Cache Any Python Object

Release 2.0.6

c0fec0de

Sep 22, 2019

Contents

Cache any python object to file using improved pickling

CHAPTER 1

Installation

To install the *anycache* module run:

```
pip install anycache
```

If you do not have write-permissions to the python installation, try:

```
pip install anycache --user
```

CHAPTER 2

API

Cache any python object to file.

class anycache.**AnyCache**(*cachedir=None*, *maxsize=None*)

Bases: object

Cache for python objects.

Keyword Arguments

- **cachedir** – Directory for cached python objects. *AnyCache* instances on the same *cachedir* share the same cache.
- **maxsize** – Maximum cache size in bytes. *None* does not limit the cache size. *0* disables caching. It the maximum size is smaller than the last cached object, this object is kept. During object write the cache size might be larger than *maxsize*. At maximum twice as large as the maximum object size.

The *AnyCache* instance mainly serves the *AnyCache.anycache* method for caching the result of functions.

```
>>> from anycache import AnyCache
>>> ac = AnyCache()
>>> @ac.anycache()
... def myfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> myfunc(4, 5)
  Calcing 4 + 5 = 9
9
>>> myfunc(4, 5)
9
>>> myfunc(4, 2)
  Calcing 4 + 2 = 6
6
```

The cache size is returned by *AnyCache.size*.

```
>>> ac.size
10
```

The cache size can be limited via *maxsize*. A *maxsize* of *0* disables caching.

```
>>> ac.maxsize = 0
>>> myfunc(4, 5)
  Calcing 4 + 5 = 9
9
```

The cache is preserved in this case, and needs to be cleared explicitly:

```
>>> ac.size
10
>>> ac.clear()
>>> ac.size
0
```

cachedir
Cache directory use for all cache files.

AnyCache instances on the same *cachedir* share the same cache.

anycache (*depfilefunc=None*)
Decorator to cache result of function depending on arguments.

Keyword Arguments **depfilefunc** – Dependency file function (see example below)

```
>>> from anycache import AnyCache
>>> ac = AnyCache()
>>> @ac.anycache()
... def myfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> myfunc(2, 5)
  Calcing 2 + 5 = 7
7
>>> myfunc(2, 5)
7
```

File I/O is not tracked by the decorator. Instead a function needs to be implemented, which returns the paths of the files, which influence the function result. The *depfilefunc* is called with the function result and all arguments. The following example, depends on the path of the source code itself:

```
>>> def mydepfilefunc(result, posarg, kwarg=3):
...     print("  Deps of %r + %r = %r" % (posarg, kwarg, result))
...     return [__file__]
>>> @ac.anycache(depfilefunc=mydepfilefunc)
... def myfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> myfunc(2, 7)
  Calcing 2 + 7 = 9
  Deps of 2 + 7 = 9
9
```

is_outdated (*func*, **args*, ***kwargs*)
Return *True* if cache is outdated for *func* used with *args* and *kwargs*.

```
>>> from anycache import AnyCache
>>> ac = AnyCache()
>>> @ac.anycache()
... def myfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> ac.is_outdated(myfunc, 2, 5)
True
>>> myfunc(2, 5)
  Calcing 2 + 5 = 7
7
>>> ac.is_outdated(myfunc, 2, 5)
False
```

remove (*func*, **args*, ***kwargs*)
Remove cache data for *func* used with *args* and *kwargs*.

```
>>> from anycache import AnyCache
>>> ac = AnyCache()
>>> @ac.anycache()
... def myfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> myfunc(2, 5)
  Calcing 2 + 5 = 7
7
>>> ac.remove(myfunc, 2, 5)
>>> myfunc(2, 5)
  Calcing 2 + 5 = 7
7
```

Removing non-existing cache entries is not an error:

```
>>> ac.remove(myfunc, 2, 5)
>>> ac.remove(myfunc, 2, 5)
```

get_ident (*func*, **args*, ***kwargs*)
Return identification string for *func* used with *args* and *kwargs*.

```
>>> from anycache import AnyCache
>>> ac = AnyCache()
>>> @ac.anycache()
... def myfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> @ac.anycache()
... def otherfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> ac.get_ident(myfunc, 2, 5)
'19044d3869955fa79d7f3db8fcdc5af84b3f55c0bdab2b4aee1bb21e1a9856c9'
>>> ac.get_ident(myfunc, 2, 6)
'1885e09f9898a1f1bd186f052d0e810693faa14b70e7b6b22de61b90c8171427'
>>> ac.get_ident(otherfunc, 2, 5)
'9b8aea26422999aaa7aed0fdb4d5145fd33b87de20f322cf175997e9b1835158'
```

clear()
Clear the cache by removing all cache files.

size
Return total size of all cache files.

anycache.**anycache**(*cachedir=None*, *maxsize=None*, *depfilefunc=None*)
Decorator to cache result of function depending on arguments.

This decorator uses one unlimited global cache within one python run. Different anycached functions have different cache name spaces and do not influence each other.

To preserve the cache result between multiple python runs, use an *AnyCache* instance with a persistent *cachedir*.

Keyword Arguments

- **cachedir** – Directory for cached python objects. *AnyCache* instances on the same *cachedir* share the same cache.
- **maxsize** – Maximum cache size in bytes. *None* does not limit the cache size. *0* disables caching. It the maximum size is smaller than the last cached object, this object is kept. During object write the cache size might be larger than *maxsize*. At maximum twice as large as the maximum object size.
- **depfilefunc** – Dependency file function (see example below)

```
>>> from anycache import anycache
>>> @anycache()
... def myfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> myfunc(2, 5)
  Calcing 2 + 5 = 7
7
>>> myfunc(2, 5)
7
```

File I/O is not tracked by the decorator. Instead a function needs to be implemented, which returns the paths of the files, which influence the function result. The *depfilefunc* is called with the function result and all arguments. The following example, depends on the path of the source code itself:

```
>>> def mydepfilefunc(result, posarg, kwarg=3):
...     print("  Deps of %r + %r = %r" % (posarg, kwarg, result))
...     return [__file__]
>>> @anycache(depfilefunc=mydepfilefunc)
... def myfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> myfunc(2, 7)
  Calcing 2 + 7 = 9
  Deps of 2 + 7 = 9
9
```

anycache.**get_defaultcache**()
Return unlimited default *AnyCache* instance.

CHAPTER 3

Getting started

To cache the result of a function, use the global unlimited anycache:

```
>>> from anycache import anycache
>>> @anycache()
... def myfunc(posarg, kwarg=3):
...     print("  Calcing %r + %r = %r" % (posarg, kwarg, posarg + kwarg))
...     return posarg + kwarg
>>> myfunc(8, 10)
  Calcing 8 + 10 = 18
18
>>> myfunc(8, 10)
18
```

anycache caches nearly any python object. Also *lambda* statements. It uses Dill as backend, an improved version of pythons build-in *pickle*.

Set a persistent cache directory to preserve the result between multiple python runs:

```
>>> from anycache import anycache
>>> @anycache(cachedir='/tmp/anycache.my')
... def myfunc(posarg, kwarg=3):
...     return posarg + kwarg
```

The *AnyCache* object serves additional functions for cache clearing and size handling.

Python Module Index

a

Index