Python Binary Memcached (bmemached) Documentation

Release 0.17

Jayson Reis

Jul 29, 2017

Contents

Contents:

CHAPTER 1

Introduction to bmemcached

A pure python module (thread safe) to access memcached via it's binary with SASL auth support.

The main purpose of this module it to be able to communicate with memcached using binary protocol and support authentication, so it can work with Heroku for example.

Latest compiled docs on Read The Docs here.

1.1 Installing

Use pip or easy_install.

```
pip install python-binary-memcached
```

1.2 Using

```
import bmemcached
client = bmemcached.Client(('127.0.0.1:11211', ), 'user',
                           'password')
client.set('key', 'value')
print client.get('key')
```

1.3 Running the tests

First run memcached with:

```
memcached -S -vvv
memcached -p5000 -S -vvv
memcached -S -s/tmp/memcached.sock -vvv
```

This is to cover all tests with socket, standard port and non standard port.

Then, run the tests.

```
cd src_dir/
py.test
```

1.4 Using with Django

If you want to use it with Django, go to django-bmemcached to get a Django backend.

1.5 Tests Status

CHAPTER 2

bmemcached Package

2.1 bmemcached Package

class bmemcached.__init__.**Client**(*servers=('127.0.0.1:11211',)*, *username=None*, *password=None*, *compression=None*, *socket_timeout=3*, *pickle_protocol=0*, *pickler=<built-in function Pickler>*, *unpickler=<built-in function Unpickler>*)

Bases: object

This is intended to be a client class which implement standard cache interface that common libs do.

Parameters

- **servers** (*list*) – A list of servers with ip[:port] or unix socket.
- **username** (*six.string_types*) – If your server requires SASL authentication, provide the username.
- **password** (*six.string_types*) – If your server requires SASL authentication, provide the password.
- **compression** (*Python module*) – This memcached client uses zlib compression by default, but you can change it to any Python module that provides *compress* and *decompress* functions, such as *bz2*.
- **pickler** (*function*) – Use this to replace the object serialization mechanism.
- **unpickler** (*function*) – Use this to replace the object deserialization mechanism.
- **socket_timeout** (*float*) – The timeout applied to memcached connections.

add(*key*, *value*, *time=0*, *compress_level=-1*)

Add a key/value to server ony if it does not exist.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.

- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True if key is added False if key already exists

Return type bool

cas(*key*, *value*, *cas*, *time=0*, *compress_level=-1*)
Set a value for a key on server if its CAS value matches cas.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True in case of success and False in case of failure

Return type bool

decr(*key*, *value*)
Decrement a key, if it exists, returns it's actual value, if it don't, return 0. Minimum value of decrement return is 0.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*int*) – Number to be decremented

Returns Actual value of the key on server

Return type int

delete(*key*, *cas=0*)
Delete a key/value from server. If key does not exist, it returns True.

Parameters **key** (*six.string_types*) – Key's name to be deleted

Returns True in case o success and False in case of failure.

Return type bool

delete_multi(*keys*)

disconnect_all()
Disconnect all servers.

Returns Nothing

Return type None

enable_retry_delay(*enable*)
Enable or disable delaying between reconnection attempts.

The first reconnection attempt will always happen immediately, so intermittent network errors don't cause caching to turn off. The retry delay takes effect after the first reconnection fails.

The reconnection delay is enabled by default for TCP connections, and disabled by default for Unix socket connections.

flush_all(*time=0*)
Send a command to server flushldelete all keys.

Parameters **time** (*int*) – Time to wait until flush in seconds.

Returns True in case of success, False in case of failure

Return type bool

get(*key*, *get_cas=False*)
Get a key from server.

Parameters

- **key** (*six.string_types*) – Key's name
- **get_cas** (*boolean*) – If true, return (value, cas), where cas is the new CAS value.

Returns Returns a key data from server.

Return type object

get_multi(*keys*, *get_cas=False*)
Get multiple keys from server.

Parameters

- **keys** (*list*) – A list of keys to from server.
- **get_cas** (*boolean*) – If get_cas is true, each value is (data, cas), with each result's CAS value.

Returns A dict with all requested keys.

Return type dict

gets(*key*)
Get a key from server, returning the value and its CAS key.

This method is for API compatibility with other implementations.

Parameters **key** (*six.string_types*) – Key's name

Returns Returns (key data, value), or (None, None) if the value is not in cache.

Return type object

incr(*key*, *value*)
Increment a key, if it exists, returns it's actual value, if it don't, return 0.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*int*) – Number to be incremented

Returns Actual value of the key on server

Return type int

replace(*key*, *value*, *time=0*, *compress_level=-1*)
Replace a key/value to server ony if it does exist.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.

- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True if key is replace False if key does not exists

Return type bool

servers

set(*key*, *value*, *time=0*, *compress_level=-1*)
Set a value for a key on server.

Parameters

- **key** (*str*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True in case of success and False in case of failure

Return type bool

set_multi(*mappings*, *time=0*, *compress_level=-1*)
Set multiple keys with it's values on server.

Parameters

- **mappings** (*dict*) – A dict with keys/values
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True in case of success and False in case of failure

Return type bool

set_servers(*servers*)
Iter to a list of servers and instantiate Protocol class.

Parameters **servers** (*list*) – A list of servers

Returns Returns nothing

Return type None

stats(*key=None*)
Return server stats.

Parameters **key** (*six.string_types*) – Optional if you want status from a key.

Returns A dict with server stats

Return type dict

2.2 client Module

class bmemcached.client.**Client**(*servers=('127.0.0.1:11211',), username=None, password=None, compression=None, socket_timeout=3, pickle_protocol=0, pickler=<built-in function Pickler>, unpickler=<built-in function Unpickler>*)

Bases: object

This is intended to be a client class which implement standard cache interface that common libs do.

Parameters

- **servers** (*list*) – A list of servers with ip[:port] or unix socket.
- **username** (*six.string_types*) – If your server requires SASL authentication, provide the username.
- **password** (*six.string_types*) – If your server requires SASL authentication, provide the password.
- **compression** (*Python module*) – This memcached client uses zlib compression by default, but you can change it to any Python module that provides *compress* and *decompress* functions, such as *bz2*.
- **pickler** (*function*) – Use this to replace the object serialization mechanism.
- **unpickler** (*function*) – Use this to replace the object deserialization mechanism.
- **socket_timeout** (*float*) – The timeout applied to memcached connections.

add(*key*, *value*, *time=0*, *compress_level=-1*)

Add a key/value to server ony if it does not exist.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True if key is added False if key already exists

Return type bool

cas(*key*, *value*, *cas*, *time=0*, *compress_level=-1*)

Set a value for a key on server if its CAS value matches cas.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True in case of success and False in case of failure

Return type bool

decr (*key*, *value*)
Decrement a key, if it exists, returns it's actual value, if it don't, return 0. Minimum value of decrement return is 0.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*int*) – Number to be decremented

Returns Actual value of the key on server

Return type int

delete (*key*, *cas=0*)
Delete a key/value from server. If key does not exist, it returns True.

Parameters **key** (*six.string_types*) – Key's name to be deleted

Returns True in case o success and False in case of failure.

Return type bool

delete_multi (*keys*)

disconnect_all ()
Disconnect all servers.

Returns Nothing

Return type None

enable_retry_delay (*enable*)
Enable or disable delaying between reconnection attempts.

The first reconnection attempt will always happen immediately, so intermittent network errors don't cause caching to turn off. The retry delay takes effect after the first reconnection fails.

The reconnection delay is enabled by default for TCP connections, and disabled by default for Unix socket connections.

flush_all (*time=0*)
Send a command to server flush|delete all keys.

Parameters **time** (*int*) – Time to wait until flush in seconds.

Returns True in case of success, False in case of failure

Return type bool

get (*key*, *get_cas=False*)
Get a key from server.

Parameters

- **key** (*six.string_types*) – Key's name
- **get_cas** (*boolean*) – If true, return (value, cas), where cas is the new CAS value.

Returns Returns a key data from server.

Return type object

get_multi (*keys*, *get_cas=False*)
Get multiple keys from server.

Parameters

- **keys** (*list*) – A list of keys to from server.
- **get_cas** (*boolean*) – If get_cas is true, each value is (data, cas), with each result's CAS value.

Returns A dict with all requested keys.

Return type dict

gets(*key*)
Get a key from server, returning the value and its CAS key.

This method is for API compatibility with other implementations.

Parameters **key** (*six.string_types*) – Key's name

Returns Returns (key data, value), or (None, None) if the value is not in cache.

Return type object

incr(*key*, *value*)
Increment a key, if it exists, returns it's actual value, if it don't, return 0.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*int*) – Number to be incremented

Returns Actual value of the key on server

Return type int

replace(*key*, *value*, *time=0*, *compress_level=-1*)
Replace a key/value to server ony if it does exist.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True if key is replace False if key does not exists

Return type bool

servers

set(*key*, *value*, *time=0*, *compress_level=-1*)
Set a value for a key on server.

Parameters

- **key** (*str*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True in case of success and False in case of failure

Return type bool

set_multi(*mappings*, *time=0*, *compress_level=-1*)
Set multiple keys with it's values on server.

Parameters

- **mappings** (*dict*) – A dict with keys/values
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True in case of success and False in case of failure

Return type bool

set_servers(*servers*)
Iter to a list of servers and instantiate Protocol class.

Parameters **servers** (*list*) – A list of servers

Returns Returns nothing

Return type None

stats(*key=None*)
Return server stats.

Parameters **key** (*six.string_types*) – Optional if you want status from a key.

Returns A dict with server stats

Return type dict

2.3 exceptions Module

exception bmemcached.exceptions.**AuthenticationNotSupported**
Bases: *bmemcached.exceptions.MemcachedException*

exception bmemcached.exceptions.**InvalidCredentials**
Bases: *bmemcached.exceptions.MemcachedException*

exception bmemcached.exceptions.**MemcachedException**
Bases: exceptions.Exception

2.4 protocol Module

class bmemcached.protocol.**Protocol**(*server*, *username=None*, *password=None*, *compression=None*, *socket_timeout=None*, *pickle_protocol=None*, *pickler=None*, *unpickler=None*)
Bases: thread._local

This class is used by Client class to communicate with server.

COMMANDS = {'auth_negotiation': {'command': 32}, 'getkq': {'command': 13, 'struct'

COMPRESSION_THRESHOLD = 128

FLAGS = {'integer': 2, 'object': 1, 'compressed': 8, 'long': 4, 'binary': 16}

HEADER_SIZE = 24

HEADER_STRUCT = '!BBHBBHLLQ'

MAGIC = {'request': 128, 'response': 129}

MAXIMUM_EXPIRE_TIME = 4294967294

STATUS = {'success': 0, 'key_exists': 2, 'server_disconnected': 4294967295, 'auth_e

add(*key*, *value*, *time*, *compress_level=-1*)
Add a key/value to server ony if it does not exist.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True if key is added False if key already exists

Return type bool

authenticate(*username*, *password*)
Authenticate user on server.

Parameters

- **username** (*six.string_types*) – Username used to be authenticated.
- **password** (*six.string_types*) – Password used to be authenticated.

Returns True if successful.

Raises InvalidCredentials, AuthenticationNotSupported, MemcachedException

Return type bool

cas(*key*, *value*, *cas*, *time*, *compress_level=-1*)
Add a key/value to server ony if it does not exist.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True if key is added False if key already exists and has a different CAS

Return type bool

decr(*key*, *value*, *default=0*, *time=100*)
Decrement a key, if it exists, returns its actual value, if it doesn't, return 0. Minimum value of decrement return is 0.

Parameters

- **key** (*six.string_types*) – Key's name

- **value** (*int*) – Number to be decremented
- **default** (*int*) – Default value if key does not exist.
- **time** (*int*) – Time in seconds to expire key.

Returns Actual value of the key on server

Return type int

delete(*key*, *cas=0*)
Delete a key/value from server. If key existed and was deleted, return True.

Parameters

- **key** (*six.string_types*) – Key's name to be deleted
- **cas** (*int*) – If set, only delete the key if its CAS value matches.

Returns True in case o success and False in case of failure.

Return type bool

delete_multi(*keys*)
Delete multiple keys from server in one command.

Parameters **keys** (*list*) – A list of keys to be deleted

Returns True in case of success and False in case of failure.

Return type bool

deserialize(*value*, *flags*)
Deserialized values based on flags or just return it if it is not serialized.

Parameters

- **value** (*six.string_types, int*) – Serialized or not value.
- **flags** (*int*) – Value flags

Returns Deserialized value

Return type six.string_typeslint

disconnect()
Disconnects from server. A new connection will be established the next time a request is made.

Returns Nothing

Return type None

flush_all(*time*)
Send a command to server flushldelete all keys.

Parameters **time** (*int*) – Time to wait until flush in seconds.

Returns True in case of success, False in case of failure

Return type bool

get(*key*)
Get a key and its CAS value from server. If the value isn't cached, return (None, None).

Parameters **key** (*six.string_types*) – Key's name

Returns Returns (value, cas).

Return type object

get_multi(*keys*)
Get multiple keys from server.

Since keys are converted to b" when six.PY3 the keys need to be decoded back into string . e.g key='test' is read as b'test' and then decoded back to 'test' This encode/decode does not work when key is already a six.binary_type hence this function remembers which keys were originally sent as str so that it only decoded those keys back to string which were sent as string

Parameters **keys** (*list*) – A list of keys to from server.

Returns A dict with all requested keys.

Return type dict

incr(*key*, *value*, *default=0*, *time=1000000*)
Increment a key, if it exists, returns its actual value, if it doesn't, return 0.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*int*) – Number to be incremented
- **default** (*int*) – Default value if key does not exist.
- **time** (*int*) – Time in seconds to expire key.

Returns Actual value of the key on server

Return type int

replace(*key*, *value*, *time*, *compress_level=-1*)
Replace a key/value to server ony if it does exist.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True if key is replace False if key does not exists

Return type bool

serialize(*value*, *compress_level=-1*)
Serializes a value based on its type.

Parameters

- **value** (*six.string_types, int, long, object*) – Something to be serialized
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns Serialized type

Return type str

server_uses_unix_socket

set(*key*, *value*, *time*, *compress_level=-1*)
Set a value for a key on server.

Parameters

- **key** (*six.string_types*) – Key's name
- **value** (*object*) – A value to be stored on server.
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True in case of success and False in case of failure

Return type bool

set_multi(*mappings*, *time=100*, *compress_level=-1*)
Set multiple keys with its values on server.

If a key is a (key, cas) tuple, insert as if cas(key, value, cas) had been called.

Parameters

- **mappings** (*dict*) – A dict with keys/values
- **time** (*int*) – Time in seconds that your key will expire.
- **compress_level** (*int*) – How much to compress. 0 = no compression, 1 = fastest, 9 = slowest but best, -1 = default compression level.

Returns True

Return type bool

set_retry_delay(*value*)

classmethod split_host_port(*server*)
Return (host, port) from server.

Port defaults to 11211.

```
>>> split_host_port('127.0.0.1:11211')
('127.0.0.1', 11211)
>>> split_host_port('127.0.0.1')
('127.0.0.1', 11211)
```

stats(*key=None*)
Return server stats.

Parameters **key** (*six.string_types*) – Optional if you want status from a key.

Returns A dict with server stats

Return type dict

CHAPTER 3

bmemcached

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

P

R

S