Redis

Indice

Introduzione

Redis è un *key-value* data structure store usato come database, cache e message broker. Come strutture dati supporta stringhe, liste, hash, set, set ordinati, bitmap, hyperloglog, e dalla versione 3.2, supporta anche indici geospaziali. Redis inoltre supporta script in Lua, transazioni, LRU eviction, replica master-slave, diversi strumenti per la persistenza dei dati, e partizione automatica tramite il Redis Cluster.

Redis è sviluppato e supportato per Linux, OS X e BSD. Non c'è supporto ufficiale né sviluppo di redis per i dispositivi Windows, ma Microsoft ha realizzato, e mantiene un port win-64 open source, tramite MS Open Tech.

Installazione

Linux e OS X

Per installare Redis su macchine Linux o su sistemi OS X, prima di tutto bisogna scaricare la versione desiderata di Redis, disponibile qui, oppure tramite Bash

```
$ wget http://download.redis.io/releases/redis-3.0.7.tar.gz
```

Una volta scaricato Redis, via download diretto o Bash, si procede ad estrarre e compilare il file.

```
$ tar xzf redis-3.0.7.tar.gz

$ cd redis-3.0.7

$ make
```

I file binari compilati sono presenti nella directory src. Per avviare il server digitare nella shell:

```
$ src/redis-server
```

Per interagire con il server, avviare il client:

```
$ src/redis-cli
```

Windows

Installare Redis su macchine Windows è più complesso, in quanto Redis non è ufficialmente supportato per Windows. Ci sono due modi per installarlo: compilare con Visual Studio, o via NuGet

Visual Studio

Per installare Redis tramite Visual Studio, bisogna scaricare il fork di Redis realizzato dal team di MS OpenTech, disponibile su github.

Una volta scaricato ed estratto l'archivio, aprire il file redisserver.sln, nella cartella msvs. Nelle proprietà del progetto, selezionare la configurazione della build desiderata (debug o release), ed il target (in genere x64).

Impostate le proprietà, si può procedere a compilare la soluzione: ciò va a creare dei file eseguibili, locati nella cartella msvs\$(Target)\$(Configuration), impostata nelle proprietà prima della compilazione. Avviare il server con il file redis-server.exe, ed il client per interagire con il server avviando redis-cli.exe.

NuGet

Un'altro modo di installare Redis è quello di scaricarlo tramite la package manager console.

```
PM > Install-Package Redis-64
```

Questo comando, chiamato nella package manager console, va a scaricare ed installare Redis nella directory tools di default di NuGet. Come per Visual Studio, per avviare il server ed il client, basta avviare gli eseguibili redis-server.exe, e redis-cli.exe.

Strutture dati

A differenza di molti key-value store, Redis consente di assegnare alle chiavi diverse strutture dati oltre alle stringhe.

Chiave

Le chiavi di Redis sono binary-safe, ovvero è possibile utilizzare, oltre a stringhe e valori, anche sequenze binarie come chiavi di Redis; è inoltre possibile utilizzare una stringa vuota come chiave. La dimensione massima di una chiave è 512MB.

È possibile, tramite alcuni comandi che non sono ristretti a delle tipologie di dati, effettuare alcune operazioni sulle chiavi: il comando `EXISTS` restituisce 1 o 0 a seconda che la chiave esista o meno nel database, mentre il comando `DEL` va ad eliminare la chiave ed il valore ad essa associato. Anche il comando `DEL` restituisce 1 o 0: se la chiave è stata rimossa, quindi esisteva, restituirà 1, altrimenti, se non è stata rimossa, quindi non eisteva una chiave con quel nome sul database, restituirà 0. Il comando `TYPE` invece restituisce il tipo di dato associato a quella chiave.

Gli esempi qui riportati mostrano due query di redis, ed il loro output.

```
> set mykey hello
OK
> exists mykey
(integer) 1
> del mykey
(integer) 1
> exists mykey
(integer) 0
```

```
> set mykey x
OK
> type mykey
string
> del mykey
(integer) 1
> del mykey
(integer) 0
```

Redis Expires, Persists e TTL

Un'altra importante funzionalità che non dipende dalla struttura dati assegnata alla chiave è il comando `EXPIRE`: questo comando imposta un timeout, espresso in secondi o millisecondi, alla scadenza del quale, viene eseguito un `DEL` e la chiave viene eliminata. Le informazioni riguardanti l'expire sono salvate sul disco; ciò significa che Redis salva su disco la data in cui quella chiave cesserà di esistere

```
> set mykey x
OK
> expire mykey 10
(integer) 1
> get mykey (subito)
"x"
> get mykey (dopo 10 secondi)
(nil)
```

L'expire può essere impostato all'inserimento del valore, in forma abbrieviata, aggiungendo `EX` e la vita della chiave, può anche essere utilizzato per prolungare la vita della chiave, o ridurla: ogni expire successivo al primo andrà a sostituire il valore precedente. È possibile inoltre verificare quanto tempo rimane alla rimozione della chiave tramite il comando `TTL`, ovvero Time To Live:

```
> set mykey x ex 10
OK
> ttl mykey
(integer) 9
> expire mykey 20
(integer) 1
> ttl mykey
(integer) 19
```

Per annullare un expire, e rendere la chiave ed il suo valore persistenti, si usa il comando `PERSIST`, che, come `EXPIRE` restituisce un valore di 1 se ha avuto successo. Dopo aver chiamato il comando `PERSIST`, `TTL` restituirà un valore di -1. Ciò indica che la variabile è persistente, e rimarrà nel database finché non verrà eliminata. Infine, per impostare il ttl, e verificarlo esprimendolo in millisecondi, vengono usati i comandi `PEXPIRE` e `PTTL`.

Stringhe

La stringa è il più semplice tipo di dato disponibile in Redis; le stesse chiavi vengono memorizzate come stringhe.

```
> set mykey x
OK
> get mykey
"x"
```

Il `SET` ed il `GET` sono i metodi con cui si assegna un valore ad una chiave, e con cui si ottiene il valore assegnato. Il `SET` va a sostituire qualsiasi valore fosse precedentemente assegnato alla chiave. I valori assegnabili come stringa possono essere di qualsiasi tipo: dal testo, al file binario, o ad esempio un file .jpg. L'unica limitazione che ha la stringa è che non può essere più grande di 512MB.

È possibile in un'unica chiamata, definire ed ottenere più chiavi, tramite `MSET` e `MGET`:

```
> mset a 10 b 20 c 7
OK
> mget a b c
1) "10"
2) "20"
3) "30"
```

`MGET` restituisce un array di valori.

Sulle stringhe è possibile compiere operazioni atomiche, come ad esempio incrementare un valore:

```
> set counter 10
OK
> type counter
"string"
> incr counter
(integer) 11
> incr counter
(integer) 12
```

Il valore di counter, pur essendo una stringa, viene interpretato come un intero, sul quale viene poi effettuato un incremento. `INCR` rappresenta un'operazione atomica, ovvero un'operazione che viene eseguita da un client alla volta, evitando così che più client entrino in una corsa.

Liste

Le liste in Redis sono di tipo linked list o lista concatenata, quindi l'aggiunta di nuovi elementi in cima o in fondo ad una lista avviene in tempo costante. Il contro però è che cercare elementi all'interno di una lista è proporzionale all'indice dell'elemento che si sta cercando.

I comandi `LPUSH` e `RPUSH` servono ad inserire elementi nella lista, rispettivamente a sinistra (in cima), e a destra (in fondo). Il comando `LRANGE` permette di ottenere valori all'interno della lista compresi tra due indici specificati nell'argomento del comando.

```
> lpush list 1
(integer) 1
> rpush list 2
(integer) 2
> rpush list 3
(integer) 3
> rpush list d
(integer) 4
> lpush list e
(integer) 5
> lrange list 2 4
1) "2"
2) "3"
3) "d"
> lrange list 0 -1
1) "e"
2) "1"
3) "2"
4) "3"
5) "d"
```

La lista contiene valori sottoforma di stringhe. `LRANGE` ha come argomenti gli indici della parte della lista da mostrare, ma se come secondo argomento viene passato -1, procederà a mostrare tutti gli elementi dal primo elemento all'indice inserito come primo argomento, all'ultimo elemento della

lista. Sia LPUSH che RPUSH sono funzioni variadiche, quindi è possibile inserire in una lista più elementi in un'unica chiamata.

Un'altra operazione possibile con le liste è il *pop*: LPOP e RPOP infatti, permettono di estrarre il primo o l'ultimo elemento della lista, togliendolo dalla lista stessa:

```
> rpush list a b c 1 2 3
(integer) 6
> lpop list
"a"
> rpop list
"3"
> lrange list 0 -1
1) "b"
2) "c"
3) "1"
4) "2"
```

In alcuni casi, ad esempio per tenere in memoria gli ultimi post o messaggi degli utenti, sono necessarie delle liste di una data lunghezza: ciò si ottiene con il *trim*; Il comando LTRIM, funziona come LRANGE, tranne che, invece di restituire i valori, elimina i valori esterni al range:

```
> rpush list a b c 1 2 3
(integer) 6
> ltrim list 0 2
OK
> lrange list 0 -1
1) "a"
2) "b"
3) "c"
```

Blocking Operations

Le liste in Redis consentono l'implementazione di code, ed un setup a produttore-consumatore. Ad esempio il produttore o produttori usano un LPUSH per inserire dei dati in una lista, ed i consumatori usano RPOP per estrarre i dati sulla lista, ed elaborarli. Nel caso la lista sia vuota però, RPOP restituisce NULL, questo potrebbe portare i consumatori a ripetere il comando finché non ottengono un dato. Questo però porta ad una serie di chiamate inutili. Per evitare ciò, Redis implementa i comandi BLPOP e BRPOP, che funzionano come LPOP e RPOP, ma, se la lista è vuota aspettano un periodo di tempo, espresso in secondi, e dichiarato come argomento, e se la lista è vuota, restituirà NULL. È possibile dare 0 come argomento per l'attesa, in questo modo, si fa sì che il client che ha chiamato il BLPOP o BRPOP rimanga in attesa indefinitamente finché la lista rimane vuota.

```
> brpop list 5
1) "list"
2) "a"
> brpop list 5
(nil)
(5.10s)
```

`BLPOP` e `BRPOP` possono ricevere come argomenti più liste, e elaborare un dato dalla prima lista che viene popolata nel caso sia vuota. I client che bloccano le liste vengono serviti in modo ordinato, ovvero il primo client che ha bloccato la lista ed è in attesa di un dato verrà servito per primo, e così via. I comandi inoltre restituiscono un array, e non un singolo valore: dato che `BLPOP` e `BRPOP` possono attendere dati da più liste, viene anche restituita la chiave a cui il valore ottenuto è associato.

Hash

Le hash sono coppie di campi e valori, ideali per rappresentare oggetti in Redis, e non vi sono limitazioni al numero di campi possibili in una hash. Per assegnare campi e valori alla hash viene utilizzato il comando `HMSET`;

```
> hmset user:0001 name pippo age 32 mail pippo@mail.com
OK
```

Per ottenere i dati dalla hash invece, abbiamo a disposizione diversi comandi:

- `HGET`
  - Restituisce il valore di un singolo campo specificato nell'argomento del comando
- `HMGET`
  - Restituisce il valore di più campi specificati negli argomenti del comando
- `HGETALL`
  - Restituisce il valore di tutti i campi

```
> hget user:0001 name
"pippo"
> hmget user:0001 name mail
1) "pippo"
2) "pippo@mail.com"
> hgetall user:0001
1) "name"
2) "pippo"
3) "age"
4) "32"
5) "mail"
6) "pippo@mail.com"
```

È possibile effettuare anche alcune operazioni sui campi, come ricerca o incremento dei valori numerici; in seguito è presente una lista dei comandi delle hash.

Set

I set sono insiemi non ordinati di strighe uniche: un set non può avere al suo interno valori uguali tra loro

```
> sadd users user:0000 user:0001 user:0002 user:0003
(integer) 4
> smembers users
1) "user:0000"
2) "user:0002"
3) "user:0001"
4) "user:0003"
> sadd users user:0000
(integer) 0
> sadd users user:0004
(integer) 1
> smembers users
1) "user:0002"
2) "user:0001"
3) "user:0003"
4) "user:0000"
5) "user:0004"
```

Come `RPUSH` per le liste, `SADD` va ad aggiungere un elemento alla fine del set; dato che i set non sono ordinati, la disposizione degli elementi di un set varia ad ogni nuovo inserimento o rimozione di un elemento. Il comando `SMEMBERS` restituisce un array contenente tutti gli elementi del set.

Dato che i set contengono solo valori unici, è possibile verificare se un elemento è già presente nel set o meno, tramite il comando `SISMEMBER`, che restituisce un valore di 1 o 0 a seconda della presenza o meno del valore all'interno del set. Per ottenere elementi dal set, viene utilizzato il comando `SPOP`, che prende un elemento a caso dal set, rimuovendolo dal set stesso.

```
> spop users
"user:0003"
> smembers users
1) "user:0000"
2) "user:0004"
3) "4"
4) "3"
5) "user:0002"
> sismember users user:0001
(integer) 0
```

Sui set si possono effettuare operazioni simili a quelle tra insiemi, come l'unione, l'intersezione, e la differenza, tramite i comandi `SINTER`, `SUNION` e `SDIFF`

```
> sadd users user:0000 user:0001 user:0002
(integer) 3
> sadd users2 user:0000 user:0001 user:0002 user:0003 user:0004
(integer) 5
> sinter users users2
1) "user:0000"
2) "user:0002"
3) "user:0001"
> sunion users users2
1) "user:0003"
2) "user:0000"
3) "user:0004"
4) "user:0002"
5) "user:0001"
> sdiff users users2
(empty list or set)
> sdiff users2 users
1) "user:0004"
2) "user:0003"
```

I comandi restituiscono tutti array di stringhe, ma i risultati di questi comandi non vengono memorizzati, permettendo di eseguire i comandi sugli stessi set più volte. Per tenere i risultati in memoria, sono disponibili delle varianti dei comandi appena visti: `SINTERSTORE`, `SUNIONSTORE` e `SDIFFSTORE`. La lista completa dei comandi dei set sono elencati più avanti.

Set Ordinati

I Sorted Set di Redis sono un mix tra i set e le hash. Come i set, sono un insieme di valori unici, ma come nelle hash, gli elementi del set sono associati ad un valore: questo valore è un numero decimale chiamato *score*. Lo score è un valore che viene definito assieme al valore assegnato, all'inserimento del valore nel set, tramite il comando `ZADD`, e, a differenza del valore ad esso associato, non deve necessariamente essere un valore unico: più elementi possono avere lo stesso score. Per l'ordinamento del set, vengono seguite due regole:

- A e B sono due elementi di un set con uno score S differente, ed I è il loro indice nel set.
- Se $A_S > B_S$ allora $I_A > I_B$
- Se $A_S = B_S$ allora A e B sono ordinati alfabeticamente.

```
> zadd StarWars 1999 "La Minaccia Fantasma"
(integer) 1
> zadd StarWars 2002 "L'Attacco Dei Cloni"
(integer) 1
> zadd StarWars 2005 "La Vendetta Dei Sith"
(integer) 1
> zadd StarWars 1977 "Guerre Stellari"
(integer) 1
> zadd StarWars 1980 "L'Impero Colpisce Ancora"
(integer) 1
> zadd StarWars 1983 "Il Ritorno Dello Jedi"
(integer) 1
> zadd StarWars 2015 "Il Risveglio Della Forza"
(integer) 1
```

`ZADD` funziona esattamente come `SADD`, ma richiede due argomenti, lo score ed il valore. Come `SADD`, è un comando variadico, quindi l'inserimento di più elementi con un unico comando è possibile. Una volta inseriti gli elementi, se viene chiamato un comando come `ZRANGE` per ottenere gli elementi del set, l'array ottenuto è già ordinato; Redis infatti ordina gli elementi all'inserimento, confrontando gli score.

```
> zrange StarWars 0 -1
1) "Guerre Stellari"
2) "L'Impero Colpisce Ancora"
3) "Il Ritorno Dello Jedi"
4) "La Minaccia Fantasma"
5) "L'Attacco Dei Cloni"
6) "La Vendetta Dei Sith"
7) "Il Risveglio Della Forza"
```

Oltre ad operazioni del genere, è anche possibile estrarre degli elementi in base al loro score. Ad esempio con il set che è appena stato creato, possiamo estrarre tutti i film usciti dopo il 1990. Per farlo viene utilizzato il comando `ZRANGEBYSCORE`:

```
> zrangebyscore StarWars 1990 +inf
1) "La Minaccia Fantasma"
2) "L'Attacco Dei Cloni"
3) "La Vendetta Dei Sith"
4) "Il Risveglio Della Forza"
```

È anche possibile eliminare un range di elementi da un set ordinato in base allo score. Possiamo ad esempio eliminare i film in cui compare Jar Jar Binks, tramite il comando `ZREMRANGEBYSCORE`:

```
> zremrangebyscore StarWars 1999 2002
2
> zrange StarWars 0 -1
1) "Guerre Stellari"
2) "L'Impero Colpisce Ancora"
3) "Il Ritorno Dello Jedi"
4) "La Vendetta Dei Sith"
5) "Il Risveglio Della Forza"
```

Score Lessicografici

Dalla versione 2.8 di Redis, è possibile ordinare elementi in ordine alfabetico. Prendiamo il set di prima, ma invece di ordinare i film per data di uscita, assegnamo loro lo stesso score:

```
> zadd StarWars 0 "La Minaccia Fantasma" 0 "L'Attacco Dei Cloni" 0 "La Vendetta Dei
Sith" 0 "Guerre Stellari" 0 "L'Impero Colpisce Ancora" 0 "Il Ritorno Dello Jedi" 0 "Il
Risveglio Della Forza"
(integer) 7
> zrange StarWars 0 -1
1) "Guerre Stellari"
2) "Il Risveglio Della Forza"
3) "Il Ritorno Dello Jedi"
4) "L'Attacco Dei Cloni"
5) "L'Impero Colpisce Ancora"
6) "La Minaccia Fantasma"
7) "La Vendetta Dei Sith"
```

Possiamo anche, sempre nel caso ci siano più elementi con lo stesso score, selezionare alcuni elementi in base alla loro iniziale, con il comando `ZRANGEBYLEX`

```
> zrangebylex StarWars [I [L
1) "Il Risveglio Della Forza"
2) "Il Ritorno Dello Jedi"
```

I comandi di ordinamento e le operazioni vengono analizzati in seugito nella sezione dedicata ai comandi

Aggiungere un elemento già presente nel set, con uno score maggiore rispetto a quello nel set, va ad aggiornare lo score dell'elemento. Questo è particolarmente utile nello use case delle leaderboards

Bitmap

Le bitmap non sono vere e proprie strutture dati, ma una serie di operazioni sui bit. Dato che le stringhe possono contenere valori binari, e la loro dimensione massima è 512MB, possono

contenere fino a 2^{32} bit. Le operazioni sui bit sono divise in due gruppi: operazioni a tempo costante, su bit singoli, come impostare un bit su 1 o 0 o ottenerne il valore, e operazioni su gruppi di bit, come contare il numero di bito con un dato valore in un dato range.

I bit sono impostati e ottenuti utilizzando i comandi `SETBIT` e `GETBIT`

```
> setbit key 10 1
(integer) 1
> getbit key 10
(integer) 1
> getbit key 11
(integer) 0
```

Il comando `SETBIT` prende come primo argomento la posizione del bit da impostare, e come secondo argomento il valore da assegnare al bit selezionato. Se il bit selezionato è "fuori" dalla stringa che contiene i bit, Redis aumenta la dimensione della stringa per ospetare il bit selezionato.

Il `GETBIT` richiede come argomento l'indice del bit che si vuole ottenere. Se il bit desiderato è fuori dalla stringa selezionata, il comando restituisce 0.

Alcuni comandi che lavorano sui gruppi di bit sono `BITCOUNT` e `BITPOS`. `BITCOUNT` conta i bit in una bitmap, e restituisce il numero di bit con valore 1, mentre `BITPOS` esamina una bitmap e restituisce del primo indice con valore 1. Entrambi possono operare in un range di byte di una stringa, piuttosto che esaminarla tutta.

```
> setbit key 0 1
0
> setbit key 100 1
0
> bitcount key
2
> bitpos key 1
(integer) 0
```

Una lista completa dei comandi relativi alle bitmap è disponibile in seguito.

Hyperloglog

Un hyperloglog è una struttura dati probabilistica utilizzata per contare elementi unici. Contare elementi unici è un compito che in genere occupa memoria proporzionalmente al numero di elementi da contare, in quanto il metodo deve tenere in memoria gli elementi singoli che ha già trovato; Redis implementa una serie di algoritmi che riducono l'uso di memoria ad una quantità costante, al costo di una precisione ridotta: nel caso di redis c'è un margine di errore di questi algoritmi dell'1%, con un uso di memoria fisso che si aggira intorno ai 12kB.

Gli hyperloglog in redis sono interpretati come stringhe, quindi si possono utilizzare i comandi `SET` e `GET`

Concettualmente gli hyperloglog sono simili al set. Con un set si può usare il comando `SADD` per popolare un ser, e poi chiamare `SCARD` per ottenerene la cardinalità, dato che un set è un insieme di elementi unici. A differenza di un set però, l'hyperloglog non contiene elementi, ma stati; ogni volta che si incontra un nuovo elemento, si chiama il comando `PFADD` per aggiungere uno stato all'hyperloglog; per ottenere il numero di elementi unici invece, si usa il comando `PFCOUNT`

```
> pfadd hll a b c d
(integer) 1
> pfcount hll
(integer) 4
```

È anche possibile unire più hyperloglog con il comando `PFMERGE`, che prende come argomenti le chiavi dei vari hyperloglog che si vuole unire. Per maggiori dettagli, riferirsi alla lista dei comandi.

Persistenza sul Disco

Redis offre due strumenti per la persistenza dei dati ed il data recovery: RDB e AOF:

- RDB
  - RDB permette di salvare degli snapshot dei dati in intervalli di tempo definiti dall'utente. Ogni snapshot viene creato da un processo diverso, o da un thread diverso all'interno dello stesso processo; viene creato ad ogni snapshot un dump.rdb, e per eseguire altri snapshot, Redis prima effettua un `fork()` del processo, crea un dump.rdb temporaneo, e quando ne ha completato la scrittura va a rinominarlo e spostarlo nella cartella di destinazione.
- AOF
  - L'Append Only File, o AOF, è un log di tutti i comandi di scrittura chiamati, che viene utilizzato per ricostruire il dataset al riavvio del server. L'AOF, può essere riscritto automaticamente da Redis o tramite una chiamata al server: questo fa si che Redis vada a creare un nuovo AOF, che contiene tutti i comandi necessari a creare il dataset presente alla chiamata, andando ad eliminare comandi superflui come eliminazione e reinserimento di un elemento.

Utilizzare solo RDB è consigliato solo nel caso in cui, se dovesse esserci un problema, perdere i dati inseriti dall'ultimo snapshot è accettabile. Ad esempio, un server può essere impostato per fare degli snapshot ogni 15 minuti. In caso di crash del server, si andranno a perdere i dati inseriti negli ultimi 15 minuti, ma non quelli inseriti in precedenza, ed al riavvio del server, Redis leggerà lo snapshot per reimpostare il dataset precedente. Uno svantaggio di RDB è che, nel caso di grandi database, il processo creato dal fork() per creare lo snapshot può impiegare del tempo, e causare un ritardo nella risposta del server ai client. La disaster recovery però è molto più rapida, soprattutto per database grandi, rispetto a quella offerta da AOF.

Molti utenti di Redis utilizzano solo AOF, anche se è un po meno efficente in caso di recovery, soprattutto a seconda di cosa stava facendo il server prima di andare offline. AOF rimane comunque un tool molto potente, potenzialmente meno efficente di RDB, ma che assicura una persistenza dei dati molto precisa, può variare a seconda della fsync policy addottata dall'utente; vi

sono 3 opzioni per quanto riguarda la fsync: disabilitata, ad ogni query e ogni secondo.

- Disabilitata:
  - Nessun tipo di sincronizzazione automatica, è l'amministratore che chiama il comando.
- Per query:
  - Ogni comando chiamato fa sì che il server lo registri nel file di log.
- Ogni secondo:
  - L'impostazione di default, il servere registra ogni secondo nel file di log tutti i comandi chiamati dall'ultima registrazione.

È consigliato utilizzare una combinazione dei due metodi di persistenza. In caso di un riavvio del server, Redis utilizzerà l'AOF file per ripristinare il server, dato che contiene i dati più completi.

Cluster

Redis consente di avere un setup master-slave, e di dividere le chiavi di un database in hash slot, tra più nodi. Il cluster rende più stabile la struttura, in quanto, ogni istanza master replica via replica asincrona, i comandi di scrittura agli slave ad esso assegnati; ciò fa in modo che se un master dovesse avere qualche problema e non essere raggiungibile dai client e dagli slave, uno dei suoi slave viene "promosso" a master, mentre la vecchia istanza master diventa uno slave della nuova istanza master. Ogni nodo del cluster ha bisogno di due porte tcp: quella che usa per comunicare con i client, ad esempio la 6379, ed una porta ottenuta aggiungendo 10000 alla porta che sta utilizzando, in questo caso 16379. Questa seconda porta è chiamata Cluster Bus ed è usata come canale di comunicazione interna al cluster: i vari nodi utilizzano questa porta per comunicare tra loro cambi di configurazione, individuazione dei problemi, autorizzazione alle promozioni e così via. Il cluster bus è un canale solo per i nodi del server, i vari client devono utilizzare la porta "bassa".

API

Le API che consentono di utilizzare Redis con altri linguaggi di programmazione sono diverse, ed un elenco è disponibile qui. In questo documento verrà trattata l'API per C# di StackExchange, sviluppata da Marc Gravell, principalemte per impiegare Redis nei siti di StackExchange, open source e disponibile su github.

Installazione

StackExchange.Redis viene installata tramite la package manager console, semplicemente digitando

```
PM> Install-Package StackExchange.Redis
```

È disponibile anche una versione strongnamed, nel caso il progetto su cui si sta lavorando sia string named

```
PM> Install-Package StackExchange.Redis.StrongName
```

Visual Studio andrà a creare una cartella chiamata StackExchange.Redis.(versione) nella cartella contenente i file del progetto che si sta utilizzando.

Utilizzo

Per conentire al programma di connettersi ed operare sul server Redis, è necessario aggiungere stackexchange.redis alle referenze, ed impostare una connessione al server stesso:

```
using StackExchange.Redis;

....

var conn = ConnectionMultiplexer.Connect("ip:port, password = password");
```

Il ConnectionMultiplexer è un oggetto particolare che viene utilizzato per gestire le connessioni al server in modo da ottimizzare i tempi, e ridurre al minimo la latenza dei singoli client connessi al server.

Una volta connessi al server, si può accedere al database utilizzando la seguente riga di codice:

```
IDatabase db = conn.GetDatabase()
```

Ora abbiamo un database `db` su cui poter effettuare operazioni di vario genere. Ad ogni operazione viene richiamato uun metodo dell'oggetto database che abbiamo creato, in questo caso `db`

Lista dei comandi

Comandi relativi alla gestione del server

BGREWRITEAOF

```
BGREWRITEAOF
```

Comiunica a Redis di iniziare a riscirvere l'Append Only File. Se è già in atto un salvataggio tramite snapshot, Redis restituisce comunque `OK`, ma notifica che l'operazione di riscrittura è messa in coda. Se è in corso un'altra operazione di riscrittura dell'AOF, Redis restituirà un errore, e non verrà messa in coda una nuova operazione. Se l'operazione dovesse fallire, l'AOF precedente non viene modificato.

BGSAVE

```
BGSAVE
```

Salva il database. Il processo si divide, il parent continua a servire i client, mentre il child esegue il salvataggio. I client possono verificare se il salvataggio è stato completato con il comando LASTSAVE.

CLIENT GETNAME

```
CLIENT GETNAME
```

Restituisce il nome della connessione, impostata con CLIENT SETNAME. Se non viene impostato un nome, il comando restituisce null.

CLIENT KILL

```
CLIENT KILL [ip:porta] [ID client-id] [TYPE normal|master|slave|pubsub] [ADDR ip:port]
[SKIPME yes/no]
```

Chiude la connessione con il client. Le varie opzioni permettono l'utilizzo di filtri per chiudere connessioni con determinati client:

`CLIENT KILL ADDR ip:porta`
: Chiude la connessione con il client all'indirizzo specificato, ottenuto dal campo `addr` di `CLIENT LIST`.

`CLIENT KILL ID id` Chiude la connessione con il client con l'id specificato, ottenuto dal campo `id` di `CLIENT LIST`.

`CLIENT KILL TYPE tipo` Chiude la connessione con tutti i client del tipo specificato. I client bloccati dal comando `MONITOR` sono considerati `normal`.

`CLIENT KILL SKIPME yes/no` Imposta se chiudere o meno la connessione con il client che ha chiamato il comando. Di default è impostato su yes, quindi non chiude la connessione.

CLIENT LIST

```
CLIENT LIST
```

Restituisce una serie di informazioni sui client connessi. I campi restituiti sono:

id	l'id del client
addr	l'indirizzo del client, indicato con ip:porta
fd	descrittore del file corrispondente al socket
age	durata totale della connessione in secondi
idle	tempo in cui il client è rimasto inattivo espresso in secondi
flags	le flag del client
db	l'id del database a cui il client è connesso
sub	numero di iscrizioni ai canali
psub	numero di iscrizioni che corrispondono ad un pattern
multi	numero di comandi in un contesto `MULTI/EXEC`
qbuf	lunghezza del buffer delle query. 0 significa che non ci sono delle query in attesa
qbuf-free	spazio libero del buffer delle query.
obl	lunghezza del buffer di output
oll	lunghezza della lista di output. Le risposte sono messe in coda in questa lista se il buffer è pieno
omem	utilizzo della memoria del buffer di output
events	eventi del descrittore di file
cmd	ultimo comando chiamato dal client
Flag	
O	il client è uno slave in modalità MONITOR
S	il client è un server slave normale
M	il client è un master

x	il client è in un contesto MULTI/EXEC
b	il client è in attesa in una blocking operation
i	il client è in attesa di un VM I/O (deprecated)
d	una o più chiavi osservate sono state modificate. EXEC fallirà
c	la connessione sarà chiusa una volta scritta tutta la risposta
u	il client è sbloccato
U	il client è connesso tramite un Unix domain socket
r	il client è in modalità readonly su un nodo del cluster
A	la connessione verrà chiusa appena possibile
N	nessun set specifico di flag
Eventi	
r	il socket del client è leggibile (event loop)
w	il socket del client è scrivibile (event loop)

CLIENT PAUSE

```
CLIENT PAUSE timeout
```

Sospende la connessione a tutti i client per un periodo specificato in millisecondi. Il comando restituisce immediatamente `OK` al caller, e non elabora più i comandi inviati dai client, ma le interazioni con gli slave proseguono normalmente. Esaurito il timeout, tutti i client sono sbloccati, e vengono eseguiti tutti i comandi messi in coda nei buffer delle query dei client.

Il `CLIENT PAUSE` è utile per modificare la configurazione master-slave, ma può anche essere inserito nelle transazioni.

CLIENT REPLY

```
CLIENT REPLY on|off|skip
```

Permette di disattivare le risposte dal server; utile nei casi in cui le risposte dal server vengono ignorate, consente di risparmiare tempo e banda. L'opzione di default è `on`, ovvero il client riceve tutte le risposte dal server e restituisce `OK`; `off` disabilita tutte le risposte da quando viene chiamato, quindi anche il comando stesso non avrà risposta. `Skip` invece permette di disattivare la risposta dal server per il comando successivo, ma come `off`, non restituisce niente.

CLIENT SETNAME

```
CLIENT SETNAME nome
```

Imposta il nome della connessione. Il nome in se ha solo due restrizioni: non deve contenere spazi, in quanto causerebbe errori nell'output di `CLIENT LIST`, ed è limitato ad una dimensione massima di 512MB, come una normale stringa di Redis. È possibile eliminare il nome di una connessione impostandolo come stringa vuota. Tutte le connessioni vengono avviate senza nome

COMMAND

```
COMMAND
```

Restituisce un array contenente tutti i comandi di Redis e relativi dettagli. Viene utilizzato per mappare le chiavi e la loro posizione . Il formato con cui vengono mostrati gli elementi è il seguente:

nome del comando	una semplice stringa in lowecase
arietà (numero di argomenti richiesti)	l'arietà comprende anche il comando stesso. Quindise il comando non richiede argomenti, sarà 1.
	positiva se il comando richiede un numero specifico di argomenti
	negativa se il comando richiede un numero minimo di argomenti
un array contenente le flag del comando	le flag sono elencate di seguito, ed indicano uno o più stati del comando
posizione della prima chiave nella lista degli argomenti	per molti comandi la prima chiave è in posizione 1. In posizione 0 c'è sempre il nome del comando.
posizione dell'ultima chiave nella lista degli argomenti	se il comando accetta solo una chiave, la posizione della prima, e dell'ultima chiave coincidono. Se il comando accetta infinite chiavi, l'ultima chiave è in posizione -1.
step count per trovare la posizione delle chiavi	permette di individuare ogni quanti elementi nella lista degli argomenti si trova una chiave. Ad esempio `MGET` avrà uno step count pari a 1, in quanto ogni argomento è una chiave, mentre `MSET` avrà uno step count pari a 2, dato che le chiavi si alternano ai valori nell'argomento.
Flag del comando	
write	il comando può causare delle modifiche

readonly	il comando non modifica le chiavi
denyoom	rifiuta il comando se OOM (out of memory)
admin	comando chiamato dall'amministratore del server
pubsub	comando chiamato da un client di tipo pub-sub
noscript	rifiuta questo comando se chiamato da uno script
random	il comando ha risultati casuali, pericoloso per gli script
sort_for_script	se chiamato da uno script, ordina l'output
loading	permette l'uso del comando anche se il database è in caricamento
stale	permette l'uso del comando mentre la replica ha stale data
skip_monitor	non mostra questo comando in modalità MONITOR
asking	comando dei cluster: - accetta anche se sta importando
fast	il comando opera in un tempo costante o pari a log(N). Utilizzato per il monitoraggio della latenza.
movablekeys	le chiavi non hanno una posizione predeterminata.

Movablekeys

A volte le chiavi non sono in posizioni prestabilite all'interno dell'argomento del comando. In questo caso il client deve esaminare tutti i comandi marcati con questa flag ed individuare le chiavi. I comandi che vengono marcati con la flag `movablekeys` sono: `SORT`, `ZUNIONSTORE`, `ZINTERSTORE`, `EVAL`, `EVALSHA`

COMMAND COUNT

```
COMMAND COUNT
```

Restituisce un intero che indica il numero totale di comandi nel server Redis.

COMMAND GETKEYS

```
COMMAND GETKEYS comando
```

Restituisce un array di chiavi utlizzate in un comando completo, ovvero un comando seguito dall'argomento

```
COMMAND GETKEYS MSET a 1 b 2 c 3 d 4
1) "a"
2) "b"
3) "c"
4) "d"
```

COMMAND INFO

```
COMMAND INFO comando1 comando2 comando3...
```

Funziona come `COMMAND`, ma invece di restituire un array contenente tutti i comandi , mostra solo i dettagli dei comandi specificati nell'argomento. Il formato di output rimane invariato.

CONFIG GET

```
CONFIG GET parametro
```

Restituisce un parametro dal file di configurazione di Redis. L'argomento accetta un parametro preciso, oppure una parte del parametro stesso, come riportato nell'esempio di seguito. In questo caso il comando restituisce tutti i parametri che corrispondono alla ricerca. Se viene passato un asterisco, il comando restituisce tutti i parametri.

```
config get *max-*-entries*
1) "hash-max-zipmap-entries"
2) "512"
3) "list-max-ziplist-entries"
4) "512"
5) "set-max-intset-entries"
6) "512"
```

CONFIG RESETSTAT

```
CONFIG RESETSTAT
```

Restituisce: `OK`

Azzera i contatori del server ottenuti tramite il comando `INFO`.

I contatori che vengono azzerati sono:

- Keyspace hits

- Keyspace misses
- Number of commands processed
- Number of connections received
- Number of expired keys
- Number of rejected connections
- Latest fork(2) time
- The aof_delayed_fsync counter

CONFIG REWRITE

```
CONFIG REWRITE
```

Riscrive il file di configurazione redis.conf per far sì che il file rispecchi la configurazione attuale del server, impostata con il comando `CONFIG SET`. Il comando è in grado di creare da zero il file, nel caso in cui il file di configurazione originale non sia più presente o sia danneggiato. Il rewrite avviene in modo da conservare il più possibile il contenuto del vecchio file:

- I commenti e la struttura generale del vecchio file sono conservati se possibile.
- Se un'opzione è già presente nel vecchio file, ma con un nuovo parametro, viene scritta sulla stessa riga del file vecchio.
- Se un'opzione non è presente, ma è impostata su un valore di default, non viene inserita nel file.
- Se un'opzione non è presente e non è un valore di default, viene inserita alla fine del file
- Le righe del vecchio file non più utilizzate vengono eliminate.

CONFIG SET

```
CONFIG SET parametro valore
```

Viene utilizzato per cambiare le impostazioni del server senza doverlo riavviare. La lista dei parametri supportati da `CONFIG SET` può essere ottenuta con il comando `CONFIG GET *`.

Tutti i parametri alterati vengono immediatamente caricati su Redis, e i cambiamenti avranno effetto dal comando successivo al cambiamento. In genere i parametri ed i valori corrispondono come struttura a quelli presenti nel redis.conf file, con due eccezzioni:

- nelle opzioni che richiedono l'inserimento di byte o altre quantità, va utilizzata la versione estesa, mentre nel file redis.conf, possono essere specificati in modo abbreviato (10k, 2gb, 20mb e così via). Dalla versione 3.0 di Redis però, è possibile usare unità di memoria con i parametri `maxmemory`, i buffer di output dei client, e la dimensione del replication backlog.
- Il parametro `save` è una singola stringa di interi separati da spazi.

DBSIZE

```
DBSIZE
```

Restituisce un intero che rappresenta il numero di chiavi contenute nel database selezionato.

DEBUG OBJECT

```
DEBUG OBJECT chiave
```

Un comando di debug che non dovrebbe essere usato dai client, che dovrebbero invece utilizzare `OBJECT`.

DEBUG SEGFAULT

```
DEBUG SEGFAULT
```

Effettua un invalid memori access che manda il server in crash. Viene utilizzato per simulare bug in Redis.

FLUSHALL

```
FLUSHALL
```

Elimina tutte le chiavi di tutti i database presenti sul server.

FLUSHDB

```
FLUSHDB
```

Elimina tutte le chiavi del database selezionato.

INFO

```
INFO [sezione]
```

Restituisce informazioni e statistiche sul server. L'argomento opzionale `sezione`, permette di filtrare le informazioni ottenute. Le varie sezioni disponibili sono:

- server:: Informazioni generali sul server
- clients:: Sezione sulle connessioni dei client

- memory:: Informazioni relative al consumo di memoria
- persistence: Informazioni relative ai metodi di persistenza RDB e AOF
- stats: Statistiche generali
- replication: Informazioni sulla replica master-slave
- cpu: Statistiche sull'utilizzo della CPU
- commandstats: Statistiche sui comandi di Redis
- cluster: Sezione sul cluster
- keyspace: Statistiche riguardanti il database
- all: Restituisce informazioni su tutte le sezioni
- default: Restituisce solo il set di sezioni di default

Di seguito sono elencati tutti i parametri restituiti con il comando `INFO all`, divisi per sezione:

Server

- redis_version: Versione del server.
- redis_git_sha1: Git SHA1.
- redis_git_dirty: Git dirty flag.
- os: Il sistema operativo che ospita il server.
- arch_bits: L'architettura del sistema operativo (32 or 64 bit).
- multiplexing_api: Meccanismo di loop di eventi utilizzato da Redis.
- gcc_version: Versione del compilatore GCC utilizzato per compilare Redis.
- process_id: PID del processo del server.
- run_id: Valore random che identifica il server (viene usato da Sentinel e Cluster).
- tcp_port: Porta TCP/IP in ascolto.
- uptime_in_seconds: Numero di secondi dall'avvio del server.
- uptime_in_days: Numero di giorni dall'avvio del server.
- lru_clock: Orologio con precisione al minuto, utilizzato per la gestione degli elementi LRU.

Client

- connected_clients: Numero di connessioni dai client (escluse le connessioni dagli slave.)
- client_longest_output_list: lista di output più lunga tra i client connessi al momento.
- client_biggest_input_buf: buffer di input più grande tra i client connessi al momento.
- blocked_clients: Numero di client in attesa in una blocking operation (`BLPOP`, `BRPOP`, `BRPOPLPUSH`).

Memory

- used_memory: Numero totale di byte allocati da Redis utilizzando il suo allocatore (standard libc, jemalloc, o un allocatore alternativo come tcmalloc).

- used_memory_human: Rappresentazione leggibile del valore precedente.
- used_memory_rss: Numero di byte allocati da Redis second il sistema operativo (resident set size).
- used_memory_peak: Picco di memoria utilizzata da Redis (in byte).
- used_memory_peak_human: Rappresentazione leggibile del valore precedente.
- used_memory_lua: Numero di byte utilizzati dal motore Lua.
- mem_fragmentation_ratio: Rapporto tra used_memory_rss e used_memory.
- mem_allocator: Allocatore di memoria, scelto alla compilazione

Idealmente, il valore di used_memory_rss dovrebbe essere di poco più grande di used_memory. Una grande differenza tra rss e used indica frammentazione della memoria, che può essere interna o esterna, la quale può essere verificata controllando mem_fragmentation_ratio. Quando la used è maggiore della rss, significa che parte della memoria di Redis è stata spostata, e ci saranno latenze.

Persistence

- loading: Flag che indica se il caricamento di un dump file è in corso.
- rdb_changes_since_last_save: Numero di cambiamenti dall'ultimo dump.
- rdb_bgsave_in_progress: Flag che indica se un RDB-save è in corso.
- rdb_last_save_time: Timestamp in tempo Unix dell'ultimo RDB save eseguito con successo.
- rdb_last_bgsave_status: Stato dell'ultima operazione di salvataggio RDB.
- rdb_last_bgsave_time_sec: Durata dell'ultima operazione di salvataggio RDB in secondi.
- rdb_current_bgsave_time_sec: Durata dell'operazione di salvataggio RDB corrente.
- aof_enabled: Flag che indica se il logging dell'AOF è attivato
- aof_rewrite_in_progress: Flag che indica se un'operazione di riscrittura dell'AOF è in corso
- aof_rewrite_scheduled: Flag che indica che un'operazione di riscrittura dell'AOF verrà eseguita una volta che l'operazione RDB save è conclusa.
- aof_last_rewrite_time_sec: Durata dell'ultima operazione di riscrittura dell'AOF in secondi.
- aof_current_rewrite_time_sec: Durata dell'operazione di riscrittura dell'AOF corrente.
- aof_last_bgrewrite_status: Stato dell'ultima operazione di riscrittura dell'AOF.

changes_since_last_save fa riferimento al numero di operazioni che hanno modificato il dataset dall'ultima chiamata di `SAVE` o `BGSAVE`.

Se l'AOF è attivo, saranno aggiunti questi campi:

- aof_current_size: La dimensione dell'AOF corrente.
- aof_base_size: La dimensione dell'AOF dall'ultimo avvio del server o dall'ultimo rewrite.
- aof_pending_rewrite: Flag che indica che un'operazione di riscrittura dell'AOF verrà eseguita una volta che l'operazione RDB save è conclusa.
- aof_buffer_length: Dimensione del buffer dell'AOF.

- aof_rewrite_buffer_length: Dimensione del buffer del rewrite dell'AOF.
- aof_pending_bio_fsync: Numero di fsync in attesa nella coda di I/O in background.
- aof_delayed_fsync: Contatore di fsync ritardati.

Se è in atto un'operazione di caricamento, saranno aggiunti questi campi:

- loading_start_time: Timestamp in tempo Unix dell'inizio dell'operazione di caricamento.
- loading_total_bytes: Dimensione totale del file.
- loading_loaded_bytes: Numero di byte già caricati.
- loading_loaded_perc: Stesso valore espresso in percentuale
- loading_eta_seconds: Tempo rimanente al completamento del caricamento espresso in secondi.

Stats

- total_connections_received: Numero di connessioni accettate dal server.
- total_commands_processed: Numero complessivo di comandi elaborati dal server.
- instantaneous_ops_per_sec: Numero di comandi elaborati ogni secondo.
- rejected_connections: Numero di connessioni rifiutate a causa del limite maxclients.
- expired_keys: Numero di eventi di scadenza delle chiavi.
- evicted_keys: Numero di chiavi rimosse a causa del limite massimo della memoria.
- keyspace_hits: Numero di ricerche di chiavi effettuate con successo nel dizionario.
- keyspace_misses: Numero di ricerche di chiavi fallite nel dizionario.
- pubsub_channels: Numero totale di canali pub/sub con iscrizioni dei client.
- pubsub_patterns: Numero totale di pattern pub/sub con iscrizioni dei client.
- latest_fork_usec: Durata dell'ultimo fork in microsecondi.

Replication

- role: Indica il ruolo dell'istanza, ovvero se è master o slave. Uno slave può essere master di altri slave.

Se l'istanza è uno slave, vengono forniti questi campi:

- master_host: Host o indirizzo IP del master.
- master_port: Porta TCP in ascolto del master.
- master_link_status: Stato della connesione con il master (up/down).
- master_last_io_seconds_ago: Numero di secondi dall'ultima interazione con il master.
- master_sync_in_progress: Indica che il master si sta sincronizzando con lo slave.

Se un'operazione di SYNC è in corso, vengono aggiunti questi campi:

- master_sync_left_bytes: Numero di byte rimanenti per il completamento dell'operazione.

- master_sync_last_io_seconds_ago: Numero di secondi dall'ultimo trasferimento I/O durante un'operazione SYNC.

Se la connessione tra il master e lo slave è down, viene aggiunto il seguente parametro:

- master_link_down_since_seconds: Downtime della connessione espresso in secondi.

Il campo seguente viene fornito sempre:

- connected_slaves: Numero di slave connessi.

Per ogni slave, viene aggiunta questa riga:

- slaveXXX: id, IP address, port, state

CPU

- used_cpu_sys: System CPU utilizzata dal server.
- used_cpu_user:User CPU utilizzata dal server.
- used_cpu_sys_children: System CPU utilizzata dal processo in background.
- used_cpu_user_children: User CPU consumed by the processo in background.

Commandstats

Fornisce statistiche in base al tipo di comando, incluso il numero di chiamate, il tempo CPU complessivo impiegato da questi comandi, e una media della CPU impiegata per esecuzione.

Per ogni tipo di comando viene aggiunta questa riga:

- cmdstat_XXX: calls=XXX,usec=XXX,usec_per_call=XXX

Cluster

- cluster_enabled: Indica se Redis è in modalità cluster

Keyspace

Fornisce statistiche sui dizionari dei database: il numero di chiavi, ed il numero di chiavi con scadenza.

Per ogni database viene aggiunta questa riga:

- dbXXX: keys=XXX,expires=XXX

LASTSAVE

```
LASTSAVE
```

Restituisce il tempo trascorso dall'ultimo salvataggio in formato Unix.

MONITOR

```
MONITOR
```

`MONITOR` è un comando di debug che restituisce ogni comando elaborato dal server. Viene utilizzato per individuare bug, o in genere vedere cosa sta facendo il server. Per interrompere il monitoraggio, viene utilizzato il comando `SIGINT`, se il monitoraggio è effettuato tramite redis-cli; se invece viene fatto tramite telnet, va utilizzato `QUIT`

ROLE

```
ROLE
```

Resstituisce un array contenente informazioni riguardo al ruolo dell'istanza:

Il primo elemento indica se l'istanza è master, slave o sentinel;

A seconda del ruolo, vengono poi mostrati campi aggiuntivi:

Master

Secondo elemento dell'array è l'offset di replica master-slave, utilizzato per la risincronizzazione.

Dopo l'offset, per ogni slave viene inserito un array di 3 elementi che contiene l'IP, la porta, e l'ultimo offset di cui lo slave è a conoscenza.

Slave

Dopo la stringa che indica che l'istanza è uno slave, viene inserito un elemento che contiene l'indirizzo IP del master, un elemento successivo che mostra la porta del master, lo stato della replica dal punto di vista del master, che può essere: connect (l'istanza si deve connettere al master) connecting (l'istanza si sta connettendo al master) sync (il master e lo slave stanno effettuando la sincronizzazione) connected (lo slave è online). Infine viene aggiunto un lemento che indica la quantità di dati ricevuti dallo slave finora in termini di offset di replica.

Sentinel

Dopo la stringa che indica che l'istanza è una sentinel, viene inserito un array che contiene i nomi di tutti i master che quella sentinel sta monitorando.

SAVE

```
SAVE
```

Il comando `SAVE` effettua un salvataggio sincrono del dataset, creando un file di tipo RDB

SHUTDOWN

```
SHUTDOWN [NOSAVE|SAVE]
```

Chiude il server, assicurandosi di uscire senza la perdita di dati: ciò è ottenuto bloccando tutti i client, e, a seconda della persistenza abilitata, va a creare un file RDB, o ad aggiornare l'AOF, o entrambi. Una volta eseguito il salvataggio, chiude il server.

Le opzioni `NOSAVE e SAVE` permettono di effettuare una chiusura del server salvando i dati anche se non sono configurati savepoint, o di uscire non salvando i dati nonostante siano stati configurati 1 o più savepoint.

SLAVEOF

```
SLAVEOF host porta
```

Il comando `SLAVEOF` imposta un'istanza come slave di un'altra istanza all'indirizzo specificato. Il comando viene utilizzato per cambiare al volo le impostazioni di replica; inoltre è possibile rendere uno slave master, con il comando `SLAVEOF NO ONE`.

SLOWLOG

```
SLOWLOG [GET|LEN|RESET] [argomento]
```

Viene utilizzato per ottenere e resettare un log delle slow query, ovvero delle query che superano un tempo di esecuzione specifico, impostato alla configurazione del server. Il comando `SLOWLOG GET [elementi]` permette di visualizzare lo slowlog intero, o un numero di elementi passati nell'argomento. `SLOWLOG LEN` restituisce la lunghezza del log, mentre `SLOWLOG RESET` elimina i dati contenuti nel log.

TIME

```
TIME
```

Restituisce un array contenente due elementi che indicano l'orario del server: un timestamp in formato Unix ed il numero di microsecondi passati nel secondo attuale.

Comandi relativi alla connessione

AUTH

```
AUTH password
```

Restituisce: `OK`

Se non è stata inserita la password nel momento della connessione, consente l'autenticazione al server.

ECHO

```
ECHO messaggio
```

Restituisce il messaggio.

PING

```
PING [messaggio]
```

Invia un ping al server. Restituisce `PONG` se non è stato passato niente nell'argomento, altrimenti restituisce il messaggio.

QUIT

```
QUIT
```

Chiude la connessione al server, appena tutte le risposte dal server sono state mandate al client. Restituisce sempre `OK`.

SELECT

```
SELECT indice
```

Permette di scegliere su quale database presente sul server lavorare. I vari database presenti sul server sono memorizzati con un indice che parte da 0. Restituisce `OK`.

Comandi generici delle chiavi

DEL

```
DEL chiave1 chiave2 chiave3...
```

Restituisce un intero delle chiavi eliminate

Elimina una chiave ed il valore ad essa associato.

DUMP

```
DUMP chiave
```

Serializza la chiave in un formato specifico a Redis e la restituisce all'utente. Se la chiave non esiste, restituisce `nil`.

EXISTS

```
EXISTS chiave1 chiave2 chiave3
```

Restituisce 1 se la chiave esiste, altrimenti 0. Se sono specificate più chiavi, restituisce il numero di chiavi che esistono tra quelle passate nell'argomento.

EXPIRE

```
EXPIRE chiave ttl
```

Restituisce 1 se l'operazione ha avuto successo, altrimenti 0.

Imposta la vita di una chiave in secondi. Allo scadere del tempo specificato, la chiave viene eliminata

EXPIREAT

```
EXPIREAT chiave timestamp
```

Restituisce 1 se l'operazione ha avuto successo, altrimenti 0.

Funziona come `EXPIRE`, ma invece di richiedere il time to live della chiave, richiede un timestamp del momento in cui la chiave scadrà. Questo timestamp va passato in formato Unix.

KEYS

```
KEYS pattern
```

Restituisce tutte le chiavi che corrispondono al pattern di ricerca.

```
> set key 1
OK
> set keey 2
OK
> set kayy 3
OK
> set k3y 4
OK
> keys k?y
1) "k3y"
2) "key"
> keys k*y
1) "k3y"
2) "key"
3) "kayy"
4) "keey"
> keys k[ae]y
1) "key"
> keys k[^e]y
1) "k3y"
> keys k[^a]y
1) "k3y"
2) "key"
```

MIGRATE

```
MIGRATE host porta chiave |"" destination-db timeout [COPY] [REPLACE] [KEYS chiave1
chiave2 chiave3...]
```

Restituisce `OK`, o se le chiavi specificate non esistono, `NOKEY`

Effettua un'operazione atomica di migrazione di una chiave da un'istanza ad un'altra. L'operazione effettua un `DUMP` della chiave o chiavi specificate, le sposta all'istanza di destinazione, effettua un `RESTORE` della chiave o chiavi, e quando il `RESTORE` restituisce `OK`, effettua un `DEL` sull'istanza di origine. Dalla versione di Redis 3.0.6, è possibile effettuare un `MIGRATE` su più chiavi: in questo caso, la prima chiave va sostituita con una stringa vuota, e le chiavi da migrare vanno dichiarate dopo l'opzione `KEYS`. L'opzione `COPY` non effettua l'eliminazione sull'istanza di origine, mentre l'opzione `REPLACE`, va a sostituire nell'istanza di destinazione, una chiave con lo stesso nome di quella in arrivo dall'istanza di origine. Il timeout indica il tempo massimo di attesa durante la comunicazione, espresso in millisecondi. Non si intende però che l'operazione debba essere compiuta nel tempo specificato, ma che l'operazione non vad a bloccarne altre per un tempo maggiore a quello specificato. Se viene oltrepassato questo limite, l'operazione viene annullata, e restituisce un errore speciale: `IOERR`.

MOVE

```
MOVE chiave db
```

Restituisce 1 se la chiave è stata spostata, altrimenti 0.

Sposta la chiave specificata dal database corrente, impostato tramite `SELECT`, al database di destinazione.

OBJECT

```
OBJECT comando [argomento]
```

Ottiene informazioni riguardo ad un oggetto di redis. I comandi disponibili sono:

`OBJECT REFCOUNT chiave`
: restituisce il numero di reference al valore associato alla chiave.

`OBJECT ENCODING chiave`
: restituisce la chiave come è stata codificata da Redis.

`OBJECT IDLETIME chiave`
: restituisce il tempo in cui la chiave non è stata richiesta da operazioni di lettura o scrittura.

Codifica

Redis codifica le sue strutture dati in modi diversi:

Stringhe
: possono essere codificate in `raw` o come `int64`.

Liste
: possono essere codificate come `ziplist` o `linkedlist`.

Set
: possono essere codificati come `intset` o `hashtable`.

Hash
: possono essere codificate come `hashtable` o `ziplist`.

Set Ordinati
: possono essere codificati come `ziplist` o `linkedlist`.

PERSIST

```
PERSIST chiave
```

Rende una chiave persistente, ovvero toglie il contatore impostato con `EXPIRE`.

PEXPIRE

```
PEXPIRE chiave ttl
```

Funziona come `EXPIRE`, ma il ttl viene espresso in millisecondi.

PEXPIREAT

```
PEXPIREAT chiave timestamp
```

Funziona come `EXPIREAT`, ma il timestamp è espresso in millisecondi.

PTTL

```
PTTL chiave
```

Restituisce il time to live della chiave espresso in millisecondi.

RANDOMKEY

```
RANDOMKEY
```

Restituisce una chiave casuale dal database corrente.

RENAME

```
RENAME chiave nome
```

Restituisce `OK`

Rinomina la chiave con il nome specificato.

RENAMENX

```
RENAMENX chiave nome
```

Resstituisce 1 se la chiave è stata rinominata, altrimenti 0.

Rinomina la chiave con il nome specificato se esso non esiste già nel database.

RESTORE

```
RESTORE chiave ttl serialized-value [REPLACE]
```

Restituisce `OK`

Crea una chiave, assegnandole un valore precedentemente serializzato con `DUMP`, ed un time to live. Se il ttl è 0, la nuova chiave non avrà scadenza. L'opzione replace permette di sostituire una chiave, se il nome della chiave specificato è già presente nel database.

SCAN

```
SCAN cursore [MATCH pattern] [COUNT count]
```

Effettua una scansione del dataset corrente di elementi che corrispondano alla pattern. La ricerca è incrementale, ovvero è possibile analizzare un dataset in piccole porzioni. Alla scansione, Redis restituisce un array di 2 elementi: il primo è il cursore da utilizzare per la scansione successiva, mentre il secondo è un array che contiene gli elementi trovati. L'opzione `COUNT count` permette al caller di gestire quanti elementi vengono restituiti nell'array dei risultati. L'opzione `MATCH` permette di cercare elementi in base ad un pattern passato nell'argomento, simile nel funzionamento al comando `KEYS`.

SORT

```
SORT chiave [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]]
[ASC|DESC] [ALPHA] [STORE destinazione]
```

Ordina gli elementi contenuti in una chiave, conservandoli oppure semplicemente restituendoli.

L'opzione `LIMIT` fa in modo che il comando restituisca un numero specifico di elementi. Richiede due argomenti: l'offset, il numero di elementi da saltare, e count, quanti elementi restituire partendo dall'offset. Le opzioni `ASC`, `DESC` e `ALPHA` consentono di visualizzare gli elementi restituiti in ordine crescente, decrescente e lessicografico. Se omessa, l'opzione di default è crescente. L'opzione by utilizza una pattern per generare delle chiavi tramite le quali effettuare l'ordinamento. I nomi delle chiavi sono ottenuti sostituendo il primo asterisco incontrato nell'argomento pattern con il valore dell'elemento nel dataset. Se una lista contiene degli elementi che rappresentano chiavi nel dataset, è possibile, invece di ottenere gli id, ottenere la chiave stessa. Queste chiavi esterne, si ottengono con l'opzione `GET pattern`. Infine è possibile conservare i risultati ottenuti dal `SORT` in un'altra chiave, tramite l'opzione `STORE destinazione`.

TTL

```
TTL chiave
```

Restituisce il time to live di una chiave espresso in secondi.

TYPE

```
TYPE chiave
```

Restituisce una stringa che indica di che tipo è la chiave specificata.

WAIT

```
WAIT numslave timeout
```

Blocca il client finché tutti i comandi di scrittura precedenti non sono stati registrati da almeno un numero specifico di slave. Se viene raggiunto il timeout specificato in millisecondi, il comando esce ed il client riprende la sua attività. Il comando restituisce sempre il numero di slave che hanno registrato i comandi di scrittura. Se il timeout viene impostato a 0, il client rimane sempre in attesa.

Comandi delle Stringhe

APPEND

```
APPEND chiave valore
```

Restituisce: intero: la lunghezza della stringa dopo l'operazione.

Se la chiave esiste, ed è una stringa, `APPEND` va ad aggiungere il valore alla fine della stringa dichiesta.

BITCOUNT

Vedi `BITCOUNT`

BITOP

vedi `BITOP`

BITPOS

vedi `BITPOS`

DECR

```
DECR chiave
```

Restituisce: intero: il valore della chiave dopo la sottrazione.

Se la chiave contiene un valore numerico, e solo un valore numerico, `DECR` sottrae 1 al valore contenuto nella chiave. Funziona solo con int64; se la chiave non esiste, viene creata una nuova chiave, con valore 0, su cui viene poi effettuata la sottrazione.

DECRBY

```
DECRBY chiave valore
```

Restituisce: intero: il valore della chiave dopo la sottrazione.

Come `DECR`, va ad effettuare una sottrazione su una stringa che contiene valori numerici. `DECRBY` però richiede come argomento il valore da sottrarre dalla stringa. Se la chiave non esiste, viene creata una nuova chiave, con valore 0, su cui viene poi effettuata la sottrazione.

GET

```
GET chiave
```

Restituisce: stringa o `nil` se la chiave non esiste.

GETBIT

Vedi `GETBIT`.

GETRANGE

```
GETRANGE chiave [inizio] [fine]
```

Restituisce: stringa.

`GETRANGE` restituisce una stringa contenente i caratteri compresi tra gli indici di inizio e fine, della stringa specificata nella chiave. Come indici accetta numeri negativi, che rappresentano i caratteri in posizione -x dalla fine della stringa. -1 è l'ultimo carattere, -2 il penultimo, -3 il terzultimo e così via. Entrambi i valori di inizio e fine sono inclusi nel range da estrarre; Le richieste che vanno oltre le dimensioni della stringa, vengono limitate alla fine della stringa stessa.

GETSET

```
GETSET chiave valore
```

Restituisce: stringa o `nil`.

Sostituisce il valore assegnato alla chiave quello specificato nell'argomento, e restituisce il valore

precedente alla sostituzione. Se la chiave prima non esisteva, restituisce `nil`.

INCR

```
INCR chiave
```

Restituisce intero: valoredella chiave dopo l'addizione.

Se la chiave contiene un valore numerico, e solo un valore numerico, `INCR` incrementa di 1 il valore contenuto nella chiave. Funziona solo con int64; se la chiave non esiste, viene creata una nuova chiave, con valore 0, su cui viene poi effettuato l'incremento.

INCRBY

```
INCRBY chiave valore
```

Restituisce: intero: il valore della chiave dopo l'addizione.

Come `INCR`, va ad effettuare un'addizione su una stringa che contiene valori numerici. `INCRBY` però richiede come argomento il valore da aggiungere dalla stringa. Se la chiave non esiste, viene creata una nuova chiave, con valore 0, su cui viene poi effettuata l'addizione.

INCRBYFLOAT

```
INCRBYFLOAT chiave valore
```

Restituisce: intero: il valore della chiave dopo l'addizione.

Come `INCRBY`, va ad effettuare l'addizione di un valore su una stringa. `INCRBYFLOAT` però accetta come argomento numeri decimali, di tipo double, con una precisione di 17 cifre. Se la chiave non esiste, viene creata una nuova chiave, con valore 0, su cui viene poi effettuata l'addizione.

MGET

```
MGET chiave1 chiave2 chiave3...
```

Restituisce: array di stringhe.

Come il `GET` restituisce il valore della chiave specificata nell'argomento; il `MGET` però accetta più chiavi nell'argomento, e restituisce il valore di ognuna.

MSET

```
MSET chiave1 valore1 chiave2 valore2 chiave3 valore3...
```

Restituisce: intero: il numero di chiavi create o modificate.

Come il `SET` imposta più chiavi e valori in un'unica chiamata. Ogni coppia chiave-valore è separata da uno spazio. Se le chiavi non esistono vengono create.

MSETNX

```
MSETNX chiave1 valore1 chiave2 valore2 chiave3 valore3...
```

Restituisce: intero: il numero di chiavi create o modificate.

Come il `MSET` imposta più chiavi e valori in un'unica chiamata, ma a differenza del `MSET`, imposta le coppie chiave-valore solo le le chiavi non eistono. Se anche solo una chiave di quelle indicate nell'argomento esiste, `MSETNX` non ne imposta nessuna.

PSETEX

```
PSETEX chiave tempo[ms]
```

`PSETEX` funziona come `SETEX`, tranne che il ttl è espresso in millisecondi.

SET

```
SET chiave "valore" [opzioni]
```

Restituisce: `OK` oppure `nil` se non viene creata o modificata la chiave (ad esempio se una condizione di esistenza non si verifica).

`SET` va a creare o modificare una chiave, impostando il valore passato nell'argomento. Qualsiasi valore inserito viene interpretato come stringa, e, a meno che non ci sia uno spazio, o della punteggiatura, i doppi apici non sono necessari. Le opzioni passabili come argomento sono:

Le condizioni di esistenza della chiave

`NX`

: Imposta la chiave se la chiave specificata non esiste.

`XX`

: Imposta la chiave se la chiave specificata esiste.

La vita della chiave

`EX` *[secondi]*

: Imposta la durata della chiave.

`PX` *[millisecondi]*

Imposta la durata della chiave in millisecondi.

SETBIT

Vedi `SETBIT`.

SETEX

```
SETEX chiave secondi valore
```

Restituisce: `OK`.

Funziona come `SET chiave valore EX secondi`, assegna un valore alla chiave specificata ed imposta il tempo durante il quale la chiave esite. Esaurito questo tempo, la chiave viene eliminata.

SETNX

Restituisce: `OK` se la creazione ha avuto successo, altrimenti restituisce `nil`.

Funziona come `SET chiave valore NX`, assegna un valore alla chiave specificata se la chiave esiste.

SETRANGE

```
SETRANGE chiave indice valore
```

Restituisce: intero: la lunghezza della stringa modificata

`SETRANGE` va a sostituire una parte della stringa a partire dall'indice specificato, con il valore passato nell'argomento.

STRLEN

```
STRLEN chiave
```

Restituisce: intero: la lunghezza della stringa.

Comandi delle Liste

BLPOP

```
BLPOP chaive1 chiave2 chiave3... secondi
```

Restituisce: un array: l'array può contenere `nil` se il tempo di attesa è scaduto, oppure due elementi:

il primo è la chiave della lista da cui ha preso il valore, il secondo è il valore.

`BLPOP` mette il client in attesa di un elemento da prendere dalla cima di una lista; il timeout, espresso in secondi nell'argomento, dopo la serie di liste da cui il client attende dati, indica per quanto il client rimane in attesa di un dato dalle liste, bloccando gli altri client. Il comando va a prendere il primo elemento che viene aggiunto ad una delle liste specificate, se sono vuote, altrimenti dalla prima lista non vuota.

BRPOP

```
BRPOP chiave1 chiave2 chiave3.... secondi
```

Restituisce: un array: l'array può contenere `nil` se il tempo di attesa è scaduto, oppure due elementi: il primo è la chiave della lista da cui ha preso il valore, il secondo è il valore.

`BRPOP` mette il client in attesa di un elemento da prendere dal fondo di una lista; il timeout, espresso in secondi nell'argomento, dopo la serie di liste da cui il client attende dati, indica per quanto il client rimane in attesa di un dato dalle liste, bloccando gli altri client. Il comando va a prendere il primo elemento che viene aggiunto ad una delle liste specificate, se sono vuote, altrimenti dalla prima lista non vuota.

BRPOPLPUSH

```
BRPOPLPUSH fonte destinazione secondi
```

Restituisce: stringa o `nil`: restituisce l'elemento spostato dalla fonte alla destinazione, o `nil` se il tempo di attesa è stato esaurito.

`BRPOPLPUSH` funziona come `RPOPLPUSH`, ma nel caso la lista fonte sia vuota, rimane in attesa per un numero di secondi specificato nell'argomento.

LINDEX

```
LINDEX chiave indice
```

Restituisce: stringa o `nil`

`LINDEX` restituisce il valore della lista all'indice desiderato. Supporta i valori negativi, dove -1 è l'ultimo elemento della lista, -2 il penultimo e così via. Se l'indice desiderato non è contenuto nella lista, restituisce `nil`

LINSERT

```
LINSERT chiave [BEFORE|AFTER] indice valore
```

Restituisce: intero: la lunghezza della lista dopo l'inserimento.

`LINSERT` va ad inserire un valore prima (BEFORE) o dopo (AFTER) l'indice selezionato. Se l'indice è al di fuori della lista restituisce -1.

LLEN

```
LLEN chiave
```

Restituisce: intero: la lunghezza della lista.

LPOP

```
LPOP chiave
```

Restituisce: stringa o `nil`

Rimuove e restituisce il primo elemento di una lista.

LPUSH

```
LPUSH chiave valore1 valore2 valore3...
```

Restituisce: intero: la lunghezza della lista modificata.

Inserisce uno o più valori in cima ad una lista. Se sono stati passati più valori, verranno inseriti da sinistra verso destra, quindi nel nostro esempio, valore3 sarà in cima alla lista.

LPUSHX

```
LPUSHX chiave valore1 valore2 valore3...
```

Restituisce: intero: la lunghezza della lista modificata.

Funziona come `LPUSH`, tranne che inserisce i valori solo le la lista esiste, altrimenti restituisce 0.

LRANGE

```
LRANGE chiave [inizio] [fine]
```

Restituisce: array.

`LRANGE` restituisce un array contenente gli elementi compresi tra gli indici di inizio e fine, della lista specificata nella chiave. Come indici accetta numeri negativi, che rappresentano i caratteri in

posizione -x dalla fine della stringa. -1 è l'ultimo carattere, -2 il penultimo, -3 il terzultimo e così via. Entrambi i valori di inizio e fine sono inclusi nel range da estrarre; Se l'inizio va oltre la fine della lista, restituisce una lista vuota, se la fine va oltre la lunghezza della lista, Redis lo interpreta come -1.

LREM

```
LREM chiave quantità valore
```

Restituisce: intero: il numero di elementi rimossi.

`LREM` rimuove un elemento da una lista un numero di volte specificato nell'argomento. Se viene passato un numero negativo, ad esempio -2, rimuove le prime due ricorrenze del valore dalla fine della lista all'inizio, mentre se viene passato un numero positivo, ad esempio 3, rimuove le prime 3 ricorrenze del valore dall'inizio della lista verso la fine.

LSET

```
LSET chiave indice valore
```

Assegna un valore all'elemento all'indice specificato.

LTRIM

```
LTRIM chiave inizio fine
```

Restituisce: intero: la lunghezza della nuova lista

Elimina tutti gli elementi di una lista non compresi tra gli indici specificati, ottenendo così una lista più corta, delle dimensioni specificate nell'argomento. Inizio e fine possono essere anche numeri negativi; se inizio > fine o è al di fuori della lista, restituisce una lista vuota, causando la perdita di tutti gli elementi al suo interno.

RPOP

```
RPOP chiave
```

Come `LPOP` rimuove e restituisce un elemento, ma a differenza di `LPOP`, lo rimuove dalla fine della lista.

RPOPLPUSH

```
RPOPLPUSH fonte destinazione
```

Restituisce: stringa: l'elemento spostato.

Rimuove l'ultimo elemento di una lista, e lo inserisce in cima ad una lista di destinazione. Se la lista da cui prende l'elemento è vuota o non esiste, restituisce `nil` e non vengono effettuate altre operazioni. Fonte e destinazione possono essere uguali; in questo caso l'elemento della lista viene spostato in cima.

RPUSH

```
RPUSH chiave valore1 valore2 valore3...
```

Restituisce: intero: la lunghezza della lista modificata.

Inserisce uno o più valori in fondo ad una lista. Se sono stati passati più valori, verranno inseriti da sinistra verso destra, quindi nel nostro esempio, valore3 sarà in fondo alla lista.

RPUSHX

```
RPUSHX chiave valore1 valore2 valore3
```

Restituisce: intero: la lunghezza della lista modificata.

Funziona come `RPUSH`, tranne che inserisce i valori solo le la lista esiste, altrimenti restituisce 0.

Comandi delle Hash

HDEL

```
HDEL chiave campo1 campo2 campo3...
```

Restituisce: intero: il numero di campi eliminati.

Elimina i campi specificati. I campi che non esistono vengono ignorati, e non sono contati tra quelli eliminati.

HEXISTS

```
HEXISTS chiave campo
```

Restituisce: intero.

Verifica se un campo esiste o meno nell'hash, restituendo 1 se il campo esiste, o 0 se non esiste.

HGET

```
HGET chiave campo
```

Restituisce il valore del campo specificato, o `nil` se il campo non esiste.

HGETALL

```
HGETALL chiave
```

Restituisce tutti i campi ed i relativi valori di una chiave in forma di array: se la chiave non esiste restituisce un array vuoto.

HINCRBY

```
HINCRBY chiave campo incremento
```

Restituisce: intero.

Come `INCRBY`, va ad incrementare di un valore specifico un campo numerico della hash, restituendo il nuovo valore del campo. Se il campo non esiste, viene creato un nuovo campo con valore 0, sul quale viene poi effettuata l'addizione. Anche `HINCRBY` accetta int64.

HINCRBYFLOAT

```
HINCRBYFLOAT chiave campo incremento
```

Restituisce: stringa.

Come `INCRBYFLOAT`, va ad incrementare di un valore di tipo double il campo di una hash. Se il campo non esiste, viene creato con un valore di 0, e poi viene effettuata l'addizione.

HKEYS

```
HKEYS chiave
```

Restituisce i nomi dei campi di una hash, ma non i valori associati. I campi vengono espressi in forma di array, se la chiave non esiste, restituisce un array vuoto.

HLEN

```
HLEN chiave
```

Restituisce un intero che indica la lunghezza della hash, ovvero il numero di campi presenti. Se la chiave non esiste, restituisce 0.

HMGET

```
HMGET chiave campo1 campo2 campo3...
```

Restituisce il valore di più campi specificati.

HMSET

```
HMSET chiave campo1 valore1 campo2 valore2 campo3 valore3...
```

Restituisce: `OK`

Imposta il valore di uno o più campi di una hash; se la chiave o i campi non esistono, vengono creati.

HSCAN

Vedi `SCAN`

HSET

```
HSET chiave campo valore
```

Restituisce: `OK`

Assegna ad un campo della hash un valore specifico. Se la hash o il campo non esistono, vengono creati.

HSETNX

```
HSETNX chiave campo valore
```

Restituisce: intero.

Assegna ad un campo della hash un valore specifico solo se il valore non esiste. Se la creazione del campo ha avuto successo, restituisce 1, altrimenti, se il campo esisteva già, restituisce 0.

HSTRLEN

```
HSTRLEN chiave campo
```

Restituisce un intero che indica la lunghezza del valore assegnato ad un campo della hash. Se il valore non esiste, restituisce 0

HVALS

```
HVALS chiave
```

Restituisce un array contenente tutti i valori della hash, senza i relativi nomi dei campi. Se la chiave non esiste, restituisce un array vuoto.

Comandi dei Set

SADD

```
SADD chiave chiave1 chiave2 chiave3...
```

Restituisce: intero: il numero di elementi unici aggiunti al set.

Aggiunge gli elementi specificati al set; se un elemento è già presente nel set, non viene aggiunto.

SCARD

```
SCARD chiave
```

Restituisce un intero che indica la cardinalità del set, ovvero il numero di elementi unici presenti al suo interno. Dato che un set contiene solo elementi unici, `SCARD` viene utilizzato per ottenere la lunghezza del set. Se la chiave non esiste, `SCARD` restituisce 0.

SDIFF

```
SDIFF chiave1 chiave2 chiave3...
```

Effettua la differenza tra più set, restituendo un array che contiene la differenza tra il primo set specificato, ed i set specificati in seguito. Nel nostro esempio, il risultato della differenza sarà (`chiave1` - `chiave2`) - `chiave3`.

SDIFFSTORE

```
SDIFFSTORE destinazione chiave1 chiave2 chiave3...
```

Restituisce: intero: il numero di elementi del set di destinazione.

Come `SDIFF`, effettua la differenza tra più set, ma, con gli elementi ottenuti va a popolare un altro

set di destinazione, contenente il risultato.

SINTER

```
SINTER chiave1 chiave2 chiave3
```

Restituisce: array.

Effettua l'intersezione tra più set, restituendo un array che contiene gli elementi che tutti i set hanno in comune tra loro. Le chiavi che non esistono vengono interpretate come set vuoti, in questo caso anche il risultato però sarà un array vuoto, dato che non ha elementi in comune con gli altri set.

SINTERSTORE

```
SINTERSTORE destinazione chiave1 chiave2 chiave3
```

Restituisce: intero: il numero di elementi nel set di destinazione

Come `SINTER`, effettua l'intersezione tra più set, ma il risultato ottenuto viene inserito in un set di destinazione.

SISMEMBER

```
SISMEBER chiave valore
```

Restituisce 1 se il valore specificato è contenuto nel set, altrimenti restituisce 0.

SMEMBERS

```
SMEMBERS chiave
```

Restituisce un array contenente tutti gli elementi di un set.

SMOVE

```
SMOVE fonte destinazione valore
```

Restituisce: intero.

Sposta un valore da un set fonte ad un set di destinazione. Se il set fonte non contiene l'elemento o non esiste, l'operazione non viene effettuata. Se l'elemento specificato è già presente nel set di destinazione, viene rimosso dalla fonte, ma non viene inserito nella destinazione, ma l'operazione viene considerata come portata a termine. Se l'operazione ha avuto successo, restituisce 1,

altrimenti restituisce 0.

SPOP

```
SPOP chiave quantità
```

Restituisce un numero specificato di elementi di un set come stringhe, rimuovendoli dal set stesso; gli elementi vengono estratti senza un ordine preciso. La quantità può essere omessa, in questo caso viene restituito solo un elemento. Se la chiave non esiste, restituisce `nil`

SRANDMEMBER

```
SRANDMEMBER chiave quantità
```

Simile a `SPOP`, restituisce un numero di elementi estratti a caso dal set, ma a differenza di `SPOP`, gli elementi ottenuti non vengono rimossi dal set. Se viene omessa la quantità, viene restituito solo un elemento, if forma di stringa, altrimenti gli elementi vengono restituiti in un array.

SREM

```
SREM valore1 valore2 valore3...
```

Restituisce: intero: il numero di elementi rimossi

Rimuove uno o più elementi da un set; se un elemento non esiste, viene ignorato. Se la chiave non esiste, restituisce 0.

SSCAN

Vedi `SCAN`.

SUNION

```
SUNION chiave1 chiave2 chiave3...
```

Effettua l'unione tra più set, restituendo un array contenente tutti gli elementi unici di tutti i set. Se lo stesso elemento è presente in più set, viene inserito solo una volta.

SUNIONSTORE

```
SUNIONSTORE destinazione chiave1 chiave2 chiave3...
```

Restituisce: intero: il numero di elementi del set di destinazione.

Effettua l'unione tra più set, andando a popolare un set di destinazione con tutti gli elementi unici di tutti i set.

Comandi dei Set Ordinati

ZADD

```
ZADD chiave valore1 valore2 valore3
```

Restituisce: intero: il numero di elementi aggiunti.

Aggiunge gli elementi specificati al SO. Gli elementi già presenti nel set non vengono aggiunti.

ZCARD

```
ZCARD chiave
```

Restituisce la cardinalità del SO, quindi il numero di elementi al suo interno, dato che i SO contengono solo elementi unici.

ZCOUNT

```
ZCOUNT chiave min max
```

Restituisce il numero di elementi in un SO con uno score compreso tra `min` e `max`. Di default, Redis interpreta min e max come valori inclusivi, quindi compresi nel range di valori tra cui cercare. Aggiungendo una parentesi tonda prima del valore però, vengono interpretati come esclusivi.

```
ZCOUNT chiave (1 5
```

Nell'esempio, verranno restituiti tutti i valori con uno score compreso tra 2 e 5. Un'altra opzione è quella di passare -inf o +inf ovvero - infinito e + infinito.

ZINCRBY

```
ZINCRBY chiave incremento valore
```

Come `INCRBY`, va ad incrementare di un valore specifico un campo numerico del SO, restituendo il nuovo valore del campo. Se il campo non esiste, viene creato un nuovo campo con valore 0, sul quale viene poi effettuata l'addizione. Anche `ZINCRBY` accetta int64.

ZINTERSTORE

```
ZINTERSTORE destinazione numchiavi chiave1 chiave2 chiave3... [WEIGHTS [weight1
weight2 weight3...]] [AGGREGATE SUM|MIN|MAX]
```

Restituisce: intero: il numero di elementi del SO di destinazione

Effettua un'intersezione tra più SO, andando a popolare un SO di destinazione con gli elementi in comune tra i SO specificati nell'argomento. Va specificato il numero di chiavi su cui effettuare l'intersezione, prima di passare le chiavi stesse. L'opzione weights richiede un intero per ogni SO da intersecare, e rappresenta il fattore per il quale moltiplicare lo score dei singoli SO; se non viene specificato, vengono moltiplicati per 1. L'opzione aggregate indica invece come ottenere gli score dei vari elementi. Di default è sum, ovvero lo score dell'elemento in comune tra i set viene sommato. Alternativamente si può scegliere di tenere lo score più basso `MIN` o quello più alto `MAX`

```
ZADD chiave1 1 a 2 b 3 c 4 d
(integer) 4
ZADD chiave2 1 a 2 2 3 d
(integer) 3
ZADD chiave3 1 1 2 2 3 3 5 a 8 d
(integer) 5
ZINTERSTORE dest 3 chiave1 chiave2 chiave3 WEIGHTS 1 2 3 AGGREGATE SUM
(integer) 2
ZRANGE dest 0 -1 WITHSCORES
1) "a"
2) "18"
3) "d"
4) "34"
```

Nell'esempio, i valori in comune a tutti e tre i SO sono `a` e `d`: i loro rispettivi score all'estrazione dal SO sono: 1*1, 1*2 e 5*3 per `a` e 4*1, 3*2 e 8*3 per `d`, come specificato nell'opzione weights. Gli scores vengono poi "passati" alla funzione `AGGREGATE`, che ne effettua la somma: per `a` quindi saranno 1+2+15 e per `d` 4+6+24.

ZLEXCOUNT

```
ZLEXCOUNT chiave inizio fine
```

Se tutti gli elementi di un SO hanno lo stesso score, `ZLEXCOUNT` restituisce un intero che indica quanti elementi sono presenti nell'intervallo compreso tra inizio e fine. `ZLEXCOUNT` esamina gli elementi non per score, ma in base al valore assegnato. Inizio e fine possono essere passati come limiti inclusivi del range, indicandoli con una parentesi quadra aperta prima del carattere, ad esempio `[f` o esclusivi, indicati con una parentesi tonda aperta prima del carattere, ad esempio `(t`. In oltre è possibile passare come argomenti `+` e `-`, che funzionano come `+inf` e `-inf` in `ZCOUNT`.

ZRANGE

```
ZRANGE chiave inizio fine [WITHSCORES]
```

Restituisce un array che contiene gli elementi di un SO compresi tra gli indici passati all'argomento. Gli elementi sono ordinati per score e, nel caso abbiano lo stesso score, lessicograficamente, e sono disposti in ordine crescente. L'opzione `WITHSCORES` restituisce un array che contiene coppie valore-score. Come gli altri `RANGE`, è possibile inserire numeri negativi.

ZRANGEBYLEX

```
ZRANGEBYLEX chiave inizio fine [LIMIT offset count]
```

Se tutti gli elementi di un SO hanno lo stesso score, `ZRANGEBYLEX` restituisce un array contenete gli elementi del SO compresi nell'intervallo specificato. Il range di valori da passare funziona come in `ZLEXCOUNT`. L'opzione `LIMIT` indica, se specificata, il numero di elementi da prendere, che rientrano nel range. `Offset` indica quanti elementi saltare all'inizio della selezione, mentre `count` indica quanti elementi prendere dalla selezione.

```
ZADD chiave 0 a 0 b 0 c 0 d 0 e 0 f 0 g
(integer) 7
redis>  ZRANGEBYLEX chiave - [c
1) "a"
2) "b"
3) "c"
ZRANGEBYLEX chiave - (c
1) "a"
2) "b"
ZRANGEBYLEX chiave [aaa (g
1) "b"
2) "c"
3) "d"
4) "e"
5) "f"
ZRANGEBYLEX chiave [aaa (g LIMIT 2 3
1) "d"
2) "e"
3) "f"
```

ZRANGEBYSCORE

```
ZRANGEBYSCORE chiave min max [WITHSCORES] [LIMIT offset count]
```

Restituisce un array contenente tutti gli elementi di un SO con uno score compreso tra `min` e `max`. Di default, Redis interpreta min e max come valori inclusivi, quindi compresi nel range di valori tra

cui cercare. Aggiungendo una parentesi tonda prima del valore però, vengono interpretati come esclusivi. L'opzione `WITHSCORES` restituisce un array che contiene coppie valore-score, mentre l'opzione `LIMIT` funziona come nel comando `ZRANGEBYLEX`.

ZRANK

```
ZRANK chiave valore
```

Restituisce un intero che indica l'indice del valore nel SO. Il SO è a base 0, il che significa che all'inidice 0 c'è l'elemento con lo score più basso.

ZREM

```
ZREM chiave valore1 valore2 valore3...
```

Rimuove gli elementi specificati dal SO, restituendo un intero che indica il numero di elementi rimossi.

ZREMRANGEBYLEX

```
ZREMRANGEBYLEX chiave inizio fine
```

Se tutti gli elementi di un SO hanno lo stesso score, `ZREMRANGEBYLEX` ordina gli elementi lessicograficamente, e rimuove gli elementi compresi tra inizio e fine, restituendo un intero che indica quanti elementi sono stati rimossi. I limiti del range funzionano come nel comando `ZRANGEBYLEX`.

ZREMRANGEBYRANK

```
ZREMRANGEBYRANK chiave inizio fine
```

Rimuove gli elementi compresi tra gli indici inizio e fine, e restituisce un intero che indica quanti elementi sono stati eliminati. I limiti sono inclusivi.

ZREMRANGEBYSCORE

```
ZREMRANGEBYSCORE chiave inizio fine
```

Rimuove gli elementi con uno score compreso tra inizio e fine. Come in `ZRANGEBYSCORE`, i limiti del range possono essere impostati come inclusivi o esclusivi.

ZREVRANGE

```
ZREVRANGE chiave inizio fine [WITHSCORES]
```

Funziona come `ZRANGE`, ma restituisce un array di elementi ordinati in ordine decrescente.

ZREVRANGEBYLEX

```
ZREVRANGEBYLEX chiave inizio fine [LIMIT offset count]
```

Funziona come `ZRANGEBYLEX`, ma restituisce un array di elementi ordinati in ordine decrescente.

ZREVRANGEBYSCORE

```
ZREVRANGEBYSCORE chiave inizio fine [WITHSCORES] [LIMIT offset count]
```

Funziona come `ZRANGEBYSCORE` ma restituisce un array di elementi ordinati in ordine decrescente.

ZREVRANK

```
ZREVRANK chiave valore
```

Funziona come `ZRANK`, ma gli elementi vengono ordinati in ordine decrescente, quindi l'elemento all'indice 0 avrà lo score più alto.

ZSCAN

Vedi `SCAN`.

ZSCORE

```
ZSCORE chiave valore
```

Restituisce una stringa che contiene lo score dell'elemento specificato. Se l'elemento non esiste, restituisce `nil`

ZUNIONSTORE

```
ZUNIONSTORE dest chiavi chiave1 chiave2 chiave3... [WEIGHTS] [AGGREGATE]
```

Effettua l'unione tra due o più SO, andando a popolare un SO con gli elementi unici dei SO passati nell'argomento. Come con `ZINTERSTORE`, è possibile impostare il comportamento dello score dei vari

elementi attraverso le opzioni `WEIGHTS` e `AGGREGATE`

Comandi delle Bitmap

BITCOUNT

```
BITCOUNT chiave [inizio] [fine]
```

Restituisce: intero.

Conta i byte di una bitmap, restituendo il numero di bit impostati su 1. Può analizzare l'intera bitmap, oppure, se vengono passati gli argomenti inizio e fine, conta gli elementi in un range specifico. Come per `GETRANGE`, accetta come argomenti anche valori negativi, dove -1 è l'ultimo indice della bitmap, -2 è il penultimo e così via.

BITOP

```
BITOP operazione destinazione chiave1, chiave2, chiave3...
```

Restituisce: intero: la dimensione della stringa di destinazione.

Effettua le operazioni AND, NOT, OR e XOR su più bitmap, e restituendo il risultato in una stringa di destinazione. L'operatore NOT, essendo unario, richiede solo una stringa di destinazione ed una chiave su cui effettuare l'operazione. Se due o più bitmap sono di lunghezze differenti, a quelle più corte vengono aggiunti degli 0 finché non raggiungono la dimensione di quella più lunga.

BITPOS

```
BITPOS chiave bit [inizio] [fine]
```

Esamina una bitmap, restituendo l'indice del primo bit impostato su 0 o 1, a seconda di cosa viene richiesto tra gli argomenti del metodo. Come `BITCOUNT`, può esaminare parte della bitmap, se vengono passati gli indici di inizio e fine del range tra gli argomenti.

GETBIT

```
GETBIT chiave indice
```

Restituisce: intero.

Restituisce il valore di un bit alla posizione passata nell'argomento. Se l'indice richiesto è oltre la fine della stringa, restituisce 0, così come restituisce 0 se la stringa è vuota.

SETBIT

```
SETBIT chiave indice valore
```

Restituisce: intero: il valore originale del bit modificato.

Imposta il bit all'indice desiderato con il valore espresso nell'argomento. Se l'indice specificato nell'argomento è fuori dalla stringa, quest'ultima viene ridimensionata per contenere l'indice desiderato.

Comandi degli Hyperloglog

PFADD

```
PFADD chiave valore1 valore2 valore3
```

Restituisce: intero: 1 se almeno un elemento unico è stato inserito, altrimenti 0.

Inserisce nell'HLL i valori dell'argomento. Se tra gli elementi inseriti c'è un elemento che non è già presente nell'HLL, restituisce 1.

PFCOUNT

```
PFCOUNT chiave1 chiave2 chiave3...
```

Restituisce: intero: la cardinalità dell'HLL

`PFCOUNT`, se chiamato con una sola chiave come argomento, restituisce la cardinalità dell'HLL, ovvero il numero di elementi unici. Se invece viene chiamato con più chiavi, effettua un'unione degli HLL passati nell'argomento, creando un HLL temporaneo, e restituendo la cardinalità di quell'HLL. Essendo una struttura dati probabilistica, può essere che restituisca una cardinalità sbagliata, ma la percentuale di errore e molto bassa, 0.81%.

PFMERGE

```
PFMERGE destinazione fonte1 fonte2 fonte3
```

`PFMERGE` unisce più HLL in un unico Hyperloglog; se non esiste viene creato.