tornadis Documentation

Release 0.7.0

Fabien MARTY

November 15, 2015

Requirements

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

Installation

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

```
pip install tornadis
```

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)
```

Second try (callbacks)

```
1   # Let's import tornado and tornadis
2   import tornado
3   import tornadis
4
5
6   def ping_callback(result):
7       if not isinstance(result, tornadis.TornadisException):
8           # For specific reasons, tornadis nearly never raises any exception
9           # they are returned as result
10          print "got exception: %s" % result
11      else:
12          # result is already a python object (a string in this simple example)
13          print "Result: %s" % result
14
15
16  @tornado.gen.coroutine
17  def main():
18      # let's (re)connect (autoconnect mode), call the ping redis command
19      # and wait the reply without blocking the tornado ioloop
20      # Note: async_call() method on Client instance does not return anything
21      # but the callback will be called later with the result.
22      client.async_call("PING", callback=ping_callback)
23      yield tornado.gen.sleep(1)
24
25
26  # Build a tornadis.Client object with some options as kwargs
27  # host: redis host to connect
28  # port: redis port to connect
29  # autoconnect=True: put the Client object in auto(re)connect mode
30  client = tornadis.Client(host="localhost", port=6379, autoconnect=True)
31
32  # Start a tornado IOLoop, execute the coroutine and end the program
33  loop = tornado.ioloop.IOLoop.instance()
34  loop.run_sync(main)
```

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)
```

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

API

7.1 Client API

class tornadis.**Client** (*autoconnect=True*, *password=None*, ***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.
> >
> > - **connection_kwargs** (*dict*) – *Connection* object kwargs (note that read_callback and close_callback args are set automatically).

 __init__ (*autoconnect=True*, *password=None*, ***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.
> > >
> > > - ****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.

7.2 PubSubClient API

class tornadis.**PubSubClient**(*autoconnect=True*, *password=None*, ***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*, ***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.

- ****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")
```

7.3 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, each 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")
```

7.4 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.

7.5 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*, *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).

__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*, *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).
>
> - **ioloop** (*IOLoop*) – the tornado ioloop to use.

t

Symbols

A

C

D

I

P

S

T