Redis

Redis là một dự án cấu trúc dữ liệu trong bộ nhớ thực hiện cơ sở dữ liệu khóa-giá trị trong bộ nhớ phân tán. Redis hỗ trợ các loại cấu trúc dữ liệu trừu tượng khác nhau, chẳng hạn như chuỗi, danh sách, bản đồ, bộ, bộ được sắp xếp, HyperLogLogs, bitmap, luồng và chỉ mục không gian.

- Redis Pttl command - Lệnh kiểm tra thời gian hết hạn của Key trong Redis
- Redis TTL command - Lệnh kiểm tra thời gian hết hạn còn lại của Key trong Redis
- Redis RANDOMKEY command - Lệnh lấy một Key ngẫu nhiên trong Redis
- Redis Rename command - Lệnh rename Key trong Redis
- Redis Renamenx command - Lệnh rename Key trong Redis
- Redis Type command - Lệnh kiểm tra loại dữ liệu giá trị của Key trong Redis

- Redis Advanced
  - Redis backup - Tạo bản backup trong Redis
  - Redis Security - Bảo mật trong Redis database
  - Redis Benchmarks - Đo hiệu năng của Redis
  - Redis Client Connection - Quản lý kết nối từ Client
  - Redis Pipelining - Thực thi nhiều lệnh đồng thời từ Client
  - Redis Partitioning

- Redis guideline
  - Hướng dẫn cài đặt Redis Master - Slave trên CentOS 7
  - Hướng dẫn cài đặt Redis trên CentOS 7/6
  - Hướng dẫn kết nối Redis sử dụng Redisson trong Java

Redis Basics

Kiến thức cơ bản về Redis

Redis overview

Redis là gì?

Redis là tên viết tắt của Remote Dictionary Server (Máy chủ từ điển từ xa), là một phần mềm mã nguồn mở được dùng để lưu trữ một cách tạm thời trên bộ nhớ (hay còn gọi là cache data) và giúp truy xuất dữ liệu một cách nhanh chóng. Do tốc độ truy xuất dữ liệu vượt trội, Redis thường được chọn sử dụng cho hoạt động lưu trữ bộ nhớ đệm Caching, quản lý phiên, trò chơi, bảng xếp hạng, phân tích theo thời gian thực, dữ liệu không gian địa lý, ứng dụng đặt xe, trò chuyện/nhắn tin, phát trực tiếp nội dung phương tiện và pub/sub.

Lợi ích của Redis

1. Lưu trữ dữ liệu trên RAM

Toàn bộ dữ liệu Redis nằm trong bộ nhớ chính của máy chủ, trái với cơ sở dữ liệu, chẳng hạn như PostgreSQL, Cassandra, MongoDB... thường lưu phần lớn dữ liệu trên ổ đĩa hoặc ổ SSD. So với cơ sở dữ liệu trên ổ đĩa truyền thống trong đó phần lớn các tác vụ đều yêu cầu truy cập qua lại tới ổ đĩa, kho dữ liệu trong bộ nhớ chẳng hạn như Redis không phải chịu hình phạt này. Do đó kho dữ liệu kiểu này có thể hỗ trợ thêm được khá nhiều tác vụ và có thời gian phản hồi nhanh hơn. Kết quả là – hiệu suất nhanh thấy rõ với các tác vụ đọc hoặc ghi thông thường mất chưa đầy một mili giây và hỗ trợ hàng triệu tác vụ mỗi giây.

2. Cấu trúc dữ liệu linh hoạt

Khác với những kho dữ liệu khóa-giá trị đơn giản có cấu trúc dữ liệu giới hạn, Redis có nhiều cấu trúc dữ liệu khác nhau nên đáp ứng được nhu cầu ứng dụng của bạn. Kiểu dữ liệu Redis gồm có:

- Chuỗi – văn bản hoặc dữ liệu nhị phân có kích thước lên tới 512MB

- Danh sách – một tập hợp các Chuỗi được sắp xếp theo thứ tự như khi được thêm vào
- Tập – một tập hợp chưa được sắp xếp các chuỗi, có khả năng giao cắt, liên kết và khác với các kiểu Tập khác
- Tập được sắp xếp – Tập được sắp xếp theo giá trị
- Mã hash – một cấu trúc dữ liệu dùng để lưu trữ danh sách các trường và giá trị
- Bitmap – một kiểu dữ liệu cho phép thực hiện các tác vụ quy mô bit
- HyperLogLogs – một cấu trúc dữ liệu xác suất để ước tính các thành phần duy nhất trong một tập dữ liệu

3. Đơn giản và dễ sử dụng

Redis đơn giản hóa mã bằng cách cho phép bạn viết ít dòng lệnh hơn để lưu trữ, truy cập và sử dụng dữ liệu trên ứng dụng của bạn. Ví dụ: nếu ứng dụng của bạn có dữ liệu được lưu trên một bảng băm và bạn muốn lưu dữ liệu đó trên kho dữ liệu – bạn chỉ cần sử dụng cấu trúc dữ liệu mã hash của Redis để lưu dữ liệu đó. Tác vụ tương tự trên kho dữ liệu không có cấu trúc dữ liệu mã hash sẽ cần nhiều dòng mã để chuyển đổi từ định dạng này sang định dạng khác. Redis được trang bị cấu trúc dữ liệu riêng và nhiều tùy chọn để điều khiển và tương tác với dữ liệu của bạn. Trên một trăm máy khách mã nguồn mở được cung cấp cho nhà phát triển Redis. Các ngôn ngữ được hỗ trợ gồm có Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go và nhiều ngôn ngữ khác.

4. Sao chép và độ bền

Redis sử dụng kiến trúc bản sao-chính và hỗ trợ sao chép không đồng bộ trong đó có thể sao chép dữ liệu sang nhiều máy chủ bản sao. Việc này mang lại hiệu suất đọc cao hơn (vì có thể chia tách các yêu cầu giữa các máy chủ) và tốc độ khôi phục nhanh hơn khi máy chủ chính gặp sự cố. Về độ bền, Redis hỗ trợ sao lưu tại một thời điểm nào đó (chép tập dữ liệu Redis sang ổ đĩa).

5. Độ khả dụng cao và quy mô linh hoạt

Redis có kiến trúc bản sao-chính theo cấu trúc liên kết dạng một nút chính hoặc cụm. Kiến trúc này cho phép bạn xây dựng những giải pháp có độ khả dụng cao, đảm bảo hiệu suất ổn định và tin cậy. Khi cần điều chỉnh kích thước cụm, bạn có nhiều tùy chọn khác nhau để thay đổi quy mô theo chiều dọc và thay đổi quy mô theo chiều ngang. Việc này cho phép tăng cụm theo nhu cầu của bạn.

6. Khả năng mở rộng

Redis là dự án mã nguồn mở được một cộng đồng đông đảo ủng hộ. Không có giới hạn về nhà cung cấp hoặc công nghệ vì Redis được có tính tiêu chuẩn mở, hỗ trợ các định dạng dữ liệu mở và có tập hợp máy khách phong phú.

Trường hợp sử dụng phổ biến của Redis

1. Lưu trữ bộ nhớ đệm

Redis là lựa chọn tuyệt vời để triển khai một bộ nhớ đệm trong bộ nhớ có độ khả dụng cao để giảm độ trễ truy cập dữ liệu, tăng năng suất và giảm tải cho cơ sở dữ liệu quan hệ hoặc NoSQL và ứng dụng của bạn. Redis có thể phục vụ những mục dữ liệu thường xuyên được yêu cầu với thời gian

phản hồi chưa đến một mili giây và cho phép bạn dễ dàng thay đổi quy mô nhằm đáp ứng mức tải cao hơn mà không cần gia tăng phần backend có chi phí tốn kém hơn. Một số ví dụ phổ biến về nhớ đệm khi sử dụng Redis bao gồm nhớ đệm kết quả truy vấn cơ sở dữ liệu, nhớ đệm phiên lâu bền, nhớ đệm trang web và nhớ đệm các đối tượng thường xuyên được sử dụng như ảnh, tập tin và siêu dữ liệu.

2. Trò chuyện, nhắn tin và danh sách tác vụ chờ xử lý

Redis hỗ trợ Pub/Sub (cấu trúc gửi nhận tin nhắn trong đó người gửi và người nhận không biết nhau) với tính năng khớp cấu trúc và nhiều cấu trúc dữ liệu như danh sách, tập được sắp xếp và mã hash. Việc này cho phép Redis hỗ trợ các phòng trò chuyện hiệu suất cao, luồng bình luận theo thời gian thực, nguồn cấp mạng xã hội và giao tiếp giữa các máy chủ. Cấu trúc dữ liệu Danh sách của Redis giúp dễ dàng triển khai một danh sách tác vụ chờ xử lý có tải trọng nhẹ. Danh sách cung cấp các hoạt động nguyên tử cũng như tính năng chặn, giúp cho chúng phù hợp với nhiều ứng dụng yêu cầu phải có trình chuyển tiếp tin nhắn tin cậy hoặc danh sách liên kết vòng.

3. Bảng xếp hạng game

Redis là giải pháp hay được các nhà phát triển game dùng để xây dựng bảng xếp hạng theo thời gian thực. Chỉ cần sử dụng cấu trúc dữ liệu Tập được sắp xếp của Redis, cấu trúc dữ liệu này đảm bảo tính duy nhất của các thành phần trong khi vẫn duy trì danh sách được sắp xếp theo điểm số của người dùng. Tạo danh sách xếp hạng theo thời gian thực dễ thực hiện như khi cập nhật điểm số của người dùng mỗi khi có thay đổi. Bạn cũng có thể sử dụng Tập được sắp xếp để xử lý dữ liệu chuỗi thời gian bằng cách dùng dấu thời gian làm điểm số.

4. Kho lưu trữ phiên

Redis là kho dữ liệu trong bộ nhớ có độ khả dụng và độ bền cao, thường được các nhà phát triển ứng dụng sử dụng để lưu trữ và quản lý dữ liệu phiên cho các ứng dụng quy mô internet. Redis có độ trễ chưa đến một mili giây, có quy mô và độ đàn hồi cần thiết để quản lý dữ liệu phiên chẳng hạn như hồ sơ người dùng, thông tin xác thực đăng nhập, trạng thái phiên và tùy chỉnh theo ý muốn người dùng.

5. Phát nội dung giàu dữ liệu

Redis cung cấp kho dữ liệu trong bộ nhớ, có tốc độ truy cập nhanh để đáp ứng các trường hợp sử dụng phát trực tiếp. Có thể sử dụng Redis để lưu trữ siêu dữ liệu về hồ sơ người dùng và xem lịch sử, thông tin/mã thông báo xác thực cho hàng triệu người dùng và hiển thị tập tin để cho phép các Mạng truyền tải nội dung (CDN) phát video cho hàng triệu người dùng di động và máy tính để bàn cùng một lúc.

6. Dữ liệu không gian địa lý

Redis cung cấp cấu trúc dữ liệu trong bộ nhớ, được tích hợp sẵn cho mục đích cụ thể và các toán tử để quản lý dữ liệu không gian địa lý theo thời gian thực ở quy mô và tốc độ mong muốn. Các lệnh như GEOADD, GEODIST, GEORADIUS và GEORADIUSBYMEMBER để lưu trữ, xử lý và phân tích dữ liệu không gian địa lý theo thời gian thực giúp cho dữ liệu không gian địa lý trở nên dễ dàng và

nhanh chóng khi sử dụng Redis. Bạn có thể sử dụng Redis để thêm các tính năng dựa trên địa điểm như thời gian lái xe, quãng đường lái xe và các điểm quan tâm cho ứng dụng của bạn.

7. Machine Learning

Các ứng dụng kiểu mới, chịu sự chi phối của dữ liệu yêu cầu machine learning phải có khả năng nhanh chóng xử lý được dữ liệu theo khối lượng lớn, đa dạng, tốc độ cao và tự động hóa quá trình ra quyết định. Đối với các trường hợp sử dụng như phát hiện lỗi trong các dịch vụ game và tài chính, đấu thầu theo thời gian thực trong công nghệ quảng cáo và mai mối trong hẹn hò và đi chung xe, khả năng xử lý dữ liệu trực tiếp và ra quyết định trong vòng vài chục mili giây có ý nghĩa hết sức quan trọng. Redis cung cấp cho bạn kho dữ liệu trong bộ nhớ, có tốc độ truy cập nhanh để xây dựng, đào tạo và triển khai mô hình machine learning một cách nhanh chóng.

8. Phân tích theo thời gian thực

Có thể dùng Redis kết hợp với các giải pháp phát trực tuyến như Apache Kafka và Amazon Kinesis làm kho dữ liệu trong bộ nhớ để tiêu thụ, xử lý và phân tích dữ liệu thời gian thực với độ trễ chưa đến một mili giây. Redis là lựa chọn lý tưởng cho các trường hợp sử dụng phân tích theo thời gian thực chẳng hạn như phân tích mạng xã hội, nhắm mục tiêu quảng cáo, cá nhân hóa và IoT.

Redis Installation

Cài đặt Redis

Phần này sẽ hướng dẫn bạn cách redis sử dụng bộ cài đặt dành cho hệ điều hành Linux. Mở chương trình dòng lệnh (hay terminal) lên và chạy các câu lệnh sau (bạn cũng có thể chuyển vào thư mục Downloads trước khi chạy các câu lệnh này):

```
$ wget http://download.redis.io/redis-stable.tar.gz
```

Câu lệnh trên sẽ tải tập tin `redis-stable.tar.gz` về máy tính bạn. Bạn cần phải giải nén tập tin này:

```
$ tar xvzf redis-stable.tar.gz
```

Sau câu lệnh trên bạn sẽ thấy một thư mục có tên `redis-stable` được tạo ra trên thư mục bạn đang đứng để thực thi câu lệnh. Bạn có thể kiểm tra bằng cách chạy lệnh ls:

```
$ ls .
```

Tiếp theo chúng ta sẽ di chuyển vào bên trong thư mục redis-stable này và thực hiện công việc biên soạn ứng dụng để có thể thực thi trực tiếp.

```
$ cd redis-stable && make
```

Bạn sẽ cần cần một khoảng thời gian vài phút để kết thúc quá trình biên soạn ra ứng dụng. Sau khi kết thúc bạn có thể kiếm tra quá trình biên soạn bằng câu lệnh:

```
$ make test
```

Kết quả của quá trình biên soan ở trên bạn sẽ thấy trong thư mục src được thêm vào 6 tập tin khác nhau là: redis-server, redis-sentinel, redis-cli, redis-benchmark, redis-check-aof và redis-check-dump. Ở đây chúng ta chỉ quan tâm 2 tập tin chính là redis-server và redis-cli. Hai tập tin này lần lượng dùng để chạy Redis server và Redis client.

Vì đây là 2 tập tin thực thi lên để tránh việc phải gõ toàn bộ đường dẫn tham chiếu tới chúng khi chạy câu lệnh sử dụng 2 chương trình này thì chúng ta lên chép chúng vào một trong các thư mục của biến $PATH. Thông thường, trên hầu hết các phiên bản của Linux thì biến $PATH sẽ bao gồm thư mục /usr/local/bin. Chúng ta sử dụng 2 câu lệnh sau để chép 2 tập tin trên vào thư mục này:

```
$ sudo cp src/redis-server /usr/local/bin/
$ sudo cp src/redis-cli /usr/local/bin/
```

Khởi động Redis

Cách đơn giản nhất để khởi động Redis là chạy tập tin thực thi redis-server trên cửa sổ dòng lệnh mà không đưa vào đối số nào trong câu lệnh này:

```
$ redis-server
```

Bạn sẽ thấy trên cửa sổ terminal có hiện thông báo như sau:

```
[id_redis_server] date_time # Warning: no config file specified, using the default config. In
order to specify a config file use 'redis-server /path/to/redis.conf'
[id_redis_server] date_time * Server started, Redis version 2.2.11
[id_redis_server] date_time * The server is now ready to accept connections on port 6379
```

Thông báo đầu tiên nói rằng bạn không đặt giá trị cho tập tin dùng để xác định các giá trị cài đặt cho Redis server, tuy nhiên điều này không quá quan trọng vào lúc này. Các thông báo tiếp theo hiển thị thông tin về phiên bản Redis server mà bạn đang sử dụng và port mà server này đang dùng.

Nếu bạn muốn đưa 1 tập tin chứa giá trị dùng để cài đặt Redis server bạn chỉ việc thêm đường dẫn của tập tin này khi chạy redis-server. Ví dụ như sau:

```
$ redis-server /etc/redis.conf
```

Kiểm tra hoạt động của Redis Server

Bây giờ để xem Redis server hoạt động như thế nào bạn có thể sử dụng phần mềm redis-cli. Trên cửa sổ dòng lệnh bạn chạy câu lệnh sau:

```
$ redis-cli ping
```

Nếu Redis server hoạt động bình thường bạn sẽ có Kết quả trả về như sau:

```
PONG
```

Khi thực thi câu lệnh bash trên, chương trình redis-cli sẽ thực hiện công việc gửi 1 câu lệnh ping tới Redis server. Khi nhận được câu lệnh này Redis server, nếu như đang hoạt động bình thường, sẽ trả về kết quả PONG như bạn đã thấy ở trên.

Ngoài ra trường hợp bạn cũng có thể sử dụng redis-cli ở chế độ interactive mode để có thể dễ dàng tương tác với Redis server hơn. Để vào chế độ này bạn chạy câu lệnh sau trên shell:

```bash
$ redis-cli
redis 127.0.0.1:6379> ping
```

Kết quả:

```
PONG
```

Redis Configuration - Thực hiện cấu hình bằng lệnh trong Python

Redis có một file để thực hiện cấu hình các tham số (redis.conf) tại thư mục root của Redis, tuy nhiên, bạn có thể get và set các tham số bằng cách sử dụng CONFIG command.

Cú pháp

Cú pháp cơ bản của Redis CONFIG command:

```
redis 127.0.0.1:6379> CONFIG GET CONFIG_SETTING_NAME
```

Ví dụ:

```
redis 127.0.0.1:6379> CONFIG GET loglevel
1) "loglevel"
2) "notice"
```

Để get tất cả cấu hình, sử dụng `*` thay cho `CONFIG_SETTING_NAME`

Ví dụ:

```
redis 127.0.0.1:6379> CONFIG GET *
  1) "dbfilename"
  2) "dump.rdb"
  3) "requirepass"
  4) ""
  5) "masterauth"
  6) ""
  7) "unixsocket"
  8) ""
  9) "logfile"
```

```
10) ""
11) "pidfile"
12) "/var/run/redis.pid"
13) "maxmemory"
14) "0"
15) "maxmemory-samples"
16) "3"
17) "timeout"
18) "0"
19) "tcp-keepalive"
20) "0"
21) "auto-aof-rewrite-percentage"
22) "100"
23) "auto-aof-rewrite-min-size"
24) "67108864"
25) "hash-max-ziplist-entries"
26) "512"
27) "hash-max-ziplist-value"
28) "64"
29) "list-max-ziplist-entries"
30) "512"
31) "list-max-ziplist-value"
32) "64"
33) "set-max-intset-entries"
34) "512"
35) "zset-max-ziplist-entries"
36) "128"
37) "zset-max-ziplist-value"
38) "64"
39) "hll-sparse-max-bytes"
40) "3000"
41) "lua-time-limit"
42) "5000"
43) "slowlog-log-slower-than"
44) "10000"
45) "latency-monitor-threshold"
46) "0"
47) "slowlog-max-len"
48) "128"
49) "port"
```

```
50) "6379"
51) "tcp-backlog"
52) "511"
53) "databases"
54) "16"
55) "repl-ping-slave-period"
56) "10"
57) "repl-timeout"
58) "60"
59) "repl-backlog-size"
60) "1048576"
61) "repl-backlog-ttl"
62) "3600"
63) "maxclients"
64) "4064"
65) "watchdog-period"
66) "0"
67) "slave-priority"
68) "100"
69) "min-slaves-to-write"
70) "0"
71) "min-slaves-max-lag"
72) "10"
73) "hz"
74) "10"
75) "no-appendfsync-on-rewrite"
76) "no"
77) "slave-serve-stale-data"
78) "yes"
79) "slave-read-only"
80) "yes"
81) "stop-writes-on-bgsave-error"
82) "yes"
83) "daemonize"
84) "no"
85) "rdbcompression"
86) "yes"
87) "rdbchecksum"
88) "yes"
89) "activerehashing"
```

```
 90) "yes"
 91) "repl-disable-tcp-nodelay"
 92) "no"
 93) "aof-rewrite-incremental-fsync"
 94) "yes"
 95) "appendonly"
 96) "no"
 97) "dir"
 98) "/home/deepak/Downloads/redis-2.8.13/src"
 99) "maxmemory-policy"
100) "volatile-lru"
101) "appendfsync"
102) "everysec"
103) "save"
104) "3600 1 300 100 60 10000"
105) "loglevel"
106) "notice"
107) "client-output-buffer-limit"
108) "normal 0 0 0 slave 268435456 67108864 60 pubsub 33554432 8388608 60"
109) "unixsocketperm"
110) "0"
111) "slaveof"
112) ""
113) "notify-keyspace-events"
114) ""
115) "bind"
116) ""
```

Sửa cấu hình

Để sửa cấu hình, bạn có thể sửa tại file redis.conf hoặc cập nhật cấu hình bằng cách sửa dụng lệnh CONFIG set.

```
redis 127.0.0.1:6379> CONFIG SET CONFIG_SETTING_NAME NEW_CONFIG_VALUE
```

Ví dụ:

```
redis 127.0.0.1:6379> CONFIG SET loglevel "notice"
OK
redis 127.0.0.1:6379> CONFIG GET loglevel
1) "loglevel"
```

```
2) "notice"
```

Redis Data types - Kiểu dữ liệu trong Redis

Redis hỗ trợ 5 kiểu dữ liệu, bao gồm:

1. Strings

Redis string là lệnh sử dụng để quản lý các key/value trong đó value có giá trị string trong redis.

Ví dụ:

```
redis 127.0.0.1:6379> SET name "laptrinh.vn"
OK
redis 127.0.0.1:6379> GET name
"laptrinh.vn"
```

Trong ví dụ trên, sử dụng SET và SET command, `name` là key được sử dụng trong Redis và `laptrinh.vn` là giá trị được lưu trữ

Chú ý: Một string value có thể được lưu trữ đến 512 megabytes.

Các lệnh thường dùng:

STT	Command	Ý nghĩa
1	SET key value	Đặt giá trị value cho key
2	GET key	Lấy giá trị lưu trữ bởi key
3	GETRANGE key start end	Lấy giá trị lưu trữ bởi key từ (start) đến (end)
4	GETSET key value	Lấy ra giá trị cũ và đặt giá trị mới cho key
5	MGET key1 key2 ..	Lấy giá trị của nhiều key theo thứ tự
6	SETEX key seconds value	Đặt giá trị và thời gian expire cho key
7	SETNX key value	Đặt giá trị cho key nếu key chưa tồn tại

STT	Command	Ý nghĩa
8	RENAMENX key newkey	Đổi tên key sang newkey nếu newkey chưa tồn tại
9	STRLEN key	Lấy độ dài giá trị lưu trữ bởi key
9	APPEND key value	Thêm vào sau giá trị lưu trữ bởi key là value
10	INCR key	Tăng giá trị lưu trữ của key (số nguyên) 1 đơn vị
11	INCRBY key n	Tăng giá trị lưu trữ của key (số nguyên) n đơn vị
12	DECR key	Giảm giá trị lưu trữ của key (số nguyên) 1 đơn vị
11	DECRBY key n	Giảm giá trị lưu trữ của key (số nguyên) n đơn vị

2. Hashes

Redis hash lưu trữ hash table của các cặp key-value, trong đó key được sắp xếp ngẫu nhiên, không theo thứ tự nào cả. Redis hỗ trợ các thao tác thêm, đọc, xóa từng phần tử, cũng như đọc tất cả giá trị.

Ví dụ:

```
redis 127.0.0.1:6379> HMSET user:1 username laptrinh.vn password
laptrinh.vn points 200
OK
redis 127.0.0.1:6379> HGETALL user:1
1) "username"
2) "laptrinh.vn"
3) "password"
4) "laptrinh.vn"
5) "points"
6) "200"
```

Chú ý: Mỗi Redis hash có thể lưu trữ đến 2^{32}-1 cặp field-value (> 4 tỷ).

Các lệnh thường dùng:

STT	Command	Ý nghĩa
1	HSET key field value	Đặt giá trị cho field là value trong hash
2	HGET key field	Lấy giá trị của field trong hash

STT	Command	Ý nghĩa
3	HDEL key field1 field2 ...	xóa field1, field2 ... trong hash
4	HEXISTS key field	Kiểm tra file có tồn tại trong hash không
5	HGETALL key	Lấy tất cả các field và value của nó trong hash
6	HINCRBY key field n	Tăng giá trị của field (số nguyên) lên n đơn vị
7	HDECRBY key field n	Giảm giá trị của field (số nguyên) lên n đơn vị
8	HINCRBYFLOAT key field f	Tăng giá trị của field (số thực) lên f
9	HDECRBYFLOAT key field n	Giảm giá trị của field (số thực) f
10	HKEYS key	Lấy tất cả các field của hash
11	HVALS key	Lấy tất cả các value của hash
12	HLEN key	Lấy số lượng field của hash
13	HMSET key field1 value1 field2 value2 ...	Đặt giá trị cho các field1 giá trị value1 field2 giá trị value2 ...
14	HMGET key field1 field2 ...	Lấy giá trị của các field1 field2 ...

3. Lists

Redis Lists là danh sách liên kết của các strings. Redis hỗ trợ các thao tác push, pop từ cả 2 phía của list, trim dựa theo offset, đọc 1 hoặc nhiều items của list, tìm kiếm và xóa giá trị.

Ví dụ:

```
redis 127.0.0.1:6379> lpush tutoriallist redis
(integer) 1
redis 127.0.0.1:6379> lpush tutoriallist mongodb
(integer) 2
redis 127.0.0.1:6379> lpush tutoriallist rabitmq
(integer) 3
redis 127.0.0.1:6379> lrange tutoriallist 0 10

1) "rabitmq"
2) "mongodb"
3) "redis"
```

Chú ý: Độ dài lớn nhất của Redis List là 2^{32}-1 phần tử (> 4 tỷ).

Các lệnh thường dùng:

STT	Command	Ý nghĩa
1	LINDEX key index	Lấy giá trị từ danh sách (list) ở vị trí index (index bắt đầu từ 0)
2	LLEN key	Lấy số lượng phần tử trong danh sách
3	LPOP key	Lấy phần tử ở đầu danh sách
4	LPUSH key value1 value2 ...	Thêm value1 value2... vào đầu danh sách
5	LRANGE key start stop	Lấy các phần tử trong list từ vị trí start đến vị trí stop
6	LSET key index value	Đặt lại giá trị tại index bằng value
7	RPOP key	Lấy giá trị ở cuối danh sách
8	RPUSH key value1 value2 ...	Thêm phần tử value1 value2 ... vào cuối danh sách
9	LINSERT key BEFORE value1 value2	Thêm phần tử value2 vào trước phần tử value1 trong danh sách
10	LINSERT key AFTER value1 value2	Thêm phần tử value2 vào sau phần tử value1 trong danh sách

4. Sets

Redis Sets là tập hợp các string (không được sắp xếp). Redis hỗ trợ các thao tác thêm, đọc, xóa từng phần tử, kiểm tra sự xuất hiện của phần tử trong tập hợp. Ngoài ra Redis còn hỗ trợ các phép toán tập hợp, gồm intersect/union/difference.

Ví dụ:

```
redis 127.0.0.1:6379> sadd tutoriallist redis
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist mongodb
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist rabitmq
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist rabitmq
(integer) 0
redis 127.0.0.1:6379> smembers tutoriallist

1) "rabitmq"
2) "mongodb"
3) "redis"
```

Chú ý: Số phần tử lớn nhất trong Redis Set được lưu trữ là 2^{32}-1 phần tử (> 4 tỷ).

Các lệnh thường dùng:

STT	Command	Ý nghĩa
1	SADD key value1 value2 ..	Thêm các giá trị value1 value2 ... vào tập hợp
2	SCARD key	Lấy số lượng phần tử trong tập hợp
3	SMEMBERS key	Lấy các phần tử trong tập hợp
4	SPOP key	Xóa bỏ ngẫu nhiên một phần tử trong tập hợp và trả về giá trị phần tử đó

5. Sorted Sets (ZSets)

Redis Sorted Sets là 1 danh sách, trong đó mỗi phần tử là map của 1 string (member) và 1 floating-point number (score), danh sách được sắp xếp theo score này. Redis hỗ trợ thao tác thêm, đọc, xóa từng phần tử, lấy ra các phần tử dựa theo range của score hoặc của string.

Ví dụ:

```
redis 127.0.0.1:6379> zadd tutoriallist 0 redis
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 mongodb
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 rabitmq
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 rabitmq
(integer) 0
redis 127.0.0.1:6379> ZRANGEBYSCORE tutoriallist 0 1000

1) "redis"
2) "mongodb"
3) "rabitmq"
```

Các lệnh thường dùng:

STT	Command	Ý nghĩa
1	ZADD key score1 value1 score2 value2 ..	Thêm các phần tử value1 value2 vào sorted set với độ ưu tiên tương ứng là score1 và score2
2	SCARD key	Lấy số lượng phần tử trong sorted set

STT	Command	Ý nghĩa
3	ZRANGE key start stop	Lấy các phần tử trong tập hợp từ start đến stop
4	ZRANGE key start stop WITHSCORES	Lấy các phần tử trong tập hợp từ start đến stop kèm theo giá trị score của chúng
5	ZSCORE key member	Lấy giá trị score của member
6	ZRANK key member	Lấy vị trí của member trong sorted set
7	ZCOUNT key score1 score2	Đếm số member có score tương ứng trong đoạn score1 đến score2

Redis Commands - Redis-cli trong Redis

Redis command được sử dụng để thực hiện các lệnh thực thi trên Redis server.

Để run command trên Redis server, bạn cần một Redis client. Redis client có sẵn trong Redis package khi chúng ta cài đặt.

Cú pháp

Cú pháp cơ bản của Redis client:

```
$redis-cli
```

Ví dụ:

Để start Redis client, mở terminal và nhập command redis-cli. Nó sẽ kết nối tới Local server và bạn có thể run bất kỳ command nào.

```
$redis-cli
redis 127.0.0.1:6379>
redis 127.0.0.1:6379> PING
PONG
```

Trong ví dụ trên, chúng ta đã kết nối đến Redis server được chạy trên máy local và thực thi lệnh PING, để kiểm tra xem server có đang chạy hay không.

Run Commands trên Remote Server

Để run command trên Redis remote server, bạn cần kết nối đến server sử dụng redis-cli kèm theo tham số host

Cú pháp:

```
$ redis-cli -h host -p port -a password
```

Ví dụ: Kết nối đến Redis remote server, với host 127.0.0.1, port 6379 và password là mypass.

```
$redis-cli -h 127.0.0.1 -p 6379 -a "mypass"
redis 127.0.0.1:6379>
redis 127.0.0.1:6379> PING
PONG
```

Redis Key command - Quản lý Key trong Redis

Redis key command được sử dụng để quản lý `key` trong Redis.

Cú pháp

```
redis 127.0.0.1:6379> COMMAND KEY_NAME
```

Ví dụ:

```
redis 127.0.0.1:6379> SET laptrinhvn redis
OK
redis 127.0.0.1:6379> DEL laptrinhvn
(integer) 1
```

Trong ví dụ trên, `DEL` là command, `laptrinhvn` là key. Nếu key `laptrinhvn` được xóa thành công, output của command là (integer) 1, ngược lại là (integer) 0.

Redis Key Command

Bảng sau đây là danh sách các command cơ bản của `key`:

STT	Command	Mô tả
1	DEL key	Xóa key nếu nó tồn tại
2	DUMP key	Trả về serialized version của giá trị được lưu trữ bởi key
3	EXISTS key	Kiểm tra key có tồn tại không

4	EXPIRE key seconds	Đặt expire time cho key sau n giây
5	EXPIREAT key timestamp	Đặt expire time cho key tại thời điểm xác định. Time có kiểu Unix timestamp
6	PEXPIRE key milliseconds	Đặt expire time cho key sau n milliseconds
7	PEXPIREAT key milliseconds-timestamp	Đặt expire time cho key tại thời điểm xác định. Time có kiểu Unix timestamp (milliseconds)
8	KEYS pattern	Tìm các key theo pattern
9	MOVE key db	Move một key sang một database Redis khác
10	PERSIST key	Xóa expire time của key
11	PTTL key	Lấy thời gian sống (còn lại) của key (milliseconds)
12	TTL key	Lấy thời gian sống (còn lại) của key (giây)
13	RANDOMKEY	Trả về một random key từ Redis

14	RENAME key newkey	Đổi tên key sang newkey, nếu newkey đã tồn tại giá trị của nó sẽ bị ghi đè bởi giá trị của key
15	RENAMENX key newkey	Đổi tên key sang newkey nếu newkey chưa tồn tại
16	TYPE key	Lấy loại dữ liệu được lưu trữ bởi key

Redis Transaction - Thực hiện nhóm lệnh trong Redis

Redis transaction cho phép một nhóm các lệnh thực hiện theo thứ tự cho đến khi lệnh cuối cùng được thực hiện xong. Khi này Redis mới cập nhật đồng thời dữ liệu thay đổi bởi nhóm lệnh này. Redis transaction bắt đầu bằng lệnh MULTI và kết thúc bằng lệnh EXEC

Ví dụ:

```
redis 127.0.0.1:6379> MULTI
OK
redis 127.0.0.1:6379> SET test redis
QUEUED
redis 127.0.0.1:6379> GET test
QUEUED
redis 127.0.0.1:6379> INCR visitors
QUEUED
redis 127.0.0.1:6379> EXEC

1) OK
2) "redis"
3) (integer) 1
```

Các lệnh thường dùng:

STT	Command	Ý nghĩa
1	MULTI	Đánh dấu bắt đầu khối lệnh transaction
2	EXEC	Thực hiện khối lệnh
3	DISCARD	Hủy tất cả các lệnh được ban hành sau MULTI
4	UNWATCH	Quên về tất cả các key đã xem
5	WATCH key [key ...]	Theo dõi các key đã cho để xác định việc thực hiện khối MULTI / EXEC

Redis HyperLogLog

Redis HyperLogLog là một thuật toán sử dụng phương pháp ngẫu nhiên để tạo ra một số lượng phần tử là duy nhất trong một tập hợp chỉ sử dụng một lượng bộ nhớ nhỏ và không đổi.

HyperLogLog cung cấp một xấp xỉ rất tốt về tính chính xác của một tập hợp ngay cả khi sử dụng một lượng bộ nhớ rất nhỏ khoảng 12 kbyte mỗi khóa với sai số chuẩn là 0,81%. Không có giới hạn về số lượng phần tử, trừ khi bạn cần một số lượng lớn hơn 2^{64} phần tử.

Ví dụ:

```
redis 127.0.0.1:6379> PFADD items "redis"
1) (integer) 1
redis 127.0.0.1:6379> PFADD items "mongodb"
1) (integer) 1
redis 127.0.0.1:6379> PFADD items "mysql"
1) (integer) 1
redis 127.0.0.1:6379> PFCOUNT items
(integer) 3
```

Các cú pháp cơ bản:

STT	Command	Mô tả
1	PFADD key element [element ...]	Thêm một phần tử vào HyperLogLog
2	PFCOUNT key [key ...]	Trả về giá trị xấp xỉ của tập hợp được quan sát bởi HyperLogLog theo key.
3	PFMERGE destkey sourcekey [sourcekey ...]	Nhập N HyperLogLog thành một

Redis Publish - Subscribe

Redis Pub / Sub được áp dụng trong các hệ thống cần gửi/nhận bản tin message nơi sender (publisher) gửi message trong khi receiver (subscriber) nhận được chúng. Liên kết mà các message được chuyển được gọi là kênh.

Trong Redis, khách hàng có thể subscribe bất kỳ số lượng kênh.

Redis-Pub-Sub-2-768x253.jpeg

Image not found or type unknown

Ví dụ:

- subscriber: 1 Client đăng ký nhận message từ redisChat:

```
redis 127.0.0.1:6379> SUBSCRIBE redisChat
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "redisChat"
3) (integer) 1
```

- publisher: 2 Client gửi message vào channel redisChat:

```
redis 127.0.0.1:6379> PUBLISH redisChat "Redis is a great caching technique"
(integer) 1
redis 127.0.0.1:6379> PUBLISH redisChat "Learn redis by Laptrinh.vn"
(integer) 1
1) "message"
2) "redisChat"
3) "Redis is a great caching technique"
1) "message"
2) "redisChat"
3) "Learn redis by Laptrinh.vn"
```

Các command cơ bản:

STT	Command	Mô tả
1	PSUBSCRIBE pattern [pattern ...]	Đăng ký (subscibe) channel theo pattern
2	PUBSUB subcommand [argument [argument ...]]	Cho biết trạng thái của hệ thống Pub / Sub. Ví dụ, client nào đang hoạt động trên máy chủ.
3	PUBLISH channel message	Gửi message tới channel
4	PUNSUBSCRIBE [pattern [pattern ...]]	Dừng nhận message của channel theo pattern
5	SUBSCRIBE channel [channel ...]	Lắng nghe message của channel
6	UNSUBSCRIBE [channel [channel ...]]	Dừng lắng nghe message của channel

Redis Scripting

Redis scripting được sử dụng để thực thi các script bằng thông dịch Lua (Lua interpreter). Nó được tích hợp vào Redis bắt đầu từ phiên bản 2.6.0. Lệnh được sử dụng cho scripting là lệnh **EVAL**.

Cú pháp:

```
redis 127.0.0.1:6379> EVAL script numkeys key [key ...] arg [arg ...]
```

Ví dụ:

```
redis 127.0.0.1:6379> EVAL "return {KEYS[1],KEYS[2],ARGV[1],ARGV[2]}" 2 key1
key2 first second
1) "key1"
2) "key2"
3) "first"
4) "second"
```

Giải thích:

- Bên trong quote "" là **nội dung của lua script** (viết bằng ngôn ngữ LUA, bạn có thể tham khảo về LUA tại đây).
- Số 2 đằng sau quote là *số parameter mô tả* `Redis key names` (ở đây chính là là key1 và key2)
- Sau key1 và key2 sẽ là các parameter **không mô tả key names**
- Các parameter mô tả keynames sẽ được access bên trong LUA script dưới hình thức một phần tử trong array `KEY`, và các parameter còn lại sẽ được access bên trong LUA script dưới dạng một phần tử trong array `ARGV`
- Tại sao lại không dùng chung parameter cho `KEY` và `ARGV`??? Câu trả lời là khi bạn set key cho redis bên trong LUA script thì key đó sẽ **chỉ dùng được thông qua KEY parameter**

Các command cơ bản của Redis Scripting:

STT	Command	Mô tả
1	EVAL script numkeys key [key ...] arg [arg ...]	Thực thi Lua script

2	EVALSHA sha1 numkeys key [key ...] arg [arg ...]	Thực thi Lua script
3	SCRIPT EXISTS script [script ...]	Kiểm tra tồn tại của một script trong cache
4	SCRIPT FLUSH	Xóa toàn bộ script trong cache
5	SCRIPT KILL	Kill việc thực thi một script
6	SCRIPT LOAD script	Load một Lua script từ cache

Redis Connection command

Redis connection là command cơ bản được sử dụng để quản lý các kết nối của client tới Redis server.

Ví dụ:

```
redis 127.0.0.1:6379> AUTH "password"
OK
redis 127.0.0.1:6379> PING
PONG
```

Các command cơ bản của Redis connection:

STT	Command	Mô tả
1	AUTH password	Thiết lập xác thực tới server sử dụng password
2	ECHO message	Print một message string
3	PING	Kiểm tra server có đang hoạt động hay không
4	QUIT	Đóng một kết nối hiện tại
5	SELECT index	Thay đổi một database cho kết nối hiện tại

Redis Server command - Lệnh quản lý Redis

Redis server command được sử dụng để quản lý Redis server.

Ví dụ;

```
redis 127.0.0.1:6379> INFO

# Server
redis_version:2.8.13
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:c2238b38b1edb0e2
redis_mode:standalone
os:Linux 3.5.0-48-generic x86_64
arch_bits:64
multiplexing_api:epoll
gcc_version:4.7.2
process_id:3856
run_id:0e61abd297771de3fe812a3c21027732ac9f41fe
tcp_port:6379
uptime_in_seconds:11554
uptime_in_days:0 hz:10
lru_clock:16651447
config_file:

# Clients
connected_clients:1
client_longest_output_list:0
client_biggest_input_buf:0
blocked_clients:0

# Memory
```

```
used_memory: 589016
used_memory_human: 575.21K
used_memory_rss: 2461696
used_memory_peak: 667312
used_memory_peak_human: 651.67K
used_memory_lua: 33792
mem_fragmentation_ratio: 4.18
mem_allocator: jemalloc-3.6.0

# Persistence
loading: 0
rdb_changes_since_last_save: 3
rdb_bgsave_in_progress: 0
rdb_last_save_time: 1409158561
rdb_last_bgsave_status: ok
rdb_last_bgsave_time_sec: 0
rdb_current_bgsave_time_sec: -1
aof_enabled: 0
aof_rewrite_in_progress: 0
aof_rewrite_scheduled: 0
aof_last_rewrite_time_sec: -1
aof_current_rewrite_time_sec: -1
aof_last_bgrewrite_status: ok
aof_last_write_status: ok

# Stats
total_connections_received: 24
total_commands_processed: 294
instantaneous_ops_per_sec: 0
rejected_connections: 0
sync_full: 0
sync_partial_ok: 0
sync_partial_err: 0
expired_keys: 0
evicted_keys: 0
keyspace_hits: 41
keyspace_misses: 82
pubsub_channels: 0
pubsub_patterns: 0
latest_fork_usec: 264
```

```
# Replication
role: master
connected_slaves: 0
master_repl_offset: 0
repl_backlog_active: 0
repl_backlog_size: 1048576
repl_backlog_first_byte_offset: 0
repl_backlog_histlen: 0

# CPU
used_cpu_sys: 10.49
used_cpu_user: 4.96
used_cpu_sys_children: 0.00
used_cpu_user_children: 0.01

# Keyspace
db0: keys = 94, expires = 1, avg_ttl = 41638810
db1: keys = 1, expires = 0, avg_ttl = 0
db3: keys = 1, expires = 0, avg_ttl = 0
```

Các command cơ bản:

STT	Command	Mô tả
1	BGREWRITEAOF	Asynchronously rewrites the append-only file
2	BGSAVE	Asynchronously saves the dataset to the disk
3	CLIENT KILL [ip:port] [ID client-id]	Kills the connection of a client
4	CLIENT LIST	Gets the list of client connections to the server

5	CLIENT GETNAME	Gets the name of the current connection
6	CLIENT PAUSE timeout	Stops processing commands from the clients for a specified time
7	CLIENT SETNAME connection-name	Sets the current connection name
8	CLUSTER SLOTS	Gets an array of Cluster slot to node mappings
9	COMMAND	Gets an array of Redis command details
10	COMMAND COUNT	Gets total number of Redis commands
11	COMMAND GETKEYS	Extracts the keys given a full Redis command
12	BGSAVE	Asynchronously saves the dataset to the disk
13	COMMAND INFO command-name [command-name ...]	Gets an array of specific Redis command details
14	CONFIG GET parameter	Gets the value of a configuration parameter

15	CONFIG REWRITE	Rewrites the configuration file with the in-memory configuration
16	CONFIG SET parameter value	Sets a configuration parameter to the given value
17	CONFIG RESETSTAT	Resets the stats returned by INFO
18	DBSIZE	Returns the number of keys in the selected database
19	DEBUG OBJECT key	Gets debugging information about a key
20	DEBUG SEGFAULT	Makes the server crash
21	FLUSHALL	Removes all the keys from all databases
22	FLUSHDB	Removes all the keys from the current database
23	INFO [section]	Gets information and statistics about the server
24	LASTSAVE	Gets the UNIX time stamp of the last successful save to the disk

25	MONITOR	Listens for all the requests received by the server in real time
26	ROLE	Returns the role of the instance in the context of replication
27	SAVE	Synchronously saves the dataset to the disk
28	SHUTDOWN [NOSAVE] [SAVE]	Synchronously saves the dataset to the disk and then shuts down the server
29	SLAVEOF host port	Makes the server a slave of another instance, or promotes it as a master
30	SLOWLOG subcommand [argument]	Manages the Redis slow queries log
31	SYNC	Command used for replication
32	TIME	Returns the current server time

Redis Key command

Các command cơ bản để quản lý key trong Redis

Redis Del command - Xóa Key trong Redis

Redis **DEL** command được sử dụng để xóa một bản ghi theo key đã tồn tại.

Return Value:

Số bản ghi (key) đã được xóa.

Cú pháp:

redis 127.0.0.1:6379> DEL KEY_NAME

Ví dụ:

```
# redis 127.0.0.1:6379> SET tutorialspoint redis
OK

# redis 127.0.0.1:6379> DEL tutorialspoint
(integer) 1
```

Redis Dump command - Lệnh Dump trong Redis

Redis DUMP command được sử dụng để trả về serialized version của giá trị được lưu trữ bởi key.

Return value:

```
Serialized value (String)
```

Cú pháp:

```
redis 127.0.0.1:6379> DUMP KEY_NAME
```

Ví dụ:

```
# redis 127.0.0.1:6379> SET tutorialspoint redis
OK

# redis 127.0.0.1:6379> DUMP tutorialspoint
"\x00\x05redis\x06\x00S\xbd\xc1q\x17z\x81\xb2"
```

Redis Exists command - Kiểm tra Key đã tồn tại trong Redis

Redis EXISTS command được sử dụng để kiểm tra xem một `key` đã tồn tại hay chưa

Return Value

Integer value:

- 1, nếu key đã tồn tại.
- 0, nếu key không tồn tại.

Syntax:

```
redis 127.0.0.1:6379> EXISTS KEY_NAME
```

Ví dụ:

```
redis 127.0.0.1:6379> EXISTS tutorialspoint-new-key
(integer) 0

redis 127.0.0.1:6379> EXISTS tutorialspoint-new-key
(integer) 1
```

Redis Expire command - Lệnh thiết lập thời gian hết hạn của Key trong Redis

Redis Expire command được sử dụng để thiết lập thời gian hết hạn của một key. Sau thời gian này, key sẽ không còn tồn tại trên Redis.

Return Value

Integer value:

- 1, nếu thời gian timeout được thiết lập cho key.
- 0, nếu key không tồn tại hoặc thời gian timeout không được thiết lập cho key.

Syntax:

```
redis 127.0.0.1:6379> Expire KEY_NAME TIME_IN_SECONDS
```

Ví dụ:

```
redis 127.0.0.1:6379> SET laptrinhvn redis
OK

redis 127.0.0.1:6379> EXPIRE laptrinhvn 60
(integer) 1
```

Trong ví dụ trên, key `laptrinhvn` được thiết lập thời gian expire là 60s (1 phút). Sau 1 phút, key `laptrinhvn` sẽ bị expire tự động

Redis Expireat command - Lệnh thiết lập thời gian hết hạn của Key trong Redis

Redis Expireat command được sử dụng để thiết lập thời gian hết hạn của một key được tính theo Unix timestamp. Sau thời gian này, key sẽ không còn tồn tại trên Redis.

Return Value

Integer value:

- 1, nếu thời gian timeout được thiết lập cho key.
- 0, nếu key không tồn tại hoặc thời gian timeout không được thiết lập cho key.

Syntax:

```
redis 127.0.0.1:6379> Expireat KEY_NAME TIME_IN_UNIX_TIMESTAMP
```

Ví dụ:

```
redis 127.0.0.1:6379> SET laptrinhvn redis
OK

redis 127.0.0.1:6379> EXPIREAT laptrinhvn 1570939008
(integer) 1
EXISTS laptrinhvn
(integer) 0
```

Trong ví dụ trên, key `laptrinhvn` được thiết lập thời gian expire là 1570939008 (2019-10-13T03:56:48+00:00). Sau thời gian này, key `laptrinhvn` sẽ bị expire tự động

Redis Pexpire command - Lệnh thiết lập thời gian hết hạn của Key trong Redis

Redis Pexpire command được sử dụng để thiết lập thời gian hết hạn của một key (tính theo miliseconds). Sau thời gian này, key sẽ không còn tồn tại trên Redis.

Return Value

Integer value:

- 1, nếu thời gian timeout được thiết lập cho key.
- 0, nếu key không tồn tại hoặc thời gian timeout không được thiết lập cho key.

Syntax:

```
redis 127.0.0.1:6379> PEXPIRE KEY_NAME TIME_IN_MILLISECONDS
```

Ví dụ:

```
redis 127.0.0.1:6379> SET laptrinhvn redis
OK

redis 127.0.0.1:6379> PEXPIRE laptrinhvn 5000
(integer) 1
```

Trong ví dụ trên, key `laptrinhvn` được thiết lập thời gian expire là 5000 ms (5s). Sau 5s, key `laptrinhvn` sẽ bị expire tự động

Redis Pexpireat command - Lệnh thiết lập thời gian hết hạn của Key trong Redis

Redis Pexpireat command được sử dụng để thiết lập thời gian hết hạn của một key được tính theo Unix timestamp miliseconds. Sau thời gian này, key sẽ không còn tồn tại trên Redis.

Return Value

Integer value:

- 1, nếu thời gian timeout được thiết lập cho key.
- 0, nếu key không tồn tại hoặc thời gian timeout không được thiết lập cho key.

Syntax:

```
redis 127.0.0.1:6379> PEXPIREAT KEY_NAME TIME_IN_MILLISECONDS_IN_UNIX_TIMESTAMP
```

Ví dụ:

```
redis 127.0.0.1:6379> SET laptrinhvn redis
OK

redis 127.0.0.1:6379> PEXPIREAT laptrinhvn 1570939008000
(integer) 1
EXISTS laptrinhvn
(integer) 0
```

Trong ví dụ trên, key `laptrinhvn` được thiết lập thời gian expire là 1570939008 (2019-10-13T03:56:48+00:00). Sau thời gian này, key `laptrinhvn` sẽ bị expire tự động

Redis Keys command - Lệnh tìm kiếm gần đúng theo Key trong Redis

Redis Keys command được sử dụng để tìm kiếm key theo pattern.

Return Value:

Danh sách (Array) các key là trùng với pattern

Syntax:

```
redis 127.0.0.1:6379> KEYS PATTERN
```

Ví dụ:

```
redis 127.0.0.1:6379> SET tutorial1 redis
OK
redis 127.0.0.1:6379> SET tutorial2 mysql
OK
redis 127.0.0.1:6379> SET tutorial3 mongodb
OK

redis 127.0.0.1:6379> KEYS tutorial*
1) "tutorial3"
2) "tutorial1"
3) "tutorial2"
```

Để get tất cả các key trong Redis, sử dung *

```
redis 127.0.0.1:6379> KEYS *
1) "tutorial3"
2) "tutorial1"
```

```
3) "tutorial2"
```

Redis Move command - Lệnh move Key sang database khác trong Redis

Redis MOVE command được sử dụng để move một key từ database hiện tại sang một database khác.

Return Value:

Integer value:

- 1, nếu key được move thành công.
- 0, nếu key không được move.

Syntax:

```
redis 127.0.0.1:6379> MOVE KEY_NAME DESTINATION_DATABASE
```

Ví dụ:

Đầu tiên, tạo một key trong Redis và set value cho nó:

```
redis 127.0.0.1:6379> SET laptrinhvn redis
OK
```

Trong Redis, mặc định database 0th là database được sử dụng, chúng ta sẽ move key laptrinhvn tới database khác (1st):

```
redis 127.0.0.1:6379> MOVE laptrinhvn 1
1) (integer) 1
```

Redis Persist command - Lệnh xóa thiết lập thời gian hết hạn của Key trong Redis

Redis Persist command được sử dụng để xóa một thiết lập expire của key.

Return value:

Integer value:

- 1, nếu timeout được xóa khỏi thiết lập của key.
- 0, nếu key không tồn tại hoặc không có thiết lập timeout cho key.

Syntax:

```
redis 127.0.0.1:6379> PERSIST KEY_NAME
```

Ví dụ:

```
redis 127.0.0.1:6379> SET tutorial1 redis
OK

redis 127.0.0.1:6379> EXPIRE tutorial1 60
1) (integer) 1
redis 127.0.0.1:6379> TTL tutorial1
1) (integer) 60
redis 127.0.0.1:6379> PERSIST tutorial1
1) (integer) 1
redis 127.0.0.1:6379> TTL tutorial1
1) (integer) -1
```

Redis Pttl command - Lệnh kiểm tra thời gian hết hạn của Key trong Redis

Redis PTTL command được sử dụng để get thời gian expiry còn lại (time to live) của key (milliseconds).

Return value:

Integer value:

- TTL (milliseconds).
- -1, nếu key không có thiết lập expire.
- -2, nếu key không tồn tại.

Syntax:

```
redis 127.0.0.1:6379> PTTL KEY_NAME
```

Vi dụ:

```
redis 127.0.0.1:6379> SET laptrinhvn redis
OK

redis 127.0.0.1:6379> EXPIRE laptrinhvn 1
1) (integer) 1
redis 127.0.0.1:6379> PTTL laptrinhvn
1) (integer) 999
```

Redis TTL command - Lệnh kiểm tra thời gian hết hạn còn lại của Key trong Redis

Redis TTL command được sử dụng để get thời gian expiry còn lại (time to live) của key (seconds).

Return value:

Integer value:

- TTL (seconds).
- -1, nếu key không có thiết lập expire.
- -2, nếu key không tồn tại.

Syntax:

```
redis 127.0.0.1:6379> TTL KEY_NAME
```

Vi dụ:

```
redis 127.0.0.1:6379> SET laptrinhvn redis
OK

redis 127.0.0.1:6379> EXPIRE laptrinhvn 60
1) (integer) 1
redis 127.0.0.1:6379> TTL laptrinhvn
1) (integer) 59
```

Redis RANDOMKEY command - Lệnh lấy một Key ngẫu nhiên trong Redis

Redis RANDOMKEY command được sử dụng để get một key ngẫu nhiên từ Redis database.

Return value:

- String - Một key ngẫu nhiên
- nill - Nếu database không có bất kỳ key nào

Syntax:

```
redis 127.0.0.1:6379> RANDOMKEY
```

Ví dụ:

```
//First, create some keys in Redis and set some values in it.

redis 127.0.0.1:6379> SET tutorial1 redis
OK
redis 127.0.0.1:6379> SET tutorial2 mysql
OK
redis 127.0.0.1:6379> SET tutorial3 mongodb
OK

//Now, get a random key from Redis.

redis 127.0.0.1:6379> RANDOMKEY
1) tutorial3
```

Redis Rename command - Lệnh rename Key trong Redis

Redis RENAME command được sử dụng để rename một tên key.

Return value:

- OK
- error, nếu tên key cũ và key mới là giống nhau hoặc khi key không tồn tại. Nếu tên key mới là tồn tại, nó sẽ ghi đè vào key này.

Syntax:

```
redis 127.0.0.1:6379> RENAME OLD_KEY_NAME NEW_KEY_NAME
```

Ví dụ:

```
# First, create some keys in Redis and set some values in it.

redis 127.0.0.1:6379> SET tutorial1 redis
OK

# Now, rename the key ‘tutorial1’ to ‘new-tutorial’.

redis 127.0.0.1:6379> RENAME tutorial1 new-tutorial
OK
```

Redis Renamenx command - Lệnh rename Key trong Redis

Redis RENAMENX command được sử dụng để rename tên của key nếu tên này chưa được sử dụng.

Return value:

Integer reply:

- 1, nếu key được rename thành tên mới.
- 0, nếu tên key cần rename đã tồn tại.

Syntax:

```
redis 127.0.0.1:6379> RENAMENX OLD_KEY_NAME NEW_KEY_NAME
```

Ví dụ:

```
#First, create some keys in Redis and set some values in it.

redis 127.0.0.1:6379> SET tutorial1 redis
OK
redis 127.0.0.1:6379> SET tutorial2 mongodb
OK

#Now, rename the key ‘tutorial1’ to ‘new-tutorial’.

redis 127.0.0.1:6379> RENAMENX tutorial1 new-tutorial
(integer) 1
redis 127.0.0.1:6379> RENAMENX tutorial2 new-tutorial
(integer) 0
```

Redis Type command - Lệnh kiểm tra loại dữ liệu giá trị của Key trong Redis

Redis TYPE command được sử dụng để get kiểu data của giá trị được lưu trữ trong key.

Return value:

String, Data type giá trị or none.

Syntax:

```
redis 127.0.0.1:6379> TYPE KEY_NAME
```

Ví dụ:

```
#First, create some keys in Redis and set some values in it.

redis 127.0.0.1:6379> SET tutorial1 redis
OK

#Now, check the type of the key.

redis 127.0.0.1:6379> TYPE tutorial1
string
```

Redis Advanced

Kiến thức nâng cao về Redis

Redis backup - Tạo bản backup trong Redis

Backup Redis data

Sử dụng Redis SAVE command để thực hiện tạo một bản backup cho Redis

Cú pháp:

```
127.0.0.1:6379> SAVE
OK
```

Thực thi command này sẽ tạo 1 file dump.rdb trong thư mục Redis.

Restore Redis Data

Để thực hiện khôi phục Redis data, move file Redis backup (dump.rdb) vào thư mục Redis và start Redis server.

Để lấy đường dẫn thư mục Redis, sử dụng command CONFIG:

```
127.0.0.1:6379> CONFIG get dir
1) "dir"
2) "/user/laptrinhvn/redis-2.8.13/src"
```

Bgsave

Một cách khác để tạo một bản backup của Redis, đó là sử dụng command BGSAVE. Đây là command sẽ start tiến trình backup và được chạy background.

Ví dụ:

```
127.0.0.1:6379> BGSAVE
Background saving started
```

Redis Security - Bảo mật trong Redis database

Redis database có cơ chế bảo mật xác thực khi client kết nối và thực thi lệnh trên Redis. Để bật tính năng bảo mật, bạn cần thiết lập mật khẩu trong config file

Kiểm tra cấu hình xác thực:

```
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) ""
```

Thiết lập xác thực:

```
127.0.0.1:6379> CONFIG set requirepass "tutorialspoint"
OK
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) "matkhau"
```

Sau khi thiết lập mật khẩu, client kết nối đến Redis mà không xác thực sẽ nhận được thông báo lỗi: `(error) NOAUTH Authentication required`

Cú pháp xác thực:

```
127.0.0.1:6379> AUTH password
```

Ví dụ:

```
127.0.0.1:6379> AUTH "tutorialspoint"
OK
127.0.0.1:6379> SET mykey "Test value"
OK
127.0.0.1:6379> GET mykey
"Test value"
```

Redis Benchmarks - Đo hiệu năng của Redis

Redis benchmark là công cụ để thực hiện đo tốc độ xử lý đồng thời của Redis database.

Cú pháp:

```
redis-benchmark [option] [option value]
```

Ví dụ: Thực hiện đồng thời 100000 lệnh

```
redis-benchmark -n 100000

PING_INLINE: 141043.72 requests per second
PING_BULK: 142857.14 requests per second
SET: 141442.72 requests per second
GET: 145348.83 requests per second
INCR: 137362.64 requests per second
LPUSH: 145348.83 requests per second
LPOP: 146198.83 requests per second
SADD: 146198.83 requests per second
SPOP: 149253.73 requests per second
LPUSH (needed to benchmark LRANGE): 148588.42 requests per second
LRANGE_100 (first 100 elements): 58411.21 requests per second
LRANGE_300 (first 300 elements): 21195.42 requests per second
LRANGE_500 (first 450 elements): 14539.11 requests per second
LRANGE_600 (first 600 elements): 10504.20 requests per second
MSET (10 keys): 93283.58 requests per second
```

Các tham số:

Sr.No	Option	Description	Default Value
1	-h	Specifies server host name	127.0.0.1
2	-p	Specifies server port	6379

3	-s	Specifies server socket	
4	-c	Specifies the number of parallel connections	50
5	-n	Specifies the total number of requests	10000
6	-d	Specifies data size of SET/GET value in bytes	2
7	-k	1=keep alive, 0=reconnect	1
8	-r	Use random keys for SET/GET/INCR, random values for SADD	
9	-p	Pipeline <numreq> requests	1
10	-h	Specifies server host name	
11	-q	Forces Quiet to Redis. Just shows query/sec values	
12	--csv	Output in CSV format	
13	-l	Generates loop, Run the tests forever	
14	-t	Only runs the comma-separated list of tests	
15	-I	Idle mode. Just opens N idle connections and wait	

Ví dụ:

```
redis-benchmark -h 127.0.0.1 -p 6379 -t set,lpush -n 100000 -q

SET: 146198.83 requests per second
LPUSH: 145560.41 requests per second
```

Redis Client Connection - Quản lý kết nối từ Client

Redis chấp nhận các kết nối từ client theo giao thức TCP socket. Khi kết nối từ client mới được chấp nhận, các thao tác sau được thực hiện:

- Client socket được đặt ở trạng thái non-blocking do Redis sử dụng ghép kênh (multiplexing) và non-blocking I/O
- TCP_NODELAY option được thiết lập để chắc chắn rằng không có delay trong connection
- Một sự kiện được tạo để Redis có thể thu thập truy vấn của client ngay khi có dữ liệu mới trên socket.

Maximum Number of Clients

Để kiểm tra số kết nối tối đa từ client, thực hiện lệnh sau:

```
config get maxclients


1) "maxclients"
2) "10000"
```

Mặc định, có 10000 kết nối (phụ thuộc vào giới hạn trong thiết lập OS), bạn có thể thay đổi thiết lập như sau:

```
redis-server --maxclients 100000
```

Client Commands

STT	Command	Mô tả
1	**CLIENT LIST**	Trả về danh sách client đang kết nối đến Redis server
2	**CLIENT SETNAME**	Thiết lập tên của connection hiện tại
3	**CLIENT GETNAME**	Trả về tên của connection hiện tại

4	**CLIENT PAUSE**	Dừng (suspend) các kết nối theo khoảng thời gian xác định (in milliseconds)
5	**CLIENT KILL**	Đóng (close) một kết nối

Redis Pipelining - Thực thi nhiều lệnh đồng thời từ Client

Redis Pipelining được sử dụng khi bạn cần gửi nhiều câu lệnh tới server trong một yêu cầu.

Ví dụ:

```
$(echo -en "PING\r\n SET tutorial redis\r\nGET tutorial\r\nINCR
visitor\r\nINCR visitor\r\nINCR visitor\r\n"; sleep 10) | nc localhost 6379
+PONG
+OK
redis
:1
:2
:3
```

Trong ví dụ trên, chúng ta thực hiện:

- Kiểm tra Redis connection bằng cách sử dụng PING command
- Set một key với name `tutorial` có giá trị `redis`
- Get giá trị của key `tutorial`
- Tăng giá trị của key `tutorial`

Redis thực hiện và output tất cả kết quả của command.

Lợi ích của Pipelining

Lợi ích của kỹ thuật này là hiệu suất giao thức được cải thiện mạnh mẽ. Việc tăng tốc đạt được bằng pipelining dao động từ 5 kết nối localhost (kết nối nhanh) cho đến ít nhất 100 kết nối internet (chậm hơn).

Redis Partitioning

Partitioning là gì

Partitioning là quá trình phân chia dữ liệu thành nhiều phân vùng riêng biệt (instance) trên Redis, do đó mọi phân vùng sẽ chỉ chứa một tập hợp con các key của bạn.

Lợi ích của việc Partitioning

- Nó cho phép Redis database lưu trữ lớn hơn nhiều bằng cách sử dụng tổng bộ nhớ của nhiều máy tính. Nếu không partitioning, bạn sẽ bị giới hạn số lượng bộ nhớ mà một máy tính có thể hỗ trợ.
- Nó cho phép mở rộng sức mạnh tính toán đa luồng (multiple cores), đa máy tính (multiple computers), và băng thông mạng cho nhiều máy tính và network.

Nhược điểm của Partitioning

- Các hoạt động liên quan đến nhiều khóa (multiple keys) thường không được hỗ trợ. Chẳng hạn, bạn không thể thực hiệnthi trên hai tập (sets) nếu chúng được lưu trữ trong các khóa được ánh xạ tới các Redis instance khác nhau.
- Giao dịch Redis liên quan đến nhiều khóa (multiple keys) không thể được sử dụng.
- Partitioning đặc biệt là khóa, vì vậy không thể loại bỏ một tập dữ liệu với một khóa lớn như một tập hợp được sắp xếp rất lớn.
- Khi Partitioning được sử dụng, xử lý dữ liệu phức tạp hơn. Chẳng hạn, bạn phải xử lý nhiều tệp RDB / AOF và để có bản sao lưu dữ liệu của mình, bạn cần tổng hợp các tệp lưu trữ từ nhiều phiên bản và máy chủ lưu trữ.
- Thêm và loại bỏ capacity có thể phức tạp. Chẳng hạn, Redis Cluster hỗ trợ cân bằng lại dữ liệu trong suốt với khả năng thêm và xóa các node khi chạy. Tuy nhiên, các hệ thống khác như Partitioning phía máy khách và proxy không hỗ trợ tính năng này. Một kỹ thuật được gọi là `Preshending` giúp về vấn đề này.

Các loại Partitioning

Có hai loại Partitioning có sẵn trong Redis. Giả sử chúng ta có bốn phiên bản Redis, R0, R1, R2, R3 và nhiều `key` đại diện cho người dùng như: user:1, user:2....

Range Partitioning

Range Partitioning được thực hiện bằng cách ánh xạ phạm vi của các đối tượng vào các Redis instance cụ thể. Giả sử trong ví dụ của chúng tôi, người dùng có từ ID.0 đến ID.10000 sẽ đi vào R0 instance, trong khi người dùng từ ID.10001 đến ID.20000 sẽ vào R1 instance...

Hash Partitioning

Trong kiểu phân vùng này, một hàm băm - hash (ví dụ: hàm mô đun) được sử dụng để chuyển đổi key thành số và sau đó dữ liệu được lưu trữ trong các Redis instance khác nhau.

Redis guideline

Hướng dẫn triển khai Redis

Hướng dẫn cài đặt Redis Master - Slave trên CentOS 7

Hướng dẫn cài đặt Redis Cluster cơ chế Master - Slave trên hệ điều hành Centos 7.

1. Yêu cầu

- Hệ điều hành: CentOS 7

- Tối thiểu 3 server, trong đó:

    - Master: 192.168.0.1
    - Slave: 192.168.0.2, 192.168.0.3

- Ghi chú: Trong hướng dẫn này sẽ không bật tính năng yêu cầu xác thực của Redis, bằng cách cấu hình `protected-mode=no`

2. Cơ chế Redis Master - Slave

- Redis sentinel cung cấp một giải pháp HA cho cụm server triển khai Redis. Nghĩa là, khi có một hoặc một số Redis instance down, thì Redis của bạn vẫn hoạt động tốt.

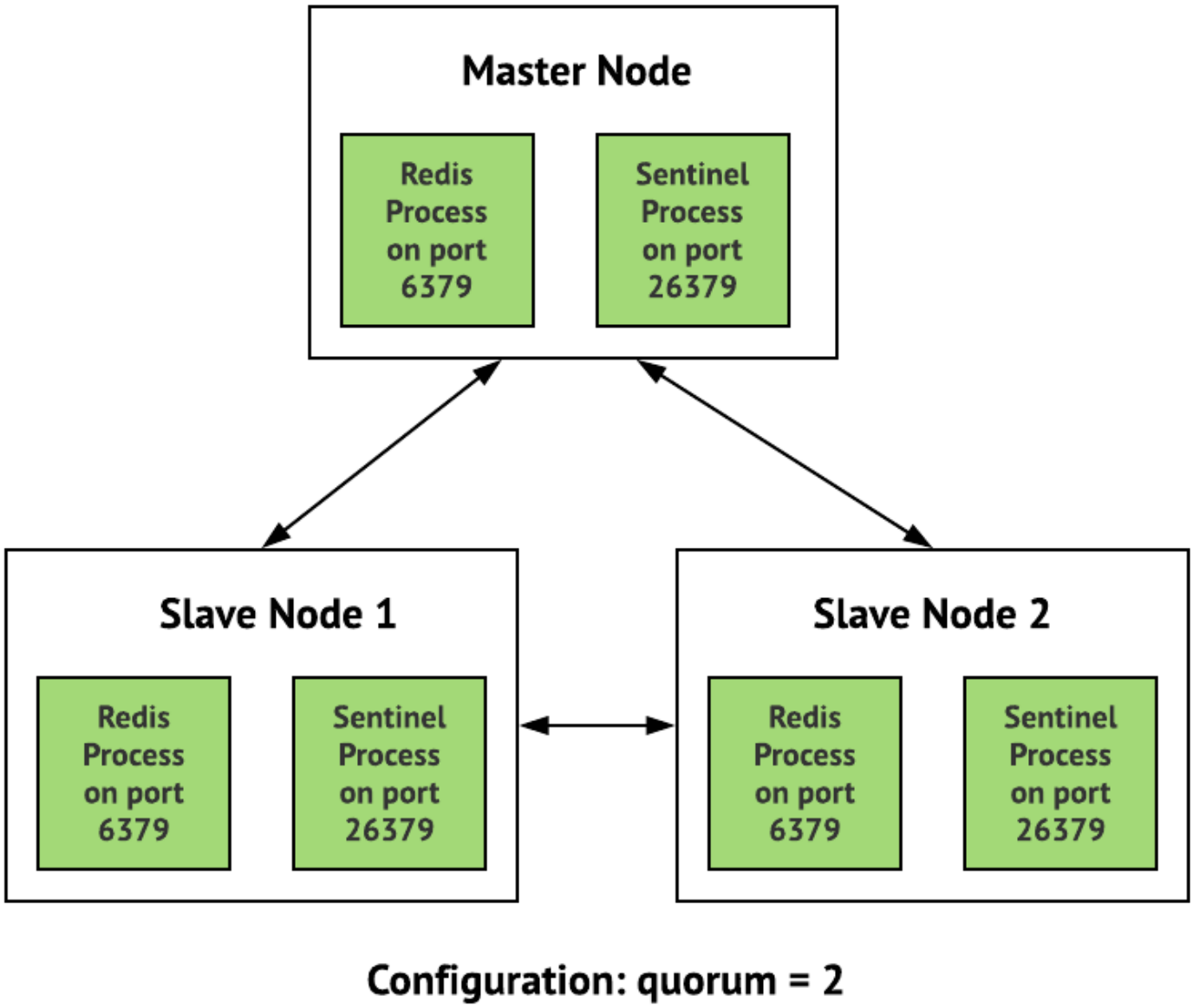
- Redis Sentinel được thiết kế để monitor và quản lý các Redis instances bằng cách thực hiện các task sau:

- Monitoring: Sentinel check các master và slave nodes theo định kì để đảm bảo các node này luôn luôn hoạt động tốt.
- Notification: Sentinel có thể send các notification alert khi có một redis instance failure thông tqua một API
- Automatic failover: Nếu một Master bị fail, Sentinel sẽ khởi tạo một "Failover" process:
    - Pick một running slave và promote SLAVE này lên làm Master.
    - Reconfigured các slaves còn lại thành slave của Master mới.
    - Reconfigured tất cả các Sentinel monitor Master mới.
    - Configuration provider: Sentinel hoạt động như một "source of authority for client’s service discovery". Client connect tới Sentinel để hỏi về địa chỉ của Current Redis Master, nếu một failover xảy ra, Sentinel sẽ report về cho client address của latest Redis Master (mới nhất).

- Các Sentinel liên lạc với nhau, và cùng nhau quyết định rằng một "Master node là unreachable". Quyết định được đưa ra dựa vào `quorum value` (Là số lượng Sentinel đồng ý rằng Master node là unreachable. Giá trị này chỉ được dùng để detect failure).

- Khi Redis Master down, các Sentinel sẽ liên lạc với nhau và start một cuộc họp bàn:

- Nếu số lượng Sentinel đồng ý promote một Slave lên làm Master thì quá trình Failover sẽ được start, một Slave sẽ trở thành Master mới. Các redis instance còn lại sẽ được reconfigure thành các slave của master mới.
- Nếu số lượng Sentinel đồng ý promote một Slave lên làm Master nhỏ hơn quá bán thì quá trình Failover sẽ không diễn ra, hệ thống redis sẽ ngừng hoạt động do không có master mới.



3. Hướng dẫn cài đặt

- Cài đặt remi repo:

```
yum -y update
yum -y install http://rpms.remirepo.net/enterprise/remi-release-7.rpm
```

- Cài đặt Redis:

```
yum --enablerepo=remi install redis
rpm -qi redis
```

- Cấu hình firewall:

```
firewall-cmd --add-port=6379/tcp --permanent
firewall-cmd --reload
```

- Cấu hình sysctl:

+ Cấu hình file `/etc/sysctl.conf`

```
vm.overcommit_memory=1
```

+ Update cấu hình:

```
sysctl vm.overcommit_memory=1
```

- Cấu hình systemctl và start service:

```
systemctl enable --now redis
```

4. Cấu hình máy chủ Master

- Mở file cấu hình /etc/redis.conf và thực hiện cập nhật các cấu hình sau:

```
#bind 127.0.0.1
maxmemory-policy noeviction
tcp-keepalive 60
appendonly yes
appendfilename "appendonly.aof"
protected-mode no
```

- Restart service:

```
systemctl restart redis.service
```

5. Cấu hình máy chủ Slave

- Mở file cấu hình /etc/redis.conf và thực hiện cập nhật các cấu hình sau:

```
#bind 127.0.0.1
replicaof 192.168.0.1 6379
appendonly yes
appendfilename "appendonly.aof"
protected-mode no
```

- Restart service:

```
systemctl restart redis.service
```

6. Cấu hình Sentinel

- Mở file cấu hình /etc/redis-sentinel.conf, **xóa các cấu hình cũ và cấu hình** như sau:

```
protected-mode no
port 26379
daemonize yes
pidfile "/var/run/redis-sentinel.pid"
logfile "/var/log/redis/sentinel.log"
dir .
sentinel deny-scripts-reconfig yes
sentinel monitor mymaster 192.168.0.1 6379 2
sentinel down-after-milliseconds mymaster 5000
sentinel failover-timeout mymaster 10000
```

- Cấu hình service và start service:

```
systemctl enable --now redis-sentinel.service
```

7. Kiểm tra replication

- Trên máy chủ `master`:

```
# redis-cli
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:2
```

```
slave0: ip=192.168.0.2,port=6379,state=online,offset=575099851,lag=0
slave1: ip=192.168.0.3,port=6379,state=online,offset=575099851,lag=0
master_replid: 7df345aed5b6aae7512865c77ee784913d3678d7
master_replid2: d4584c75d2c5e03284e98511dba94cc095ada933
master_repl_offset: 575099851
second_repl_offset: 16516
repl_backlog_active: 1
repl_backlog_size: 1048576
repl_backlog_first_byte_offset: 574051276
repl_backlog_histlen: 1048576
```

- Trên 2 máy chủ `slave`:

```
$ redis-cli
127.0.0.1:6379> info replication
# Replication
role: slave
master_host: 192.168.0.1
master_port: 6379
master_link_status: up
master_last_io_seconds_ago: 1
master_sync_in_progress: 0
slave_repl_offset: 575120130
slave_priority: 100
slave_read_only: 1
connected_slaves: 0
master_replid: 7df345aed5b6aae7512865c77ee784913d3678d7
master_replid2: 0000000000000000000000000000000000000000
master_repl_offset: 575120130
second_repl_offset: -1
repl_backlog_active: 1
repl_backlog_size: 1048576
repl_backlog_first_byte_offset: 574071555
repl_backlog_histlen: 1048576
```

- Giờ bạn thực hiện stop redis trên máy chủ `192.168.0.1`, nếu 1 trong 2 máy chủ `192.168.0.2` hoặc `192.168.0.3` được up thành máy chủ redis master là quá trình cài đặt diễn ra thành công

Xem thêm: Hướng dẫn cài đặt Redis trên CentOS 7/6

Hướng dẫn cài đặt Redis trên CentOS 7/6

Redis là tên viết tắt của Remote Dictionary Server (Máy chủ từ điển từ xa), là một phần mềm mã nguồn mở được dùng để lưu trữ một cách tạm thời trên bộ nhớ (hay còn gọi là cache data) và giúp truy xuất dữ liệu một cách nhanh chóng. Do tốc độ truy xuất dữ liệu vượt trội, Redis thường được chọn sử dụng cho hoạt động lưu trữ bộ nhớ đệm Caching, quản lý phiên, trò chơi, bảng xếp hạng, phân tích theo thời gian thực, dữ liệu không gian địa lý, ứng dụng đặt xe, trò chuyện/nhắn tin, phát trực tiếp nội dung phương tiện và pub/sub.

1. Hướng dẫn cài đặt Redis

- Cài đặt remi repository:

```
## CentOS 7 ##
yum install epel-release
rpm -Uvh http://rpms.famillecollet.com/enterprise/remi-release-7.rpm

## CentOS 6 ##
yum install epel-release
rpm -Uvh http://rpms.famillecollet.com/enterprise/remi-release-6.rpm
```

- Cài đặt redis:

```
yum --enablerepo=remi install redis
rpm -qi redis
```

- Chạy Redis và tự động khởi động khi boot:

```
## Centos 7 ##
systemctl enable --now redis

## Centos 6 ##
chkconfig redis on
service redis start
```

2. Kiểm tra lại quá trình cài đặt Redis

- Check lại Redis server:

```
redis-cli ping
```

Nếu kết quả về `PONG` là ok

- Redis shell tools:

Mặc định Redis cài đặt với một công cụ comment là `redis-cli`

Sau khi khởi động Redis, các bạn có thể sử dụng một số command như:

- FLUSHALL – clear all databases
- SELECT # – select database under index #
- FLUSHDB – empty currently selected database
- KEYS * – list all keys from currently selected

Xem danh sách đầy đủ command ở đây.

Hướng dẫn kết nối Redis sử dụng Redisson trong Java

1. Giới thiệu

Redisson là một Redis client trong Java. Trong bài viết này, chúng ta sẽ tìm hiểu một số tính năng và demo cách xây dựng một ứng dụng sử dụng Redisson.

2. Dependency

Khai báo dependency trong maven pom.xml

```
<dependency>
    <groupId>org.redisson</groupId>
    <artifactId>redisson</artifactId>
    <version>3.3.0</version>
</dependency>
```

3. Cấu hình

Chúng ta cần cấu hình Redisson để kết nối đến Redis. Redisson hỗ trợ các kết nối đến Redis như sau:

- Single node (Kết nối đến Redis dạng singleton) - Get single node settings here
- Master with slave nodes (Kết nối đến Redis dạng master-slave) - Get master-slave node settings here

- Sentinel nodes (Kết nối đến Redis kiểu sentinel) - Get sentinel node settings here
- Clustered nodes (Kết nối đến Redis cluster) - Get clustered node settings here
- Replicated nodes (Kết nối đến Redis replicated) - Get replicated node settings here

3.1. Java Configuration - Khai báo cấu hình trực tiếp trong Java

Chúng ta có thể cấu hình để tạo kết nối đến Redis trực tiếp trong Java như sau:

```
Config config = new Config();
config.useSingleServer()
  .setAddress("127.0.0.1:6379");

RedissonClient client = Redisson.create(config);
```

3.2. File Configuration - Khai báo cấu hình trên file

Redisson hỗ trợ load cấu hình từ file định dạng JSON hoặc YAML.

```
Config config = Config.fromJSON(new File("singleNodeConfig.json"));
RedissonClient client = Redisson.create(config);
```

Đây là mẫu config cho `singleNodeConfig.json`

```
{
    "singleServerConfig": {
        "idleConnectionTimeout": 10000,
        "pingTimeout": 1000,
        "connectTimeout": 10000,
        "timeout": 3000,
        "retryAttempts": 3,
        "retryInterval": 1500,
        "reconnectionTimeout": 3000,
        "failedAttempts": 3,
        "password": null,
        "subscriptionsPerConnection": 5,
        "clientName": null,
        "address": "redis://127.0.0.1:6379",
        "subscriptionConnectionMinimumIdleSize": 1,
        "subscriptionConnectionPoolSize": 50,
        "connectionMinimumIdleSize": 10,
        "connectionPoolSize": 64,
```

```
        "database": 0,
        "dnsMonitoring": false,
        "dnsMonitoringInterval": 5000
    },
    "threads": 0,
    "nettyThreads": 0,
    "codec": null,
    "useLinuxNativeEpoll": false
}
```

Đây là mẫu config cho `singleNodeConfig.yaml`

```
singleServerConfig:
    idleConnectionTimeout: 10000
    pingTimeout: 1000
    connectTimeout: 10000
    timeout: 3000
    retryAttempts: 3
    retryInterval: 1500
    reconnectionTimeout: 3000
    failedAttempts: 3
    password: null
    subscriptionsPerConnection: 5
    clientName: null
    address: "redis://127.0.0.1:6379"
    subscriptionConnectionMinimumIdleSize: 1
    subscriptionConnectionPoolSize: 50
    connectionMinimumIdleSize: 10
    connectionPoolSize: 64
    database: 0
    dnsMonitoring: false
    dnsMonitoringInterval: 5000
threads: 0
nettyThreads: 0
codec: !<org.redisson.codec.JsonJacksonCodec> {}
useLinuxNativeEpoll: false
```

Để lưu từ cấu hình Java sang file JSON hoặc YAML config, chúng ta thực hiện như sau:

```
Config config = new Config();
// ... we configure multiple settings here in Java
String jsonFormat = config.toJSON();
String yamlFormat = config.toYAML();
```

4. Redisson operation

Redisson hỗ trợ các interface để thực thi synchronous, asynchronous and reactive interfaces.

VD: Synchronous

```
RedissonClient client = Redisson.create();
RAtomicLong myLong = client.getAtomicLong('myLong');
```

VD: Asynchronous

```
RFuture<Boolean> isSet = myLong.compareAndSetAsync(6, 27);
isSet.handle((result, exception) -> {
    // handle the result or exception here.
});
```

VD: Reactive

```
RedissonReactiveClient client = Redisson.createReactive();
RAtomicLongReactive myLong = client.getAtomicLong("myLong");

Publisher<Boolean> isSetPublisher = myLong.compareAndSet(5, 28);
```

5. Objects

Redisson định nghĩa các Objec như sau:

- ObjectHolder
- BinaryStreamHolder
- GeospatialHolder
- BitSet
- AtomicLong
- AtomicDouble
- Topic
- BloomFilter
- HyperLogLog

5.1. ObjectHolder

ObjectHolder có thể lưu trữ tất cả các kiểu dữ liệu.

```
RBucket<Ledger> bucket = client.getBucket("ledger");
bucket.set(new Ledger());
Ledger ledger = bucket.get();
```

5.2. BinaryStreamHolder

5.3. GeospatialHolder

5.4. BitSet

5.5. AtomicLong

AtomicLong lưu trữ giá trị kiểu long

```
RAtomicLong atomicLong = client.getAtomicLong("myAtomicLong");
atomicLong.set(5);
atomicLong.incrementAndGet();
```

5.6. AtomicDouble

5.7. Topic

Topic để lưu trữ các bản tin trong cơ chế pub-sub của Redis.

```
RTopic<CustomMessage> subscribeTopic = client.getTopic("baeldung");
subscribeTopic.addListener(
  (channel, customMessage)
  -> future.complete(customMessage.getMessage()));
```

5.8. BloomFilter

5.9. HyperLogLog

6. Collections

Redisson định nghĩa và hỗ trợ các kiểu Collection như sau:

- Map
- Multimap
- Set
- SortedSet

- ScoredSortedSet
- LexSortedSet
- List
- Queue
- Deque
- BlockingQueue
- BoundedBlockingQueue
- BlockingDeque
- BlockingFairQueue
- DelayedQueue
- PriorityQueue
- PriorityDeque

6.1. Map

Redisson Map được implement từ *java.util.concurrent.ConcurrentMap* và *java.util.Map* interfaces.

Bao gồm: *RMap*, *RMapCache*, *RLocalCachedMap* và *RClusteredMap*

```
RMap<String, Ledger> map = client.getMap("ledger");
Ledger newLedger = map.put("123", new Ledger());
```

6.2. Multimap

6.3. Set

Redisson *Set* được implement từ *java.util.Set* interface.

Bao gồm: *RSet*, *RSetCache*, và *RClusteredSet*.

```
RSet<Ledger> ledgerSet = client.getSet("ledgerSet");
ledgerSet.add(new Ledger());
```

6.4. SortedSet

6.5. ScoredSortedSet

6.6. LexSortedSet

6.7. List

Redisson *Lists* được implement từ *java.util.List* interface.

Let's create a *List* with Redisson:

```
RList<Ledger> ledgerList = client.getList("ledgerList");
ledgerList.add(new Ledger());
```

6.8. Queue

6.9. Deque

6.10. BlockingQueue

6.11. BoundedBlockingQueue

6.12. BlockingDeque

6.13. BlockingFairQueue

6.14. DelayedQueue

6.15. PriorityQueue

6.16. PriorityDeque

7. Locks and Synchronizers

Redisson cho phép các thread thực hiện khóa (lock), đồng bộ (synchronization) qua applications/servers. Bao gồm:

- Lock
- FairLock
- MultiLock
- ReadWriteLock
- Semaphore
- PermitExpirableSemaphore
- CountDownLatch

7.1. Lock

Redisson's *Lock* được implement từ *java.util.concurrent.locks.Lock* interface.

```
RLock lock = client.getLock("lock");
lock.lock();
// perform some long operations...
lock.unlock();
```

7.2. FairLock

7.3. MultiLock

Redisson's *RedissonMultiLock* nhòm nhiều đối tượng kiểu *RLock* để trở thành 1 lock block:

```
RLock lock1 = clientInstance1.getLock("lock1");
RLock lock2 = clientInstance2.getLock("lock2");
RLock lock3 = clientInstance3.getLock("lock3");

RedissonMultiLock lock = new RedissonMultiLock(lock1, lock2, lock3);
lock.lock();
// perform long running operation...
lock.unlock();
```

7.4. ReadWriteLock

7.5. Semaphore

7.6. PermitExpirableSemaphore

7.7. CountDownLatch

8. Service

8.1. Remote Service

Cơ chế này hỗ trợ kiểu thực thi trên server-side từ remote method được gọi từ client-side:

Server-side register:

```
RRemoteService remoteService = client.getRemoteService();
LedgerServiceImpl ledgerServiceImpl = new LedgerServiceImpl();

remoteService.register(LedgerServiceInterface.class, ledgerServiceImpl);
```

Client-side call remote method:

```
RRemoteService remoteService = client.getRemoteService();
LedgerServiceInterface ledgerService
  = remoteService.get(LedgerServiceInterface.class);
```

```
List<String> entries = ledgerService.getEntries(10);
```

8.2. Live Object Service

9. Pipelining

Redisson hỗ trợ pipelining để thực thi một batch các lệnh:

```
RBatch batch = client.createBatch();
batch.getMap("ledgerMap").fastPutAsync("1", "2");
batch.getMap("ledgerMap").putAsync("2", "5");

List<?> result = batch.execute();
```

10. Scripting

Redisson hỗ trợ LUA scripting. Chúng ta có thể thực thi LUA scripts đến Redis:

```
client.getBucket("foo").set("bar");
String result = client.getScript().eval(Mode.READ_ONLY,
"return redis.call('get', 'foo')", RScript.ReturnType.VALUE);
```

11. Low-Level Client

Redisson hỗ trợ Low-level Client để thực thi các native Redis command:

```
RedisClient client = new RedisClient("localhost", 6379);
RedisConnection conn = client.connect();
conn.sync(StringCodec.INSTANCE, RedisCommands.SET, "test", 0);

conn.closeAsync();
client.shutdown();
```

12. Ví dụ mẫu

Xem thêm tại đây