Redis: In-Memory Caching

Redis (Remote Dictionary Server) is an open-source, in-memory data structure store, used as a database, cache, and message broker. It supports various data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, and geospatial indexes with radius queries.

Key Features of Redis

- **In-Memory Storage**: Redis stores data in memory, which makes it extremely fast for read and write operations.
- **Persistence**: Redis can persist data to disk, allowing it to recover data after a restart.
- **Replication**: Redis supports master-slave replication, enabling high availability and scalability.
- **Data Structures**: Redis supports a variety of data structures, making it versatile for different use cases.
- **Atomic Operations**: All Redis operations are atomic, ensuring data integrity.

Redis Commands with Examples

1. **Strings**

Strings are the most basic type of Redis value. They are binary-safe and can store any kind of data.

- **SET: Set the value of a key.**

```
\# Command
SET mykey "Hello"

\# Result
OK
```

- **GET: Get the value of a key.**

```
\# Previous Value
"Hello" (set using \`SET mykey "Hello"\`)

\# Command
GET mykey

\# Result
"Hello"
```

- **INCR: Increment the integer value of a key by 1.**

```
\# Previous Value
10 (set using \`SET counter 10\`)

\# Command
INCR counter

\# Result
(integer) 11
```

- **APPEND: Append a value to a key.**

```
\# Previous Value
"Hello" (set using \`SET mykey "Hello"\`)

\# Command
APPEND mykey " World"

\# Result
(integer) 11  \# Length of the new string
```

- **DECR: Decrement the integer value of a key by 1.**

```
\# Previous Value
10 (set using \`SET counter 10\`)

\# Command
DECR counter

\# Result
(integer) 9
```

- **DECRBY: Decrement the integer value of a key by a specified amount.**

```
\# Previous Value
10 (set using \`SET counter 10\`)

\# Command
DECRBY counter 5

\# Result
(integer) 5
```

- **GETDEL: Get the value of a key and then delete it.**

```
\# Previous Value
"Hello" (set using \`SET mykey "Hello"\`)

\# Command
GETDEL mykey

\# Result
"Hello"  \# Key is deleted after this command
```

- **GETEX: Get the value of a key and optionally set its expiration.**

```
\# Previous Value
"Hello" (set using \`SET mykey "Hello"\`)

\# Command
GETEX mykey EX 60  \# Set expiration to 60 seconds

\# Result
"Hello"
```

- **GETRANGE: Get a substring of the string stored at a key.**

```
\# Previous Value
"Hello World" (set using \`SET mykey "Hello World"\`)

\# Command
GETRANGE mykey 0 4

\# Result
"Hello"
```

- **GETSET: Set a new value for a key and return its old value.**

```
\# Previous Value
"Hello" (set using \`SET mykey "Hello"\`)

\# Command
GETSET mykey "World"
```

```
\# Result
"Hello"  \# Old value
```

- **INCRBY: Increment the integer value of a key by a specified amount.**

```
\# Previous Value
10 (set using \`SET counter 10\`)

\# Command
INCRBY counter 5

\# Result
(integer) 15
```

- **INCRBYFLOAT: Increment the float value of a key by a specified amount.**

```
\# Previous Value
10.5 (set using \`SET counter 10.5\`)

\# Command
INCRBYFLOAT counter 2.3

\# Result
"12.8"  \# Result is returned as a string
```

- **LCS: Find the longest common substring between two strings.**

```
\# Previous Values
"Hello" (set using \`SET key1 "Hello"\`)
"Hallo" (set using \`SET key2 "Hallo"\`)

\# Command
LCS key1 key2

\# Result
"allo"
```

- **MGET: Get the values of multiple keys.**

```
\# Previous Values
"Hello" (set using \`SET key1 "Hello"\`)
"World" (set using \`SET key2 "World"\`)

\# Command
MGET key1 key2

\# Result
1) "Hello"
2) "World"
```

- **MSET: Set multiple keys to multiple values.**

```
\# Command
MSET key1 "Hello" key2 "World"

\# Result
OK
```

- **MSETNX: Set multiple keys to multiple values only if none of the keys exist.**

```
\# Previous Values
key1 does not exist
key2 does not exist

\# Command
MSETNX key1 "Hello" key2 "World"

\# Result
(integer) 1  \# Success
```

- **PSETEX: Set the value and expiration in milliseconds for a key.**

```
\# Command
PSETEX mykey 5000 "Hello"  \# Expires in 5000 milliseconds (5 seconds)

\# Result
OK
```

- **SETEX: Set the value and expiration in seconds for a key.**

```
\# Command
SETEX mykey 10 "Hello"  \# Expires in 10 seconds

\# Result
OK
```

- **SETNX: Set the value of a key only if it does not exist.**

```
\# Previous Value
key1 does not exist

\# Command
SETNX key1 "Hello"

\# Result
(integer) 1  \# Success
```

- **SETRANGE: Overwrite part of a string at a key starting at a specified offset.**

```
\# Previous Value
"Hello World" (set using \`SET mykey "Hello World"\`)

\# Command
SETRANGE mykey 6 "Redis"

\# Result
(integer) 11  \# Length of the new string
```

- **STRLEN: Get the length of the value stored at a key.**

```
\# Previous Value
"Hello" (set using \`SET mykey "Hello"\`)

\# Command
STRLEN mykey

\# Result
(integer) 5
```

- **SUBSTR: Get a substring of the string stored at a key (deprecated, use `GETRANGE` instead).**

```
\# Previous Value
"Hello World" (set using \`SET mykey "Hello World"\`)

\# Command
SUBSTR mykey 0 4

\# Result
"Hello"
```

---

2. **Hashes**

Hashes are maps between string fields and string values.

- **HSET**: Set the value of a field in a hash.
  - Syntax: `HSET key field value [field value ...]`

```
# Set a single field
HSET user:1000 name "John Doe"

# Set multiple fields
HSET user:1000 age 30 email "john.doe@example.com"
```

- **HGET**: Get the value of a field in a hash.
  - Syntax: `HGET key field`

```
HGET user:1000 name
Output =>
"John Doe"
```

- **HGETALL**: Get all fields and values in a hash.
  - Syntax: `HGETALL key`

```
 HGETALL user:1000
 Output =>
1) "name"
2) "John Doe"
3) "age"
4) "30"
5) "email"
6) "john.doe@example.com"
```

- **HKEYS**: Get all fields in a hash.
  - Syntax: `HKEYS key`

```
 HKEYS user:1000
 Output =>
1) "name"
2) "age"
3) "email"
```

- **HVALS**: Get all values in a hash.
  - Syntax: `HVALS key`

```
HVALS user:1000
Output =>
1) "John Doe"
2) "30"
3) "john.doe@example.com"
```

- **HEXISTS**: Check if a field exists in a hash.
  - Syntax: `HEXISTS key field`
  - Returns `1` if the field exists, `0` otherwise.

```
HEXISTS user:1000 name
Output =>
(integer) 1
```

- **HDEL**: Delete one or more fields from a hash.
  - Syntax: `HDEL key field [field ...]`

```
HDEL user:1000 email
HGETALL user:1000
Output =>
1) "name"
2) "John Doe"
3) "age"
4) "30"
```

- **DEL**: Removes the entire key and its associated data.
  - Syntax: `DEL key [key ...]`

```
DEL user:1000
HGETALL user:1000
Output =>
(empty list or set)
```

3. **Lists**

Lists are collections of string elements sorted by insertion order **unique value not required** it's work like **queue** push at **start** pop at **end** of queue.

- **LPUSH**: Insert one or more elements at the head of a list.

```
LPUSH mylist "world"
LPUSH mylist "hello"
LRANGE mylist 0 -1
Output:
1) "hello"
2) "world"
```

- **RPUSH**: Insert one or more elements at the tail of a list.

```
RPUSH mylist "end"
LRANGE mylist 0 -1
Output:
1) "hello"
2) "world"
3) "end"
```

- **LPOP**: Remove and return the first element from the head of a list.
  - Syntax: **LPOP key [count]**
  - Hint count default **1**

```
LPOP mylist
LRANGE mylist 0 -1
1) "world"
2) "end"
```

- **RPOP**: Remove and return the last element from the tail of a list.
  - Syntax: **RPOP key [count]**
  - Hint count default **1**

```

```

```
RPOP mylist
LRANGE mylist 0 -1
Output:
1) "world"
```

- **LRANGE**: Get a range of elements from a list.
  - Syntax: **LLEN key**

```
LRANGE mylist 0 -1
```

- **LLEN**: Get the length of a list.

```
LLEN mylist
Output:
(integer) 1
```

- **LPOS**: Returns the index of the first occurrence of a specified element.
  - Syntax: **LPOS key element [RANK rank] [COUNT num-matches] [MAXLEN len]**

```
LPUSH mylist "hello"
LPUSH mylist "world"
LPUSH mylist "hello"
LPOS mylist "hello"
Output:
(integer) 0
```

- **LINDEX**: Get an element by its index in the list.
  - Syntax: **LINDEX key index**

```
LINDEX mylist 1
Output:
"world"
```

- **LREM**: Removes the first `count` occurrences of the specified element.
  - Syntax: **LREM key count element**

```
LREM mylist 1 "hello"
Output:
```

```
1) "world"
2) "hello"
```

- **LTRIM**: Trims the list to contain only the elements within the specified range.
    - Syntax: **LTRIM key start stop**

```
RPUSH mylist "Mustafa"
LTRIM mylist 1 -1
LRANGE mylist 0 -1
Output:
1) "hello"
2) "Mustafa"
```

4. **Sets**

Sets are unordered collections of **unique strings**.

- **SADD**: Add one or more members to a set.

```
SADD myset "Hello"
SADD myset "World"
```

- **SMEMBERS**: Get all members of a set.

```
SMEMBERS myset
```

- **SISMEMBER**: Check if a member exists in a set.

```
SISMEMBER myset "Hello"
```

- **SREM**: Remove one or more members from a set.

```
SREM myset "Hello"
SMEMBERS myset
OUTPUT => 1) "World"
```

- **SCARD**: Get the number of members in a set.

```
SCARD myset
OUTPUT => (integer) 2
```

- **SRANDMEMBER**: Get one or more random members from a set.

```
SRANDMEMBER myset 2
OUTPUT=>
1) "Mustafa"
2) "slam"
```

- **SUNION**: Perform a union of multiple sets.

```
SADD set1 "A"
SADD set1 "B"
SADD set2 "B"
SADD set2 "C"
SUNION set1 set2
1) "A"
2) "B"
3) "C"
```

- **SINTER**: Perform an intersection of multiple sets.

```
SINTER set1 set2
OUTPUT=>
1) "B"
```

- **SDIFF**: Perform a difference between sets.

```
SDIFF set1 set2
OUTPUT=>
1) "A"
```

- **SMOVE**: Move a member from one set to another.

```
SMOVE source destination member
SMOVE set1 set2 "A"
OUTPUT=>
(integer) 1
```

- **SSCAN**: Iterate over members of a set.

```
SSCAN key cursor [MATCH pattern] [COUNT count]
SSCAN myset 0 MATCH "H*"
OUTPUT=>
1) "A"
```

5. **Sorted Sets**

Sorted sets are sets where each member is associated with a score, allowing for range queries.

- **ZADD**: Add one or more members to a sorted set.

```
ZADD myzset 1 "one"
ZADD myzset 2 "two"
```

- **ZRANGE**: Get a range of members from a sorted set.

```
ZRANGE myzset 0 \-1 WITHSCORES
```

- **ZREM**: Remove one or more members from a sorted set.

```
ZREM myzset "one"
```

6. **HyperLogLog**

HyperLogLog is a probabilistic data structure used to estimate the cardinality of a set.

- **PFADD**: Add an element to a HyperLogLog.

```
PFADD myloglog "user1"
```

- **PFCOUNT**: Estimate the number of unique elements in a HyperLogLog.

```
PFCOUNT myloglog
```

7. **Geospatial**

Redis supports geospatial indexing, allowing you to store and query locations.

- **GEOADD**: Add a location to a geospatial index.

```
GEOADD cities 13.361389 38.115556 "Palermo"
```

- **GEODIST**: Get the distance between two locations.

```
GEODIST cities "Palermo" "Catania" km
```

8. **Transactions**

Redis supports transactions, allowing you to execute multiple commands atomically.

- **MULTI**: Start a transaction.

```
SET key1 "value1"
SET key2 "value2"
EXEC
```

- **EXEC**: Execute all commands in a transaction.

```
EXEC
```

9. **Pub/Sub**

Redis supports Publish/Subscribe messaging patterns.

- **PUBLISH**: Publish a message to a channel.

```
PUBLISH mychannel "Hello, World!"
```

- **SUBSCRIBE**: Subscribe to one or more channels.

```
SUBSCRIBE mychannel
```

10. **Scripting**

Redis supports Lua scripting for complex operations.

- **EVAL**: Execute a Lua script.

```
EVAL "return redis.call('GET', KEYS\[1\])" 1 mykey
```

11. **Server Management**

Redis provides commands for managing the server.

- **INFO**: Get information and statistics about the server.

```
INFO
```

- **FLUSHALL**: Remove all keys from all databases.

```
FLUSHALL
```

- **SAVE**: Synchronously save the dataset to disk.

```
SAVE
```

Conclusion

Redis is a powerful in-memory data store that supports a wide range of data structures and operations. Its speed and versatility make it an excellent choice for caching, session storage, real-time analytics, and more.

For more detailed information, refer to the Redis documentation.