Method Cache Documentation

Release 0.0.1

Stefan Eiermann

Feb 27, 2019

Contents:

1 Cache **1**

2 Exceptions **3**

3 Helper **5**

4 Store **7**

5 How to use **9**
- 5.1 TLDR . . . 9
- 5.2 @cache . . . 9
- 5.3 Store . . . 11
- 5.4 Full Example . . . 11

6 Hack **13**

7 Indices and tables **15**

CHAPTER 1

Cache

methodcache.cache.**cache**(***options*)

Decorator method for caching

Parameters

- **ttl** – Time to life for this Cache Entry in Seconds
- **store** – Storage Object. To this object the cache entry was saved
- **category** – String in which Category the cache object should be sorted

methodcache.cache.**add_to_cache**(*options={}*, *func=None*, *params=None*)

Seperated Add To Cache Method; Contains the function stack to add a function and there result to cache

Parameters

- **options** – dict with keys of store,category,ttl (previously validated)
- **func** – orginal function
- **params** – WrapperParameters object with *args and **kwargs from orginal function

Return any Result created by orignal function

CHAPTER 2

Exceptions

exception methodcache.exceptions.**GeneralMethodCacheException**
General MethodCache Exception

exception methodcache.exceptions.**NoMethod**
Raised when Method not registered in Cache

exception methodcache.exceptions.**TTLExpired**
Raised when TTL of an MethodObject are expired

CHAPTER 3

Helper

class methodcache.helper.**WrapperFunction**(*func*)

get_func()
Return given function :return func: “raw” function

get_hash()
Get hash of function :return string: Numeric Hash

get_name()
Return Name of function :return str: Name of given function

class methodcache.helper.**WrapperParameters**(*arguments=()*, *keyword_arguments={}*)

get_args()
Return list of given args :return list: args

get_kwargs()
Return list of given kwargs :return list: kwargs

santize_args()
Create a dict from args. Every dict key start with the word “arg” followed by the value index of tuple. The value of this key are hashed :return dict: arguments as dict

santize_kwargs()
Return a dict wich every value are hashed :return dict: kwargs with hashed values

santize_parameters()
Give a common list of args and kwargs. see santize_args and santize_kwargs :return dict: Merged dict of args and kwargs

CHAPTER 4

Store

class methodcache.store.**Store**(*ttl=60*)

get_all_categorys()
List of all categories on root level

Returns

get_category(**category_tuple*, *full=False*)
GoTo category level given in category_tuple and return this level content :param category_tuple: List of Categorys :param full: return the full cateogry not only the names :return:

get_method_store(**category*)
Get the MethodStore from category/subcategory If no methodstore exists, create a new one

Parameters **category** – list of category and thier subcategorys

Return MethodStore Return new or old MethodStore Object

class methodcache.store.**MethodStore**(*store*, **args*, ***kwargs*)

create(*func*, *params*, *result*)
Create a new Cache Entry

Parameters

- **func** – WrappedFunction for wich function the entry created
- **params** – WrappedParameter which parameters used to get this result
- **result** – string Result

get_method(*func*, *params*)
Returns the MethodObject identified by func and params

Parameters

- **func** – WrappedFunction Object to identifiy search method

- **params** – WrappedParameters Object to check if a object with this parameters is stored

Return MethodObject the MethodObject with the cache Information

has_method(*func*)
Check if function is stored in this MethodStore Object

Parameters **func** – WrapperFunction with the searched funciton

Return bool True if method existes in this store

has_method_call(*func*, *params*)
Like has_method. This Checks also if a method with the given arguments exists

Parameters

- **func** – WrapperFunction contains the function
- **params** – WrapperParameters contains arguments and keyword arguments

Return bool True if method call exists and stored in this MEthodStore

class methodcache.store.**MethodObject**(*func*, *params*, *result*)

has_params__exactly(*params*)
Check if the Parameters are excatly the same wich are given as argument :param params: WrapperParameters to compare :return bool: True if the params are the same

CHAPTER 5

How to use

5.1 TLDR

Just simple put @cache `@cache` over your method.

Example:

```
from methodcache import cache

@cache
def dothings(*args, **kwargs):
    pass
```

5.2 @cache

Is the caching decorator. use this decorator on methods you want to cache with the following optional kwargs:

- store
- category
- ttl

5.2.1 store

For Attributes and thier documentaiton see *methodcache.cache.cache*

`@cache` can used without any parameters.

If the kwarg `store` is missing, it creates a default Store. This default Store are safed as static attribute named `_default_store` in the class *methodcache.store.Store*

5.2.2 category

If the kwarg `category` not set, a category named `default` was used

Categories can be seperated in "groups". You can define in which group the current method is cached. The groups can be interleaved via `:`. eg. `car:manufacturer:model`

Demo

```
from methodcache import cache

@cache(category="car:manufacturer")
def get_manufacturer(*args, **kwargs):
    return ["BMW", "Opel", "VW", "Honda"]

@cache(category="car:manufacturer:model")
def get_model(manufacturer):
    models = {
        "BMW": ["CarType1", "CarType2", "CarType3"]
        "Opel": ["CarType4", "CarType5", "CarType6"]
    }
    return models[manufacturer]

manufacturer = get_manufacturer()
get_model(manufacturer[0])
```

If these example code was executed the Store objects the informations like:

```
{
    "car": {
        "manufacturer": {
            "method_store":{
                "get_manufacturer": ["BMW", "Opel", "VW", "Honda"]
            },
            "model": {
                "method_store": {
                    "get_model;BMW": ["CarType1", "CarType2", "CarType3"]
                }
            }
        }

    }
}
```

5.2.3 ttl

If the kwarg `ttl` is not set a default ttl of 5 minutes are set.

You can define a TTL by createing a Store object. This TTL from Store can be overwriten by `@cache`

TTLs are always in Seconds

```
# TTL of 5 Minutes
@cache(ttl=60*5)
```

The code Above is strongly simplified. In orginal software the arguments are also check.

The code Above is strongly simplified. In orginal software the arguments are also check.

5.3 Store

The Store object contains all caching information.

By initializeation you can set a default ttl.

```
st = Store(ttl=60*5)
```

to use the store as cache in for an app define it as handover parameters `store`

```
st = Store(ttl=60*5)

@cache(store=st)
def dothings()
    pass
```

You can define multiple Stores to seperate your Application section from each other

5.4 Full Example

```
import time
from methodcache import cache, Store

st = Store(ttl=5)


class Car:

    def __init__(self, serial_number, milage):
        self.serial_number = serial_number
        self.milage = milage


class CarManager:

    @cache(store=st, category="car")
    def get_cars(self):
        time.sleep(2)
        return [Car("JKaHO3hoHOe4GHOy4", 234214), Car("AJAHWho4HOI46HIOA4t", 34571)]

    @cache(store=st, category="car:manufacturer", ttl=5)
    def get_manufacturer(self):
        time.sleep(3)
        return ["BMW", "Opel", "Ford"]


class FruitManager:

    @cache(store=st, category="fruit")
    def get_fruits(self):
        time.sleep(3)
        return ["Apple", "Banana", "Pineapple"]

    @cache(store=st, category="fruit:country")
    def get_country(self):
```

(continues on next page)

(continued from previous page)

```
        time.sleep(3)
        return ["Germany", "Ecuador", "Costa Rica"]


fruits = FruitManager()
print()
print("Without Cache")
print(fruits.get_fruits())
print(fruits.get_country())
print()
print("With Cache")
print("-"*10)
print(fruits.get_fruits())
print(fruits.get_country())


cars = CarManager()
print()
print("Without Cache")
print(cars.get_cars())
print(cars.get_manufacturer())
print()
print("With Cache")
print("-"*10)
print(cars.get_cars())
print(cars.get_manufacturer())
print()
print("With Expired TTL")
print("-"*10)
print(fruits.get_country())
print()
print("Get All Categorys")
print("-"*10)
print(st.get_all_categorys())
print(st.get_category("car"))
```

CHAPTER 6

Hack

You can simply write your own decorator, or imlement in your code just simply call `add_to_cache`

```
from methodcache.cache import add_to_cache
main_store = Store(ttl=5)

def dothings(a,b,c):
    time.sleep(5)
    return a+b+c


add_to_cache(options={}, store=main_store, func=dothings, WrapperParameters((1,2,3)))
```

This is useful for the following Example

```
from methodcache.cache import add_to_cache
from methodcache import Store

# ToDo Write a Example
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

A

add_to_cache() (*in module methodcache.cache*), 1

C

cache() (*in module methodcache.cache*), 1
create() (*methodcache.store.MethodStore method*), 7

G

GeneralMethodCacheException, 3
get_all_categorys() (*methodcache.store.Store method*), 7
get_args() (*methodcache.helper.WrapperParameters method*), 5
get_category() (*methodcache.store.Store method*), 7
get_func() (*methodcache.helper.WrapperFunction method*), 5
get_hash() (*methodcache.helper.WrapperFunction method*), 5
get_kwargs() (*methodcache.helper.WrapperParameters method*), 5
get_method() (*methodcache.store.MethodStore method*), 7
get_method_store() (*methodcache.store.Store method*), 7
get_name() (*methodcache.helper.WrapperFunction method*), 5

H

has_method() (*methodcache.store.MethodStore method*), 8
has_method_call() (*methodcache.store.MethodStore method*), 8
has_params__exactly() (*methodcache.store.MethodObject method*), 8

M

MethodObject (*class in methodcache.store*), 8
MethodStore (*class in methodcache.store*), 7

N

NoMethod, 3

S

santize_args() (*methodcache.helper.WrapperParameters method*), 5
santize_kwargs() (*methodcache.helper.WrapperParameters method*), 5
santize_parameters() (*methodcache.helper.WrapperParameters method*), 5
Store (*class in methodcache.store*), 7

T

TTLExpired, 3

W

WrapperFunction (*class in methodcache.helper*), 5
WrapperParameters (*class in methodcache.helper*), 5