# Timed Dictionary Documentation

***Release 0.1.0***

**Zachary Ernst**

**Oct 11, 2018**

# CONTENTS:

CHAPTER

ONE

# TIMED DICTIONARY MODULE

**class** timed_dict.timed_dict.**Empty**

Bases: object

Just to provide a unique default class when asking for a key that's been deleted.

**__weakref__**

list of weak references to the object (if defined)

**class** timed_dict.timed_dict.**TimedDict**(*timeout=None*, *checks_per_second=0.5*, *sample_probability=0.25*, *callback=None*, *expired_keys_ratio=0.25*, *sweep_flag=True*, *callback_args=None*, *callback_kwargs=None*)

Bases: collections.abc.MutableMapping

A dictionary whose keys time out. After a pre-determined number of seconds, the key and value will be deleted from the dictionary. Optionally, a callback function is executed, which is passed the key and the associated value.

When it is instantiated, this class creates a thread which runs all the time, looking for expired keys. Each TimedDict object gets its own thread.

The algorithm is the same one that Redis uses. It is semi-lazy and probabilistic. After sleeping for a set interval, it iterates through a random sample of the keys (which is determined by the sample_probability kwarg in the class constructor). It expires any keys it finds during the sweep which have existed for more then timeout seconds. If at least expired_keys_ratio of the sampled keys have to be expired, then the process is repeated again immediately. If not, then it sleeps for the interval again before restarting.

Additionally, a check is made to any specific key that's accessed. If the key should be expired, then it does so and returns an Empty object.

**__getitem__**(*key*)

Gets the item. Before it does so, checks whether the key has expired. If so, it expires the key _first_, before returning a value.

If the key does not exist (or gets expired during this call) the method returns an instance of the *Empty* class. We use the *Empty* class (defined above) because we want to avoid raising exceptions, but we also want to allow that the legitimate value of a key might be *None* (or any other default value we like).

**__init__**(*timeout=None*, *checks_per_second=0.5*, *sample_probability=0.25*, *callback=None*, *expired_keys_ratio=0.25*, *sweep_flag=True*, *callback_args=None*, *callback_kwargs=None*)

Initialize self. See help(type(self)) for accurate signature.

**__len__**()

Returns the number of items currently in the TimedDict.

**__repr__**()

String representation of the TimedDict. It returns the keys, values, and time of expiration for each.

**__setitem__**(*key*, *value*)
Replaces the __setitem__ from the parent class. Sets both the base_dict (which holds the values) and the timed_dict (which holds the expiration time.

**__weakref__**
list of weak references to the object (if defined)

**expire_key**(*key*)
Expire the key, delete the value, and call the callback function if one is specified.

**Parameters** **key** – The TimedDict key

**keys**()
Replaces the keys method. There's probably a better way to accomplish this.

**set_expiration**(*key*, *ignore_missing=False*, *additional_seconds=None*, *seconds=None*)
Alters the expiration time for a key. If the key is not present, then raise an Exception unless *ignore_missing* is set to *True*.

**Parameters**

- **key** – The key whose expiration we are changing.
- **ignore_missing** (*bool*) – If set, then return silently if the key does not exist. Default is *False*.
- **additional_seonds** (*int*) – Add this many seconds to the current expiration time.
- **seconds** (*int*) – Expire the key this many seconds from now.

**stop_sweep**()
Stops the thread that periodically tests the keys for expiration.

**sweep**()
This methods runs in a separate thread. So long as *self.sweep_flag* is set, it expires keys according to the process explained in the docstring for the *TimedDict* class. The thread is halted by calling *self.stop_sweep()*, which sets the *self.sweep_flag* to *False*.

**values**()
Replaces the values method. There's probably a better way to accomplish this.

timed_dict.timed_dict.**cleanup_sweep_threads**()
Not used. Keeping this function in case we decide not to use daemonized threads and it becomes necessary to clean up the running threads upon exit.

timed_dict.timed_dict.**my_callback**(*key*, *value*)
Simple test of callback function.

CHAPTER

TWO

# OVERVIEW

## 2.1 What it is

"Timed Dictionary" provides a class – `TimedDict` – which is a dictionary whose keys time out. After a predetermined number of seconds, the key and value will be deleted from the dictionary. Optionally, a callback function may be specified, which is called whenever a key is expired.

When it is instantiated, this class creates a thread which runs all the time, looking for expired keys. Each `TimedDict` object gets its own thread.

The algorithm is the same one that Redis uses. It is semi-lazy and probabilistic. After sleeping for a set interval, it iterates through a random sample of the keys (which is determined by the `sample_probability` kwarg in the class constructor). It expires any keys it finds during the sweep which have existed for more then `timeout` seconds. If at least `expired_keys_ratio` of the sampled keys have to be expired, then the process is repeated again immediately. If not, then it sleeps for the interval again before restarting.

Additionally, a check is made to any specific key that's accessed. If the key should be expired, then it does so and returns an `Empty` object.

## 2.2 What it is not

This module is **not** a replacement for Redis, MEMCACHE, or any other such things. It is intended to be simple, lightweight, free of any infrastructure requirements, stand-alone, pure Python, and above all, functional. If you need failover, distributed caching, guarantees, a magical solution to the CAP theorem, or you're processing huge volumes of data, you should not use this.

## 2.3 Why it is

As a data engineer, I'm constantly coming across use-cases that look like they require heavyweight tools, but which aren't demanding enough to justify the investment. Redis, although it isn't exactly "heavyweight", requires a server to run, with a dedicated IP address, and so on for virtually any production use-case. So engineers who would like to have a key-value store with expiration and callbacks either roll their own one-off kludgey solution, or they stand up a Redis instance somewhere and have one more thing to worry about.

This module provides that core functionality, implemented as a Python dictionary. But it does not require anything other than the standard Python library to run. It is dead-simple, reliable, and tested.

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## t

# INDEX

## Symbols



## C



## E



## K



## M



## S



## T



## V