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.

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 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
```

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

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

```
CONFIG SET loglevel "notice"
```

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.

```
lpush kolaparthi redis
```

```
lpush srini redis
```

```
lpush rajus redis
```

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.

```
sadd Wiillis redis
```

```
sadd Abc redis
```

```
sadd Abc redis
```

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.

```
SET name "kolaparthi"
```

```
GET name
```

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
```

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 tut redis
OK
```

Now, set timeout of the previously created key.

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

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 tut redis
OK
```

Now, set timeout of the previously created key.

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