Riakcached Documentation

Release 0.1.0

Brett Langdon

September 13, 2013

CONTENTS

Contents:

CHAPTER

ONE

RIAKCACHED.CLIENTS

class riakcached.clients.**RiakClient**(*bucket*, *pool=None*)

A Memcache like client to the Riak HTTP Interface

Constructor for a new riakcached.clients.RiakClient

Pool - if no pool is provided then a default riakcached.pools.Urllib3Pool is used

Parameters

- **bucket** (*str*) – The name of the Riak bucket to use
- **pool** (riakcached.pools.Pool) – The riakcached.pools.Pool to use for requests

add_deserializer(*content_type*, *deserializer*)

Add a content-type deserializer to the client

The *deserializer* function should have the following definition:

```
def deserializer(data):
    return undo_something(data)
```

Example:

```
def base64_deserializer(data):
    return base64.b64decode(data)
client.add_deserializer("application/base64", base64_deserializer)
```

Parameters

- **content_type** (*str*) – the content-type to associate *deserializer* with
- **deserializer** (*function*) – the deserializer function to use with *content_type*

add_serializer(*content_type*, *serializer*)

Add a content-type serializer to the client

The *serializer* function should have the following definition:

```
def serializer(data):
    return do_something(data)
```

and should return a *str*

Example:

```
def base64_serializer(data):
    return base64.b64encode(data)
client.add_serializer("application/base64", base64_serializer)
```

Parameters

- **content_type** (*str*) – the content-type to associate *serializer* with
- **serializer** (*function*) – the serializer function to use with *content_type*

delete(*key*)
Delete the provided key from the client's *bucket*

Parameters **key** (*str*) – the key to delete

Returns bool - True if the key was removed, False otherwise

Raises riakcached.exceptions.RiakcachedBadRequest

delete_many(*keys*)
Delete multiple keys at once from the client's *bucket*

Parameters **keys** (*list*) – list of *str* keys to delete

Returns dict - the keys are the keys provided and the values are True or False from the calls to delete()

Raises riakcached.exceptions.RiakcachedBadRequest

deserialize(*data*, *content_type*)
Deserialize the provided *data* from *content_type*

This method will lookup the registered deserializer for the provided Content-Type (defaults to str(data)) and passes *data* through the deserializer.

Parameters

- **data** (*str*) – the data to deserialize
- **content_type** (*str*) – the Content-Type to deserialize *data* from

Returns object - whatever the deserializer returns

get(*key*, *counter=False*)
Get the value of the key from the client's *bucket*

Parameters

- **key** (*str*) – the key to get from the bucket
- **counter** (*bool*) – whether or not the *key* is a counter

Returns object - the deserialized value of *key*

Returns None - if the call was not successful or the key was not found

Raises riakcached.exceptions.RiakcachedBadRequest

Raises riakcached.exceptions.RiakcachedServiceUnavailable

get_many(*keys*)
Get the value of multiple keys at once from the client's *bucket*

Parameters **keys** (*list*) – the list of keys to get

Returns dict - the keys are the keys provided and the values are the results from calls to `get()`, except keys whose values are *None* are not included in the result

Raises `riakcached.exceptions.RiakcachedBadRequest`

Raises `riakcached.exceptions.RiakcachedServiceUnavailable`

incr(*key*, *value=1*)
Increment the counter with the provided key

Parameters

- **key** (*str*) – the counter to increment
- **value** (*int*) – how much to increment by

Returns bool - True/False whether or not it was successful

Raises `riakcached.exceptions.RiakcachedConflict`

Raises `riakcached.exceptions.RiakcachedBadRequest`

keys()
Get a list of all keys

Returns list - list of keys on the server

Returns None - when the call is not successful

ping()
Ping the server to ensure it is up

Returns bool - True if it is successful, False otherwise

props()
Get the properties for the client's *bucket*

Returns dict - the *bucket*'s set properties

Returns None - when the call is not successful

serialize(*data*, *content_type*)
Serialize the provided *data* to *content_type*

This method will lookup the registered serializer for the provided Content-Type (defaults to str(data)) and passes *data* through the serializer.

Parameters

- **data** (*object*) – the data to serialize
- **content_type** (*str*) – the desired Content-Type for the provided *data*

Returns str - the serialized data

set(*key*, *value*, *content_type='text/plain'*)
Set the value of a key for the client's *bucket*

Parameters

- **key** (*str*) – the key to set the value for
- **value** (*object*) – the value to set, this will get serialized for the *content_type*
- **content_type** (*str*) – the Content-Type for *value*

Returns bool - True if the call is successful, False otherwise

Raises `riakcached.exceptions.RiakcachedBadRequest`

Raises riakcached.exceptions.RiakcachedPreconditionFailed

set_many(*values*, *content_type='text/plain'*)
Set the value of multiple keys at once for the client's *bucket*

Parameters

- **values** (*dict*) – the key -> value pairings for the keys to set
- **content_type** (*str*) – the Content-Type for all of the values provided

Returns dict - the keys are the keys provided and the values are True or False from the calls to set()

Raises riakcached.exceptions.RiakcachedBadRequest

Raises riakcached.exceptions.RiakcachedPreconditionFailed

set_props(*props*)
Set the properties for the client's *bucket*

Parameters **props** (*dict*) – the properties to set

Returns bool - True if it is successful otherwise False

stats()
Get the server stats

Returns dict - the stats from the server

Returns None - when the call is not successful

class riakcached.clients.**ThreadedRiakClient**(*bucket*, *pool=None*)
A threaded version of riakcached.clients.RiakClient

The threaded version uses threads to try to parallelize the {set,get,delete}_many method calls

Constructor for a new riakcached.clients.RiakClient

Pool - if no pool is provided then a default riakcached.pools.Urllib3Pool is used

Parameters

- **bucket** (*str*) – The name of the Riak bucket to use
- **pool** (riakcached.pools.Pool) – The riakcached.pools.Pool to use for requests

delete_many(*keys*)
Delete multiple keys at once from the client's *bucket*

Parameters **keys** (*list*) – list of *str* keys to delete

Returns dict - the keys are the keys provided and the values are True or False from the calls to delete()

Raises riakcached.exceptions.RiakcachedBadRequest

get_many(*keys*)
Get the value of multiple keys at once from the client's *bucket*

Parameters **keys** (*list*) – the list of keys to get

Returns dict - the keys are the keys provided and the values are the results from calls to get(), except keys whose values are *None* are not included in the result

Raises riakcached.exceptions.RiakcachedBadRequest

Raises riakcached.exceptions.RiakcachedServiceUnavailable

set_many(*values*)

Set the value of multiple keys at once for the client's *bucket*

Parameters

- **values** (*dict*) – the key -> value pairings for the keys to set
- **content_type** (*str*) – the Content-Type for all of the values provided

Returns dict - the keys are the keys provided and the values are True or False from the calls to `set()`

Raises riakcached.exceptions.RiakcachedBadRequest

Raises riakcached.exceptions.RiakcachedPreconditionFailed

CHAPTER

TWO

RIAKCACHED.EXCEPTIONS

exception riakcached.exceptions.**RiakcachedBadRequest**
Exception that is raised for unexpected HTTP 400 status responses

Inherits from riakcached.exceptions.RiakcachedException

exception riakcached.exceptions.**RiakcachedConflict**
Exception that is raised for unexpected HTTP 409 status responses

Inherits from riakcached.exceptions.RiakcachedException

exception riakcached.exceptions.**RiakcachedConnectionError**
Exception that is raised when pool requests raises an *HTTPError*

Inherits from riakcached.exceptions.RiakcachedException

exception riakcached.exceptions.**RiakcachedException**
Base class for Riakcached Exceptions

exception riakcached.exceptions.**RiakcachedNotFound**
Exception that is raised for unexpected HTTP 204 status responses

Inherits from riakcached.exceptions.RiakcachedException

exception riakcached.exceptions.**RiakcachedPreconditionFailed**
Exception that is raised for unexpected HTTP 412 status responses

Inherits from riakcached.exceptions.RiakcachedException

exception riakcached.exceptions.**RiakcachedServiceUnavailable**
Exception that is raised for unexpected HTTP 503 status responses

Inherits from riakcached.exceptions.RiakcachedException

exception riakcached.exceptions.**RiakcachedTimeout**
Exception that is raised when pool requests takes too long

Inherits from riakcached.exceptions.RiakcachedException

CHAPTER

THREE

RIAKCACHED.POOLS

class riakcached.pools.**Pool**(*base_url='http://127.0.0.1:8098'*, *timeout=2*, *auto_connect=True*)
A Riak HTTP request connection pool base class

This is the base class that should be used for any custom connection pool to be used by any of the riakcached.clients.RiakClient

Constructs a new riakcached.pools.Pool

Parameters

- **base_url** (*str*) – the base url that the client should use for requests
- **timeout** (*int*) – the connection timeout to use
- **auto_connect** (*bool*) – whether or not to call connect() on __init__

close()
Closes the connection pool if it is opened

connect()
Create the connection pool

request(*method*, *url*, *body=None*, *headers=None*)
Makes a single HTTP request

Parameters

- **method** (*str*) – the HTTP method to make the requets with
- **url** (*str*) – the full url for the request
- **body** (*str*) – the data to POST or PUT with the request
- **headers** (*dict*) – extra headers to add to the request

Returns tuple - status, data, headers

Raises riakcached.exceptions.RiakcachedTimeout

Raises riakcached.exceptions.RiakcachedConnectionError

class riakcached.pools.**Urllib3Pool**(*base_url='http://127.0.0.1:8098'*, *timeout=2*, *auto_connect=True*)
A subclass of riakcached.pools.Pool which uses *urllib3* for requests

Constructs a new riakcached.pools.Pool

Parameters

- **base_url** (*str*) – the base url that the client should use for requests

- **timeout** (*int*) – the connection timeout to use
- **auto_connect** (*bool*) – whether or not to call `connect()` on __init__

close()
Closes the connection pool if it is opened

connect()
Create the connection pool

request(*method*, *url*, *body=None*, *headers=None*)
Makes a single HTTP request

Parameters

- **method** (*str*) – the HTTP method to make the requets with
- **url** (*str*) – the full url for the request
- **body** (*str*) – the data to POST or PUT with the request
- **headers** (*dict*) – extra headers to add to the request

Returns tuple - status, data, headers

Raises `riakcached.exceptions.RiakcachedTimeout`

Raises `riakcached.exceptions.RiakcachedConnectionError`

A Memcached like interface to the Riak HTTP Client. Read The Docs

CHAPTER

FOUR

INSTALLING

4.1 From PyPI

```
pip install riakcached
```

4.2 From Git

```
git clone git://github.com/brettlangdon/riakcached.git
cd ./riakcached
pip install -r requirements.txt
python setup.py install
```

CHAPTER

FIVE

USAGE

5.1 Basic Usage

```
from riakcached.clients import RiakClient

client = RiakClient("my_bucket")

client.set("hello", "world")
print client.get("hello")
# 'hello'

client.delete("hello")
print client.get("hello")
# None

values = {
    "hello": "world",
    "foo": "bar",
}
client.set_many(values)

keys = ["hello", "foo", "test"]
print client.get_many(keys)
# {'foo': 'bar', 'hello': 'world'}

client.close()
```

5.2 Connection Pool Settings

```
from riakcached.clients import RiakClient
from riakcached.pools import Urllib3Pool

pool = Urllib3Pool(base_url="http://my-host.com:8098/", timeout=1)
client = RiakClient("my_bucket", pool=pool)

client.get("foo")
```

5.3 Custom Connection Pool

```
from riakcached.clients import RiakClient
from riakcache.pools import Pool

class CustomPool(Pool):
    __slots__ = ["connection"]

    def connect(self):
        self.connection = make_a_connection()

    def close(self):
        if self.connection:
            close_connection(self.connection)

    def request(self, method, url, body=None, headers=None):
        results = make_request(self.connection, method, url, body, headers, timeout=self.timeout)
        return results.status, results.data, results.headers


custom_pool = CustomPool(base_url="http://my-host.com:8098", timeout=1)
client = RiakClient("my_bucket", pool=pool)
```

5.4 Threaded Client

The exists a `riakcached.clients.ThreadedRiakClient` which inherits from `riakcached.clients.RiakClient` and which uses threading to try to parallelize calls to `get_many`, `set_many` and `delete_many`.

CHAPTER

SIX

DOCUMENTATION

The documentation can be found in the `/docs` directory in this repository and should be fairly complete for the codebase.

6.1 Building Documentation

```
git clone git://github.com/brettlangdon/riakcached.git
cd riakcached
pip install -r docs-requirements.txt
cd ./docs
make html
```

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

r