Nội dung

[**I.** **Giới thiệu** 1](#_Toc503297207)

[**1.** **Khái quát** 1](#_Toc503297208)

[**2.** **Đặc trưng của Redis** 2](#_Toc503297209)

[**3.** **Lịch sử ra đời Redis** 2](#_Toc503297210)

[**4.** **Những ứng dụng nên dùng Redis** 3](#_Toc503297211)

[**5.** **Cài đặt Redis trên windows** 3](#_Toc503297212)

[**II.** **Các lệnh cơ bản trong Redis** 6](#_Toc503297213)

[**1.** **Redis Key** 6](#_Toc503297214)

[**2.** **Redis String** 7](#_Toc503297215)

[**3.** **Redis List** 8](#_Toc503297216)

[**4.** **Redis Set** 10](#_Toc503297217)

[**5.** **Redis Hash** 10](#_Toc503297218)

[**6.** **Redis Zset** 12](#_Toc503297219)

[**7.** **Redis Transaction** 13](#_Toc503297220)

[**III.** **Kiến trúc và cơ chế** 13](#_Toc503297221)

[**1.** **In – Memory Key – Value Store: Lưu trữ dưới cặp khóa và giá trị trên bộ nhớ RAM.** 13](#_Toc503297222)

[**2.** **Kiến trúc đơn sơ của Redis** 14](#_Toc503297223)

[**3.** **Persistence (Duy trì, lưu trữ dữ liệu cứng)** 14](#_Toc503297224)

[**4.** **Replication(Nhân bản)** 15](#_Toc503297225)

[**5.** **Partition** 17](#_Toc503297226)

[**6.** **Redis connection client** 18](#_Toc503297227)

[**IV.** **Redis với Java** 19](#_Toc503297228)

[**1.** **Redis STRING** 20](#_Toc503297229)

[**2.** **Redis LISTS** 21](#_Toc503297230)

[**3.** **Redis HASH** 21](#_Toc503297231)

[**4.** **Redis SET** 21](#_Toc503297232)

[**5.** **Redis ZSET** 22](#_Toc503297233)

[**V.** **Nguồn tham khảo** 22](#_Toc503297234)

[**VI.** **Link hướng dẫn cài cặt và sử dụng Redis với Java** 23](#_Toc503297235)

# **Giới thiệu**

## **Khái quát**

* Nếu bạn đã từng làm việc với Object của Javascript, .Net hay đơn giản bạn đã từng đọc một file JSON, thì chắc hẳn bạn đã biết đến cấu trúc dạng key-value, thì Redis cũng tương tự như vậy.
* **Redis** là một cơ sở dữ liệu nguồn mở lưu trữ dữ liệu theo dạng key-value.
* **Redis** cũng thường được gọi là server lưu trữ các cấu trúc dữ liệu (data structure server) do các khóa có thể bao gồm các dữ liệu kiểu strings, hashes, lists, sets và sorted sets.
* **Redis** được lưu trữ key-value trên RAM với hiệu năng cao, redis còn hỗ trợ lưu trữ dữ liệu trên đĩa cứng (persistent redis) cho phép phục hồi dữ liệu khi gặp sự cố
* Redis hỗ trợ thêm mới, cập nhật và loại bỏ dữ liệu nhanh chóng
* Redis có những đặc điểm giống như Memcached như:
* Lưu trữ dạng key /value.
* Tất cả data được lưu trên Memory(RAM)
* Key có thể hết hạn(expire) hoặc không
* Nhanh(Fast), nhẹ nhàng(light-weight)
* Redis có thêm nhiều đặc điểm, chức năng khác mang lại lợi ích khi sử dụng và triển khai
  + Persistence
  + Hỗ trợ nhiều Databases
  + Truy vấn theo Key
  + Hỗ trợ counters dữ liệu kiểu integer
  + Cấu trúc dữ liệu cấp cao
  + Thao tác dữ liệu chuyên biệt
  + Tự động phân trang danh sách
  + Nhân rộng master-slave
* Redis lấy và nạp dữ liệu trên Memory(RAM), nhưng tại một thời điểm thì dữ liệu có thể được lưu trữ trên disk(Data in memory, but saved on disk).
* Điểm khác biệt dễ nhận thấy của Redis là: Key là một string nhưng value thì không giới hạn ở một string mà có thể là List, Sets, Sorted sets, ....
* Redis hỗ trợ “Multiple database” với nhiều commands để tự động remove key từ một database tới database khác.
* Mặc định thì DB 0 sẽ được lựa chọn cho mỗi lần kết nối(connection), nhưng khi sử dụng lệnh SELECT(SELECT command) thì nó có thể select/create một database khác. Thao tác MOVE(MOVE operation) có thể chuyển một item từ một DB tới DB khác một cách tự động.
* Redis rất nhanh trong các thao tác lấy và nạp dữ liệu do redis hỗ trợ nhiều lệnh mang tính chất chuyên biệt.
* Redis hỗ trợ mở rộng master-slave nếu chúng ta muốn sự an toàn hoặc mở rộng, co giãn trong việc lưu trữ data

## **Đặc trưng của Redis**

* Redis không có table mà chỉ có Data model
* Điểm đặc biệt nhất của Redis đó chính là không có table, vì chúng ta đã nói ở phần trước Redis lưu trữ data dưới dạng key-value. Thực thế thì memcache cũng làm vậy, có thể dễ hiểu hơn Memcached là một hệ thống lưu trữ bản sao các đối tượng (objects) và dữ liệu được truy cập nhiều lần để tăng tốc độc truy xuất. Mục đích chính của nó là để tăng tốc độ ứng dụng web bằng cách truy vấn cơ sở dữ liệu bộ nhớ đệm, nội dung, hoặc kết quả tính toán khác.
* Tuy nhiên kiểu dữ liệu của memcache lại bị hạn chế, không đa dạng như Redis, do đó không hỗ trợ được nhiều thao tác từ phía client, bạn có thể tham khảo các kiểu dữ liệu của Redis để lưu value:
* **STRING**: Có thể là string, integer hoặc float. Redis có thể làm việc với cả string, từng phần của string, cũng như tăng/giảm giá trị của integer, float.
* **LIST**: 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.
* **SET**: 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.
* **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.
* **ZSET** **(sorted set):** 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.

1. **Lịch sử ra đời Redis**

* Tác giá của Redis là Salvatore Sanfilippo (nickname: antirez), hiện đang phát triển Redis và được sponsor bởi VMWare.



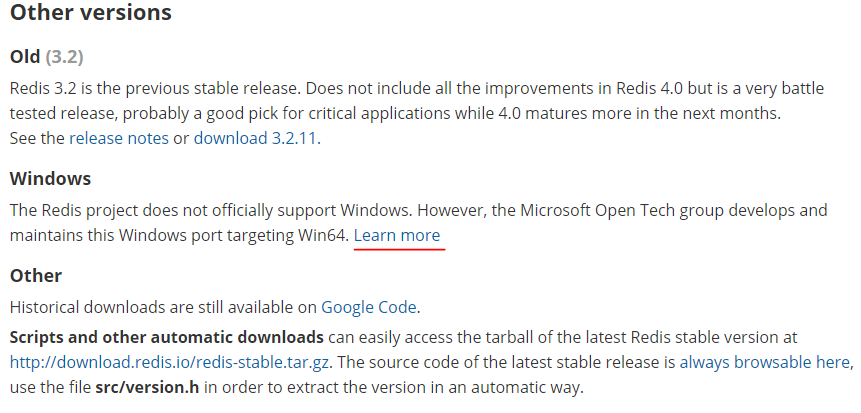
* Redis ra đời khi mà SQL Database không thể nào đáp ứng được công việc mà @antiez làm. Cụ thể Server của antirez nhận 1 lượng lớn thông tin từ nhiều trang web khác nhau thông qua JavaScript tracker, lưu trữ n page view cho trừng trang và hiển thị chúng theo thời gian thực cho user, kèm theo đó là lưu trữ 1 lượng nhỏ lịch sử hiển thị của trang web. Khi số lượng page view tăng đến hàng nghìn page trên 1 giây, antirez không thể tìm ra cách tiếp cận nào thực sự tối ưu cho việc thiết kế database của mình.
* Tuy nhiên, anh ta nhận ra rằng, việc lưu trữ 1 danh sách bị giới hạn các bản ghi không phải là vấn đề quá khó khăn. Từ đó, ý tưởng lưu trữ thông tin trên RAM và quản lý các page views dưới dạng native data với thời gian pop và push là hằng số được ra đời. Antirez bắt tay vào việc xây dựng prototype bằng C, bổ sung tính năng lưu trữ thông tin trên đĩa cứng và… Redis ra đời.
* Redis được ra đời để giải quyết các vấn đề mà chính tác giả của nó gặp phải trong công việc của mình.

1. **Những ứng dụng nên dùng Redis**

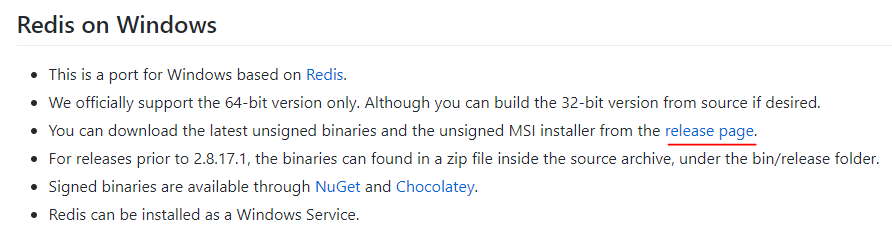
* Không phải ứng dụng nào chúng ta cũng áp dụng Redis vào được, Redis là một lựa chọn tuyệt vời nếu ta cần đến một server lưu trữ dữ liệu đòi hỏi tính mở rộng cao (scaleable) và chia sẻ bởi nhiều tiền trình, nhiều ứng dụng và nhiều server khác nhau.
* Các ứng dụng yêu cầu cơ sở dữ liệu với lượng record được lưu vào rất lớn, Redis rất nhanh có thể thực hiện khoảng 110000 set dữ liệu trên giây, và 81000 get dữ liệu/giây.
* Ứng dụng có thể tương tác cross-platform (đa nền tảng), cross-server (đa máy chủ), và cross-application (đa ứng dung) đã làm Redis trở thành một lựa chọn đúng đắn cho rất rất nhiều công việc khác nhau. Tốc độ cực cao của Redis cũng có thể được lợi dụng để làm caching layer

1. **Cài đặt Redis trên windows**

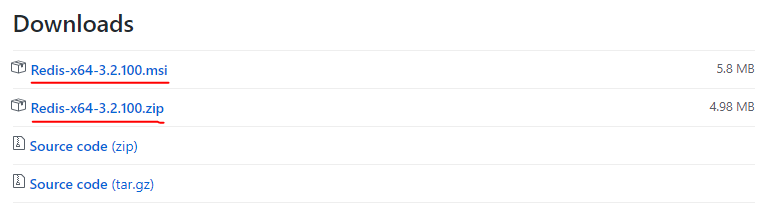
* Vào đường dẫn <https://redis.io/download>
* Tìm đến phần dành cho windows -> chọn Learn more



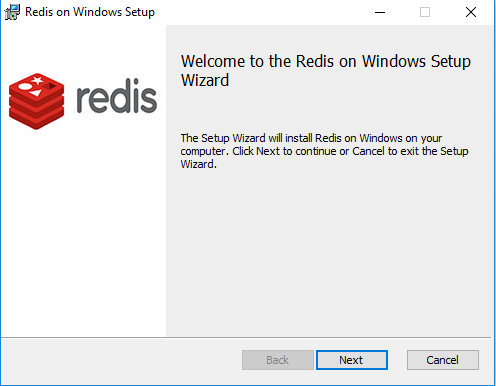
* Sẽ hiện ra một trang github chứa source cài đặt redis -> tìm đến phần Redis on windows -> chọn release page



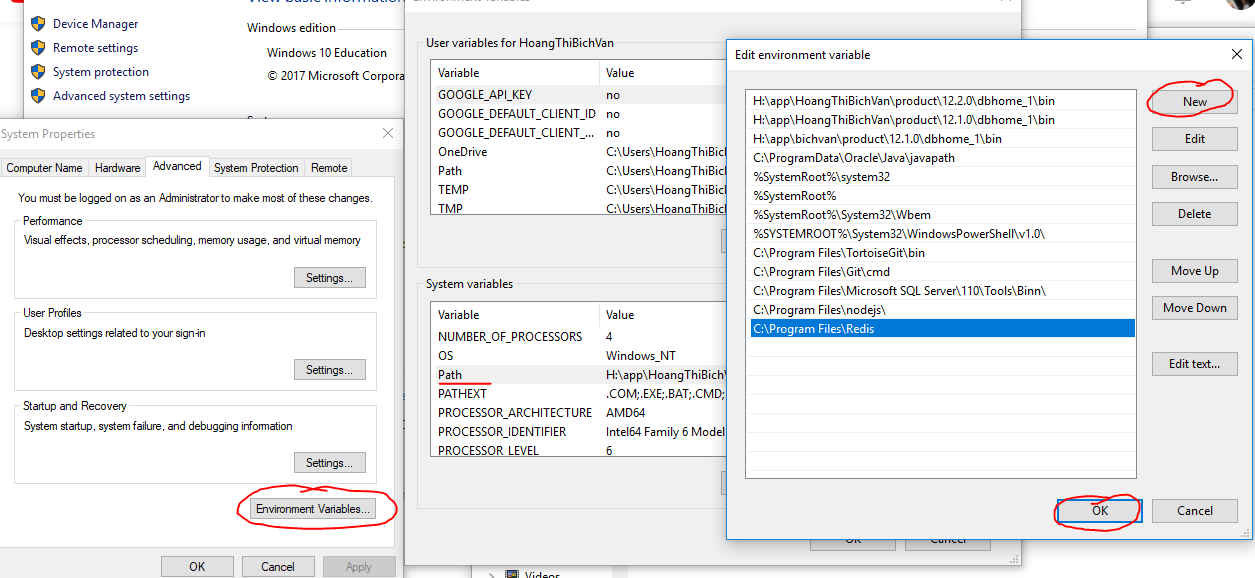
* Tìm đến phần Downloads -> có thể chọn 1 trong 2 file đuôi .msi hoặc .zip



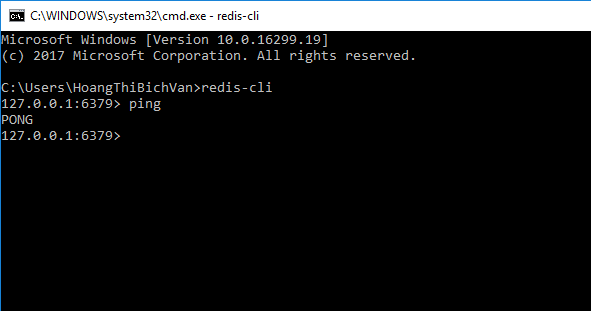
* ở đây mình download file .msi -> click vào file vừa download để tiến hành cài đặt, cấu hình-> như cài các phần mềm khác thì thao tát rất đơn giản là chỉ cần nhấn next… và install là đã có thể cài redis thành công



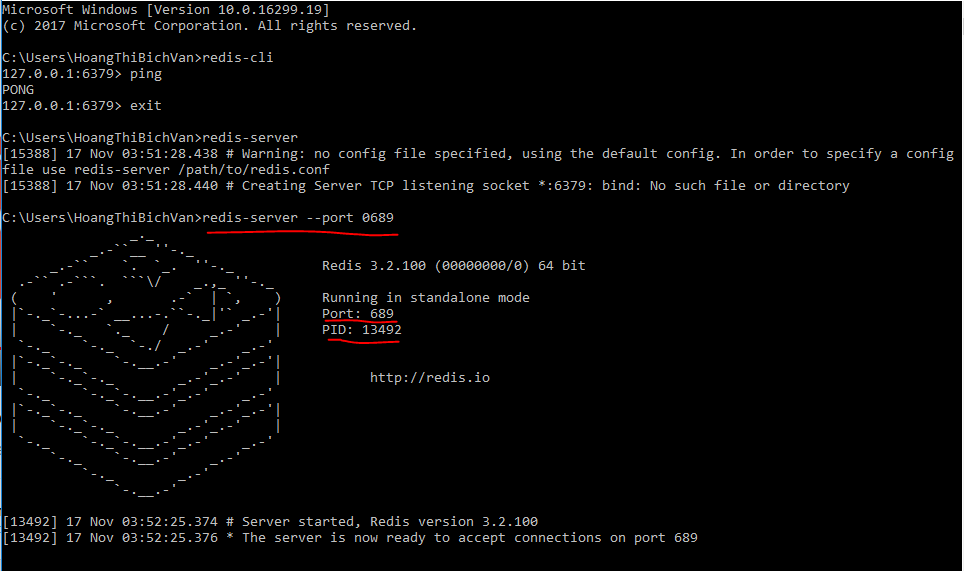
* Copy đường dẫn C:\Program Files\Redis tiến hành add vào path như hình bên dưới



* Sau đó mở cmd lên gõ câu lệnh **redis-cli** để kiểm tra ta có được thông tin 127.0.0.1 là server(localhost), 6379 là port mặc định -> tiếp đến gõ **ping** nếu hiện **PONG** là đã cài đặt thành công.



* Run server redis bằng câu lệnh **redis-server** nhận được thông báo chưa config tiếng hành đổi port bằng lệnh **redis-server –port 8080**  số port bạn muốn -> run server thành công. Nếu báo lỗi thì ta tiếng hành đổi port vì port đó có thể là đã được sử dụng



1. **Các lệnh cơ bản trong Redis**
2. **Redis Key**

Redis key là các lệnh sử dụng để quản lý các key trong redis. Với cú pháp như sau:

COMMAND KEY\_NAME [VALUE]

Các lệnh thường dùng:

|  |  |  |
| --- | --- | --- |
| **STT** | **Command** | **Ý nghĩa** |
| 1 | DEL key | Xóa key nếu nó tồn tại |
| 2 | EXISTS key | Kiểm tra key có tồn tại không |
| 3 | EXPIRE key n | Đặt expire time cho key sau n giây |
| 4 | PERSIST key | Xóa expire time của key |
| 6 | TTL key | Lấy thời gian sống của key (giây) |
| 7 | 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 |
| 8 | RENAMENX key newkey | Đổi tên key sang newkey nếu newkey chưa tồn tại |
| 9 | TYPE key | Lấy loại dữ liệu được lưu trữ bởi key |

## **Redis String**

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

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 | DEL key | Xoá key và giá trị tương ứng |
| 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 |
| 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 |

## **Redis List**

List trong Redis là linked list, lưu trữ 1 danh sách có thứ tự (trước sau) của các string. Cách lưu trữ này giúp cho thời gian add thêm 1 phần tử vào đầu hoặc cuối list là hằng số, bất kể size của list là bao nhiêu. Lợi thế này cũng có 1 mặt trái là việc truy xuất đến phần tử theo index của linked list là lâu hơn rất nhiều so với array.

Redis list là lệnh sử dụng để quản lý các key/value trong đó value có giá trị là một list (danh sách). List là kiểu dữ liệu khá phổ biến, có 2 kiểu list thường dùng là stack (vào sau ra trước) và queue (vào trước ra trước).

Các lệnh thường dùng:

|  |  |  |
| --- | --- | --- |
| **STT** | **Command** | **Ý nghĩa** |
| 1 | RPUSH | Cho thêm 1 giá trị vào phía bên phải của list |
| 2 | LRANGE | Lấy 1 dải giá trị của list |
| 3 | LPOP key | Lấy phần tử ở đầu danh sách |
| 4 | LINDEX | Lấy ra phần tử của list dựa theo index |
| 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 |

Trang redis.io đưa ra 2 trường hợp phổ biến cho việc dùng list. 1 là lưu lại các post mới nhất của users trên mạng xã hội, điển hình là mạng xã hội nổi tiếng Twitter dùng Redis cho việc này (lưu các tweets mới nhất của users). 2 là xây dựng mô hình tương tác giữa consumer và producer, trong đó producer đưa item vào list và consumer dùng các item này theo thứ tự quy định trong list.

## **Redis Set**

Set trong Redis khá giống với list, nhưng khác 1 điều là các phần tử trong set không được sắp xếp theo thứ tự nào cả. Tuy nhiên, Redis đã tăng performance khi làm việc với set bằng cách sử dụng 1 bảng băm (hash table) để lưu trữ các phần tử của set. Hiểu đơn giản thì mỗi item được add vào set sẽ là 1 key trong bảng băm, còn value thì không có. Việc làm này giúp theo tác truy xuất dữ liệu trên SET nhanh hơn nhiều (do tận dụng ưu thế về tốc độ tìm kiếm trên bảng băm), nhất là khi muốn đảm bảo không bị trùng lặp phần tử trong set.

Redis set là lệnh sử dụng để quản lý các key/value trong đó value có giá trị là một set (tập hợp). Các giá trị trong tập hợp là duy nhất không bị trùng lặp

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 |

Cũng cần nói thêm rằng độ phức tạp tính toán của SISMEMBER là O(1), đây là ưu thế rất lớn của set so với dùng list khi sử dụng bảng băm. Chi tiết hơn về độ phức tạp tính toán trong Redis sẽ được nhắc đến ở phần sau của bài viết này.

## **Redis Hash**

Không giống như LIST và SET lưu trữ 1 tập dữ liệu là các string, HASH lưu trữ tập các map của key và value. Key vẫn là string, còn value có thể là string hoặc số. Nếu là số thì chúng ta có thể làm các thao tác tăng, giảm giá trị 1 cách đơn giản. HASH được coi là 1 mô hình thu nhỏ của Redis, khi dữ liệu được tổ chức dạng key-value.

Redis hash là lệnh sử dụng để quản lý các key/value trong đó value có giá trị là hash. Hash là kiểu dữ liệu khá phổ biến, thường được dùng để lưu trữ các object.

|  |  |  |
| --- | --- | --- |
| **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 |
| 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 ... |

## **Redis Zset**

Sorted Set (ZSET) là 1 phiên bản đầy đủ của set, khi mà phần value của item được thiết lập, và bắt buộc là 1 số (float number) được gọi là score. Ở điểm này thì zset khá giống với hash khi lưu trữ 1 cặp key, value (trong zset gọi là member và score). Và vì là “sorted”, nên các cặp member-score được add vào sorted set sẽ được sắp xếp theo thứ tự của các score, nếu score trùng nhau thì tiếp tục sắp xếp theo member. Ngoài ra cũng cần chú ý là không cho phép 2 phần tử khác nhau của zset có member trùng nhau.

Redis sorted set là lệnh sử dụng để quản lý các key/value trong đó value có giá trị là một sorted set (tập hợp được sắp xếp theo điểm/độ ưu tiên từ thấp đến cao). Các giá trị trong sorted set là duy nhất không bị trùng lặp.

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 |
| 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 |

ZSET là 1 cấu trúc dữ liệu đặc biệt của riêng Redis, nó chuyên dùng cho các bài toán dạng tìm “top”. Top người dùng theo score, top webpage theo số lượng view, “top whatever”. Cách Redis lưu trữ dữ liệu zset cũng rất thú vị, chúng ta sẽ tìm hiểu ở phần tiếp theo của bài viết.

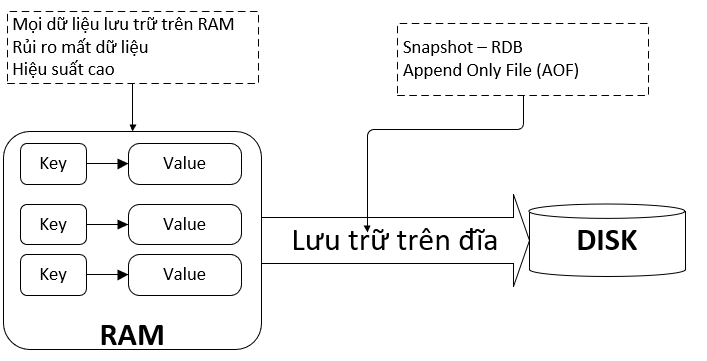
## **Redis Transaction**

Một điểm khá thú vị trong Redis là transaction. 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.

Các lệnh thường dùng:

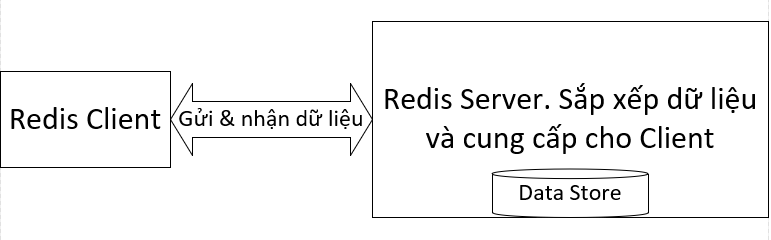
|  |  |  |
| --- | --- | --- |
| **STT** | **Command** | **Ý nghĩa** |
| 1 | MULTI | Đánh dấu bắt đầu khối lệnh transaction |
| 2 | EXEC | Thực hiện khối lệnh |

1. **Kiến trúc và cơ chế**
2. **In – Memory Key – Value Store: Lưu trữ dưới cặp khóa và giá trị trên bộ nhớ RAM.**



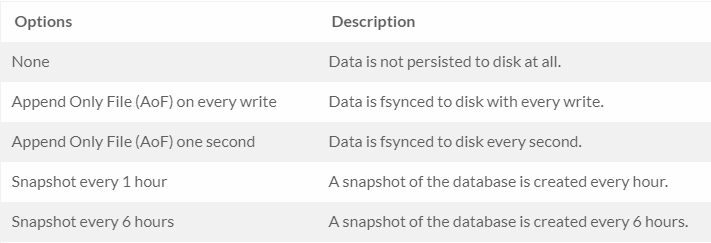
* + - Key: phải là một chuỗi
    - Value: có thể là string, list, set, sorted set hoặc hash.
    - Ưu điểm: Redis lưu trữ dữ liệu trong bộ nhớ chính (RAM) nên tốc độ đọc ghi dữ liệu rất nhanh.
    - Nhược điểm: Vì bộ nhớ RAM hạn chế, nên Redis không thể lưu được các dữ liệu lớn mà chỉ lưu những văn bản nhỏ mà cần truy cập, sữa đổi và chèn vào tới một tốc độ rất nhanh.

1. **Kiến trúc đơn sơ của Redis**



* + - Redis client: điều khiển truy vấn dữ liệu từ giao diện hoặc bất kỳ ngôn ngữ lập trình khác thông qua ngôn ngữ API của Redis để gửi yêu cầu đến server.
    - Redis server: chịu trách nhiệm cho việc sử lý và lưu trưc dữ liệu trong bộ nhớ. Nó xử lý tất cả các loại hình quản lý và tạo thành phần chính của kiến trúc.
    - Client và Server có thể nằm trên 1 máy hoặc 2 máy vật lý khác nhau.

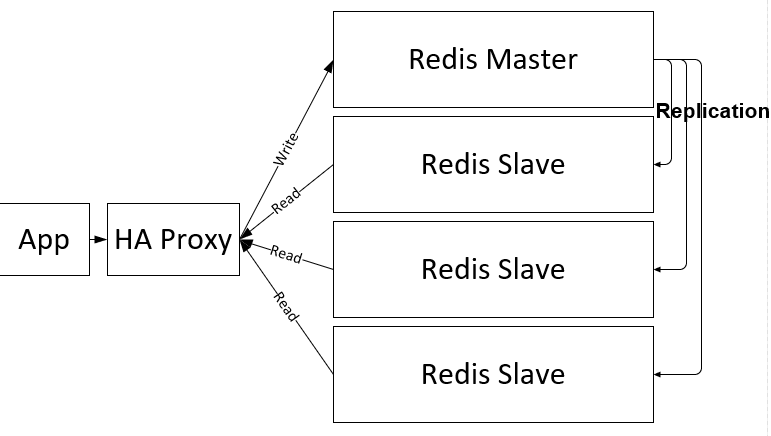
1. **Persistence (Duy trì, lưu trữ dữ liệu cứng)**
   * + RDB: tạo 1 bản sao toàn bộ dữ liệu từ ram và cất sang bộ nhớ đĩa. Điều này được thực hiện định kỳ, do đó sẽ mất dữ liệu kể từ bản sao cuối. Phù hợp cho việc phục hồi từ thảm họa, dể dàng cất dữ trên trung tâm dữ liệu. Khởi động nhanh.
     + AOF: ghi log tất cả các hoạt động ghi mà được nhận bởi server, mọi thứ đều bền bỉ nhưng file có kích thước lớn hơn so với kiểu RDB.
     + SAVE Command: Người dùng có thể ra lệnh cho máy chủ thực hiện RDB bất cứ lúc nào.
     + Các tham số lựa chọn cấu hình



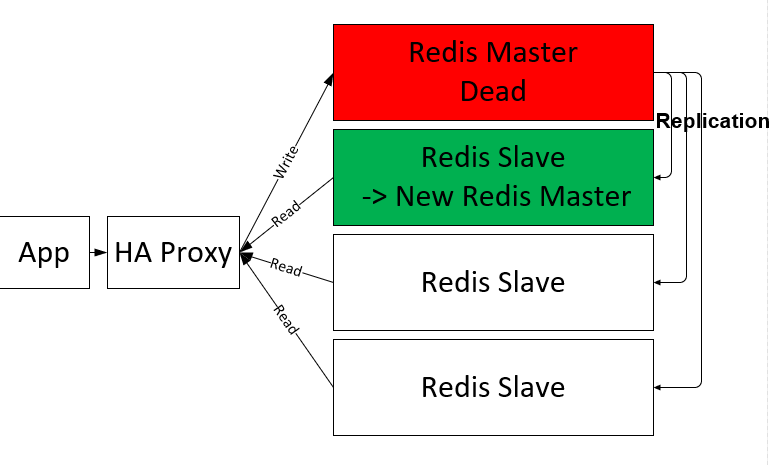
* + - So sánh RDB với AOF

|  |  |  |
| --- | --- | --- |
|  | AOF | RDB |
|  | Hao bộ nhớ đĩa | Ít tốn bộ nhớ đĩa |
|  | Cung cấp độ bền tốt hơn (phục hồi điểm mới nhất trong thời gian) | Ít bền |
|  | Thời gian phục hồi lâu do kích thước file lớn | Phục hồi nhanh |
|  | Không gian đĩa cần mở rộng, vì File tăng kích cỡ theo thời gian | Không gian đĩa ít mở rộng, vì thực hiện một lần vài giờ |

1. **Replication(Nhân bản)**
   * + Là một kỹ thuật liên quan đến nhiều máy tính để cho phép chia sẻ khả năng truy cập dữ liệu. Trong một môi trường nhân rộng, nhiều máy tính chia sẻ cùng một dữ liệu với nhau để ngay cả khi một vài máy tính bị hỏng, tất cả dữ liệu sẽ có sẵn.
     + Tất cả các nô lệ có chứa chính xác cùng một dữ liệu như tổng thể. Có thể có nhiều như nô lệ cho mỗi máy chủ chính. Khi một nô lệ mới được đưa vào môi trường, thì nó sẽ tự động đồng bộ hóa tất cả dữ liệu.
     + Mọi truy vấn được đưa đến master, master sẽ thực hiện các hoạt động. Khi có thao tác ghi master sẽ phân phối dữ liệu đó đến Slave.



* + - Khi một master và một slave kết nối tốt, master giữ nô lệ cho việc cập nhật bằng cách gửi một luồng lệnh để tái tạo các thay đổi trên tập dữ liệu xảy ra trong bộ dữ liệu tổng thể. (lệnh ghi, key hết hạn, thu hồi key ….).
    - Khi mối kết nối giữa master và nô lệ bị đứt, do mạng hoặc timeout được cảm nhận trong master hoặc nô lệ, nô lệ kết nối lại và cố gắng tiến hành đồng bộ hóa, nó sẽ cố gắng để có được một phần của dòng lệnh từ server mà nó bị mất trong quá trình ngắt kết nối.
    - Khi không thể đồng bộ hoá một phần, nô lệ sẽ yêu cầu đồng bộ hóa lại toàn bộ. Điều này sẽ liên quan đến một quá trình phức tạp hơn, master cần tạo một ảnh chụp nhanh tất cả dữ liệu của nó, gửi nó đến nô lệ. Sau đó mọi tiến trình vẫn như củ.
    - Khi một Slave được đưa vào môi trường thì hệ thống sẽ tự đồng bộ hóa dữ liệu với Slave mới.
    - Master chết thì đưa Slave củ lên làm master, công việc được tiếp tục.



1. **Partition**

Phân vùng là quá trình chia tách dữ liệu của bạn thành nhiều Instances Redis, do đó Instances sẽ chỉ chứa một tập hợp các keys của bạn.

Lợi ích:

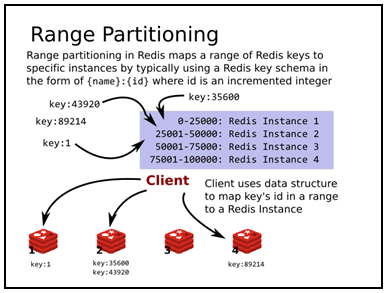
* Mở rộng bộ nhớ chứa dữ liệu, nhờ vào sử dụng nhiều máy tính
* Tăng tốc độ tính toán, nhờ vào nhiều lõi của các máy tính, băng thông mạng…

Bất lợi:

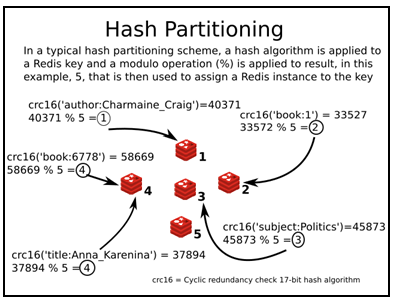
* Không thực hiện giao dữ liệu được, vì chứng ở khác Instances
* Không thể thực hiện được các transactions (1 hiện 1 nhóm các lệnh trong 1 bước).
* Khi phân vùng được sử dụng, xử lý dữ liệu phức tạp hơn, ví dụ bạn phải xử lý nhiều tệp RDB / AOF và sao lưu dữ liệu của bạn cần để tổng hợp các tệp persistence từ nhiều trường hợp và máy chủ.

Có 2 loại:

* Partitioning basics (Range Partitioning): Một trong những cách đơn giản nhất để thực hiện phân vùng là phân vùng theo phạm vi, và được thực hiện bằng cách ánh xạ các phạm vi của các đối tượng vào các trường hợp Redis cụ thể. Ví dụ, tôi có thể nói người dùng từ ID 0 đến ID 25000 sẽ đi vào ví dụ R1, trong khi người dùng ID 25001 đến ID 50000 sẽ đi vào ví dụ R1 và vân vân.



* Hash Partitioning: Phân vùng Hash là một sự lựa chọn của phân vùng Range. Trong phân vùng Hash, một hàm băm(CRC16, CRC32, MD5, SHA1) được sử dụng để chuyển đổi khóa thành một số và sau đó dữ liệu được lưu trữ trong các Instances nhau Redis khác nhau băng phương pháp mod.



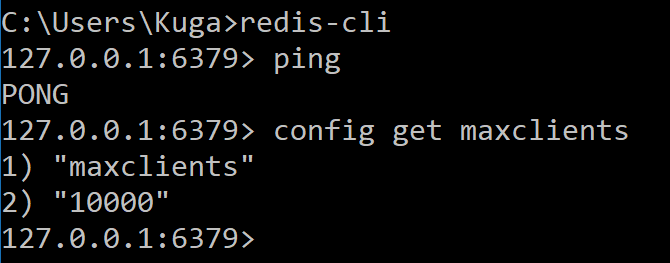
1. **Redis connection client**

Redis chấp nhận kết nối của client trên cổng TCP và trên ổ cắm Unix, nếu được kích hoạt. Khi một kết nối client mới được chấp nhận, các thao tác sau được thực hiện:

* Client socket được đặt trong trạng thái không khóa vì Redis sử dụng ghép kênh và không chặn I/O.
* Tùy chọn TCP\_NODELAY được thiết lập để đảm bảo không có sự chậm trể trong kết nối của chúng.
* Một tập tin sự kiện có thể đọc được tạo ra để Redis có thể thu thập các truy vấn của khách hàng ngay khi có dữ liệu mới được đọc trên ổ cắm.

Số lượng client tối đa:

* Trong Redis config (redis.conf), có một thuộc tính được gọi là maxclients, mô tả số lượng khách hàng tối đa có thể kết nối với Redis.



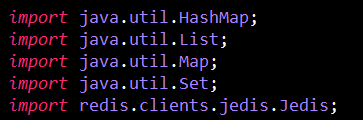
Các lệnh cơ bản của Client

* CLIENT LIST: trả về danh sách các client được kết nối tới server.
* CLIENT SETNAME: đặt tên cho kết nối hiện tại.
* CLIENT GETNAME: trả về tên của kết nối hiện tại mà được thiết lập bởi CLIENT SETNAME.
* CLIENT PAUSE: Đây là lệnh kiểm soát kết nối, có thể tạm ngừng tất cả các máy khách Redis trong một khoảng thời gian nhất định (tính bằng mili giây):
* CLIENT KILL <filter> <value> ... ... <filter> <value>: đóng kết nối client theo điều kiện lọc. CLIENT KILL addr 127.0.0.1:6379 type slave => đóng kết nối với client tại địa chỉ 127.0.0.1:6379 mà có type là slave
* CLIENT REPLY ON|OFF|SKIP: Kiểm soát xem máy chủ sẽ trả lời các lệnh của client hay không. Các chế độ sau có sẵn:

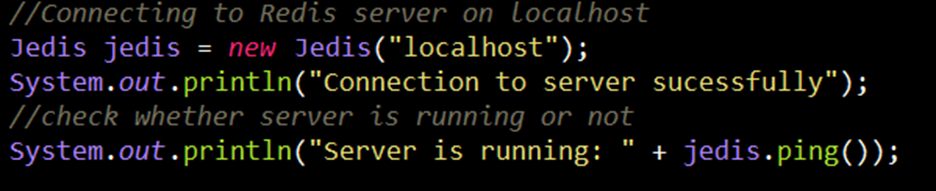
1. **Redis với Java**
   * Trước khi bắt đầu sử dụng Redis trong các chương trình Java của bạn, bạn cần đảm bảo rằng bạn có trình điều khiển Redis Java và Java được cài đặt trên máy.
   * Tải thư viện tại Jedis: <https://mvnrepository.com/artifact/redis.clients/jedis>



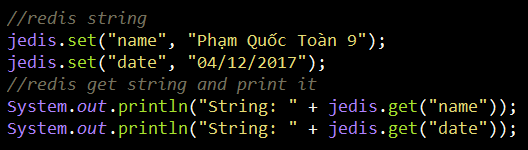
* + Trong Project ta thêm jedis.jar vào mục Libraries
  + Khai báo thư viện trong mã nguồn



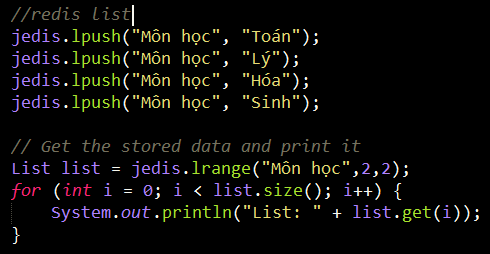
* + Kết nối java với Redis



1. **Redis STRING**

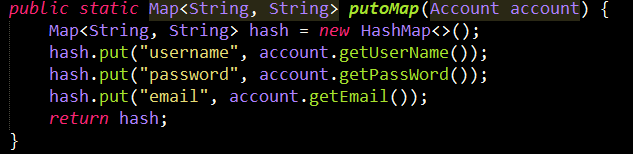


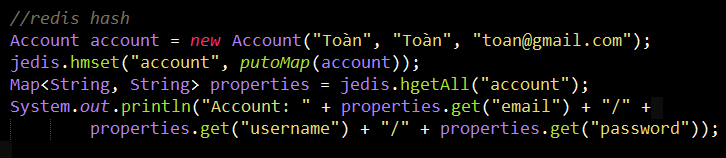
1. **Redis LISTS**



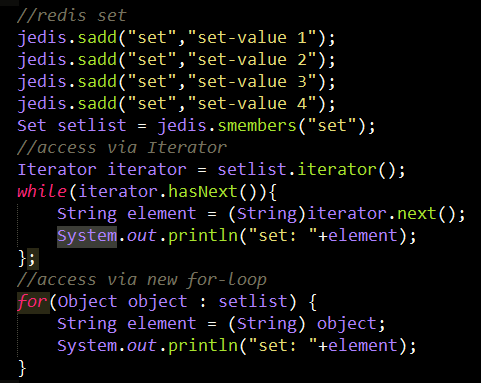
Vì là list nên các values mới trùng với value có trong list vẫn thực hiện push được.

1. **Redis HASH**



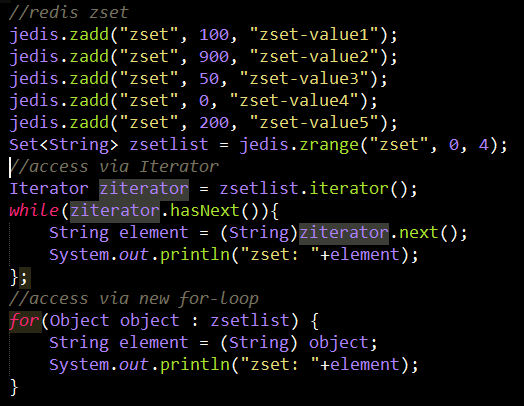


1. **Redis SET**



Nếu value mới đã có trong SET thì value mới không được thêm.

1. **Redis ZSET**



1. **Nguồn tham khảo**

[**https://www.tutorialspoint.com/redis/redis\_client\_connection.htm**](https://www.tutorialspoint.com/redis/redis_client_connection.htm)

[**https://www.tutorialspoint.com/redis/redis\_java.htm**](https://www.tutorialspoint.com/redis/redis_java.htm)

[**https://www.tutorialspoint.com/redis/redis\_partitioning.htm**](https://www.tutorialspoint.com/redis/redis_partitioning.htm)

[**https://redis.io/topics/cluster-tutorial**](https://redis.io/topics/cluster-tutorial)

[**https://redis.io/commands/client-reply**](https://redis.io/commands/client-reply)

[**https://redis.io/topics/replication**](https://redis.io/topics/replication)

[**https://www.packtpub.com/books/content/scaling-redis-cluster-and-sentinel**](https://www.packtpub.com/books/content/scaling-redis-cluster-and-sentinel)

[**http://qnimate.com/overview-of-redis-architecture/**](http://qnimate.com/overview-of-redis-architecture/)

1. **Link hướng dẫn cài cặt và sử dụng Redis với Java**

[**https://youtu.be/iC3SXRXUoPo**](https://youtu.be/iC3SXRXUoPo)