asyncio_redis Documentation

Release 0.1

Jonathan Slenders

Aug 23, 2017

Contents

Asynchronous Redis client for Python.

This Redis library is a completely asynchronous, non-blocking client for a Redis server. It depends on asyncio (PEP 3156) and therefor it requires Python 3.3 or 3.4. If you're new to asyncio, it can be helpful to check out the asyncio documentation first.

CHAPTER 1

Features

- Works for the asyncio (PEP3156) event loop
- No dependencies except asyncio
- Connection pooling and pipelining
- Automatic conversion from native Python types (unicode or bytes) to Redis types (bytes).
- Blocking calls and transactions supported
- Pubsub support
- Streaming of multi bulk replies
- Completely tested

CHAPTER 2

Installation

```
pip install asyncio_redis
```

Start by taking a look at *some examples*.

CHAPTER 3

Author and License

The `asyncio_redis` package is written by Jonathan Slenders. It's BSD licensed and freely available. Feel free to improve this package and send a pull request.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Examples

The `Connection` class

A *`Connection`* instance will take care of the connection and will automatically reconnect, using a new transport when the connection drops. This connection class also acts as a proxy to at *`RedisProtocol`* instance; any Redis command of the protocol can be called directly at the connection.

```
import asyncio
import asyncio_redis

@asyncio.coroutine
def example():
    # Create Redis connection
    connection = yield from asyncio_redis.Connection.create(host='localhost',␣
↪port=6379)

    # Set a key
    yield from connection.set('my_key', 'my_value')

    # When finished, close the connection.
    connection.close()

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(example())
```

See *the reference* to learn more about the other Redis commands.

Connection pooling

Requests will automatically be distributed among all connections in a `Pool`. If a connection is blocking because of –for instance– a blocking rpop, another connection will be used for new commands.

Note: This is the recommended way to connect to the Redis server.

```
import asyncio
import asyncio_redis

@asyncio.coroutine
def example():
    # Create Redis connection
    connection = yield from asyncio_redis.Pool.create(host='localhost', port=6379,␣
↪poolsize=10)

    # Set a key
    yield from connection.set('my_key', 'my_value')

    # When finished, close the connection pool.
    connection.close()
```

Transactions

A transaction can be started by calling `multi`. This returns a `Transaction` instance which is in fact just a proxy to the `RedisProtocol`, except that every Redis method of the protocol now became a coroutine that returns a future. The results of these futures can be retrieved after the transaction is commited with `exec`.

```
import asyncio
import asyncio_redis

@asyncio.coroutine
def example(loop):
    # Create Redis connection
    connection = yield from asyncio_redis.Pool.create(host='localhost', port=6379,␣
↪poolsize=10)

    # Create transaction
    transaction = yield from connection.multi()

    # Run commands in transaction (they return future objects)
    f1 = yield from transaction.set('key', 'value')
    f2 = yield from transaction.set('another_key', 'another_value')

    # Commit transaction
    yield from transaction.exec()

    # Retrieve results
    result1 = yield from f1
    result2 = yield from f2
```

```
    # When finished, close the connection pool.
    connection.close()
```

It's recommended to use a large enough poolsize. A connection will be occupied as long as there's a transaction running in there.

Pubsub

By calling *start_subscribe* (either on the protocol, through the *Connection* class or through the *Pool* class), you can start a pubsub listener.

```
import asyncio
import asyncio_redis

@asyncio.coroutine
def example():
    # Create connection
    connection = yield from asyncio_redis.Connection.create(host='localhost',␣
↪port=6379)

    # Create subscriber.
    subscriber = yield from connection.start_subscribe()

    # Subscribe to channel.
    yield from subscriber.subscribe([ 'our-channel' ])

    # Inside a while loop, wait for incoming events.
    while True:
        reply = yield from subscriber.next_published()
        print('Received: ', repr(reply.value), 'on channel', reply.channel)

    # When finished, close the connection.
    connection.close()
```

LUA Scripting

The *register_script* function – which can be used to register a LUA script – returns a *Script* instance. You can call its *run* method to execute this script.

```
import asyncio
import asyncio_redis

code = \
"""
local value = redis.call('GET', KEYS[1])
value = tonumber(value)
return value * ARGV[1]
"""

@asyncio.coroutine
def example():
    connection = yield from asyncio_redis.Connection.create(host='localhost',␣
↪port=6379)

    # Set a key
```

```
    yield from connection.set('my_key', '2')

    # Register script
    multiply = yield from connection.register_script(code)

    # Run script
    script_reply = yield from multiply.run(keys=['my_key'], args=['5'])
    result = yield from script_reply.return_value()
    print(result) # prints 2 * 5

    # When finished, close the connection.
    connection.close()
```

Raw bytes or UTF-8

The redis protocol only knows about bytes, but normally you want to use strings in your Python code. `asyncio_redis` is helpful and installs an encoder that does this conversion automatically, using the UTF-8 codec. However, sometimes you want to access raw bytes. This is possible by passing a *`BytesEncoder`* instance to the connection, pool or protocol.

```
import asyncio
import asyncio_redis

from asyncio_redis.encoders import BytesEncoder

@asyncio.coroutine
def example():
    # Create Redis connection
    connection = yield from asyncio_redis.Connection.create(host='localhost',␣
↪port=6379, encoder=BytesEncoder())

    # Set a key
    yield from connection.set(b'my_key', b'my_value')

    # When finished, close the connection.
    connection.close()
```

Scanning for keys

Redis has a few nice scanning utilities to discover keys in the database. They are rather low-level, but `asyncio_redis` exposes a simple *`Cursor`* class that allows you to iterate over all the keys matching a certain pattern. Each call of the *`fetchone()`* coroutine will return the next match. You don't have have to worry about accessing the server every x pages.

The following example will print all the keys in the database:

```
import asyncio
import asyncio_redis

from asyncio_redis.encoders import BytesEncoder

@asyncio.coroutine
def example():
    cursor = yield from protocol.scan(match='*')
```

```
    while True:
        item = yield from cursor.fetchone()
        if item is None:
            break
        else:
            print(item)
```

See the scanning utilities: *scan()*, *sscan()*, *hscan()* and *zscan()*

The RedisProtocol class

The most low level way of accessing the redis server through this library is probably by creating a connection with the *RedisProtocol* yourself. You can do it as follows:

```
import asyncio
import asyncio_redis

@asyncio.coroutine
def example():
    loop = asyncio.get_event_loop()

    # Create Redis connection
    transport, protocol = yield from loop.create_connection(
                asyncio_redis.RedisProtocol, 'localhost', 6379)

    # Set a key
    yield from protocol.set('my_key', 'my_value')

    # Get a key
    result = yield from protocol.get('my_key')
    print(result)

if __name__ == '__main__':
    asyncio.get_event_loop().run_until_complete(example())
```

Note: It is not recommended to use the Protocol class directly, because the low-level Redis implementation could change. Prefer the *Connection* or *Pool* class as demonstrated above if possible.

Reference

You can either use the *RedisProtocol* class directly, use the *Connection* class, or use the *Pool* wrapper which also offers connection pooling.

The Protocol

class asyncio_redis.**RedisProtocol**(*, *password=None*, *db=0*, *encoder=None*, *connection_lost_callback=None*, *enable_typechecking=True*, *loop=None*)

The Redis Protocol implementation.

```
self.loop = asyncio.get_event_loop()
transport, protocol = yield from loop.create_connection(RedisProtocol, 'localhost
↪', 6379)
```

Parameters

- **password** (Native Python type as defined by the `encoder` parameter) – Redis database password
- **encoder** (*BaseEncoder* instance.) – Encoder to use for encoding to or decoding from redis bytes to a native type. (Defaults to *UTF8Encoder*)
- **db** (*int*) – Redis database
- **enable_typechecking** (*bool*) – When `True`, check argument types for all redis commands. Normally you want to have this enabled.

append (*self*, *tr*, *key*, *value*)
Append a value to a key

Parameters

- **value** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

auth (*self*, *tr*, *password*)
Authenticate to the server

Parameters **password** – Native Python type, as defined by *native_type*

Returns (Future of) *StatusReply*

bgrewriteaof (*self*, *tr*)
Asynchronously rewrite the append-only file

Returns (Future of) *StatusReply*

bgsave (*self*, *tr*)
Asynchronously save the dataset to disk

Returns (Future of) *StatusReply*

bitcount (*self*, *tr*, *key*, *start=0*, *end=-1*)
Count the number of set bits (population counting) in a string.

Parameters

- **end** – int
- **start** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

bitop_and (*self*, *tr*, *destkey*, *srckeys*)
Perform a bitwise AND operation between multiple keys.

Parameters

- **destkey** – Native Python type, as defined by *native_type*
- **srckeys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) int

bitop_not (*self*, *tr*, *destkey*, *key*)
Perform a bitwise NOT operation between multiple keys.

Parameters

- **destkey** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

bitop_or (*self*, *tr*, *destkey*, *srckeys*)
Perform a bitwise OR operation between multiple keys.

Parameters

- **destkey** – Native Python type, as defined by *native_type*
- **srckeys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) int

bitop_xor (*self*, *tr*, *destkey*, *srckeys*)
Perform a bitwise XOR operation between multiple keys.

Parameters

- **destkey** – Native Python type, as defined by *native_type*
- **srckeys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) int

blpop (*self*, *tr*, *keys*, *timeout=0*)

Remove and get the first element in a list, or block until one is available. This will raise *TimeoutError* when the timeout was exceeded and Redis returns *None*.

Parameters

- **timeout** – int
- **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) *BlockingPopReply*

brpop (*self*, *tr*, *keys*, *timeout=0*)

Remove and get the last element in a list, or block until one is available. This will raise *TimeoutError* when the timeout was exceeded and Redis returns *None*.

Parameters

- **timeout** – int
- **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) *BlockingPopReply*

brpoplpush (*self*, *tr*, *source*, *destination*, *timeout=0*)
Pop a value from a list, push it to another list and return it; or block until one is available

Parameters

- **timeout** – int

- **source** – Native Python type, as defined by *native_type*
- **destination** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type*

client_getname (*self*, *tr*)
Get the current connection name

Returns (Future of) Native Python type, as defined by *native_type*

client_kill (*self*, *tr*, *address*)

Kill the connection of a client *address* should be an "ip:port" string.

Parameters **address** – str

Returns (Future of) *StatusReply*

client_list (*self*, *tr*)
Get the list of client connections

Returns (Future of) *InfoReply*

client_setname (*self*, *tr*, *name*)
Set the current connection name

Returns (Future of) *StatusReply*

config_get (*self*, *tr*, *parameter*)
Get the value of a configuration parameter

Parameters **parameter** – str

Returns (Future of) ConfigPairReply

config_resetstat (*self*, *tr*)
Reset the stats returned by INFO

Returns (Future of) *StatusReply*

config_rewrite (*self*, *tr*)
Rewrite the configuration file with the in memory configuration

Returns (Future of) *StatusReply*

config_set (*self*, *tr*, *parameter*, *value*)
Set a configuration parameter to the given value

Parameters

- **value** – str
- **parameter** – str

Returns (Future of) *StatusReply*

connection_made (*transport*)

dbsize (*self*, *tr*)
Return the number of keys in the currently-selected database.

Returns (Future of) int

decr (*self*, *tr*, *key*)
Decrement the integer value of a key by one

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

decrby (*self*, *tr*, *key*, *increment*)
Decrement the integer value of a key by the given number

Parameters

- **increment** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

delete (*self*, *tr*, *keys*)
Delete a key

Parameters **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) int

echo (*self*, *tr*, *string*)
Echo the given string

Parameters **string** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type*

evalsha (*self*, *tr*, *sha*, *keys=None*, *args=None*)

Evaluates a script cached on the server side by its SHA1 digest. Scripts are cached on the server side using the SCRIPT LOAD command.

The return type/value depends on the script.

This will raise a *ScriptKilledError* exception if the script was killed.

Parameters

- **args** – List or iterable of Native Python type, as defined by *native_type* or None
- **keys** – List or iterable of Native Python type, as defined by *native_type* or None
- **sha** – str

Returns (Future of) `EvalScriptReply`

exists (*self*, *tr*, *key*)
Determine if a key exists

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) bool

expire (*self*, *tr*, *key*, *seconds*)
Set a key's time to live in seconds

Parameters

- **seconds** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

expireat (*self*, *tr*, *key*, *timestamp*)
Set the expiration for a key as a UNIX timestamp

Parameters

- **timestamp** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

flushall (*self*, *tr*)
Remove all keys from all databases

Returns (Future of) *StatusReply*

flushdb (*self*, *tr*)
Delete all the keys of the currently selected DB. This command never fails.

Returns (Future of) *StatusReply*

get (*self*, *tr*, *key*)
Get the value of a key

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type* or None

getbit (*self*, *tr*, *key*, *offset*)
Returns the bit value at offset in the string value stored at key

Parameters

- **offset** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) bool

getset (*self*, *tr*, *key*, *value*)
Set the string value of a key and return its old value

Parameters

- **value** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type* or None

hdel (*self*, *tr*, *key*, *fields*)
Delete one or more hash fields

Parameters

- **fields** – List or iterable of Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

hexists (*self*, *tr*, *key*, *field*)
Returns if field is an existing field in the hash stored at key.

Parameters

- **field** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) bool

hget (*self*, *tr*, *key*, *field*)
Get the value of a hash field

Parameters

- **field** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type* or None

hgetall (*self*, *tr*, *key*)
Get the value of a hash field

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) *DictReply*

hgetall_asdict (*self*, *tr*, *key*)
Get the value of a hash field

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) dict

hincrby (*self*, *tr*, *key*, *field*, *increment*)

Increment the integer value of a hash field by the given number Returns: the value at field after the increment operation.

Parameters

- **field** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

hincrbyfloat (*self*, *tr*, *key*, *field*, *increment*)

Increment the float value of a hash field by the given amount Returns: the value at field after the increment operation.

Parameters

- **field** – Native Python type, as defined by *native_type*
- **increment** – int or float
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) float

hkeys (*self*, *tr*, *key*)
Get all the keys in a hash. (Returns a set)

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) *SetReply*

hkeys_asset (*self*, *tr*, *key*)
Get all the keys in a hash. (Returns a set)

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) set

hlen (*self*, *tr*, *key*)
Returns the number of fields contained in the hash stored at key.

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

hmget (*self*, *tr*, *key*, *fields*)
Get the values of all the given hash fields

Parameters

- **fields** – List or iterable of Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *ListReply*

hmget_aslist (*self*, *tr*, *key*, *fields*)
Get the values of all the given hash fields

Parameters

- **fields** – List or iterable of Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) list

hmset (*self*, *tr*, *key*, *values*)
Set multiple hash fields to multiple values

Parameters

- **values** – dict
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *StatusReply*

hscan (*self*, *tr*, *key*, *match=None*)

Incrementally iterate hash fields and associated values Also see: *scan()*

Parameters

- **match** – Native Python type, as defined by *native_type* or None
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *DictCursor*

hset (*self*, *tr*, *key*, *field*, *value*)
Set the string value of a hash field

Parameters

- **value** – Native Python type, as defined by *native_type*
- **field** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

hsetnx (*self*, *tr*, *key*, *field*, *value*)
Set the value of a hash field, only if the field does not exist

Parameters

- **value** – Native Python type, as defined by *native_type*
- **field** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

hvals (*self*, *tr*, *key*)
Get all the values in a hash. (Returns a list)

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) *ListReply*

hvals_aslist (*self*, *tr*, *key*)
Get all the values in a hash. (Returns a list)

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) list

in_blocking_call
True when waiting for answer to blocking command.

in_pubsub
True when the protocol is in pubsub mode.

in_transaction
True when we're inside a transaction.

in_use
True when this protocol is in use.

incr (*self*, *tr*, *key*)
Increment the integer value of a key by one

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

incrby (*self*, *tr*, *key*, *increment*)
Increment the integer value of a key by the given amount

Parameters

- **increment** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

info (*self*, *tr*, *section=None*)
Get information and statistics about the server

Parameters **section** – Native Python type, as defined by *native_type* or None

Returns (Future of) *InfoReply*

is_connected
True when the underlying transport is connected.

keys (*self*, *tr*, *pattern*)

Find all keys matching the given pattern.

Note: Also take a look at *scan()*.

Parameters **pattern** – Native Python type, as defined by *native_type*

Returns (Future of) *ListReply*

keys_aslist (*self*, *tr*, *pattern*)

Find all keys matching the given pattern.

Note: Also take a look at *scan()*.

Parameters **pattern** – Native Python type, as defined by *native_type*

Returns (Future of) list

lastsave (*self*, *tr*)

Get the UNIX time stamp of the last successful save to disk

Returns (Future of) int

lindex (*self*, *tr*, *key*, *index*)

Get an element from a list by its index

Parameters

- **index** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type* or None

linsert (*self*, *tr*, *key*, *pivot*, *value*, *before=False*)

Insert an element before or after another element in a list

Parameters

- **value** – Native Python type, as defined by *native_type*
- **pivot** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

llen (*self*, *tr*, *key*)

Returns the length of the list stored at key.

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

lpop (*self*, *tr*, *key*)

Remove and get the first element in a list

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type* or None

lpush (*self*, *tr*, *key*, *values*)
Prepend one or multiple values to a list

Parameters

- **values** – List or iterable of Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

lpushx (*self*, *tr*, *key*, *value*)
Prepend a value to a list, only if the list exists

Parameters

- **value** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

lrange (*self*, *tr*, *key*, *start=0*, *stop=-1*)
Get a range of elements from a list.

Parameters

- **stop** – int
- **start** – int

Returns (Future of) *ListReply*

lrange_aslist (*self*, *tr*, *key*, *start=0*, *stop=-1*)
Get a range of elements from a list.

Parameters

- **stop** – int
- **start** – int

Returns (Future of) list

lrem (*self*, *tr*, *key*, *count=0*, *value=''*)
Remove elements from a list

Parameters

- **count** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

lset (*self*, *tr*, *key*, *index*, *value*)
Set the value of an element in a list by its index.

Parameters

- **value** – Native Python type, as defined by *native_type*
- **index** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *StatusReply*

ltrim(*self*, *tr*, *key*, *start=0*, *stop=-1*)
Trim a list to the specified range

Parameters

- **stop** – int
- **start** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *StatusReply*

mget(*self*, *tr*, *keys*)
Returns the values of all specified keys.

Parameters **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) *ListReply*

mget_aslist(*self*, *tr*, *keys*)
Returns the values of all specified keys.

Parameters **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) list

move(*self*, *tr*, *key*, *database*)
Move a key to another database

Parameters

- **key** – Native Python type, as defined by *native_type*
- **database** – int

Returns (Future of) int

multi(*self*, *tr*, *watch=None*)

Start of transaction.

```
transaction = yield from protocol.multi()

# Run commands in transaction
f1 = yield from transaction.set('key', 'value')
f2 = yield from transaction.set('another_key', 'another_value')

# Commit transaction
yield from transaction.exec()

# Retrieve results (you can also use asyncio.tasks.gather)
result1 = yield from f1
result2 = yield from f2
```

returns A *asyncio_redis.Transaction* instance.

Parameters **watch** – List or iterable of Native Python type, as defined by *native_type* or None

Returns (Future of) *asyncio_redis.Transaction*

persist(*self*, *tr*, *key*)
Remove the expiration from a key

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

pexpire (*self*, *tr*, *key*, *milliseconds*)
Set a key's time to live in milliseconds

Parameters

- **milliseconds** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

pexpireat (*self*, *tr*, *key*, *milliseconds_timestamp*)
Set the expiration for a key as a UNIX timestamp specified in milliseconds

Parameters

- **key** – Native Python type, as defined by *native_type*
- **milliseconds_timestamp** – int

Returns (Future of) int

ping (*self*, *tr*)
Ping the server (Returns PONG)

Returns (Future of) *StatusReply*

pttl (*self*, *tr*, *key*)
Get the time to live for a key in milliseconds

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

publish (*self*, *tr*, *channel*, *message*)

Post a message to a channel (Returns the number of clients that received this message.)

Parameters

- **channel** – Native Python type, as defined by *native_type*
- **message** – Native Python type, as defined by *native_type*

Returns (Future of) int

pubsub_channels (*self*, *tr*, *pattern=None*)

Lists the currently active channels. An active channel is a Pub/Sub channel with one ore more subscribers (not including clients subscribed to patterns).

Parameters **pattern** – Native Python type, as defined by *native_type* or None

Returns (Future of) *ListReply*

pubsub_channels_aslist (*self*, *tr*, *pattern=None*)

Lists the currently active channels. An active channel is a Pub/Sub channel with one ore more subscribers (not including clients subscribed to patterns).

Parameters **pattern** – Native Python type, as defined by *native_type* or None

Returns (Future of) list

pubsub_numpat (*self*, *tr*)

Returns the number of subscriptions to patterns (that are performed using the PSUBSCRIBE command). Note that this is not just the count of clients subscribed to patterns but the total number of patterns all the clients are subscribed to.

Returns (Future of) int

pubsub_numsub (*self*, *tr*, *channels*)

Returns the number of subscribers (not counting clients subscribed to patterns) for the specified channels.

Parameters **channels** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) *DictReply*

pubsub_numsub_asdict (*self*, *tr*, *channels*)

Returns the number of subscribers (not counting clients subscribed to patterns) for the specified channels.

Parameters **channels** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) dict

randomkey (*self*, *tr*)
Return a random key from the keyspace

Returns (Future of) Native Python type, as defined by *native_type*

register_script (*self*, *tr*, *script*)

Register a LUA script.

```
script = yield from protocol.register_script(lua_code)
result = yield from script.run(keys=[...], args=[...])
```

Parameters **script** – str

Returns (Future of) *Script*

rename (*self*, *tr*, *key*, *newkey*)
Rename a key

Parameters

- **newkey** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *StatusReply*

renamenx (*self*, *tr*, *key*, *newkey*)

Rename a key, only if the new key does not exist (Returns 1 if the key was successfully renamed.)

Parameters

- **newkey** – Native Python type, as defined by *native_type*

- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

rpop (*self*, *tr*, *key*)
Remove and get the last element in a list

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type* or None

rpoplpush (*self*, *tr*, *source*, *destination*)
Remove the last element in a list, append it to another list and return it

Parameters

- **source** – Native Python type, as defined by *native_type*
- **destination** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type* or None

rpush (*self*, *tr*, *key*, *values*)
Append one or multiple values to a list

Parameters

- **values** – List or iterable of Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

rpushx (*self*, *tr*, *key*, *value*)
Append a value to a list, only if the list exists

Parameters

- **value** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

sadd (*self*, *tr*, *key*, *members*)
Add one or more members to a set

Parameters

- **members** – List or iterable of Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

save (*self*, *tr*)
Synchronously save the dataset to disk

Returns (Future of) *StatusReply*

scan (*self*, *tr*, *match=None*)

Walk through the keys space. You can either fetch the items one by one or in bulk.

```
cursor = yield from protocol.scan(match='*')
while True:
    item = yield from cursor.fetchone()
    if item is None:
```

```
        break
    else:
        print(item)
```

```
cursor = yield from protocol.scan(match='*')
items = yield from cursor.fetchall()
```

It's possible to alter the COUNT-parameter, by assigning a value to `cursor.count`, before calling `fetchone` or `fetchall`. For instance:

```
cursor.count = 100
```

Also see: *sscan()*, *hscan()* and *zscan()*

Redis reference: http://redis.io/commands/scan

Parameters **match** – Native Python type, as defined by *native_type* or None

Returns (Future of) *Cursor*

scard (*self*, *tr*, *key*)
Get the number of members in a set

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

script_exists (*self*, *tr*, *shas*)
Check existence of scripts in the script cache.

Parameters **shas** – List or iterable of str

Returns (Future of) List or iterable of bool

script_flush (*self*, *tr*)
Remove all the scripts from the script cache.

Returns (Future of) *StatusReply*

script_kill (*self*, *tr*)

Kill the script currently in execution. This raises *NoRunningScriptError* when there are no scrips running.

Returns (Future of) *StatusReply*

script_load (*self*, *tr*, *script*)
Load script, returns sha1

Parameters **script** – str

Returns (Future of) str

sdiff (*self*, *tr*, *keys*)
Subtract multiple sets

Parameters **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) *SetReply*

sdiff_asset (*self*, *tr*, *keys*)
Subtract multiple sets

Parameters **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) set

sdiffstore(*self*, *tr*, *destination*, *keys*)
Subtract multiple sets and store the resulting set in a key

Parameters

- **keys** – List or iterable of Native Python type, as defined by *native_type*
- **destination** – Native Python type, as defined by *native_type*

Returns (Future of) int

select(*self*, *tr*, *db*)
Change the selected database for the current connection

Parameters **db** – int

Returns (Future of) *StatusReply*

set(*self*, *tr*, *key*, *value*, *expire=None*, *pexpire=None*, *only_if_not_exists=False*, *only_if_exists=False*)

Set the string value of a key

```
yield from protocol.set('key', 'value')
result = yield from protocol.get('key')
assert result == 'value'
```

To set a value and its expiration, only if key not exists, do:

```
yield from protocol.set('key', 'value', expire=1, only_if_not_
↪exists=True)
```

This will send: `SET key value EX 1 NX` at the network. To set value and its expiration in milliseconds, but only if key already exists:

```
yield from protocol.set('key', 'value', pexpire=1000, only_if_
↪exists=True)
```

Parameters

- **pexpire** – int or None
- **only_if_not_exists** – bool
- **value** – Native Python type, as defined by *native_type*
- **expire** – int or None
- **only_if_exists** – bool
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *StatusReply* or None

setbit(*self*, *tr*, *key*, *offset*, *value*)
Sets or clears the bit at offset in the string value stored at key

Parameters

- **value** – bool
- **offset** – int

- **key** – Native Python type, as defined by *native_type*

Returns (Future of) bool

setex (*self*, *tr*, *key*, *seconds*, *value*)
Set the string value of a key with expire

Parameters

- **value** – Native Python type, as defined by *native_type*
- **seconds** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *StatusReply*

setnx (*self*, *tr*, *key*, *value*)

Set the string value of a key if it does not exist. Returns True if value is successfully set

Parameters

- **value** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) bool

shutdown (*self*, *tr*, *save=False*)
Synchronously save the dataset to disk and then shut down the server

Returns (Future of) *StatusReply*

sinter (*self*, *tr*, *keys*)
Intersect multiple sets

Parameters **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) *SetReply*

sinter_asset (*self*, *tr*, *keys*)
Intersect multiple sets

Parameters **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) set

sinterstore (*self*, *tr*, *destination*, *keys*)
Intersect multiple sets and store the resulting set in a key

Parameters

- **keys** – List or iterable of Native Python type, as defined by *native_type*
- **destination** – Native Python type, as defined by *native_type*

Returns (Future of) int

sismember (*self*, *tr*, *key*, *value*)
Determine if a given value is a member of a set

Parameters

- **value** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) bool

smembers (*self*, *tr*, *key*)
Get all the members in a set

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) *SetReply*

smembers_asset (*self*, *tr*, *key*)
Get all the members in a set

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) set

smove (*self*, *tr*, *source*, *destination*, *value*)
Move a member from one set to another

Parameters

- **value** – Native Python type, as defined by *native_type*
- **source** – Native Python type, as defined by *native_type*
- **destination** – Native Python type, as defined by *native_type*

Returns (Future of) int

spop (*self*, *tr*, *key*)
Removes and returns a random element from the set value stored at key.

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) Native Python type, as defined by *native_type* or None

srandmember (*self*, *tr*, *key*, *count=1*)

Get one or multiple random members from a set (Returns a list of members, even when count==1)

Parameters

- **count** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *SetReply*

srandmember_asset (*self*, *tr*, *key*, *count=1*)

Get one or multiple random members from a set (Returns a list of members, even when count==1)

Parameters

- **count** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) set

srem (*self*, *tr*, *key*, *members*)
Remove one or more members from a set

Parameters

- **members** – List or iterable of Native Python type, as defined by *native_type*

- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

sscan (*self*, *tr*, *key*, *match=None*)

Incrementally iterate set elements

Also see: *scan()*

Parameters

- **match** – Native Python type, as defined by *native_type* or None
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *SetCursor*

start_subscribe (*self*, *tr*, **a*)

Start a pubsub listener.

```
# Create subscription
subscription = yield from protocol.start_subscribe()
yield from subscription.subscribe(['key'])
yield from subscription.psubscribe(['pattern*'])

while True:
    result = yield from subscription.next_published()
    print(result)
```

returns *Subscription*

Returns (Future of) *asyncio_redis.Subscription*

strlen (*self*, *tr*, *key*)

Returns the length of the string value stored at key. An error is returned when key holds a non-string value.

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

sunion (*self*, *tr*, *keys*)
Add multiple sets

Parameters **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) *SetReply*

sunion_asset (*self*, *tr*, *keys*)
Add multiple sets

Parameters **keys** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) set

sunionstore (*self*, *tr*, *destination*, *keys*)
Add multiple sets and store the resulting set in a key

Parameters

- **keys** – List or iterable of Native Python type, as defined by *native_type*
- **destination** – Native Python type, as defined by *native_type*

Returns (Future of) int

ttl (*self*, *tr*, *key*)
Get the time to live for a key

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

type (*self*, *tr*, *key*)
Determine the type stored at key

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) *StatusReply*

watch (**args*, ***kw*)

Watch keys.

```
# Watch keys for concurrent updates
yield from protocol.watch(['key', 'other_key'])

value = yield from protocol.get('key')
another_value = yield from protocol.get('another_key')

transaction = yield from protocol.multi()

f1 = yield from transaction.set('key', another_value)
f2 = yield from transaction.set('another_key', value)

# Commit transaction
yield from transaction.exec()

# Retrieve results
yield from f1
yield from f2
```

Returns (Future of) None

zadd (*self*, *tr*, *key*, *values*)

Add one or more members to a sorted set, or update its score if it already exists

```
yield protocol.zadd('myzset', { 'key': 4, 'key2': 5 })
```

Parameters

- **values** – dict
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

zcard (*self*, *tr*, *key*)
Get the number of members in a sorted set

Parameters **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

zcount (*self*, *tr*, *key*, *min*, *max*)
Count the members in a sorted set with scores within the given values

Parameters

- **min** – ZScoreBoundary
- **max** – ZScoreBoundary
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

zincrby (*self*, *tr*, *key*, *increment*, *member*)
Increment the score of a member in a sorted set

Parameters

- **member** – Native Python type, as defined by *native_type*
- **increment** – float
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) float

zinterstore (*self*, *tr*, *destination*, *keys*, *weights=None*, *aggregate='SUM'*)
Intersect multiple sorted sets and store the resulting sorted set in a new key

Parameters

- **weights** – None or List or iterable of float
- **keys** – List or iterable of Native Python type, as defined by *native_type*
- **destination** – Native Python type, as defined by *native_type*

Returns (Future of) int

zrange (*self*, *tr*, *key*, *start=0*, *stop=-1*)

Return a range of members in a sorted set, by index.

You can do the following to receive the slice of the sorted set as a python dict (mapping the keys to their scores):

```
result = yield protocol.zrange('myzset', start=10, stop=20)
my_dict = yield result.asdict()
```

or the following to retrieve it as a list of keys:

```
result = yield protocol.zrange('myzset', start=10, stop=20)
my_dict = yield result.aslist()
```

Parameters

- **stop** – int
- **start** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *ZRangeReply*

zrange_asdict (*self*, *tr*, *key*, *start=0*, *stop=-1*)

Return a range of members in a sorted set, by index.

You can do the following to receive the slice of the sorted set as a python dict (mapping the keys to their scores):

```
result = yield protocol.zrange('myzset', start=10, stop=20)
my_dict = yield result.asdict()
```

or the following to retrieve it as a list of keys:

```
result = yield protocol.zrange('myzset', start=10, stop=20)
my_dict = yield result.aslist()
```

Parameters

- **stop** – int
- **start** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) dict

zrangebyscore(*self*, *tr*, *key*, *min=ZScoreBoundary(value='-inf', exclude_boundary=False)*, *max=ZScoreBoundary(value='+inf', exclude_boundary=False)*, *offset=0*, *limit=-1*)

Return a range of members in a sorted set, by score

Parameters

- **offset** – int
- **limit** – int
- **min** – ZScoreBoundary
- **max** – ZScoreBoundary
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *ZRangeReply*

zrangebyscore_asdict(*self*, *tr*, *key*, *min=ZScoreBoundary(value='-inf', exclude_boundary=False)*, *max=ZScoreBoundary(value='+inf', exclude_boundary=False)*, *offset=0*, *limit=-1*)

Return a range of members in a sorted set, by score

Parameters

- **offset** – int
- **limit** – int
- **min** – ZScoreBoundary
- **max** – ZScoreBoundary
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) dict

zrank(*self*, *tr*, *key*, *member*)

Determine the index of a member in a sorted set

Parameters

- **member** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int or None

zrem(*self*, *tr*, *key*, *members*)
Remove one or more members from a sorted set

Parameters

- **members** – List or iterable of Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

zremrangebyrank(*self*, *tr*, *key*, *min=0*, *max=-1*)
Remove all members in a sorted set within the given indexes

Parameters

- **min** – int
- **max** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

zremrangebyscore(*self*, *tr*, *key*, *min=ZScoreBoundary(value='-inf', exclude_boundary=False)*, *max=ZScoreBoundary(value='+inf', exclude_boundary=False)*)
Remove all members in a sorted set within the given scores

Parameters

- **min** – ZScoreBoundary
- **max** – ZScoreBoundary
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int

zrevrange(*self*, *tr*, *key*, *start=0*, *stop=-1*)

Return a range of members in a reversed sorted set, by index.

You can do the following to receive the slice of the sorted set as a python dict (mapping the keys to their scores):

```
my_dict = yield protocol.zrevrange_asdict('myzset', start=10, stop=20)
```

or the following to retrieve it as a list of keys:

```
zrange_reply = yield protocol.zrevrange('myzset', start=10, stop=20)
my_dict = yield zrange_reply.aslist()
```

Parameters

- **stop** – int
- **start** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *ZRangeReply*

zrevrange_asdict (*self*, *tr*, *key*, *start=0*, *stop=-1*)

Return a range of members in a reversed sorted set, by index.

You can do the following to receive the slice of the sorted set as a python dict (mapping the keys to their scores):

```
my_dict = yield protocol.zrevrange_asdict('myzset', start=10, stop=20)
```

or the following to retrieve it as a list of keys:

```
zrange_reply = yield protocol.zrevrange('myzset', start=10, stop=20)
my_dict = yield zrange_reply.aslist()
```

Parameters

- **stop** – int
- **start** – int
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) dict

zrevrangebyscore (*self*, *tr*, *key*, *max=ZScoreBoundary(value='+inf', exclude_boundary=False)*, *min=ZScoreBoundary(value='-inf', exclude_boundary=False)*, *offset=0*, *limit=-1*)

Return a range of members in a sorted set, by score, with scores ordered from high to low

Parameters

- **offset** – int
- **limit** – int
- **min** – ZScoreBoundary
- **max** – ZScoreBoundary
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *ZRangeReply*

zrevrangebyscore_asdict (*self*, *tr*, *key*, *max=ZScoreBoundary(value='+inf', exclude_boundary=False)*, *min=ZScoreBoundary(value='-inf', exclude_boundary=False)*, *offset=0*, *limit=-1*)

Return a range of members in a sorted set, by score, with scores ordered from high to low

Parameters

- **offset** – int
- **limit** – int
- **min** – ZScoreBoundary
- **max** – ZScoreBoundary
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) dict

zrevrank (*self*, *tr*, *key*, *member*)

Determine the index of a member in a sorted set, with scores ordered from high to low

Parameters

- **member** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) int or None

zscan(*self*, *tr*, *key*, *match=None*)

Incrementally iterate sorted sets elements and associated scores Also see: *scan()*

Parameters

- **match** – Native Python type, as defined by *native_type* or None
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) *DictCursor*

zscore(*self*, *tr*, *key*, *member*)

Get the score associated with the given member in a sorted set

Parameters

- **member** – Native Python type, as defined by *native_type*
- **key** – Native Python type, as defined by *native_type*

Returns (Future of) float or None

zunionstore(*self*, *tr*, *destination*, *keys*, *weights=None*, *aggregate='SUM'*)

Add multiple sorted sets and store the resulting sorted set in a new key

Parameters

- **weights** – None or List or iterable of float
- **keys** – List or iterable of Native Python type, as defined by *native_type*
- **destination** – Native Python type, as defined by *native_type*

Returns (Future of) int

class asyncio_redis.**HiRedisProtocol**(*, *password=None*, *db=0*, *encoder=None*, *connection_lost_callback=None*, *enable_typechecking=True*, *loop=None*)

Protocol implementation that uses the *hiredis* library for parsing the incoming data. This will be faster in many cases, but not necessarily always.

It does not (yet) support streaming of multibulk replies, which means that you won't see the first item of a multi bulk reply, before the whole response has been parsed.

Encoders

class asyncio_redis.encoders.**BaseEncoder**

Abstract base class for all encoders.

decode_to_native(*data*)

Decodes network bytes to a Python native type. It should always be the reverse operation of *encode_from_native*.

encode_from_native(*data*)

Encodes the native Python type to network bytes. Usually this will encode a string object to bytes using the UTF-8 encoding. You can either override this function, or set the *encoding* attribute.

native_type = None

class asyncio_redis.encoders.**UTF8Encoder**
Encode strings to and from utf-8 bytes.

class asyncio_redis.encoders.**BytesEncoder**
For raw access to the Redis database.

native_type
alias of bytes

Connection

class asyncio_redis.**Connection**
Wrapper around the protocol and transport which takes care of establishing the connection and reconnecting it.

```
connection = yield from Connection.create(host='localhost', port=6379)
result = yield from connection.set('key', 'value')
```

close()
Close the connection transport.

classmethod create(*host='localhost'*, *port=6379*, ***, *password=None*, *db=0*, *encoder=None*, *auto_reconnect=True*, *loop=None*, *protocol_class=<class 'asyncio_redis.protocol.RedisProtocol'>*)

Parameters

- **host** (*str*) – Address, either host or unix domain socket path
- **port** (*int*) – TCP port. If port is 0 then host assumed to be unix socket path
- **password** (*bytes*) – Redis database password
- **db** (*int*) – Redis database
- **encoder** (*BaseEncoder* instance.) – Encoder to use for encoding to or decoding from redis bytes to a native type.
- **auto_reconnect** (*bool*) – Enable auto reconnect
- **loop** – (optional) asyncio event loop.
- **protocol_class** (*RedisProtocol*) – (optional) redis protocol implementation

transport
The transport instance that the protocol is currently using.

Connection pool

class asyncio_redis.**Pool**
Pool of connections. Each Takes care of setting up the connection and connection pooling.

When poolsize > 1 and some connections are in use because of transactions or blocking requests, the other are preferred.

```
pool = yield from Pool.create(host='localhost', port=6379, poolsize=10)
result = yield from connection.set('key', 'value')
```

close()
Close all the connections in the pool.

connections_connected
The amount of open TCP connections.

connections_in_use
Return how many protocols are in use.

classmethod create(*host='localhost'*, *port=6379*, ***, *password=None*, *db=0*, *encoder=None*, *poolsize=1*, *auto_reconnect=True*, *loop=None*, *protocol_class=<class 'asyncio_redis.protocol.RedisProtocol'>*)
Create a new connection pool instance.

Parameters

- **host** (*str*) – Address, either host or unix domain socket path
- **port** (*int*) – TCP port. If port is 0 then host assumed to be unix socket path
- **password** (*bytes*) – Redis database password
- **db** (*int*) – Redis database
- **encoder** (*BaseEncoder* instance.) – Encoder to use for encoding to or decoding from redis bytes to a native type.
- **poolsize** (*int*) – The number of parallel connections.
- **auto_reconnect** (*bool*) – Enable auto reconnect
- **loop** – (optional) asyncio event loop.
- **protocol_class** (*RedisProtocol*) – (optional) redis protocol implementation

poolsize
Number of parallel connections in the pool.

register_script(*self*, *tr*, *script*)

Register a LUA script.

```
script = yield from protocol.register_script(lua_code)
result = yield from script.run(keys=[...], args=[...])
```

Parameters **script** – str

Returns (Future of) *Script*

Command replies

class asyncio_redis.replies.**StatusReply**(*status*)
Wrapper for Redis status replies. (for messages like OK, QUEUED, etc...)

class asyncio_redis.replies.**DictReply**(*multibulk_reply*)
Container for a dict reply.

The content can be retrieved by calling *asdict()* which returns a Python dictionary. Or by iterating over it:

```
for f in dict_reply:
    key, value = yield from f
    print(key, value)
```

asdict()
Return the result as a Python dictionary.

class asyncio_redis.replies.**ListReply**(*multibulk_reply*)
Redis list result. The content can be retrieved by calling *aslist()* or by iterating over it or by iterating over it

```
for f in list_reply:
    item = yield from f
    print(item)
```

aslist()
Return the result as a Python `list`.

class asyncio_redis.replies.**SetReply**(*multibulk_reply*)
Redis set result. The content can be retrieved by calling *asset()* or by iterating over it

```
for f in set_reply:
    item = yield from f
    print(item)
```

asset()
Return the result as a Python `set`.

class asyncio_redis.replies.**ZRangeReply**(*multibulk_reply*)
Container for a zrange query result.

class asyncio_redis.replies.**PubSubReply**(*channel*, *value*, *, *pattern=None*)
Received pubsub message.

channel
Channel name

pattern
The pattern to which we subscribed or *None* otherwise

value
Received PubSub value

class asyncio_redis.replies.**BlockingPopReply**(*list_name*, *value*)
blpop() or *brpop()* reply

list_name
List name.

value
Popped value

class asyncio_redis.replies.**InfoReply**(*data*)
info() reply.

class asyncio_redis.replies.**ClientListReply**(*data*)
client_list() reply.

Cursors

class asyncio_redis.cursors.**Cursor**(*name*, *scanfunc*)
Cursor for walking through the results of a *scan* query.

fetchall()
Coroutine that reads all the items in one list.

fetchone()
Coroutines that returns the next item. It returns *None* after the last item.

class asyncio_redis.cursors.**SetCursor**(*name*, *scanfunc*)
Cursor for walking through the results of a *sscan* query.

class asyncio_redis.cursors.**DictCursor**(*name*, *scanfunc*)
Cursor for walking through the results of a *hscan* query.

fetchall()
Coroutine that reads all the items in one dictionary.

fetchone()
Get next { key: value } tuple It returns *None* after the last item.

class asyncio_redis.cursors.**ZCursor**(*name*, *scanfunc*)
Cursor for walking through the results of a *zscan* query.

Utils

class asyncio_redis.**ZScoreBoundary**(*value*, *exclude_boundary=False*)
Score boundary for a sorted set. for queries like zrangebyscore and similar

Parameters

- **value** (*float*) – Value for the boundary.
- **exclude_boundary** (*bool*) – Exclude the boundary.

class asyncio_redis.**Transaction**(*protocol*)
Transaction context. This is a proxy to a *RedisProtocol* instance. Every redis command called on this object will run inside the transaction. The transaction can be finished by calling either `discard` or `exec`.

More info: http://redis.io/topics/transactions

discard()
Discard all commands issued after MULTI

exec()
Execute transaction.

This can raise a *TransactionError* when the transaction fails.

unwatch()
Forget about all watched keys

class asyncio_redis.**Subscription**(*protocol*)
Pubsub subscription

next_published()
Coroutine which waits for next pubsub message to be received and returns it.

Returns instance of *PubSubReply*

psubscribe(*tr*, *patterns: ListOf(<class 'asyncio_redis.protocol.NativeType'>)*) → NoneType

_psubscribe(self, tr, patterns) Listen for messages published to channels matching the given patterns

Parameters **patterns** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) None

punsubscribe(*tr*, *patterns: ListOf(<class 'asyncio_redis.protocol.NativeType'>)*) → NoneType

_punsubscribe(self, tr, patterns) Stop listening for messages posted to channels matching the given patterns

Parameters **patterns** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) None

subscribe(*tr*, *channels: ListOf(<class 'asyncio_redis.protocol.NativeType'>)*) → NoneType

_subscribe(self, tr, channels) Listen for messages published to the given channels

Parameters **channels** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) None

unsubscribe(*tr*, *channels: ListOf(<class 'asyncio_redis.protocol.NativeType'>)*) → NoneType

_unsubscribe(self, tr, channels) Stop listening for messages posted to the given channels

Parameters **channels** – List or iterable of Native Python type, as defined by *native_type*

Returns (Future of) None

class asyncio_redis.**Script**(*sha*, *code*, *get_evalsha_func*)
Lua script.

run(*keys=[]*, *args=[]*)
Returns a coroutine that executes the script.

```
script_reply = yield from script.run(keys=[], args=[])

# If the LUA script returns something, retrieve the return value
result = yield from script_reply.return_value()
```

This will raise a *ScriptKilledError* exception if the script was killed.

class asyncio_redis.**ZAggregate**
Aggregation method for zinterstore and zunionstore.

Exceptions

class asyncio_redis.exceptions.**TransactionError**
Transaction failed.

class asyncio_redis.exceptions.**NotConnectedError**(*message='Not connected'*)
Protocol is not connected.

class asyncio_redis.exceptions.**TimeoutError**
Timeout during blocking pop.

class asyncio_redis.exceptions.**ConnectionLostError**(*exc*)
Connection lost during query. (Special case of NotConnectedError.)

class asyncio_redis.exceptions.**NoAvailableConnectionsInPoolError**(*message='Not connected'*)
When the connection pool has no available connections.

class asyncio_redis.exceptions.**ScriptKilledError**
Script was killed during an evalsha call.

class asyncio_redis.exceptions.**NoRunningScriptError**
script_kill was called while no script was running.

Index

R

S

T

U

V

W

Z