CTools

Release 0.2.0

ko-han

Aug 10, 2020

CONTENTS

1 Install **3**

2 Table of contents **5**

2.1 API 5

2.2 Changelog 11

3 Indices and tables **13**

Index **15**

Welcome! This is the documentation for CTools 0.2.0, last updated Mon Aug 10 2020.

CHAPTER

ONE

INSTALL

Install and update using *pip*

```
pip install -U ctools
```

CHAPTER

TWO

TABLE OF CONTENTS

2.1 API

2.1.1 Functions

ctools.**strhash**(*s*, *method='fnv1a'*, */*)
Hash str with consistent value.

Parameters

s [str] The string to hash.

method [{'fnv1a', 'fnv1', 'djb2', 'murmur'}, optional] Choice in method, default first when optional.

Returns

int hash number

Raises

ValueError **If method not supported.**

ctools.**jump_consistent_hash**(*key*, *num_buckets*, */*)
Generate a number in the range [0, num_buckets).

Parameters

key [int] The key to hash.

num_buckets [int] Number of buckets to use.

Returns

int hash number.

ctools.**int8_to_datetime**(*date_integer*, */*)
Convert integer like 20180101 to datetime.datetime(2018, 1, 1)).

Parameters

date_integer [int] The string to hash.

Returns

datetime.datetime parsed datetime

Examples

```
>>> import ctools
>>> ctools.int8_to_datetime(20010101)
datetime.datetime(2001, 1, 1, 0, 0)
```

2.1.2 Classes

class ctools.**CacheMap**(*capacity=None*)

A fast LFU (least frequently used) mapping.

Parameters

capacity [int, optional] Max size of cache, default is C INT32_MAX.

Examples

```
>>> import ctools
>>> cache = ctools.CacheMap(1)
>>> cache['foo'] = 'bar'
>>> cache['foo']
'bar'
>>> cache['bar'] = 'foo'
>>> 'foo' in cache
False
```

clear()

Clean cache.

evict()

Evict a item. raise error if no item in cache.

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

Get item from cache.

hit_info()

Return capacity, hits, and misses count.

items()

Iter keys and values.

keys()

Iter keys.

next_evict_key()

Return the most unused key.

pop(*key*, *default=None*, */*)

Pop an item from cache, if key not exists return default.

popitem()

Remove and return some (key, value) pairas a 2-tuple; but raise KeyError if mapping is empty.

set_capacity(*capacity*, */*)

Reset capacity of cache.

setdefault(*key*, *default=None*, */*)

Get item in cache, if key not exists, set default to cache and return it.

setnx(*key*, *fn=None*)
Like setdefault but accept a callable.

Parameters

key [object] Hash key.

fn [typing.Callable[[typing.Any], typing.Any], optional] It's a callable that accept key as only one argument, called when key not exists.

Returns

object The found value or what setnx return.

update(*map*, */*)
Update item to cache. Unlike dict.update, only accept a dict object.

values()
Iter values.

class ctools.**TTLCache**(*ttl=None*)
A mapping that keys expire and unreachable after ttl seconds.

Parameters

ttl [int, optional] Key will expire after this many seconds, default is 60 (1 minute).

Examples

```
>>> import ctools
>>> import time
>>> cache = ctools.TTLCache(5)
>>> cache['foo'] = 'bar'
>>> cache['foo']
'bar'
>>> time.sleep(5)
>>> 'foo' in cache
False
```

clear()
Clear cache.

get(*key*, *default=None*, */*)
Get item from cache.

get_default_ttl()
Return default ttl.

items()
Iter items.

keys()
Iter keys.

pop(*key*, *default=None*)
Pop item from cache.

popitem()
Remove and return some (key, value) pairas a 2-tuple; but raise KeyError if mapping is empty.

set_default_ttl(*ttl*, */*)
Reset default ttl.

Parameters

ttl [int] Expire seconds.

Notes

Exist keys won't change their expire.

setdefault(*key*, *default=None*, */*)
Get item in cache, if key not exists, set default to cache and return it.

setnx(*key*, *fn=None*)
Like setdefault but accept a callable.

Parameters

key [object] Hash key.

fn [typing.Callable[[typing.Any], typing.Any], optional] It's a callable that accept key as only one argument, called when key not exists.

Returns

object The found value or what setnx return.

update(*mp*, ***kwargs*)
Update item to cache. Unlike dict.update, only accept a dict object.

values()
Iter values.

class ctools.**Channel**(*size=None*, */*)
A channel support sending and safe consuming.

Parameters

size [int, optional] The max size of channel, default to C MAX_INT32.

Examples

```
>>> import ctools
>>> ch = ctools.Channel(1)
>>> ch.send('foo')
True
>>> ch.send('bar')
False
>>> ch.recv()
('foo', True)
>>> ch.recv()
(None, False)
```

clear()
Clear channel.

close(*send=True*, *recv=True*)
Close channel.

recv()
Receive an object from channel.

Returns

o Object that received. Return None if no items in channel.

ok [bool] Return False if no items in channel else True.

Raises

IndexError If the channel is closing for receive.

recvable()
Return channel is available to receive.

safe_consume(*fn*, */*)
Safe consume with a callable.

Parameters

fn [typing.Callable[[typing.Any], bool]] The *fn* receive an item as only one argument and must return True on success, False on fail.

Returns

bool Return True if consume success, and False on fail.Return False if no item in the channel.

send(*obj*, */*)
Send an object to channel.

Returns

bool Return True if send success else False

Raises

IndexError If the channel is closing for sending.

sendable()
Return channel is available to send.

size()
Return the size of channel.

class ctools.**SortedMap**(*cmp=None*, */*)
A sorted map base on red-black tree.

New in version 0.2.0.

Parameters

cmp [typing.Callable[[typing.Any], int], optional] A optional callable receive two keys, that

return negative integer if *k1* < *k2*,

return positive integer if *k1* > *k2*,

return 0 if *k1* == *k2*.

It's every similar to standard C library qsort comparator.

Examples

```
>>> import ctools
>>> foo = ctools.SortedMap()
>>> foo[1] = 1
>>> foo[2] = 2
>>> foo[1]
1
>>> foo.max()
(2, 2)
>>> foo.min()
(1, 1)
>>> foo.keys()
[1, 2]
>>> foo.popitem()
(1, 1)
```

clear()
Clear mapping.

get(*key*, *default=None*)
Return value if find else default.

items()
Iterate items in order of keys.

keys()
Iterate sorted keys.

max()
Return maximum (key, value) pairas a 2-tuple; but raise KeyError if mapping is empty.

min()
Return minimum (key, value) pairas a 2-tuple; but raise KeyError if mapping is empty.

pop(*key*, *default=None*)
Pop an item, if key not exists, return default.

popitem()
Remove and return some (key, value) pairas a 2-tuple; but raise KeyError if mapping is empty. Ensure key is the smallest in the mapping.

setdefault(*key*, *default=None*)
Return value if find else default and put default to mapping.

setnx(*key*, *fn=None*)
Like setdefault but accept a callable.

Parameters

key [object] Hash key.

fn [typing.Callable[[typing.Any], typing.Any], optional] It's a callable that accept key as only one argument, called when key not exists.

Returns

object The found value or what setnx return.

update(*mp*, ***kwargs*)
Like dict.update, but only accept dict.

values()
Iterate over values in order of keys.

2.2 Changelog

2.2.1 0.2.0

New Feature

- New class *SortedMap*. A sorted map base on red-black tree.

Changes

- *CacheMap* and *TTLCache* now support all of MutableMapping methods.
- *CacheMap.setnx()* and *TTLCache.setnx()* must accept key as their only one argument.
- *CacheMap* default size is setting to MAX_INT32.
- *Channel* default size is setting to MAX_INT32.
- *TTLCache* default ttl is setting to 1 minute.

2.2.2 0.1.0

New Feature

- New class *CacheMap*, it's a lfu cache behaving much like dict.
- New class *TTLCache*, a cache mapping storing items for a certain time.
- *strhash()* now add a method argument which supported fnv1a(default), fnv1, djb2 and murmur algorithms.

2.2.3 0.0.4

Improve

- Decrease *strhash()* collisions, but note that we will get different hash value in this version compare with old version.

2.2.4 0.0.3

- First release.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

INDEX

C

CacheMap (*class in ctools*), 6
Channel (*class in ctools*), 8
clear() (*ctools.CacheMap method*), 6
clear() (*ctools.Channel method*), 8
clear() (*ctools.SortedMap method*), 10
clear() (*ctools.TTLCache method*), 7
close() (*ctools.Channel method*), 8

E

evict() (*ctools.CacheMap method*), 6

G

get() (*ctools.CacheMap method*), 6
get() (*ctools.SortedMap method*), 10
get() (*ctools.TTLCache method*), 7
get_default_ttl() (*ctools.TTLCache method*), 7

H

hit_info() (*ctools.CacheMap method*), 6

I

int8_to_datetime() (*in module ctools*), 5
items() (*ctools.CacheMap method*), 6
items() (*ctools.SortedMap method*), 10
items() (*ctools.TTLCache method*), 7

J

jump_consistent_hash() (*in module ctools*), 5

K

keys() (*ctools.CacheMap method*), 6
keys() (*ctools.SortedMap method*), 10
keys() (*ctools.TTLCache method*), 7

M

max() (*ctools.SortedMap method*), 10
min() (*ctools.SortedMap method*), 10

N

next_evict_key() (*ctools.CacheMap method*), 6

P

pop() (*ctools.CacheMap method*), 6
pop() (*ctools.SortedMap method*), 10
pop() (*ctools.TTLCache method*), 7
popitem() (*ctools.CacheMap method*), 6
popitem() (*ctools.SortedMap method*), 10
popitem() (*ctools.TTLCache method*), 7

R

recv() (*ctools.Channel method*), 8
recvable() (*ctools.Channel method*), 9

S

safe_consume() (*ctools.Channel method*), 9
send() (*ctools.Channel method*), 9
sendable() (*ctools.Channel method*), 9
set_capacity() (*ctools.CacheMap method*), 6
set_default_ttl() (*ctools.TTLCache method*), 7
setdefault() (*ctools.CacheMap method*), 6
setdefault() (*ctools.SortedMap method*), 10
setdefault() (*ctools.TTLCache method*), 8
setnx() (*ctools.CacheMap method*), 6
setnx() (*ctools.SortedMap method*), 10
setnx() (*ctools.TTLCache method*), 8
size() (*ctools.Channel method*), 9
SortedMap (*class in ctools*), 9
strhash() (*in module ctools*), 5

T

TTLCache (*class in ctools*), 7

U

update() (*ctools.CacheMap method*), 7
update() (*ctools.SortedMap method*), 10
update() (*ctools.TTLCache method*), 8

V

values() (*ctools.CacheMap method*), 7
values() (*ctools.SortedMap method*), 10
values() (*ctools.TTLCache method*), 8