dataflake.cache Documentation

Release 1.0

Jens Vagelpohl

January 19, 2010

CONTENTS

`dataflake.cache` provides caching implementations based on a very simple API.

CHAPTER

ONE

NARRATIVE DOCUMENTATION

Narrative documentation explaining how to use `dataflake.cache`.

1.1 Installation

You will need Python version 2.4 or better to run `dataflake.cache`. Development of `dataflake.cache` is done primarily under Python 2.6, so that version is recommended.

> **Warning:** To successfully install `dataflake.cache`, you will need an environment capable of compiling Python C code. See the documentation about installing, e.g. `gcc` and `python-devel` for your system. You will also need *setuptools* installed on within your Python system in order to run the `easy_install` command.

It is advisable to install `dataflake.cache` into a *virtualenv* in order to obtain isolation from any "system" packages you've got installed in your Python version (and likewise, to prevent `dataflake.cache` from globally installing versions of packages that are not compatible with your system Python).

After you've got the requisite dependencies installed, you may install `dataflake.cache` into your Python environment using the following command:

```
$ easy_install dataflake.cache
```

1.2 Using dataflake.cache

`dataflake.cache` provides several cache implementations with a shared simplified API.

Using a SimpleCache object:

```
>>> from dataflake.cache.simple import SimpleCache
>>> cache = SimpleCache()
>>> cache.set('key1', 'value1')
>>> cache.get('key1')
'value1'
>>> cache.invalidate('key1')
>>> cache.get('key1', default='not available')
'not available'
```

To attach a specific lifetime to cached items, a cache implementation with built-in timeout is provided as well:

```
>>> import time
>>> from dataflake.cache.timeout import TimeoutCache
>>> cache = TimeoutCache()
>>> cache.setTimeout(1)
```

```
>>> cache.set('key1', 'value1')
>>> cache.get('key1')
'value1'
>>> time.sleep(1)
>>> cache.get('key1', default='not available')
'not available'
```

Both the simple and timeout caches are available as thread-safe implementations using locks, see the *dataflake.cache.simple* and *dataflake.cache.timeout* documentation.

The *Interfaces* page contains more information about the cache APIs.

1.3 Development

1.3.1 Getting the source code

The source code is maintained in the Dataflake Subversion repository. To check out the trunk:

```
$ svn co http://svn.dataflake.org/svn/dataflake.cache/trunk/
```

You can also browse the code online at http://svn.dataflake.org/viewvc/dataflake.cache.

When using setuptools or zc.buildout you can use the following URL to retrieve the latest development code as Python egg:

```
$ http://svn.dataflake.org/svn/dataflake.cache/trunk#egg=dataflake.cache
```

1.3.2 Bug tracker

For bug reports, suggestions or questions please use the dataflake bug tracker at http://www.dataflake.org/tracker.

1.3.3 Setting up a development sandbox and testing

Once you've obtained a source checkout, you can follow these instructions to perform various development tasks. All development requires that you run the buildout from the package root directory:

```
$ python bootstrap.py
$ bin/buildout
```

Once you have a buildout, the tests can be run as follows:

```
$ bin/test
```

1.3.4 Building the documentation

The Sphinx documentation is built by doing the following from the directory containing setup.py:

```
$ cd docs
$ make html
```

1.3.5 Making a release

The first thing to do when making a release is to check that the ReST to be uploaded to PyPI is valid:

```
$ bin/docpy setup.py --long-description | bin/rst2 html \
  --link-stylesheet \
  --stylesheet=http://www.python.org/styles/styles.css > build/desc.html
```

Once you're certain everything is as it should be, the following will build the distribution, upload it to PyPI, register the metadata with PyPI and upload the Sphinx documentation to PyPI:

```
$ bin/buildout -o
$ bin/docpy setup.py sdist upload register upload_sphinx --upload-dir=docs/_build/html
```

The `bin/buildout` will make sure the correct package information is used.

1.4 Change log

1.4.1 1.0 (2010-01-18)

- Initial release based on caching code formerly residing inside Products.LDAPUserFolder

CHAPTER

TWO

API DOCUMENTATION

API documentation for `dataflake.cache`.

2.1 Interfaces

interface dataflake.cache.interfaces.ICache
Simple cache interface

invalidate(*key=None*)
Invalidate the given key, or all key/values if no key is passed.

set(*key, value*)
Store a key/value pair

get(*key, default=None*)
Get value for the given key

If no value is found or the value is invalid, the default value will be returned.

keys()
Return all cache keys

items()
Return all cached keys and values

Returns a sequence of (key, value) tuples.

values()
Return all cached values

interface dataflake.cache.interfaces.ITimeoutCache
Extends: dataflake.cache.interfaces.ICache

Simple cache with a timeout

Only records younger than the configured timeout are returned

setTimeout(*timeout*)
Set a timeout value in seconds

getTimeout()
Get the timeout value

2.2 dataflake.cache.simple

class SimpleCache()
Simple instance-level cache

get (*key, default=None*)
Get value for the given key

If no value is found the default value will be returned.

invalidate (*key=None*)
Invalidate the given key, or all key/values if no key is passed.

items ()
Return all cached keys and values

Returns a sequence of (key, value) tuples.

keys ()
Return all cache keys

set (*key, value*)
Store a key/value pair

values ()
Return all cached values

class **LockingSimpleCache** ()
Simple module-level cache protected by a lock serializing access

get (**args, **kw*)
Get value for the given key

If no value is found the default value will be returned.

invalidate (**args, **kw*)
Invalidate the given key, or all key/values if no key is passed.

items ()
Return all cached keys and values

Returns a sequence of (key, value) tuples.

keys ()
Return all cache keys

set (**args, **kw*)
Store a key/value pair

values ()
Return all cached values

2.3 dataflake.cache.timeout

class **TimeoutCache** ()
A simple non-persistent cache with timeout

get (*key, default=None*)
Get value for the given key

If no value is found or the value is older than the allowed timeout, the default value will be returned.

getTimeout ()
Get the timeout value

invalidate (*key=None*)
Invalidate the given key, or all key/values if no key is passed.

items ()
Return all cached keys and values

Returns a sequence of (key, value) tuples.

keys()
Return all cache keys

set(*key, object*)
Store a key/value pair

setTimeout(*timeout*)
Set a timeout value in seconds

values()
Return all cached values

class LockingTimeoutCache()
Simple module-level cache protected by a lock serializing access

get(**args, **kw*)
Get value for the given key

If no value is found the default value will be returned.

getTimeout()
Get the timeout value

invalidate(**args, **kw*)
Invalidate the given key, or all key/values if no key is passed.

items()
Return all cached keys and values

Returns a sequence of (key, value) tuples.

keys()
Return all cache keys

set(**args, **kw*)
Store a key/value pair

setTimeout(*timeout*)
Set a timeout value in seconds

values()
Return all cached values

CHAPTER

THREE

SUPPORT

If you need commercial support for this software package, please contact zetwork GmbH at http://www.zetwork.com.

CHAPTER

FOUR

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*
- *Glossary*

MODULE INDEX

D

INDEX