# tornadis Documentation

***Release 0.8.0***

**Fabien MARTY**

**Mar 09, 2017**

# Contents

CHAPTER 1

# Requirements

- Python 2.7 or Python >= 3.2
- unix operating system (linux, osx...)
- a running redis server (>= 2.0)

CHAPTER 2

# Installation

With pip (without pip see at then end of this document):

```
pip install tornadis
```

# CHAPTER 3

## First try (coroutines)

```python
# Let's import tornado and tornadis
import tornado
import tornadis


@tornado.gen.coroutine
def talk_to_redis():
    # let's (re)connect (autoconnect mode), call the ping redis command
    # and wait the reply without blocking the tornado ioloop
    # Note: call() method on Client instance returns a Future object (and
    # should be used as a coroutine).
    result = yield client.call("PING")
    if isinstance(result, tornadis.TornadisException):
        # For specific reasons, tornadis nearly never raises any exception
        # they are returned as result
        print "got exception: %s" % result
    else:
        # result is already a python object (a string in this simple example)
        print "Result: %s" % result


# Build a tornadis.Client object with some options as kwargs
# host: redis host to connect
# port: redis port to connect
# autoconnect=True: put the Client object in auto(re)connect mode
client = tornadis.Client(host="localhost", port=6379, autoconnect=True)

# Start a tornado IOLoop, execute the coroutine and end the program
loop = tornado.ioloop.IOLoop.instance()
loop.run_sync(talk_to_redis)
```

# CHAPTER 4

## Second try (callbacks)

```python
# Let's import tornado and tornadis
import tornado
import tornadis


def ping_callback(result):
    if not isinstance(result, tornadis.TornadisException):
        # For specific reasons, tornadis nearly never raises any exception
        # they are returned as result
        print "got exception: %s" % result
    else:
        # result is already a python object (a string in this simple example)
        print "Result: %s" % result


@tornado.gen.coroutine
def main():
    # let's (re)connect (autoconnect mode), call the ping redis command
    # and wait the reply without blocking the tornado ioloop
    # Note: async_call() method on Client instance does not return anything
    # but the callback will be called later with the result.
    client.async_call("PING", callback=ping_callback)
    yield tornado.gen.sleep(1)


# Build a tornadis.Client object with some options as kwargs
# host: redis host to connect
# port: redis port to connect
# autoconnect=True: put the Client object in auto(re)connect mode
client = tornadis.Client(host="localhost", port=6379, autoconnect=True)

# Start a tornado IOLoop, execute the coroutine and end the program
loop = tornado.ioloop.IOLoop.instance()
loop.run_sync(main)
```

# CHAPTER 5

## Go further: Pipeline

```python
# Let's import tornado and tornadis
import tornado
import tornadis


@tornado.gen.coroutine
def pipeline_coroutine():
    # Let's make a pipeline object to stack commands inside
    pipeline = tornadis.Pipeline()
    pipeline.stack_call("SET", "foo", "bar")
    pipeline.stack_call("GET", "foo")

    # At this point, nothing is sent to redis

    # let's (re)connect (autoconnect mode), send the pipeline of requests
    # (atomic mode) and wait all replies without blocking the tornado ioloop.
    results = yield client.call(pipeline)

    if isinstance(results, tornadis.TornadisException):
        # For specific reasons, tornadis nearly never raises any exception
        # they are returned as results
        print "got exception: %s" % results
    else:
        # The two replies are in the results array
        print results
        # >>> ['OK', 'bar']


# Build a tornadis.Client object with some options as kwargs
# host: redis host to connect
# port: redis port to connect
# autoconnect=True: put the Client object in auto(re)connect mode
client = tornadis.Client(host="localhost", port=6379, autoconnect=True)

# Start a tornado IOLoop, execute the coroutine and end the program
```

```
loop = tornado.ioloop.IOLoop.instance()
loop.run_sync(pipeline_coroutine)
```

CHAPTER 6

# Installation without pip

- install tornado >= 4.2
- install python wrapper for hiredis
- install six
- download and uncompress a tornadis release
- run `python setup.py install` in the tornadis directory

CHAPTER 7

# API

## Client API

**class** `tornadis.`**`Client`**(*autoconnect=True*, *password=None*, *db=0*, ***connection_kwargs*)
Bases: `object`

High level object to interact with redis.

**Variables**

- **`autoconnect`** (*`boolean`*) – True if the client is in autoconnect mode (and in autoreconnection mode) (default True).
- **`password`** (*`string`*) – the password to authenticate with.
- **`db`** (*`int`*) – database number.
- **`connection_kwargs`** (*`dict`*) – *`Connection`* object kwargs (note that read_callback and close_callback args are set automatically).

**`__init__`**(*autoconnect=True*, *password=None*, *db=0*, ***connection_kwargs*)
Constructor.

**Parameters**

- **autoconnect** (*boolean*) – True if the client is in autoconnect mode (and in autoreconnection mode) (default True).
- **password** (*string*) – the password to authenticate with.
- **db** (*int*) – database number.
- ****connection_kwargs** – *`Connection`* object kwargs.

**`async_call`**(**args*, ***kwargs*)
Calls a redis command, waits for the reply and call a callback.

Following options are available (not part of the redis command itself):

•**callback** Function called (with the result as argument) when the result is available. If not set, the reply is silently discarded. In case of errors, the callback is called with a TornadisException object as argument.

**Parameters**

- ***args** – full redis command as variable length argument list or a Pipeline object (as a single argument).
- ****kwargs** – options as keyword parameters.

### Examples

```
>>> def cb(result):
        pass
>>> client.async_call("HSET", "key", "field", "val", callback=cb)
```

**call**(**args*, ***kwargs*)

Calls a redis command and returns a Future of the reply.

**Parameters**

- ***args** – full redis command as variable length argument list or a Pipeline object (as a single argument).
- ****kwargs** – internal private options (do not use).

**Returns**

**a Future with the decoded redis reply as result (when available) or** a ConnectionError object in case of connection error.

**Raises** *ClientError* – your Pipeline object is empty.

### Examples

```
>>> @tornado.gen.coroutine
    def foobar():
        client = Client()
        result = yield client.call("HSET", "key", "field", "val")
```

**connect**(**args*, ***kwargs*)

Connects the client object to redis.

It's safe to use this method even if you are already connected. Note: this method is useless with autoconnect mode (default).

**Returns** a Future object with True as result if the connection was ok.

**disconnect**()

Disconnects the client object from redis.

It's safe to use this method even if you are already disconnected.

**is_connected**()

Returns True is the client is connected to redis.

**Returns** True if the client if connected to redis.

## PubSubClient API

**class** `tornadis.`**`PubSubClient`**(*autoconnect=True*, *password=None*, *db=0*, ***connection_kwargs*)

Bases: `tornadis.client.Client`

High level specific object to interact with pubsub redis.

The call() method is forbidden with this object.

More informations on the redis side: http://redis.io/topics/pubsub

**`__init__`**(*autoconnect=True*, *password=None*, *db=0*, ***connection_kwargs*)

Constructor.

**Parameters**

- **autoconnect** (*boolean*) – True if the client is in autoconnect mode (and in autoreconnection mode) (default True).
- **password** (*string*) – the password to authenticate with.
- **db** (*int*) – database number.
- ****connection_kwargs** – *`Connection`* object kwargs.

**`async_call`**(**args*, ***kwargs*)

Not allowed method with PubSubClient object.

**`call`**(**args*, ***kwargs*)

Not allowed method with PubSubClient object.

**`connect`**(**args*, ***kwargs*)

Connects the client object to redis.

It's safe to use this method even if you are already connected. Note: this method is useless with autoconnect mode (default).

**Returns** a Future object with True as result if the connection was ok.

**`disconnect`**()

Disconnects the client object from redis.

It's safe to use this method even if you are already disconnected.

**`is_connected`**()

Returns True is the client is connected to redis.

**Returns** True if the client if connected to redis.

**`pubsub_pop_message`**(**args*, ***kwargs*)

Pops a message for a subscribed client.

**Parameters** **deadline** (*int*) – max number of seconds to wait (None => no timeout)

**Returns**

**Future with the popped message as result (or None if timeout** or ConnectionError object in case of connection errors or ClientError object if you are not subscribed)

**`pubsub_psubscribe`**(**args*)

Subscribes to a list of patterns.

http://redis.io/topics/pubsub

**Parameters** ***args** – variable list of patterns to subscribe.

**Returns** Future with True as result if the subscribe is ok.

**Return type** Future

### Examples

```
>>> yield client.pubsub_psubscribe("channel*", "foo*")
```

**pubsub_punsubscribe**(**args*)
Unsubscribes from a list of patterns.

http://redis.io/topics/pubsub

**Parameters** ***args** – variable list of patterns to unsubscribe.

**Returns** Future with True as result if the unsubscribe is ok.

**Return type** Future

### Examples

```
>>> yield client.pubsub_punsubscribe("channel*", "foo*")
```

**pubsub_subscribe**(**args*)
Subscribes to a list of channels.

http://redis.io/topics/pubsub

**Parameters** ***args** – variable list of channels to subscribe.

**Returns** Future with True as result if the subscribe is ok.

**Return type** Future

### Examples

```
>>> yield client.pubsub_subscribe("channel1", "channel2")
```

**pubsub_unsubscribe**(**args*)
Unsubscribes from a list of channels.

http://redis.io/topics/pubsub

**Parameters** ***args** – variable list of channels to unsubscribe.

**Returns** Future with True as result if the unsubscribe is ok.

**Return type** Future

### Examples

```
>>> yield client.pubsub_unsubscribe("channel1", "channel2")
```

# Pipeline API

**class** tornadis.**Pipeline**
Bases: object

Pipeline class to stack redis commands.

A pipeline object is just a kind of stack. You stack complete redis commands (with their corresponding arguments) inside it.

Then, you use the call() method of a Client object to process the pipeline (which must be the only argument of this call() call).

More informations on the redis side: http://redis.io/topics/pipelining

**Variables**

- **pipelined_args** – A list of tuples, earch tuple is a complete redis command.
- **number_of_stacked_calls** – the number of stacked redis commands (integer).

**__init__**()
Constructor.

**stack_call**(**args*)
Stacks a redis command inside the object.

The syntax is the same than the call() method a Client class.

**Parameters** ***args** – full redis command as variable length argument list.

### Examples

```
>>> pipeline = Pipeline()
>>> pipeline.stack_call("HSET", "key", "field", "value")
>>> pipeline.stack_call("PING")
>>> pipeline.stack_call("INCR", "key2")
```

# Pool API

**class** tornadis.**ClientPool**(*max_size=-1*, *client_timeout=-1*, *autoclose=False*, ***client_kwargs*)
Bases: object

High level object to deal with a pool of redis clients.

**__init__**(*max_size=-1*, *client_timeout=-1*, *autoclose=False*, ***client_kwargs*)
Constructor.

**Parameters**

- **max_size** (*int*) – max size of the pool (-1 means "no limit").
- **client_timeout** (*int*) – timeout in seconds of a connection released to the pool (-1 means "no timeout").
- **autoclose** (*boolean*) – automatically disconnect released connections with lifetime > client_timeout (test made every client_timeout/10 seconds).
- **client_kwargs** (*dict*) – Client constructor arguments.

**connected_client**()
Returns a ContextManagerFuture to be yielded in a with statement.

**Returns** A ContextManagerFuture object.

### Examples

```
>>> with (yield pool.connected_client()) as client:
        # client is a connected tornadis.Client instance
        # it will be automatically released to the pool thanks to
        # the "with" keyword
        reply = yield client.call("PING")
```

**destroy**()
Disconnects all pooled client objects.

**get_client_nowait**()
Gets a Client object (not necessary connected).

If max_size is reached, this method will return None (and won't block).

**Returns** A Client instance (not necessary connected) as result (or None).

**get_connected_client**(**args*, ***kwargs*)
Gets a connected Client object.

If max_size is reached, this method will block until a new client object is available.

**Returns**

**A Future object with connected Client instance as a result** (or ClientError if there was a connection problem)

**preconnect**(**args*, ***kwargs*)
(pre)Connects some or all redis clients inside the pool.

**Parameters** **size** (*int*) – number of redis clients to build and to connect (-1 means all clients if pool max_size > -1)

**Raises** *ClientError* – when size == -1 and pool max_size == -1

**release_client**(*client*)
Releases a client object to the pool.

**Parameters** **client** – Client object.

# Exceptions

**class** tornadis.**TornadisException**
Bases: exceptions.Exception

Base Exception class.

**class** tornadis.**ConnectionError**
Bases: tornadis.exceptions.TornadisException

Exception raised when there is a connection error.

**class** tornadis.**ClientError**
Bases: tornadis.exceptions.TornadisException

Exception raised when there is a client error.

## Connection API

Warning: this class is not public, it appears here just to document some kwargs. Do not use directly.

**class** tornadis.**Connection**(*read_callback*, *close_callback*, *host='127.0.0.1'*, *port=6379*, *unix_domain_socket=None*, *read_page_size=65536*, *write_page_size=65536*, *connect_timeout=20*, *tcp_nodelay=False*, *aggressive_write=False*, *read_timeout=0*, *ioloop=None*)
Low level connection object.

**Variables**

- **host** (*string*) – the host name to connect to.
- **port** (*int*) – the port to connect to.
- **unix_domain_socket** (*string*) – path to a unix socket to connect to (if set, overrides host/port parameters).
- **read_page_size** (*int*) – page size for reading.
- **write_page_size** (*int*) – page size for writing.
- **connect_timeout** (*int*) – timeout (in seconds) for connecting.
- **tcp_nodelay** (*boolean*) – set TCP_NODELAY on socket.
- **aggressive_write** (*boolean*) – try to minimize write latency over global throughput (default False).
- **read_timeout** (*int*) – timeout (in seconds) to read something on the socket (if nothing is read during this time, the connection is closed) (default: 0 means no timeout)

**__init__**(*read_callback*, *close_callback*, *host='127.0.0.1'*, *port=6379*, *unix_domain_socket=None*, *read_page_size=65536*, *write_page_size=65536*, *connect_timeout=20*, *tcp_nodelay=False*, *aggressive_write=False*, *read_timeout=0*, *ioloop=None*)
Constructor.

**Parameters**

- **read_callback** – callback called when there is something to read (private, do not use from Client constructor).
- **close_callback** – callback called when the connection is closed (private, do not use from Client constructor).
- **host** (*string*) – the host name to connect to.
- **port** (*int*) – the port to connect to.
- **unix_domain_socket** (*string*) – path to a unix socket to connect to (if set, overrides host/port parameters).
- **read_page_size** (*int*) – page size for reading.
- **write_page_size** (*int*) – page size for writing.
- **connect_timeout** (*int*) – timeout (in seconds) for connecting.

- **tcp_nodelay** (*boolean*) – set TCP_NODELAY on socket.
- **aggressive_write** (*boolean*) – try to minimize write latency over global throughput (default False).
- **read_timeout** (*int*) – timeout (in seconds) to read something on the socket (if nothing is read during this time, the connection is closed) (default: 0 means no timeout)
- **ioloop** (*IOLoop*) – the tornado ioloop to use.

# Python Module Index

## t

# Index

## Symbols



## A



## C



## D



## G



## I



## P



## R



## S



## T