를 실행한다.(slave read only 모드는 자동으로 disable 된다.)
- 새 Master 장비에 info 명령어로 master 설정이 role:master 인지 connected_slaves에 제대로 slave가 연결되었는지 확인한다. (Redis의 replication 설정이 제대로 되었는지 확인한다.)
- 애플리케이션 서버에서 Redis 연결하는 부분을 장애가 발생한 Master에서 새 Master로 변경한다.
- 애플리케이션 서버에서 새 Master 장비로 제대로 접속하는지 새 Master에서 info 명령어를 통해 connected_clients가 접속되었는지 확인하고 키 값이 올라가는지 확인한다. monitor 명령어를 이용하여 커맨드가 제대로 들어오는지 확인한다.

자동 장애 조치(Automatic Failover) 시스템은 위에 나열한 장애 조치를 자동으로 시스템에서 할 수 있도록 할 수 있다. 〈그림 7〉에 자동으로 장애를 처리할 수 있는 구조 중 하나를 설명한다. Redis-management 라는 서버를 두어 Redis 장비에 모니터링을 하여 status를 확인한다. 각 Redis 장비의 Master/Slave 정보를 zookeeper에 저장한다. 애플리케이션 서버는 zookeeper의 정보를 보고 Redis Master의 장비 이름을 확인하고 사용한다.

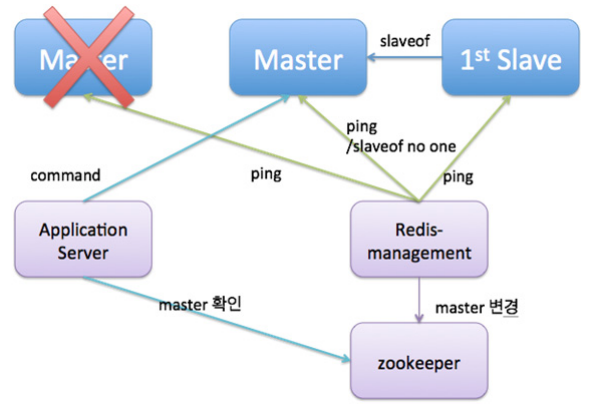
애플리케이션 서버는 Redis 장비 이름을 저장할 필요 없이 Zookeeper 주소만 알면 된다.

**그림 7_Redis 자동 장애 조치를 위한 구조**




〈그림 7〉의 Redis Master에서 하드웨어 손상으로 더 이상 서비스를 할 수 없다고 판단하면, Redis-management 서버에서 체크하고 첫 번째 Slave였던 Redis 장비를 Master로 승격한다. 이를 위해 Redis-management 서버에서 첫 번째 Slave에 "slaveof no one" 명령어를 내린다. 그리고 새로운 Master를 Zookeeper에 저장한다. 애플리케이션 서버는 Zookeeper를 보고 Redis Master가 변경되었다는 것을 이벤트로 받아 새로운 Master로 연결을 한다. 이 내용은 〈그림 8〉에서 보여주고 있다.

**그림 8_Redis 자동 장애 조치 구조에서 Redis Master 장애 시 새로운 Master로 변경되는 구조**




참고로 저자가 작성한 자동 장애 조치 구조 말고 TwemProxy와 HAProxy[8]를 사용할 수 있다. 참고로 Redis 2.8.19 버전에서는 Sentinel[9]을 포함하여 자동 장애 조치를 지원하고 있으나, 현재 산업계에서는 적용 초창기이라 활성화하기에는 좀 더 시간이 필요할 것 같다.

8) TwemProxy(https://github.com/twitter/twemproxy), HAProxy(http://www.haproxy.org)
9) http://redis.io/topics/sentinel

## 2.4 샤딩 (Sharding)

Redis 캐시 데이터를 1개의 Master가 아닌 여러 대의 Master로 분산시켜 대용량 트래픽을 처리하는 방법이다. 이 방법은 주로 3가지가 쓰인다. 계속 늘어날 수 있는 구조에서는 Range 방식을 추천한다.

- Range 방식: 순차적으로 키를 생성한 후, 키의 범위 안에 있는 데이터를 Redis에 저장하지만, 집중적인 트래픽으로 Redis에 부하를 줄 수 있다.
- Hash 방식: 키를 생성하고 키를 분산하는 해쉬 함수를 만들고 어느 Redis에 저장할 지를 결정하지만, migration이 어렵다.
- Index 방식: Hash 방식을 근간으로 하고 있지만 Index를 두어 어느 Redis에 저장할지 결정한다. Index 방식이기 때문에 migration이 Hash 방식에 비해서 쉽다.

## 2.5 키-값 스캐닝 이슈

Redis는 TTL(expire)을 주어 일정 시간이 지나면 자동으로 삭제할 수 있고, 애플리케이션 서버에서 수동으로 데이터를 지우게 할 수 있다. 데이터를 삭제할 때 너무 많은 데이터를 지우면 Redis에 많은 부하를 줄 수 있다. Redis가 단일 쓰레드만을 사용해서 동작하는 구조라, 데이터를 모두 접근하는(Full Scanning) 방식을 사용하는 keys 또는 flushall과 같은 커맨드는 데이터가 많으면 Redis에 부하를 줄 수 있으니, 조심스럽게 사용할 필요가 있다. 저자는 배치 작업을 진행하는 처리 용량이 많은 상태의 Redis에 redis-cli에서 smember(set의 요소를 출력) 커맨드를 실행했다가 커맨드 결과도 얻지 못하고 처리도 못하는 상태를 겪은 적이 있다.

## 2.6 설정 정보

Redis 설정을 환경에 맞게 써야 하니 무엇보다 중요하다. Redis에서 공식으로 내놓은 설정은 https://raw.githubusercontent.com/antirez/redis/2.8/redis.conf 이다. 참고로 내용에 대한 한글 설정 설명은 http://moss.tistory.com/150 을 참조하도록 한다. 설정 정보 없이 Redis 서버를 실행하면 디폴트 내용 때문에 성능 이슈를 부딪칠 수 있으니 항상 설정을

주고 아래와 같이 실행하는 것이 좋다.

```
$ redis-server redis.conf
```

- Maxmemory : 사용할 수 있는 최대 메모리 값
- maxclients : 클라이언트가 접속할 수 있는 최대 값
- save : RDB(snapshot) 사용 시 저장 주기
- 구조체(예, list, set)의 최대 키 크기

설정 정보 외 동적 정보를 설정할 수 있는 커맨드는 "config set" / "config get" 이다. master 정보, 로그 레벨, 메모리 LRU 정책, slave read only 모드 등과 같은 정적 설정을 동적 설정을 통해 정보를 변경할 수 있다. 예를 들어 timeout 정보를 동적으로 변경하고 볼 수 있다.

```
127.0.0.1:6379> config get timeout
1) "timeout"
2) "0"
127.0.0.1:6379> config set timeout 1000
OK
127.0.0.1:6379> config get timeout
1) "timeout"
2) "1000"
```

## 2.7 기타 이슈

분산 캐시의 공통된 단점인데, Redis는 간단하게 만들어진 만큼 메모리 파편화가 발생하고 남아있는 공간을 다 활용하지 못한다. 즉, 캐시 데이터 파편화를 정리하는 로직이 없다. 구조체의 최대 키 크기를 잘 설정해야 하고 Hash를 잘 활용해서 메모리를 적게 쓸 수 있다. 메모리를 관리하는 방법(Eviction 알고리즘 : LRU)을 활용할 수 있다.[10)]

Redis는 단일 쓰레드 기반의 프로세스로 실행하기 때문에 멀티 코어 CPU를 잘 활용할 수 없다. 성능을 최대로 높이기 위해서는 반드시 한 대의 물리 장비에 CPU 당 한 개의 Redis를 실행시킬 수 있다. 그러나 일반적인 한 개의 Redis에서 메모리를 많이 쓰는 운영 환경이 많이 잘 쓰이지 않는 듯하다.

10) http://redis.io/topics/memory-optimization, http://redis.io/topics/lru-cache

## Ⅲ. 결론

Redis는 단일 쓰레드로 동작하지만, 안정성과 성능이 좋은 편이고 분산 캐시 시스템이라 많은 개발자들이 활용하고 있다. 특히 사용할 수 있는 구조체가 많아 기능성이 풍부하다. 이번 Part 2에서는 Redis에 대한 구조체와 명령어를 살펴보았고, 운영 환경에서 필요한 설정, 복제, 장애 처리 등을 설명하였다. Part 3에서는 Memcached에 대해서 살펴보도록 하겠다.

### 참고 자료


1. 레디스 홈페이지 (http://redis.io)
2. 레디스 설명 (http://en.wikipedia.org/wiki/Redis)
3. 하이퍼 로그로그 (http://helloworld.naver.com/helloworld/textyle/7113010
4. 강대명님의 '대용량 서버 구축을 위한 Memcached와 Redis'(http://www.hanbit.co.kr/ebook/look.html?isbn=9788979149425)
5. redis 3.0 한글 번역(http://www.programkr.com/blog/MUzM3ADMwYT2.html)