REDIS

tutorialspoint

SIMPLYEASYLEARNING

About the Tutorial

Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server, since the keys can contain strings, hashes, lists, sets and sorted sets. Redis is written in C.

This tutorial provides good understanding on Redis concepts, needed to create and deploy a highly scalable and performance-oriented system.

Audience

This tutorial is designed for Software Professionals who are willing to learn Redis in simple and easy steps. After completing this tutorial, you will be at an intermediate level of expertise from where you can take yourself to a higher level of expertise.

Prerequisites

Before proceeding with this tutorial, you should have basic knowledge of Data Structures.

Disclaimer & Copyright

Table of Contents

Redis - Basics

1. Redis – Overview

Redis is an open source, advanced key-value store and an apt solution for building high-performance, scalable web applications.

Redis has three main peculiarities that sets it apart.

- Redis holds its database entirely in the memory, using the disk only for persistence.
- Redis has a relatively rich set of data types when compared to many key-value data stores.
- Redis can replicate data to any number of slaves.

Redis Advantages

Following are certain advantages of Redis.

- **Exceptionally fast**: Redis is very fast and can perform about 110000 SETs per second, about 81000 GETs per second.
- **Supports rich data types**: Redis natively supports most of the datatypes that developers already know such as list, set, sorted set, and hashes. This makes it easy to solve a variety of problems as we know which problem can be handled better by which data type.
- **Operations are atomic**: All Redis operations are atomic, which ensures that if two clients concurrently access, Redis server will receive the updated value.
- **Multi-utility tool**: Redis is a multi-utility tool and can be used in a number of use cases such as caching, messaging-queues (Redis natively supports Publish/Subscribe), any short-lived data in your application, such as web application sessions, web page hit counts, etc.

Redis Versus Other Key-value Stores

- Redis is a different evolution path in the key-value DBs, where values can contain more complex data types, with atomic operations defined on those data types.
- Redis is an in-memory database but persistent on disk database, hence it represents a different trade off where very high write and read speed is achieved with the limitation of data sets that can't be larger than the memory.

  Another advantage of in-memory databases is that the memory representation of complex data structures is much simpler to manipulate compared to the same data structure on disk. Thus, Redis can do a lot with little internal complexity.

2. Redis – Environment

In this chapter, you will learn about the environmental setup for Redis.

Install Redis on Ubuntu

To install Redis on Ubuntu, go to the terminal and type the following commands:

```
$sudo apt-get update
$sudo apt-get install redis-server
```

This will install Redis on your machine.

Start Redis

```
$redis-server
```

Check If Redis is Working

```
$redis-cli
```

This will open a redis prompt.

```
redis 127.0.0.1:6379>
```

In the above prompt, **127.0.0.1** is your machine's IP address and **6379** is the port on which Redis server is running. Now type the following **PING** command.

```
redis 127.0.0.1:6379> ping
PONG
```

This shows that Redis is successfully installed on your machine.

Install Redis Desktop Manager on Ubuntu

To install Redis desktop manager on Ubuntu, just download the package from http://redisdesktop.com/download

Open the downloaded package and install it.

Redis desktop manager will give you UI to manage your Redis keys and data.

3. Redis – Configuration

In Redis, there is a configuration file (redis.conf) available at the root directory of Redis. Although you can get and set all Redis configurations by Redis **CONFIG** command.

Syntax

Following is the basic syntax of Redis **CONFIG** command.

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

Example

```
redis 127.0.0.1:6379> CONFIG GET loglevel


1) "loglevel"
2) "notice"
```

To get all configuration settings, use ***** in place of CONFIG_SETTING_NAME

Example

```
redis 127.0.0.1:6379> CONFIG GET *


  1) "dbfilename"
  2) "dump.rdb"
  3) "requirepass"
  4) ""
  5) "masterauth"
  6) ""
  7) "unixsocket"
  8) ""
  9) "logfile"
 10) ""
 11) "pidfile"
 12) "/var/run/redis.pid"
 13) "maxmemory"
 14) "0"
```

```
15) "maxmemory-samples"
16) "3"
17) "timeout"
18) "0"
19) "tcp-keepalive"
20) "0"
21) "auto-aof-rewrite-percentage"
22) "100"
23) "auto-aof-rewrite-min-size"
24) "67108864"
25) "hash-max-ziplist-entries"
26) "512"
27) "hash-max-ziplist-value"
28) "64"
29) "list-max-ziplist-entries"
30) "512"
31) "list-max-ziplist-value"
32) "64"
33) "set-max-intset-entries"
34) "512"
35) "zset-max-ziplist-entries"
36) "128"
37) "zset-max-ziplist-value"
38) "64"
39) "hll-sparse-max-bytes"
40) "3000"
41) "lua-time-limit"
42) "5000"
43) "slowlog-log-slower-than"
44) "10000"
45) "latency-monitor-threshold"
46) "0"
47) "slowlog-max-len"
48) "128"
49) "port"
50) "6379"
```

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

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

Edit Configuration

To update configuration, you can edit **redis.conf** file directly or you can update configurations via **CONFIG set** command.

Syntax

Following is the basic syntax of **CONFIG SET** command.

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

Example

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

1) "loglevel"
2) "notice"
```

4. Redis – Data Types

Redis supports 5 types of data types.

Strings

Redis string is a sequence of bytes. Strings in Redis are binary safe, meaning they have a known length not determined by any special terminating characters. Thus, you can store anything up to 512 megabytes in one string.

Example

```
redis 127.0.0.1:6379> SET name "tutorialspoint"

OK

redis 127.0.0.1:6379> GET name

"tutorialspoint"
```

In the above example, **SET** and **GET** are Redis commands, **name** is the key used in Redis and **tutorialspoint** is the string value that is stored in Redis.

Note: A string value can be at max 512 megabytes in length.

Hashes

A Redis hash is a collection of key value pairs. Redis Hashes are maps between string fields and string values. Hence, they are used to represent objects.

Example

```
redis 127.0.0.1:6379> HMSET user:1 username tutorialspoint password
tutorialspoint points 200

OK

redis 127.0.0.1:6379> HGETALL user:1


1) "username"

2) "tutorialspoint"

3) "password"

4) "tutorialspoint"

5) "points"

6) "200"
```

In the above example, hash data type is used to store the user's object which contains basic information of the user. Here **HMSET, HGETALL** are commands for Redis, while **user:1** is the key.

Every hash can store up to 232 - 1 field-value pairs (more than 4 billion).

Lists

Redis Lists are simply lists of strings, sorted by insertion order. You can add elements to a Redis List on the head or on the tail.

Example

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

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

The max length of a list is 232 - 1 elements (4294967295, more than 4 billion of elements per list).

Sets

Redis Sets are an unordered collection of strings. In Redis, you can add, remove, and test for the existence of members in O(1) time complexity.

Example

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

```
(integer) 0
redis 127.0.0.1:6379> smembers tutoriallist

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

Note: In the above example, **rabitmq** is added twice, however due to unique property of the set, it is added only once.

The max number of members in a set is 232 - 1 (4294967295, more than 4 billion of members per set).

Sorted Sets

Redis Sorted Sets are similar to Redis Sets, non-repeating collections of Strings. The difference is, every member of a Sorted Set is associated with a score, that is used in order to take the sorted set ordered, from the smallest to the greatest score. While members are unique, the scores may be repeated.

Example

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

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

Redis – Commands

5. Redis – Commands

Redis commands are used to perform some operations on Redis server.

To run commands on Redis server, you need a Redis client. Redis client is available in Redis package, which we have installed earlier.

Syntax

Following is the basic syntax of Redis client.

```
$redis-cli
```

Example

Following example explains how we can start Redis client.

To start Redis client, open the terminal and type the command **redis-cli**. This will connect to your local server and now you can run any command.

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


PONG
```

In the above example, we connect to Redis server running on the local machine and execute a command **PING**, that checks whether the server is running or not.

Run Commands on the Remote Server

To run commands on Redis remote server, you need to connect to the server by the same client **redis-cli**

Syntax

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

Example

Following example shows how to connect to Redis remote server, running on host 127.0.0.1, port 6379 and has password mypass.

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


PONG
```

6. Redis – Keys

Redis keys commands are used for managing keys in Redis. Following is the syntax for using redis keys commands.

Syntax

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

Example

```
redis 127.0.0.1:6379> SET tutorialspoint redis

OK

redis 127.0.0.1:6379> DEL tutorialspoint

(integer) 1
```

In the above example, **DEL** is the command, while **tutorialspoint** is the key. If the key is deleted, then the output of the command will be (integer) 1, otherwise it will be (integer) 0.

Redis Keys Commands

Following table lists some basic commands related to keys.

Sr. No.	Command & Description
1	**DEL key** This command deletes the key, if it exists
2	**DUMP key** This command returns a serialized version of the value stored at the specified key
3	**EXISTS key** This command checks whether the key exists or not
4	**EXPIRE key** seconds Sets the expiry of the key after the specified time
5	**EXPIREAT key timestamp** Sets the expiry of the key after the specified time. Here time is in Unix timestamp format

6	**PEXPIRE key milliseconds** Set the expiry of key in milliseconds
7	**PEXPIREAT key milliseconds-timestamp** Sets the expiry of the key in Unix timestamp specified as milliseconds
8	**KEYS pattern** Finds all keys matching the specified pattern
9	**MOVE key db** Moves a key to another database
10	**PERSIST key** Removes the expiration from the key
11	**PTTL key** Gets the remaining time in keys expiry in milliseconds
12	**TTL key** Gets the remaining time in keys expiry
13	**RANDOMKEY** Returns a random key from Redis
14	**RENAME key newkey** Changes the key name
15	**RENAMENX key newkey** Renames the key, if a new key doesn't exist
16	**TYPE key** Returns the data type of the value stored in the key

Keys Del Command

Redis **DEL** command is used to delete the existing key in Redis.

Return Value

Number of keys that were removed.

Syntax

Following is the basic syntax of Redis **DEL** command.

```
redis 127.0.0.1:6379> DEL KEY_NAME
```

Example

First, create a key in Redis and set some value in it.

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

Now, delete the previously created key.

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

Keys Dump Command

Redis **DUMP** command is used to get a serialized version of data stored at specified key in Redis.

Return Value

Serialized value (String)

Syntax

Following is the basic syntax of Redis **DUMP** command.

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

Example

First, create a key in Redis and set some value in it.

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

Now, create dump of the previously created key.

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

Keys Exists Command

Redis **EXISTS** command is used to check whether the key exists in Redis or not.

Return Value

Integer value

- 1, if the key exists.
- 0, if the key does not exist.

Syntax

Following is the basic syntax of Redis **EXISTS** command.

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

Example

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

Now, create a key with the name tutorialspoint-new-key and check for its existence.

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

Keys Expire Command

Redis **Expire** command is used to set the expiry of a key. After the expiry time, the key will not be available in Redis.

Return Value

Integer value 1 or 0

- 1, if timeout is set for the key.
- 0, if the key does not exist or timeout could not be set.

Syntax

Following is the basic syntax of Redis **Expire** command.

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

Example

First, create a key in Redis and set some value in it.

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

Now, set timeout of the previously created key.

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

In the above example, 1 minute (or 60 seconds) time is set for the key tutorialspoint. After 1 minute, the key will expire automatically.

Keys Expireat Command

Redis **Expireat** command is used to set the expiry of key in Unix timestamp format. After the expiry time, the key will not be available in Redis.

Return Value

Integer value 1 or 0

- 1, if timeout is set for key.
- 0, if key does not exists or timeout could not set.

Syntax

Following is the basic syntax of Redis **Expireat** command.

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

Example

First, create a key in Redis and set some value in it.

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

Now, set timeout of the previously created key.

```
redis 127.0.0.1:6379> EXPIREAT tutorialspoint 1293840000
(integer) 1
EXISTS tutorialspoint
(integer) 0
```

Keys Pexpire Command

Redis **Pexpire** command is used to set the expiry of the key in milliseconds. After the expiry time, the key will not be available in Redis.

Return Value

Integer value 1 or 0

- 1, if the timeout is set for the key.
- 0, if the key does not exist or timeout could not be set.

Syntax

Following is the basic syntax of Redis **Expire** command.

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

Example

First, create a key in Redis and set some value in it.

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

Now, set timeout of the previously created key.

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

In the above example, 5 seconds time is set for the key tutorialspoint. After 5 seconds, the key will expire automatically.

Keys Pexpireat Command

Redis **Pexpireat** command is used to set the expiry of the key in Unix timestamp specified in milliseconds. After the expiry time, the key will not be available in Redis.

Return Value

Integer value 1 or 0

- 1, if timeout is set for the key.
- 0, if the key does not exist or timeout could not be set.

Syntax

Following is the basic syntax of Redis **Pexpireat** command.

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

Example

First, create a key in Redis and set some value in it.

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

```
OK
```

Now, set timeout of the previously created key.

```
redis 127.0.0.1:6379> PEXPIREAT tutorialspoint 1555555555005
(integer) 1
```