

---

# **pymemcache Documentation**

***Release 1.4.0***

**Charles Gordon, Nicholas Charriere**

**Sep 16, 2018**



<b>1</b>	<b>Getting started!</b>	<b>3</b>
1.1	Basic Usage . . . . .	3
1.2	Using a memcached cluster . . . . .	3
1.3	Serialization . . . . .	3
1.4	Deserialization with python3 . . . . .	4
1.5	Key Constraints . . . . .	4
1.6	Best Practices . . . . .	5
<b>2</b>	<b>pymemcache</b>	<b>7</b>
2.1	pymemcache package . . . . .	7
<b>3</b>	<b>Indices and tables</b>	<b>19</b>
	<b>Python Module Index</b>	<b>21</b>



Contents:



---

## Getting started!

---

A comprehensive, fast, pure-Python memcached client library.

### Basic Usage

```
from pymemcache.client.base import Client

client = Client(('localhost', 11211))
client.set('some_key', 'some_value')
result = client.get('some_key')
```

### Using a memcached cluster

This will use a consistent hashing algorithm to choose which server to set/get the values from. It will also automatically rebalance depending on if a server goes down.

```
from pymemcache.client.hash import HashClient

client = HashClient([
    ('127.0.0.1', 11211),
    ('127.0.0.1', 11212)
])
client.set('some_key', 'some value')
result = client.get('some_key')
```

### Serialization

```
import json
from pymemcache.client.base import Client

def json_serializer(key, value):
    if type(value) == str:
        return value, 1
    return json.dumps(value), 2

def json_deserializer(key, value, flags):
```

```
if flags == 1:
    return value
if flags == 2:
    return json.loads(value)
raise Exception("Unknown serialization format")

client = Client(('localhost', 11211), serializer=json_serializer,
               deserializer=json_deserializer)
client.set('key', {'a': 'b', 'c': 'd'})
result = client.get('key')
```

pymemcache provides a default pickle-based serializer:

```
from pymemcache.client.base import Client
from pymemcache import serde

class Foo(object):
    pass

client = Client(('localhost', 11211),
               serializer=serde.python_memcache_serializer,
               deserializer=serde.python_memcache_deserializer)
client.set('key', Foo())
result = client.get('key')
```

The serializer uses the highest pickle protocol available. In order to make sure multiple versions of Python can read the protocol version, you can specify the version with `get_python_memcache_serializer`

```
client = Client(('localhost', 11211),
               serializer=serde.get_python_memcache_serializer(pickle_version=2),
               deserializer=serde.python_memcache_deserializer)
```

## Deserialization with python3

```
def json_deserializer(key, value, flags):
    if flags == 1:
        return value.decode('utf-8')
    if flags == 2:
        return json.loads(value.decode('utf-8'))
    raise Exception("Unknown serialization format")
```

## Key Constraints

This client implements the ASCII protocol of memcached. This means keys should not contain any of the following illegal characters: > Keys cannot have spaces, new lines, carriage returns, or null characters. We suggest that if you have unicode characters, or long keys, you use an effective hashing mechanism before calling this client. At Pinterest, we have found that murmur3 hash is a great candidate for this. Alternatively you can set *allow\_unicode\_keys* to support unicode keys, but beware of what unicode encoding you use to make sure multiple clients can find the same key.



## Best Practices

- Always set the `connect_timeout` and `timeout` arguments in the `pymemcache.client.base.Client` constructor to avoid blocking your process when memcached is slow. You might also want to enable the `no_delay` option, which sets the `TCP_NODELAY` flag on the connection's socket.
- Use the “noreply” flag for a significant performance boost. The “noreply” flag is enabled by default for “set”, “add”, “replace”, “append”, “prepend”, and “delete”. It is disabled by default for “cas”, “incr” and “decr”. It obviously doesn't apply to any get calls.
- Use `get_many` and `gets_many` whenever possible, as they result in less round trip times for fetching multiple keys.
- Use the “ignore\_exc” flag to treat memcache/network errors as cache misses on calls to the `get*` methods. This prevents failures in memcache, or network errors, from killing your web requests. Do not use this flag if you need to know about errors from memcache, and make sure you have some other way to detect memcache server failures.



---

## pymemcache

---

### pymemcache package

#### Subpackages

##### pymemcache.client package

#### Submodules

##### pymemcache.client.base module

```
class pymemcache.client.base.Client(server,    serializer=None,    deserializer=None,    connect_timeout=None,    timeout=None,    no_delay=False,    ignore_exc=False,    socket_module=<module 'socket' from '/usr/lib/python2.7/socket.pyc'>,    key_prefix='',    default_noreply=True,    allow_unicode_keys=False)
```

Bases: `object`

A client for a single memcached server.

#### *Keys and Values*

Keys must have a `__str__()` method which should return a str with no more than 250 ASCII characters and no whitespace or control characters. Unicode strings must be encoded (as UTF-8, for example) unless they consist only of ASCII characters that are neither whitespace nor control characters.

Values must have a `__str__()` method to convert themselves to a byte string. Unicode objects can be a problem since `str()` on a Unicode object will attempt to encode it as ASCII (which will fail if the value contains code points larger than U+127). You can fix this with a serializer or by just calling `encode` on the string (using UTF-8, for instance).

If you intend to use anything but str as a value, it is a good idea to use a serializer and deserializer. The `pymemcache.serde` library has some already implemented serializers, including one that is compatible with the `python-memcache` library.

#### *Serialization and Deserialization*

The constructor takes two optional functions, one for “serialization” of values, and one for “deserialization”. The serialization function takes two arguments, a key and a value, and returns a tuple of two elements, the serialized value, and an integer in the range 0-65535 (the “flags”). The deserialization function takes three parameters, a key, value and flags and returns the deserialized value.

Here is an example using JSON for non-str values:

```
def serialize_json(key, value):
    if type(value) == str:
        return value, 1
    return json.dumps(value), 2

def deserialize_json(key, value, flags):
    if flags == 1:
        return value

    if flags == 2:
        return json.loads(value)

    raise Exception("Unknown flags for value: {1}".format(flags))
```

### Error Handling

All of the methods in this class that talk to memcached can throw one of the following exceptions:

- `MemcacheUnknownCommandError`
- `MemcacheClientError`
- `MemcacheServerError`
- `MemcacheUnknownError`
- `MemcacheUnexpectedCloseError`
- `MemcacheIllegalInputError`
- `socket.timeout`
- `socket.error`

Instances of this class maintain a persistent connection to memcached which is terminated when any of these exceptions are raised. The next call to a method on the object will result in a new connection being made to memcached.

**add** (*key*, *value*, *expire*=0, *noreply*=None)

The memcached “add” command.

#### Parameters

- **key** – str, see class docs for details.
- **value** – str, see class docs for details.
- **expire** – optional int, number of seconds until the item is expired from the cache, or zero for no expiry (the default).
- **noreply** – optional bool, True to not wait for the reply (defaults to `self.default_noreply`).

**Returns** If `noreply` is True, the return value is always True. Otherwise the return value is True if the value was stored, and False if it was not (because the key already existed).

**append** (*key*, *value*, *expire*=0, *noreply*=None)

The memcached “append” command.

#### Parameters

- **key** – str, see class docs for details.
- **value** – str, see class docs for details.

- **expire** – optional int, number of seconds until the item is expired from the cache, or zero for no expiry (the default).
- **noreply** – optional bool, True to not wait for the reply (defaults to self.default\_noreply).

**Returns** True.

**cas** (*key, value, cas, expire=0, noreply=False*)

The memcached “cas” command.

**Parameters**

- **key** – str, see class docs for details.
- **value** – str, see class docs for details.
- **cas** – int or str that only contains the characters ‘0’-‘9’.
- **expire** – optional int, number of seconds until the item is expired from the cache, or zero for no expiry (the default).
- **noreply** – optional bool, False to wait for the reply (the default).

**Returns** If noreply is True, always returns True. Otherwise returns None if the key didn’t exist, False if it existed but had a different cas value and True if it existed and was changed.

**check\_key** (*key*)

Checks key and add key\_prefix.

**close** ()

Close the connection to memcached, if it is open. The next call to a method that requires a connection will re-open it.

**decr** (*key, value, noreply=False*)

The memcached “decr” command.

**Parameters**

- **key** – str, see class docs for details.
- **value** – int, the amount by which to increment the value.
- **noreply** – optional bool, False to wait for the reply (the default).

**Returns** If noreply is True, always returns None. Otherwise returns the new value of the key, or None if the key wasn’t found.

**delete** (*key, noreply=None*)

The memcached “delete” command.

**Parameters**

- **key** – str, see class docs for details.
- **noreply** – optional bool, True to not wait for the reply (defaults to self.default\_noreply).

**Returns** If noreply is True, always returns True. Otherwise returns True if the key was deleted, and False if it wasn’t found.

**delete\_many** (*keys, noreply=None*)

A convenience function to delete multiple keys.

**Parameters**

- **keys** – list(str), the list of keys to delete.
- **noreply** – optional bool, True to not wait for the reply (defaults to self.default\_noreply).

**Returns** True. If an exception is raised then all, some or none of the keys may have been deleted. Otherwise all the keys have been sent to memcache for deletion and if `noreply` is False, they have been acknowledged by memcache.

**delete\_multi** (*keys, noreply=None*)

A convenience function to delete multiple keys.

**Parameters**

- **keys** – list(str), the list of keys to delete.
- **noreply** – optional bool, True to not wait for the reply (defaults to `self.default_noreply`).

**Returns** True. If an exception is raised then all, some or none of the keys may have been deleted. Otherwise all the keys have been sent to memcache for deletion and if `noreply` is False, they have been acknowledged by memcache.

**flush\_all** (*delay=0, noreply=None*)

The memcached “flush\_all” command.

**Parameters**

- **delay** – optional int, the number of seconds to wait before flushing, or zero to flush immediately (the default).
- **noreply** – optional bool, True to not wait for the reply (defaults to `self.default_noreply`).

**Returns** True.

**get** (*key, default=None*)

The memcached “get” command, but only for one key, as a convenience.

**Parameters**

- **key** – str, see class docs for details.
- **default** – value that will be returned if the key was not found.

**Returns** The value for the key, or default if the key wasn’t found.

**get\_many** (*keys*)

The memcached “get” command.

**Parameters** **keys** – list(str), see class docs for details.

**Returns** A dict in which the keys are elements of the “keys” argument list and the values are values from the cache. The dict may contain all, some or none of the given keys.

**get\_multi** (*keys*)

The memcached “get” command.

**Parameters** **keys** – list(str), see class docs for details.

**Returns** A dict in which the keys are elements of the “keys” argument list and the values are values from the cache. The dict may contain all, some or none of the given keys.

**gets** (*key, default=None, cas\_default=None*)

The memcached “gets” command for one key, as a convenience.

**Parameters**

- **key** – str, see class docs for details.
- **default** – value that will be returned if the key was not found.
- **cas\_default** – same behaviour as default argument.

**Returns** A tuple of (value, cas) or (default, cas\_defaults) if the key was not found.

**gets\_many** (*keys*)

The memcached “gets” command.

**Parameters** **keys** – list(str), see class docs for details.

**Returns** A dict in which the keys are elements of the “keys” argument list and the values are tuples of (value, cas) from the cache. The dict may contain all, some or none of the given keys.

**incr** (*key, value, noreply=False*)

The memcached “incr” command.

**Parameters**

- **key** – str, see class docs for details.
- **value** – int, the amount by which to increment the value.
- **noreply** – optional bool, False to wait for the reply (the default).

**Returns** If noreply is True, always returns None. Otherwise returns the new value of the key, or None if the key wasn’t found.

**prepend** (*key, value, expire=0, noreply=None*)

The memcached “prepend” command.

**Parameters**

- **key** – str, see class docs for details.
- **value** – str, see class docs for details.
- **expire** – optional int, number of seconds until the item is expired from the cache, or zero for no expiry (the default).
- **noreply** – optional bool, True to not wait for the reply (defaults to self.default\_noreply).

**Returns** True.

**quit** ()

The memcached “quit” command.

This will close the connection with memcached. Calling any other method on this object will re-open the connection, so this object can be re-used after quit.

**replace** (*key, value, expire=0, noreply=None*)

The memcached “replace” command.

**Parameters**

- **key** – str, see class docs for details.
- **value** – str, see class docs for details.
- **expire** – optional int, number of seconds until the item is expired from the cache, or zero for no expiry (the default).
- **noreply** – optional bool, True to not wait for the reply (defaults to self.default\_noreply).

**Returns** If noreply is True, always returns True. Otherwise returns True if the value was stored and False if it wasn’t (because the key didn’t already exist).

**set** (*key, value, expire=0, noreply=None*)

The memcached “set” command.

**Parameters**

- **key** – str, see class docs for details.
- **value** – str, see class docs for details.
- **expire** – optional int, number of seconds until the item is expired from the cache, or zero for no expiry (the default).
- **noreply** – optional bool, True to not wait for the reply (defaults to self.default\_noreply).

**Returns** If no exception is raised, always returns True. If an exception is raised, the set may or may not have occurred. If noreply is True, then a successful return does not guarantee a successful set.

**set\_many** (*values*, *expire=0*, *noreply=None*)

A convenience function for setting multiple values.

**Parameters**

- **values** – dict(str, str), a dict of keys and values, see class docs for details.
- **expire** – optional int, number of seconds until the item is expired from the cache, or zero for no expiry (the default).
- **noreply** – optional bool, True to not wait for the reply (defaults to self.default\_noreply).

**Returns** Returns a list of keys that failed to be inserted. If noreply is True, always returns empty list.

**set\_multi** (*values*, *expire=0*, *noreply=None*)

A convenience function for setting multiple values.

**Parameters**

- **values** – dict(str, str), a dict of keys and values, see class docs for details.
- **expire** – optional int, number of seconds until the item is expired from the cache, or zero for no expiry (the default).
- **noreply** – optional bool, True to not wait for the reply (defaults to self.default\_noreply).

**Returns** Returns a list of keys that failed to be inserted. If noreply is True, always returns empty list.

**stats** (\*args)

The memcached “stats” command.

The returned keys depend on what the “stats” command returns. A best effort is made to convert values to appropriate Python types, defaulting to strings when a conversion cannot be made.

**Parameters** **\*arg** – extra string arguments to the “stats” command. See the memcached protocol documentation for more information.

**Returns** A dict of the returned stats.

**touch** (*key*, *expire=0*, *noreply=None*)

The memcached “touch” command.

**Parameters**

- **key** – str, see class docs for details.
- **expire** – optional int, number of seconds until the item is expired from the cache, or zero for no expiry (the default).
- **noreply** – optional bool, True to not wait for the reply (defaults to self.default\_noreply).



**Returns** True if the expiration time was updated, False if the key wasn't found.

**version()**

The memcached "version" command.

**Returns** A string of the memcached version.

```
class pymemcache.client.base.PooledClient(server, serializer=None, deserializer=None,
                                         connect_timeout=None, timeout=None,
                                         no_delay=False, ignore_exc=False,
                                         socket_module=<module 'socket' from
                                         '/usr/lib/python2.7/socket.pyc'>, key_prefix='',
                                         max_pool_size=None, lock_generator=None, de-
                                         fault_noreply=True, allow_unicode_keys=False)
```

Bases: `object`

A thread-safe pool of clients (with the same client api).

#### Parameters

- **max\_pool\_size** – maximum pool size to use (going above this amount triggers a runtime error), by default this is 2147483648L when not provided (or none).
- **lock\_generator** – a callback/type that takes no arguments that will be called to create a lock or semaphore that can protect the pool from concurrent access (for example a eventlet lock or semaphore could be used instead)

Further arguments are interpreted as for `Client` constructor.

**add** (key, value, expire=0, noreply=None)

**append** (key, value, expire=0, noreply=None)

**cas** (key, value, cas, expire=0, noreply=False)

**check\_key** (key)

Checks key and add key\_prefix.

**close** ()

**decr** (key, value, noreply=False)

**delete** (key, noreply=None)

**delete\_many** (keys, noreply=None)

**delete\_multi** (keys, noreply=None)

**flush\_all** (delay=0, noreply=None)

**get** (key, default=None)

**get\_many** (keys)

**get\_multi** (keys)

**gets** (key)

**gets\_many** (keys)

**incr** (key, value, noreply=False)

**prepend** (key, value, expire=0, noreply=None)

**quit** ()

**replace** (key, value, expire=0, noreply=None)

```
set (key, value, expire=0, noreply=None)
set_many (values, expire=0, noreply=None)
set_multi (values, expire=0, noreply=None)
stats (*args)
touch (key, expire=0, noreply=None)
version ()
```

## pymemcache.client.hash module

```
class pymemcache.client.hash.HashClient (servers, hasher=<class 'pymem-  
cache.client.rendezvous.RendezvousHash'>, serializer=None, deserializer=None, connect_timeout=None, timeout=None, no_delay=False, socket_module=<module 'socket' from  
'/usr/lib/python2.7/socket.pyc'>, key_prefix='', max_pool_size=None, lock_generator=None, retry_attempts=2, retry_timeout=1, dead_timeout=60, use_pooling=False, ignore_exc=False, allow_unicode_keys=False)
```

Bases: `object`

A client for communicating with a cluster of memcached servers

```
add (key, *args, **kwargs)
add_server (server, port)
append (key, *args, **kwargs)
cas (key, *args, **kwargs)
decr (key, *args, **kwargs)
delete (key, *args, **kwargs)
delete_many (keys, *args, **kwargs)
delete_multi (keys, *args, **kwargs)
flush_all ()
get (key, *args, **kwargs)
get_many (keys, gets=False, *args, **kwargs)
get_multi (keys, gets=False, *args, **kwargs)
gets (key, *args, **kwargs)
gets_many (keys, *args, **kwargs)
gets_multi (keys, *args, **kwargs)
incr (key, *args, **kwargs)
prepend (key, *args, **kwargs)
remove_server (server, port)
replace (key, *args, **kwargs)
```

```
set (key, *args, **kwargs)  
set_many (values, *args, **kwargs)  
set_multi (values, *args, **kwargs)
```

## pymemcache.client.murmur3 module

`pymemcache.client.murmur3.murmur3_32` (*data*, *seed*=0)

MurmurHash3 was written by Austin Appleby, and is placed in the public domain. The author hereby disclaims copyright to this source code.

## pymemcache.client.rendezvous module

```
class pymemcache.client.rendezvous.RendezvousHash (nodes=None, seed=0,  
                                                hash_function=<function mur-  
                                                mur3_32>)
```

Bases: `object`

Implements the Highest Random Weight (HRW) hashing algorithm most commonly referred to as rendezvous hashing.

Originally developed as part of python-clandestined.

Copyright (c) 2014 Ernest W. Durbin III

**add\_node** (*node*)

**get\_node** (*key*)

**remove\_node** (*node*)

## Module contents

### pymemcache.test package

#### Submodules

#### pymemcache.test.conftest module

#### pymemcache.test.test\_benchmark module

#### pymemcache.test.test\_client module

#### pymemcache.test.test\_client\_hash module

#### pymemcache.test.test\_integration module

#### pymemcache.test.test\_rendezvous module

#### pymemcache.test.test\_serde module

## pymemcache.test.test\_utils module

### pymemcache.test.utils module

Useful testing utilities.

This module is considered public API.

```
class pymemcache.test.utils.MockMemcacheClient (server=None, serializer=None, deseri-  
alizer=None, connect_timeout=None,  
timeout=None, no_delay=False, ig-  
nore_exc=False, default_noreply=True,  
allow_unicode_keys=False)
```

Bases: `object`

A (partial) in-memory mock for Clients.

**add** (key, value, expire=None, noreply=True)

**decr** (key, value, noreply=False)

**delete** (key, noreply=True)

**delete\_many** (keys, noreply=True)

**delete\_multi** (keys, noreply=True)

**get** (key, default=None)

**get\_many** (keys)

**get\_multi** (keys)

**incr** (key, value, noreply=False)

**set** (key, value, expire=0, noreply=True)

**set\_many** (values, expire=None, noreply=True)

**set\_multi** (values, expire=None, noreply=True)

**stats** ()

## Module contents

## Submodules

### pymemcache.exceptions module

**exception** pymemcache.exceptions.**MemcacheClientError**

Bases: `pymemcache.exceptions.MemcacheError`

Raised when memcached fails to parse the arguments to a request, likely due to a malformed key and/or value, a bug in this library, or a version mismatch with memcached.

**exception** pymemcache.exceptions.**MemcacheError**

Bases: `exceptions.Exception`

Base exception class

**exception** `pymemcache.exceptions.MemcacheIllegalInputError`

Bases: `pymemcache.exceptions.MemcacheClientError`

Raised when a key or value is not legal for Memcache (see the class docs for Client for more details).

**exception** `pymemcache.exceptions.MemcacheServerError`

Bases: `pymemcache.exceptions.MemcacheError`

Raised when memcached reports a failure while processing a request, likely due to a bug or transient issue in memcached.

**exception** `pymemcache.exceptions.MemcacheUnexpectedCloseError`

Bases: `pymemcache.exceptions.MemcacheServerError`

Raised when the connection with memcached closes unexpectedly.

**exception** `pymemcache.exceptions.MemcacheUnknownCommandError`

Bases: `pymemcache.exceptions.MemcacheClientError`

Raised when memcached fails to parse a request, likely due to a bug in this library or a version mismatch with memcached.

**exception** `pymemcache.exceptions.MemcacheUnknownError`

Bases: `pymemcache.exceptions.MemcacheError`

Raised when this library receives a response from memcached that it cannot parse, likely due to a bug in this library or a version mismatch with memcached.

## pymemcache.fallback module

A client for falling back to older memcached servers when performing reads.

It is sometimes necessary to deploy memcached on new servers, or with a different configuration. In these cases, it is undesirable to start up an empty memcached server and point traffic to it, since the cache will be cold, and the backing store will have a large increase in traffic.

This class attempts to solve that problem by providing an interface identical to the Client interface, but which can fall back to older memcached servers when reads to the primary server fail. The approach for upgrading memcached servers or configuration then becomes:

1. Deploy a new host (or fleet) with memcached, possibly with a new configuration.
2. From your application servers, use FallbackClient to write and read from the new cluster, and to read from the old cluster when there is a miss in the new cluster.
3. Wait until the new cache is warm enough to support the load.
4. Switch from FallbackClient to a regular Client library for doing all reads and writes to the new cluster.
5. Take down the old cluster.

## Best Practices:

- Make sure that the old client has “ignore\_exc” set to True, so that it treats failures like cache misses. That will allow you to take down the old cluster before you switch away from FallbackClient.

**class** `pymemcache.fallback.FallbackClient` (*caches*)

Bases: `object`

**add** (*key*, *value*, *expire*=0, *noreply*=True)

**append** (*key*, *value*, *expire*=0, *noreply*=True)

**cas** (*key, value, cas, expire=0, noreply=True*)

**close** ()  
Close each of the memcached clients

**decr** (*key, value, noreply=True*)

**delete** (*key, noreply=True*)

**flush\_all** (*delay=0, noreply=True*)

**get** (*key*)

**get\_many** (*keys*)

**gets** (*key*)

**gets\_many** (*keys*)

**incr** (*key, value, noreply=True*)

**prepend** (*key, value, expire=0, noreply=True*)

**quit** ()

**replace** (*key, value, expire=0, noreply=True*)

**set** (*key, value, expire=0, noreply=True*)

**stats** ()

**touch** (*key, expire=0, noreply=True*)

## pymemcache.pool module

**class** `pymemcache.pool.ObjectPool` (*obj\_creator, after\_remove=None, max\_size=None, lock\_generator=None*)

Bases: `object`

A pool of objects that release/creates/destroys as needed.

**clear** ()

**destroy** (*obj, silent=True*)

**free**

**get** ()

**get\_and\_release** (*\*args, \*\*kws*)

**release** (*obj, silent=True*)

**used**

## pymemcache.serde module

`pymemcache.serde.get_python_memcache_serializer` (*pickle\_version=2*)  
Return a serializer using a specific pickle version

`pymemcache.serde.python_memcache_deserializer` (*key, value, flags*)

## Module contents

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`





## p

- `pymemcache`, 18
  - `pymemcache.client`, 15
    - `pymemcache.client.base`, 7
    - `pymemcache.client.hash`, 14
    - `pymemcache.client.murmur3`, 15
    - `pymemcache.client.rendezvous`, 15
  - `pymemcache.exceptions`, 16
  - `pymemcache.fallback`, 17
  - `pymemcache.pool`, 18
  - `pymemcache.serde`, 18
  - `pymemcache.test`, 16
    - `pymemcache.test.utils`, 16



## A

[add\(\)](#) ([pymemcache.client.base.Client](#) method), 8  
[add\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[add\(\)](#) ([pymemcache.client.hash.HashClient](#) method), 14  
[add\(\)](#) ([pymemcache.fallback.FallbackClient](#) method), 17  
[add\(\)](#) ([pymemcache.test.utils.MockMemcacheClient](#) method), 16  
[add\\_node\(\)](#) ([pymemcache.client.rendezvous.RendezvousHash](#) method), 15  
[add\\_server\(\)](#) ([pymemcache.client.hash.HashClient](#) method), 14  
[append\(\)](#) ([pymemcache.client.base.Client](#) method), 8  
[append\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[append\(\)](#) ([pymemcache.client.hash.HashClient](#) method), 14  
[append\(\)](#) ([pymemcache.fallback.FallbackClient](#) method), 17

## C

[cas\(\)](#) ([pymemcache.client.base.Client](#) method), 9  
[cas\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[cas\(\)](#) ([pymemcache.client.hash.HashClient](#) method), 14  
[cas\(\)](#) ([pymemcache.fallback.FallbackClient](#) method), 17  
[check\\_key\(\)](#) ([pymemcache.client.base.Client](#) method), 9  
[check\\_key\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[clear\(\)](#) ([pymemcache.pool.ObjectPool](#) method), 18  
[Client](#) (class in [pymemcache.client.base](#)), 7  
[close\(\)](#) ([pymemcache.client.base.Client](#) method), 9  
[close\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[close\(\)](#) ([pymemcache.fallback.FallbackClient](#) method), 18

## D

[decr\(\)](#) ([pymemcache.client.base.Client](#) method), 9  
[decr\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[decr\(\)](#) ([pymemcache.client.hash.HashClient](#) method), 14

[decr\(\)](#) ([pymemcache.fallback.FallbackClient](#) method), 18  
[decr\(\)](#) ([pymemcache.test.utils.MockMemcacheClient](#) method), 16  
[delete\(\)](#) ([pymemcache.client.base.Client](#) method), 9  
[delete\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[delete\(\)](#) ([pymemcache.client.hash.HashClient](#) method), 14  
[delete\(\)](#) ([pymemcache.fallback.FallbackClient](#) method), 18  
[delete\(\)](#) ([pymemcache.test.utils.MockMemcacheClient](#) method), 16  
[delete\\_many\(\)](#) ([pymemcache.client.base.Client](#) method), 9  
[delete\\_many\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[delete\\_many\(\)](#) ([pymemcache.client.hash.HashClient](#) method), 14  
[delete\\_many\(\)](#) ([pymemcache.test.utils.MockMemcacheClient](#) method), 16  
[delete\\_multi\(\)](#) ([pymemcache.client.base.Client](#) method), 10  
[delete\\_multi\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[delete\\_multi\(\)](#) ([pymemcache.client.hash.HashClient](#) method), 14  
[delete\\_multi\(\)](#) ([pymemcache.test.utils.MockMemcacheClient](#) method), 16  
[destroy\(\)](#) ([pymemcache.pool.ObjectPool](#) method), 18

## F

[FallbackClient](#) (class in [pymemcache.fallback](#)), 17  
[flush\\_all\(\)](#) ([pymemcache.client.base.Client](#) method), 10  
[flush\\_all\(\)](#) ([pymemcache.client.base.PooledClient](#) method), 13  
[flush\\_all\(\)](#) ([pymemcache.client.hash.HashClient](#) method), 14  
[flush\\_all\(\)](#) ([pymemcache.fallback.FallbackClient](#) method), 18

`free` (pymemcache.pool.ObjectPool attribute), [18](#)

## G

`get()` (pymemcache.client.base.Client method), [10](#)

`get()` (pymemcache.client.base.PooledClient method), [13](#)

`get()` (pymemcache.client.hash.HashClient method), [14](#)

`get()` (pymemcache.fallback.FallbackClient method), [18](#)

`get()` (pymemcache.pool.ObjectPool method), [18](#)  
`get()` (pymemcache.test.utils.MockMemcacheClient method), [16](#)

`get_and_release()` (pymemcache.pool.ObjectPool method), [18](#)

`get_many()` (pymemcache.client.base.Client method), [10](#)

`get_many()` (pymemcache.client.base.PooledClient method), [13](#)

`get_many()` (pymemcache.client.hash.HashClient method), [14](#)

`get_many()` (pymemcache.fallback.FallbackClient method), [18](#)

`get_many()` (pymemcache.test.utils.MockMemcacheClient method), [16](#)

`get_multi()` (pymemcache.client.base.Client method), [10](#)

`get_multi()` (pymemcache.client.base.PooledClient method), [13](#)

`get_multi()` (pymemcache.client.hash.HashClient method), [14](#)

`get_multi()` (pymemcache.test.utils.MockMemcacheClient method), [16](#)

`get_node()` (pymemcache.client.rendezvous.RendezvousHash method), [15](#)

`get_python_memcache_serializer()` (in module pymemcache.serde), [18](#)

`gets()` (pymemcache.client.base.Client method), [10](#)

`gets()` (pymemcache.client.base.PooledClient method), [13](#)

`gets()` (pymemcache.client.hash.HashClient method), [14](#)

`gets()` (pymemcache.fallback.FallbackClient method), [18](#)

`gets_many()` (pymemcache.client.base.Client method), [11](#)

`gets_many()` (pymemcache.client.base.PooledClient method), [13](#)

`gets_many()` (pymemcache.client.hash.HashClient method), [14](#)

`gets_many()` (pymemcache.fallback.FallbackClient method), [18](#)

`gets_multi()` (pymemcache.client.hash.HashClient method), [14](#)

## H

`HashClient` (class in pymemcache.client.hash), [14](#)

## I

`incr()` (pymemcache.client.base.Client method), [11](#)

`incr()` (pymemcache.client.base.PooledClient method), [13](#)

`incr()` (pymemcache.client.hash.HashClient method), [14](#)

`incr()` (pymemcache.fallback.FallbackClient method), [18](#)

`incr()` (pymemcache.test.utils.MockMemcacheClient method), [16](#)

## M

`MemcacheClientError`, [16](#)

`MemcacheError`, [16](#)

`MemcacheIllegalInputError`, [16](#)

`MemcacheServerError`, [17](#)

`MemcacheUnexpectedCloseError`, [17](#)

`MemcacheUnknownCommandError`, [17](#)

`MemcacheUnknownError`, [17](#)

`MockMemcacheClient` (class in pymemcache.test.utils), [16](#)

`murmur3_32()` (in module pymemcache.client.murmur3), [15](#)

## O

`ObjectPool` (class in pymemcache.pool), [18](#)

## P

`PooledClient` (class in pymemcache.client.base), [13](#)

`prepend()` (pymemcache.client.base.Client method), [11](#)

`prepend()` (pymemcache.client.base.PooledClient method), [13](#)

`prepend()` (pymemcache.client.hash.HashClient method), [14](#)

`prepend()` (pymemcache.fallback.FallbackClient method), [18](#)

`pymemcache` (module), [18](#)

`pymemcache.client` (module), [15](#)

`pymemcache.client.base` (module), [7](#)

`pymemcache.client.hash` (module), [14](#)

`pymemcache.client.murmur3` (module), [15](#)

`pymemcache.client.rendezvous` (module), [15](#)

`pymemcache.exceptions` (module), [16](#)

`pymemcache.fallback` (module), [17](#)

`pymemcache.pool` (module), [18](#)

`pymemcache.serde` (module), [18](#)

`pymemcache.test` (module), [16](#)

`pymemcache.test.utils` (module), [16](#)

`python_memcache_deserializer()` (in module pymemcache.serde), [18](#)

## Q

`quit()` (pymemcache.client.base.Client method), [11](#)

`quit()` (pymemcache.client.base.PooledClient method), [13](#)

`quit()` (pymemcache.fallback.FallbackClient method), [18](#)

## R

`release()` (pymemcache.pool.ObjectPool method), [18](#)

`remove_node()` (pymemcache.client.rendezvous.RendezvousHash method), [15](#)

remove\_server() (pymemcache.client.hash.HashClient method), 14

RendezvousHash (class in pymemcache.client.rendezvous), 15

replace() (pymemcache.client.base.Client method), 11

replace() (pymemcache.client.base.PooledClient method), 13

replace() (pymemcache.client.hash.HashClient method), 14

replace() (pymemcache.fallback.FallbackClient method), 18

## S

set() (pymemcache.client.base.Client method), 11

set() (pymemcache.client.base.PooledClient method), 13

set() (pymemcache.client.hash.HashClient method), 14

set() (pymemcache.fallback.FallbackClient method), 18

set() (pymemcache.test.utils.MockMemcacheClient method), 16

set\_many() (pymemcache.client.base.Client method), 12

set\_many() (pymemcache.client.base.PooledClient method), 14

set\_many() (pymemcache.client.hash.HashClient method), 15

set\_many() (pymemcache.test.utils.MockMemcacheClient method), 16

set\_multi() (pymemcache.client.base.Client method), 12

set\_multi() (pymemcache.client.base.PooledClient method), 14

set\_multi() (pymemcache.client.hash.HashClient method), 15

set\_multi() (pymemcache.test.utils.MockMemcacheClient method), 16

stats() (pymemcache.client.base.Client method), 12

stats() (pymemcache.client.base.PooledClient method), 14

stats() (pymemcache.fallback.FallbackClient method), 18

stats() (pymemcache.test.utils.MockMemcacheClient method), 16

## T

touch() (pymemcache.client.base.Client method), 12

touch() (pymemcache.client.base.PooledClient method), 14

touch() (pymemcache.fallback.FallbackClient method), 18

## U

used (pymemcache.pool.ObjectPool attribute), 18

## V

version() (pymemcache.client.base.Client method), 13

version() (pymemcache.client.base.PooledClient method), 14