pyredis Documentation

Release 0.0.5

Stephan Schultchen

Oct 09, 2021

Contents

Contents:

CHAPTER 1

Introduction

Redis Client implementation for Python. The Client only supports Python 3 for the moment. If there is enough interest, i will make it work with Python 2.

Currently implemented Features:

- Base Redis Client
- Publish Subscribe Client
- Sentinel Client
- Connection Pool
- Sentinel Backed Connection Pool
- Client & Pool for Redis Cluster
- Bulk Mode (Not supported with Redis Cluster)
- Client & Pool with Static Hash Cluster (Supports Bulk Mode)
- Sentinel Backed Client & Pool with Static Hash Cluster (Supports Bulk Mode)

1.1 Installing

pyredis can be installed via pip as follows:

```
pip install python_redis
```

1.2 Author

Stephan Schultchen <stephan.schultchen@gmail.com>

1.3 License

Unless stated otherwise on-file pyredis uses the MIT license, check LICENSE file.

1.4 Contributing

If you'd like to contribute, fork the project, make a patch and send a pull request.

CHAPTER 2

Usage Example

2.1 Simple Client Usage

```
from pyredis import Client

client = Client(host="localhost")
client.ping()
```

2.2 Bulk Mode

Bulk Mode can be used to import large amounts of data in a short time. With bulk mode enabled sending requests and fetching results is separated from each other. Which will save many network round trips, improving query performance.

All executed commands will return None.

There is a threshold, which defaults to 5000 requests, after which the results of the previous requests are fetched into a list. If you are not interested in the results, this can be disabled by calling bulk_start with the parameter keep_results=False.

Fetching results, when the threshold is reached is a transparent operation. The client will only notice that the execution of the the request triggering the threshold will take a little longer.

Calling bulk_stop() will fetch all remaining results, and return a list with fetched results. This list can also contain exceptions from failed commands.

```
from pyredis import Client

client = Client(host="localhost")
client.bulk_start()
client.set('key1', 'value1')
client.set('key2', 'value2')
```

(continues on next page)

(continued from previous page)

```
client.set('key3', 'value3')
client.bulk_stop()
[b'OK', b'OK', b'OK']


from pyredis import HashClient

client = Client(buckets=[('host1', 6379), ('host2', 6379), ('host3', 6379)])
client.bulk_start()
client.set('key1', 'value1')
client.set('key2', 'value2')
client.set('key3', 'value3')
client.bulk_stop()
[b'OK', b'OK', b'OK']
```

2.3 Using a Connection Pool

```
from pyredis import Pool

pool = Pool(host="localhost")
client = pool.acquire()
client.ping()
b'PONG'
pool.release(client)
```

2.4 Using a Cluster Connection Pool

```
from pyredis import ClusterPool

pool = ClusterPool(seeds=[('seed1', 6379), ('seed2', 6379), ('seed3', 6379)])
client = pool.acquire()
client.ping(shard_key='test')
b'PONG'
pool.release(client)
```

2.5 Using a Hash Connection Pool

```
from pyredis import HashPool

pool = HashPool(buckets=[('host1', 6379), ('host2', 6379), ('host3', 6379)])
client = pool.acquire()
client.ping(shard_key='test')
b'PONG'
pool.release(client)
```

2.6 Using a Sentinel backed Connection Hash Pool

```
from pyredis import SentinelHashPool

pool = SentinelHashPool(sentinels=[('sentinel1', 26379), ('sentinel2', 26379), (
↪'sentinel3', 26379)], buckets=['bucket1', 'bucket2', 'bucket3'])
client = pool.acquire()
client.ping(shard_key='test')
b'PONG'
pool.release(client)
```

2.7 Using a Sentinel backed Connection Pool

```
from pyredis import SentinelPool

pool = SentinelPool(sentinels=[('sentinel1', 26379), ('sentinel2', 26379), ('sentinel3
↪', 26379)], name=pool_name)
client = pool.acquire()
client.ping()
b'PONG'
pool.release(client)
```

2.8 Getting Pool by URL

```
from pyredis import get_by_url
pool1 = get_by_url('redis://localhost?password=topsecret')
pool1 = get_by_url('redis://localhost:6379?db=0&password=topsecret')
sentinel = get_by_url('sentinel://seed1:6379,seed2,seed3:4711?name=pool_name&db=0&
↪password=topsecret')
cluster = get_by_url('redis://seed1:6379,seed2:4711,seed3?db=0')
```

2.9 Getting PubSubClient by URL

```
from pyredis import get_by_url
# it is not save to share this client between threads
pubsub = get_by_url('pubsub://localhost?password=topsecret')
```

2.10 Publish Subscribe

```
from pyredis import Client, PubSubClient

client = Client(host='localhost')
subscribe = PubSubClient(host='localhost')

subscribe.subscribe('/blub')
```

(continues on next page)

(continued from previous page)

```
subscribe.get()
[b'subscribe', b'/blub', 1]

client.publish('/blub', 'test')
1

subscribe.get()
[b'message', b'/blub', b'test']
```

CHAPTER 3

API Documentation

3.1 Connection

class pyredis.connection.**Connection**(*host=None*, *port=6379*, *unix_sock=None*, *database=None*, *password=None*, *encoding=None*, *conn_timeout=2*, *read_only=False*, *read_timeout=2*, *sentinel=False*, *username=None*)

Low level client for talking to a Redis Server.

This class is should not be used directly to talk to a Redis server, unless you know what you are doing. In most cases it should be sufficient to use one of the Client classes, or one of the Connection Pools.

Parameters

- **host** (*str*) – Host IP or Name to connect, can only be set when unix_sock is None.
- **port** (*int*) – Port to connect, only used when host is also set.
- **unix_sock** (*str*) – Unix Socket to connect, can only be set when host is None.
- **database** (*int*) – Select which db should be used for this connection, defaults to None, so we use the default database 0
- **password** (*str*) – Password used for authentication. If None, no authentication is done
- **encoding** (*str*) – Convert result strings with this encoding. If None, no encoding is done.
- **conn_timeout** (*float*) – Connect Timeout.
- **read_timeout** (*float*) – Read Timeout.
- **sentinel** (*bool*) – If True, authentication and database selection is skipped.
- **username** (*str*) – Username used for acl scl authentication. If not set, fall back use legacy auth.

close()

Close Client Connection.

This closes the underlying socket, and mark the connection as closed.

Returns None

read(*close_on_timeout=True*, *raise_on_result_err=True*)
Read result from the socket.

Parameters

- **close_on_timeout** (*book*) – Close the connection after a read timeout
- **raise_on_result_err** (*bool*) – Raise exception on protocol errors

Returns result, exception

write(**args*)
Write commands to socket.

Parameters **args** (*str, int, float*) – Accepts a variable number of arguments

Returns None

3.2 Client

3.2.1 Client

class pyredis.**Client**(***kwargs*)
Base Client for Talking to Redis.

Inherits the following Command classes:

- commands.Connection,
- commands.Hash,
- commands.HyperLogLog,
- commands.Key,
- commands.List,
- commands.Publish,
- commands.Scripting,
- commands.Set,
- commands.SSet,
- commands.String,
- commands.Transaction

Parameters **kwargs** – pyredis.Client takes the same arguments as pyredis.connection.Connection.

bulk
True if bulk mode is enabled.

Returns bool

bulk_start(*bulk_size=5000*, *keep_results=True*)
Enable bulk mode

Put the client into bulk mode. Instead of executing a command & waiting for the reply, all commands are send to Redis without fetching the result. The results get fetched whenever $bulk_size commands have been executed, which will also resets the counter, or of bulk_stop() is called.

Parameters

- **bulk_size** (*int*) – Number of commands to execute, before fetching results.
- **keep_results** (*bool*) – If True, keep the results. The Results will be returned when calling bulk_stop.

Returns None

bulk_stop()
Stop bulk mode.

All outstanding results from previous commands get fetched. If bulk_start was called with keep_results=True, return a list with all results from the executed commands in order. The list of results can also contain Exceptions, hat you should check for.

Returns None, list

close()
Close client.

Returns None

closed
Check if client is closed.

Returns bool

execute(**args*)
Execute arbitrary redis command.

Parameters **args** (*list, int, float*) –

Returns result, exception

3.2.2 ClusterClient

class pyredis.**ClusterClient**(*seeds=None*, *database=0*, *password=None*, *encoding=None*, *slave_ok=False*, *conn_timeout=2*, *read_timeout=2*, *cluster_map=None*, *username=None*)
Base Client for Talking to Redis Cluster.

Inherits the following Commmand classes:

- commands.Connection,
- commands.Hash,
- commands.HyperLogLog,
- commands.Key,
- commands.List,
- commands.Scripting,
- commands.Set,
- commands.SSet,
- commands.String,

- commands.Transaction

Parameters

- **seeds** (*list*) – Accepts a list of seed nodes in this form: [('seed1', 6379), ('seed2', 6379), ('seed3', 6379)]
- **slave_ok** (*bool*) – Set to True if this Client should use slave nodes.
- **database** (*int*) – Select which db should be used for this connection
- **password** (*str*) – Password used for authentication. If None, no authentication is done
- **encoding** (*str*) – Convert result strings with this encoding. If None, no encoding is done.
- **conn_timeout** (*float*) – Connect Timeout.
- **read_timeout** (*float*) – Read Timeout.
- **username** (*str*) – Username used for acl scl authentication. If not set, fall back use legacy auth.

execute(**args*, *shard_key=None*, *sock=None*, *asking=False*, *retries=3*)

Execute arbitrary redis command.

Parameters

- **args** (*list, int, float*) –
- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.
- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

Returns result, exception

3.2.3 HashClient

class pyredis.**HashClient**(*buckets*, *database=None*, *password=None*, *encoding=None*, *conn_timeout=2*, *read_timeout=2*, *username=None*)

Client for Talking to Static Hashed Redis Cluster.

The Client will calculate a crc16 hash using the shard_key, which is be default the first Key in case the command supports multiple keys. If the Key is using the TAG annotation "bla{tag}blarg", then only the tag portion is used, in this case "tag". The key space is split into 16384 buckets, so in theory you could provide a list with 16384 ('host', port) pairs to the "buckets" parameter. If you have less then 16384 ('host', port) pairs, the client will try to distribute the key spaces evenly between available pairs.

— Warning — Since this is static hashing, the order of pairs has to match on each client you use! Also changing the number of pairs will change the mapping between buckets and pairs, rendering your data inaccessible!

Inherits the following Commmand classes:

- commands.Connection,
- commands.Hash,
- commands.HyperLogLog,
- commands.Key,
- commands.List,

- commands.Publish,
- commands.Scripting,
- commands.Set,
- commands.SSet,
- commands.String,
- commands.Transaction

bulk
True if bulk mode is enabled.

Returns bool

bulk_start(*bulk_size=5000*, *keep_results=True*)
Enable bulk mode

Put the client into bulk mode. Instead of executing a command & waiting for the reply, all commands are send to Redis without fetching the result. The results get fetched whenever $bulk_size commands have been executed, which will also resets the counter, or of bulk_stop() is called.

Parameters

- **bulk_size** (*int*) – Number of commands to execute, before fetching results.
- **keep_results** (*bool*) – If True, keep the results. The Results will be returned when calling bulk_stop.

Returns None

bulk_stop()
Stop bulk mode.

All outstanding results from previous commands get fetched. If bulk_start was called with keep_results=True, return a list with all results from the executed commands in order. The list of results can also contain Exceptions, hat you should check for.

Returns None, list

close()
Close client.

Returns None

closed
Check if client is closed.

Returns bool

execute(**args*, *shard_key=None*, *sock=None*)
Execute arbitrary redis command.

Parameters

- **args** (*list, int, float*) –
- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.
- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

Returns result, exception

3.2.4 PubSubClient

class pyredis.**PubSubClient**(***kwargs*)
Pub/Sub Client.

Subscribe part of the Redis Pub/Sub System.

Parameters **kwargs** – pyredis.PubSubClient takes the same arguments as pyredis.connection.Connection.

close()
Close Client

Returns None

closed
Check if Client is closed.

Returns bool

get()
Fetch published item from Redis.

Returns list

3.2.5 SentinelClient

class pyredis.**SentinelClient**(*sentinels*, *password=None*, *username=None*)
Redis Sentinel Client.

Parameters

- **sentinels** (*list*) – Accepts a list of sentinels in this form: [('sentinel1', 26379), ('sentinel2', 26379), ('sentinel3', 26379)]
- **password** (*str*) – Password used for authentication of Sentinel instance itself. If None, no authentication is done. Only available starting with Redis 5.0.1.
- **username** (*str*) – Username used for acl scl authentication. If not set, fall back use legacy auth.

close()
Close Connection.

Returns None

execute(**args*)
Execute sentinel command.

Parameters **args** (*string, int, float*) –

Returns result, exception

get_master(*name*)
Get Master Info.

Return dictionary with master details.

Parameters **name** (*str*) – Name of Redis service

Returns dict

get_masters()
Get list of masters.

Returns list of dicts

get_slaves(*name*)
Get slaves.

Return a list of dictionaries, with slave details.

Parameters **name** (*str*) – Name of Redis service

Returns

next_sentinel()
Switch to the Next Sentinel.

Returns None

sentinels
Return configured sentinels.

Returns deque

3.3 Pool

3.3.1 BasePool

class pyredis.pool.**BasePool**(*database=0*, *password=None*, *encoding=None*, *conn_timeout=2*, *read_timeout=2*, *pool_size=16*, *lock=<unlocked _thread.lock object>*, *username=None*)
Base Class for all other pools.

All other pools inherit from this base class. This class itself, cannot be used directly.

Parameters

- **database** (*int*) – Select which db should be used for this pool
- **password** (*str*) – Password used for authentication. If None, no authentication is done
- **encoding** (*str*) – Convert result strings with this encoding. If None, no encoding is done.
- **conn_timeout** (*float*) – Connect Timeout.
- **read_timeout** (*float*) – Read Timeout.
- **pool_size** (*int*) – Upper limit of connections this pool can handle.
- **lock** (*_lock object, defaults to threading.Lock*) – Class implementing a Lock.
- **username** (*str*) – Username used for acl scl authentication. If not set, fall back use legacy auth.

acquire()
Acquire a client connection from the pool.

Returns redis.Client, exception

conn_timeout
Return configured connection timeout

Returns float

database
Return configured database.

Returns int

encoding
Return configured encoding

Returns str, None

password
Return configured password for this pool.

Returns str, None

pool_size
Return, or adjust the current pool size.

shrinking is implemented via closing unused connections. if there not enough unused connections to fulfil the shrink request, connections returned via pool.release are closed.

Returns int, None

read_timeout
Return configured read timeout

Returns float

release(*conn*)
Return a client connection to the pool.

Parameters **conn** – redis.Client instance, managed by this pool.

Returns None

3.3.2 ClusterPool

class pyredis.**ClusterPool**(*seeds*, *slave_ok=False*, *password=None*, *username=None*, ***kwargs*)
Redis Cluster Pool.

Inherits all the arguments, methods and attributes from BasePool.

Parameters

- **seeds** – Accepts a list of seed nodes in this form: [('host1', 6379), ('host2', 6379), ('host3', 6379)]
- **slave_ok** (*bool*) – Defaults to False. If True, this pool will return connections to slave instances.
- **retries** (*int*) – In case there is a chunk move ongoing, while executing a command, how many times should we try to find the right node, before giving up.

execute(**args*, ***kwargs*)
Execute arbitrary redis command.

Parameters **args** (*list, int, float*) –

Returns result, exception

slave_ok
True if this pool will return slave connections

Returns bool

3.3.3 HashPool

class pyredis.**HashPool**(*buckets*, ***kwargs*)
Pool for straight connections to Redis

Inherits all the arguments, methods and attributes from BasePool.

The Client will calculate a crc16 hash using the shard_key, which is be default the first Key in case the command supports multiple keys. If the Key is using the TAG annotation "bla{tag}blarg", then only the tag portion is used, in this case "tag". The key space is split into 16384 buckets, so in theory you could provide a list with 16384 ('host', port) pairs to the "buckets" parameter. If you have less then 16384 ('host', port) pairs, the client will try to distribute the key spaces evenly between available pairs.

— Warning — Since this is static hashing, the the order of pairs has to match on each client you use! Also changing the number of pairs will change the mapping between buckets and pairs, rendering your data inaccessible!

Parameters **buckets** – list of ('host', port) pairs, where each pair represents a bucket example: [('localhost', 7001), ('localhost', 7002), ('localhost', 7003)]

buckets
Return configured buckets.

Returns list

execute(**args*, ***kwargs*)
Execute arbitrary redis command.

Parameters **args** (*list, int, float*) –

Returns result, exception

3.3.4 Pool

class pyredis.**Pool**(*host=None*, *port=6379*, *unix_sock=None*, ***kwargs*)
Pool for straight connections to Redis

Inherits all the arguments, methods and attributes from BasePool.

Parameters

- **host** (*str*) – Host IP or Name to connect, can only be set when unix_sock is None.
- **port** (*int*) – Port to connect, only used when host is also set.
- **unix_sock** (*str*) – Unix Socket to connect, can only be set when host is None.

execute(**args*)
Execute arbitrary redis command.

Parameters **args** (*list, int, float*) –

Returns result, exception

host
Return configured host.

Returns str, None

port
Return configured port.

Returns int

unix_sock
Return configured Unix socket.

Returns str, None

3.3.5 SentinelHashPool

class pyredis.**SentinelPool**(*sentinels*, *name*, *slave_ok=False*, *retries=3*, *sentinel_password=None*, *sentinel_username=None*, ***kwargs*)

Sentinel backed Pool.

Inherits all the arguments, methods and attributes from BasePool.

Parameters

- **sentinels** (*list*) – Accepts a list of sentinels in this form: [('sentinel1', 26379), ('sentinel2', 26379), ('sentinel3', 26379)]
- **name** (*str*) – Name of the cluster managed by sentinel, that this pool should manage.
- **slave_ok** (*bool*) – Defaults to False. If True, this pool will return connections to slave instances.
- **retries** (*int*) – In case a sentinel delivers stale data, how many other sentinels should be tried.
- **sentinel_password** (*str*) – Password used for authentication of Sentinel instance itself. If None, no authentication is done. Only available starting with Redis 5.0.1.
- **sentinel_username** (*str*) – Username used for acl style authentication of Sentinel instance itself. If None, no authentication is done. Only available starting with Redis 5.0.1.

execute(**args*)
Execute arbitrary redis command.

Parameters **args** (*list, int, float*) –

Returns result, exception

name
Name of the configured Sentinel managed cluster.

Returns str

retries
Number of retries in case of stale sentinel.

Returns int

sentinels
Deque with configured sentinels.

Returns deque

slave_ok
True if this pool return slave connections

Returns bool

3.3.6 SentinelPool

class pyredis.**SentinelPool**(*sentinels*, *name*, *slave_ok=False*, *retries=3*, *sentinel_password=None*, *sentinel_username=None*, ***kwargs*)

Sentinel backed Pool.

Inherits all the arguments, methods and attributes from BasePool.

Parameters

- **sentinels** (*list*) – Accepts a list of sentinels in this form: [('sentinel1', 26379), ('sentinel2', 26379), ('sentinel3', 26379)]
- **name** (*str*) – Name of the cluster managed by sentinel, that this pool should manage.
- **slave_ok** (*bool*) – Defaults to False. If True, this pool will return connections to slave instances.
- **retries** (*int*) – In case a sentinel delivers stale data, how many other sentinels should be tried.
- **sentinel_password** (*str*) – Password used for authentication of Sentinel instance itself. If None, no authentication is done. Only available starting with Redis 5.0.1.
- **sentinel_username** (*str*) – Username used for acl style authentication of Sentinel instance itself. If None, no authentication is done. Only available starting with Redis 5.0.1.

execute(**args*)

Execute arbitrary redis command.

Parameters **args** (*list, int, float*) –

Returns result, exception

name

Name of the configured Sentinel managed cluster.

Returns str

retries

Number of retries in case of stale sentinel.

Returns int

sentinels

Deque with configured sentinels.

Returns deque

slave_ok

True if this pool return slave connections

Returns bool

3.4 Commands

3.4.1 Connection

class pyredis.commands.**Connection**

echo(**args*, *shard_key=None*, *sock=None*)
Execute ECHO Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

ping(*shard_key=None*, *sock=None*)
Execute PING Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result,exception

3.4.2 Hash

class pyredis.commands.**Hash**

hdel(**args*)
Execute HDEL Command, consult Redis documentation for details.

Returns result, exception

hexists(**args*)
Execute HEXISTS Command, consult Redis documentation for details.

Returns result, exception

hget(**args*)
Execute HGET Command, consult Redis documentation for details.

Returns result, exception

hgetall(**args*)
Execute HGETALL Command, consult Redis documentation for details.

Returns result, exception

hincrby(**args*)
Execute HINCRBY Command, consult Redis documentation for details.

Returns result, exception

hincrbyfloat(**args*)
Execute HINCRBYFLOAT Command, consult Redis documentation for details.

Returns result, exception

hkeys(**args*)
Execute HKEYS Command, consult Redis documentation for details.

Returns result, exception

hlen(**args*)
Execute HLEN Command, consult Redis documentation for details.

Returns result, exception

hmget(**args*)
Execute HMGET Command, consult Redis documentation for details.

Returns result, exception

hmset(**args*)
Execute HMSET Command, consult Redis documentation for details.

Returns result, exception

hscan(**args*)
Execute HSCAN Command, consult Redis documentation for details.

Returns result, exception

hset(**args*)
Execute HSET Command, consult Redis documentation for details.

Returns result, exception

hsetnx(**args*)
Execute HSETNX Command, consult Redis documentation for details.

Returns result, exception

hstrlen(**args*)
Execute HSTRLEN Command, consult Redis documentation for details.

Returns result, exception

hvals(**args*)
Execute HVALS Command, consult Redis documentation for details.

Returns result, exception

3.4.3 HyperLogLog

class pyredis.commands.**HyperLogLog**

pfadd(**args*)
Execute PFADD Command, consult Redis documentation for details.

Returns result, exception

pfcount(**args*)
Execute PFCOUNT Command, consult Redis documentation for details.

Returns result, exception

pfmerge(**args*)
Execute PFMERGE Command, consult Redis documentation for details.

Returns result, exception

3.4.4 Geo

class pyredis.commands.**Geo**

geoadd(**args*)
Execute GEOADD Command, consult Redis documentation for details.

Returns result, exception

geodist(**args*)
Execute GEODIST Command, consult Redis documentation for details.

Returns result, exception

geohash(**args*)
Execute GEOHASH Command, consult Redis documentation for details.

Returns result, exception

geopos(**args*)
Execute GEOPOS Command, consult Redis documentation for details.

Returns result, exception

georadius(**args*)
Execute GEORADIUS Command, consult Redis documentation for details.

Returns result, exception

georadiusbymember(**args*)
Execute GEORADIUSBYMEMBER Command, consult Redis documentation for details.

Returns result, exception

3.4.5 Key

class pyredis.commands.**Key**

delete(**args*)
Execute DEL Command, consult Redis documentation for details.

Returns result, exception

dump(**args*)
Execute DUMP Command, consult Redis documentation for details.

Returns result, exception

exists(**args*)
Execute EXISTS Command, consult Redis documentation for details.

Returns result, exception

expire(**args*)
Execute EXPIRE Command, consult Redis documentation for details.

Returns result, exception

expireat(**args*)
Execute EXPIREAT Command, consult Redis documentation for details.

Returns result, exception

keys(**args*, *shard_key=None*, *sock=None*)
Execute KEYS Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

migrate(**args*)
Execute MIGRATE Command, consult Redis documentation for details.

Returns result, exception

move(**args*)
Execute MOVE Command, consult Redis documentation for details.

Returns result, exception

object(**args*, *shard_key=None*, *sock=None*)
Execute OBJECT Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

persist(**args*)
Execute PERSIST Command, consult Redis documentation for details.

Returns result, exception

pexpire(**args*)
Execute PEXPIRE Command, consult Redis documentation for details.

Returns result, exception

pexpireat(**args*)
Execute PEXPIREAT Command, consult Redis documentation for details.

Returns result, exception

pttl(**args*)
Execute PTTL Command, consult Redis documentation for details.

Returns result, exception

randomkey(**args*, *shard_key=None*, *sock=None*)
Execute RANDOMKEY Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

rename(**args*)
Execute RENAME Command, consult Redis documentation for details.

Returns result, exception

renamenx(**args*)
Execute RENAMENX Command, consult Redis documentation for details.

Returns result, exception

restore(**args*)
Execute RESTORE Command, consult Redis documentation for details.

Returns result, exception

scan(**args*, *shard_key=None*, *sock=None*)
Execute SCAN Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

sort(**args*)
Execute SORT Command, consult Redis documentation for details.

Returns result, exception

ttl(**args*)
Execute TTL Command, consult Redis documentation for details.

Returns result, exception

type(**args*)
Execute TYPE Command, consult Redis documentation for details.

Returns result, exception

wait(**args*)
Execute WAIT Command, consult Redis documentation for details.

Returns result, exception

3.4.6 List

class pyredis.commands.**List**

blpop(**args*)
Execute BLPOP Command, consult Redis documentation for details.

Returns result, exception

brpop(**args*)
Execute BRPOP Command, consult Redis documentation for details.

Returns result, exception

brpoplpush(**args*)
Execute BRPOPPUSH Command, consult Redis documentation for details.

Returns result, exception

lindex(**args*)
Execute LINDEX Command, consult Redis documentation for details.

Returns result, exception

linsert(**args*)
Execute LINSERT Command, consult Redis documentation for details.

Returns result, exception

llen(**args*)
Execute LLEN Command, consult Redis documentation for details.

Returns result, exception

lpop(**args*)
Execute LPOP Command, consult Redis documentation for details.

Returns result, exception

lpush(**args*)
Execute LPUSH Command, consult Redis documentation for details.

Returns result, exception

lpushx(**args*)
Execute LPUSHX Command, consult Redis documentation for details.

Returns result, exception

lrange(**args*)
Execute LRANGE Command, consult Redis documentation for details.

Returns result, exception

lrem(**args*)
Execute LREM Command, consult Redis documentation for details.

Returns result, exception

lset(**args*)
Execute LSET Command, consult Redis documentation for details.

Returns result, exception

ltrim(**args*)
Execute LTRIM Command, consult Redis documentation for details.

Returns result, exception

rpop(**args*)
Execute RPOP Command, consult Redis documentation for details.

Returns result, exception

rpoplpush(**args*)
Execute RPOPLPUSH Command, consult Redis documentation for details.

Returns result, exception

rpush(**args*)
Execute RPUSH Command, consult Redis documentation for details.

Returns result, exception

rpushx(**args*)
Execute RPUSHX Command, consult Redis documentation for details.

Returns result, exception

3.4.7 Publish

class pyredis.commands.**Publish**

publish(**args*)
Execute PUBLISH Command, consult Redis documentation for details.

Returns result, exception

3.4.8 Scripting

class pyredis.commands.**Scripting**

eval(**args*, *shard_key=None*, *sock=None*)
Execute EVAL Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

evalsha(**args*, *shard_key=None*, *sock=None*)
Execute EVALSHA Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_debug(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT DEBUG Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_exists(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT EXISTS Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_flush(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT FLUSH Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_kill(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT KILL Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_load(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT LOAD Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

3.4.9 Set

class pyredis.commands.**Set**

sadd(**args*)
Execute SADD Command, consult Redis documentation for details.

Returns result, exception

scard(**args*)
Execute SCARD Command, consult Redis documentation for details.

Returns result, exception

sdiff(**args*)
Execute SDIFF Command, consult Redis documentation for details.

Returns result, exception

sdiffstore(**args*)
Execute SDIFFSTORE Command, consult Redis documentation for details.

Returns result, exception

sinter(**args*)
Execute SINTER Command, consult Redis documentation for details.

Returns result, exception

sinterstore(**args*)
Execute SINTERSTORE Command, consult Redis documentation for details.

Returns result, exception

sismember(**args*)
Execute SISMEMBER Command, consult Redis documentation for details.

Returns result, exception

smembers(**args*)
Execute SMEMBERS Command, consult Redis documentation for details.

Returns result, exception

smove(**args*)
Execute SMOVE Command, consult Redis documentation for details.

Returns result, exception

spop(**args*)
Execute SPOP Command, consult Redis documentation for details.

Returns result, exception

srandmember(**args*)
Execute SRANDMEMBER Command, consult Redis documentation for details.

Returns result, exception

srem(**args*)
Execute SREM Command, consult Redis documentation for details.

Returns result, exception

sscan(**args*)
Execute SSCAN Command, consult Redis documentation for details.

Returns result, exception

sunion(**args*)
Execute SUNION Command, consult Redis documentation for details.

Returns result, exception

sunoinstore(**args*)
Execute SUNIONSTORE Command, consult Redis documentation for details.

Returns result, exception

3.4.10 SSet

class pyredis.commands.**SSet**

zadd(**args*)
Execute ZADD Command, consult Redis documentation for details.

Returns result, exception

zcard(**args*)
Execute ZCARD Command, consult Redis documentation for details.

Returns result, exception

zcount(**args*)
Execute ZCOUNT Command, consult Redis documentation for details.

Returns result, exception

zincrby(**args*)
Execute ZINCRBY Command, consult Redis documentation for details.

Returns result, exception

zinterstore(**args*)
Execute ZINTERSTORE Command, consult Redis documentation for details.

Returns result, exception

zlexcount(**args*)
Execute ZLEXCOUNT Command, consult Redis documentation for details.

Returns result, exception

zrange(**args*)
Execute ZRANGE Command, consult Redis documentation for details.

Returns result, exception

zrangebylex(**args*)
Execute ZRANGEBYLEX Command, consult Redis documentation for details.

Returns result, exception

zrangebyscore(**args*)
Execute ZRANGEBYSCORE Command, consult Redis documentation for details.

Returns result, exception

zrank(**args*)
Execute ZRANK Command, consult Redis documentation for details.

Returns result, exception

zrem(**args*)
Execute ZREM Command, consult Redis documentation for details.

Returns result, exception

zremrangebylex(**args*)
Execute ZREMRANGEBYLEX Command, consult Redis documentation for details.

Returns result, exception

zremrangebyrank(**args*)
Execute ZREMRANGEBYRANK Command, consult Redis documentation for details.

Returns result, exception

zremrangebyscrore(**args*)
Execute ZREMRANGEBYSCORE Command, consult Redis documentation for details.

Returns result, exception

zrevrange(**args*)
Execute ZREVRANGE Command, consult Redis documentation for details.

Returns result, exception

zrevrangebylex(**args*)
Execute ZREVRANGEBYLEX Command, consult Redis documentation for details.

Returns result, exception

zrevrangebyscore(**args*)
Execute ZREVRANGEBYSCORE Command, consult Redis documentation for details.

Returns result, exception

zrevrank(**args*)
Execute ZREVRANK Command, consult Redis documentation for details.

Returns result, exception

zscan(**args*)
Execute ZSCAN Command, consult Redis documentation for details.

Returns result, exception

zscore(**args*)
Execute ZSCORE Command, consult Redis documentation for details.

Returns result, exception

zunionstore(**args*)
Execute ZUNIONSTORE Command, consult Redis documentation for details.

Returns result, exception

3.4.11 String

class pyredis.commands.**String**

append(**args*)
Execute APPEND Command, consult Redis documentation for details.

Returns result, exception

bitcount(**args*)
Execute BITCOUNT Command, consult Redis documentation for details.

Returns result, exception

bitfield(**args*)
Execute BITFIELD Command, consult Redis documentation for details.

Returns result, exception

bitop(**args*)
Execute BITOP Command, consult Redis documentation for details.

Returns result, exception

bitpos(**args*)
Execute BITPOS Command, consult Redis documentation for details.

Returns result, exception

decr(**args*)
Execute DECR Command, consult Redis documentation for details.

Returns result, exception

decrby(**args*)
Execute DECRBY Command, consult Redis documentation for details.

Returns result, exception

get(**args*)
Execute GET Command, consult Redis documentation for details.

Returns result, exception

getbit(**args*)
Execute GETBIT Command, consult Redis documentation for details.

Returns result, exception

getrange(**args*)
Execute GETRANGE Command, consult Redis documentation for details.

Returns result, exception

getset(**args*)
Execute GETSET Command, consult Redis documentation for details.

Returns result, exception

incr(**args*)
Execute INCR Command, consult Redis documentation for details.

Returns result, exception

incrby(**args*)
Execute INCRBY Command, consult Redis documentation for details.

Returns result, exception

incrbyfloat(**args*)
Execute INCRBYFLOAT Command, consult Redis documentation for details.

Returns result, exception

mget(**args*)
Execute MGET Command, consult Redis documentation for details.

Returns result, exception

mset(**args*)
Execute MSET Command, consult Redis documentation for details.

Returns result, exception

msetnx(**args*)
Execute MSETNX Command, consult Redis documentation for details.

Returns result, exception

psetex(**args*)
Execute PSETEX Command, consult Redis documentation for details.

Returns result, exception

set(**args*)
Execute SET Command, consult Redis documentation for details.

Returns result, exception

setbit(**args*)
Execute SETBIT Command, consult Redis documentation for details.

Returns result, exception

setex(**args*)
Execute SETEX Command, consult Redis documentation for details.

Returns result, exception

setnx(**args*)
Execute SETNX Command, consult Redis documentation for details.

Returns result, exception

setrange(**args*)
Execute SETRANGE Command, consult Redis documentation for details.

Returns result, exception

strlen(**args*)
Execute STRLEN Command, consult Redis documentation for details.

Returns result, exception

3.4.12 Subscribe

class pyredis.commands.**Subscribe**

psubscribe(**args*)
Execute PSUBSCRIBE Command, consult Redis documentation for details.

Returns result, exception

punsubscribe(**args*)
Execute PUNSUBSCRIBE Command, consult Redis documentation for details.

Returns result, exception

subscribe(**args*)
Execute SUBSCRIBE Command, consult Redis documentation for details.

Returns result, exception

unsubscribe(**args*)
Execute UNSUBSCRIBE Command, consult Redis documentation for details.

Returns result, exception

3.4.13 Transaction

class pyredis.commands.**Transaction**

discard(**args*, *shard_key=None*, *sock=None*)
Execute DISCARD Command, consult Redis documentation for details.

Returns result, exception

exec(**args*, *shard_key=None*, *sock=None*)
Execute EXEC Command, consult Redis documentation for details.

Returns result, exception

multi(**args*, *shard_key=None*, *sock=None*)
Execute MULTI Command, consult Redis documentation for details.

Returns result, exception

unwatch(**args*, *shard_key=None*, *sock=None*)
Execute UNWATCH Command, consult Redis documentation for details.

Returns result, exception

watch(**args*)
Execute WATCH Command, consult Redis documentation for details.

Returns result, exception

3.4.14 Scripting

class pyredis.commands.**Scripting**

eval(**args*, *shard_key=None*, *sock=None*)
Execute EVAL Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

evalsha(**args*, *shard_key=None*, *sock=None*)
Execute EVALSHA Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_debug(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT DEBUG Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_exists(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT EXISTS Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_flush(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT FLUSH Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_kill(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT KILL Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

script_load(**args*, *shard_key=None*, *sock=None*)
Execute SCRIPT LOAD Command, consult Redis documentation for details.

Parameters

- **shard_key** (*string*) – (optional) Should be set to the key name you try to work with. Can not be used if sock is set.

  Only used if used with a Cluster Client

- **sock** (*string*) – (optional) The string representation of a socket, the command should be executed against. For example: "testhost_6379" Can not be used if shard_key is set.

  Only used if used with a Cluster Client

Returns result, exception

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

H

I

K

L

M

N

O

P

R

S

T

U

W

Z