zope.proxy Documentation

Release 4.0

Zope Foundation Contributors

Sep 27, 2017

Contents

1 zope.proxy Narrative Documentation **3**
- 1.1 Subclassing ProxyBase . . . 3
- 1.2 Using get descriptors in proxy classes . . . 4
- 1.3 Marking proxy attributes as non-overridable . . . 4
- 1.4 Changing the proxied object . . . 5

2 zope.proxy API **7**
- 2.1 zope.proxy.interfaces . . . 7
- 2.2 zope.proxy . . . 7
- 2.3 zope.proxy.decorator . . . 7

3 Hacking on zope.proxy **9**
- 3.1 Getting the Code . . . 9
- 3.2 Working in a virtualenv . . . 9
- 3.3 Using zc.buildout . . . 11
- 3.4 Using tox . . . 12
- 3.5 Contributing to zope.proxy . . . 13

4 Indices and tables **15**

Contents:

CHAPTER 1

zope.proxy Narrative Documentation

Subclassing ProxyBase

If you subclass a proxy, instances of the subclass have access to data defined in the class, including descriptors.

Your subclass instances don't get instance dictionaries, but they can have slots.

```
>>> from zope.proxy import ProxyBase
>>> class MyProxy(ProxyBase):
...     __slots__ = 'x', 'y'
...
...     def f(self):
...         return self.x

>>> l = [1, 2, 3]
>>> p = MyProxy(l)
```

You can use attributes defined by the class, including slots:

```
>>> p.x = 'x'
>>> p.x
'x'
>>> p.f()
'x'
```

You can also use attributes of the proxied object:

```
>>> p
[1, 2, 3]
>>> p.pop()
3
>>> p
[1, 2]
```

Using get descriptors in proxy classes

A non-data descriptor in a proxy class doesn't hide an attribute on a proxied object or prevent writing the attribute.

```
>>> class ReadDescr(object):
...     def __get__(self, i, c):
...         return 'read'

>>> from zope.proxy import ProxyBase
>>> class MyProxy(ProxyBase):
...     __slots__ = ()
...
...     z = ReadDescr()
...     q = ReadDescr()

>>> class MyOb:
...     q = 1

>>> o = MyOb()
>>> p = MyProxy(o)
>>> p.q
1

>>> p.z
'read'

>>> p.z = 1
>>> o.z, p.z
(1, 1)
```

Marking proxy attributes as non-overridable

Normally, methods defined in proxies are overridden by methods of proxied objects. This applies to all non-data descriptors. The non_overridable function can be used to convert a non-data descriptor to a data descriptor that disallows writes. This function can be used as a decorator to make functions defined in proxy classes take precedence over functions defined in proxied objects.

```
>>> from zope.proxy import ProxyBase
>>> from zope.proxy import non_overridable
>>> class MyProxy(ProxyBase):
...     __slots__ = ()
...
...     @non_overridable
...     def foo(self):
...         return 'MyProxy foo'

>>> class MyOb:
...     def foo(self):
...         return 'MyOb foo'

>>> o = MyOb()
>>> p = MyProxy(o)
>>> p.foo()
'MyProxy foo'
```

Changing the proxied object

```
>>> from zope.proxy import ProxyBase
>>> from zope.proxy import setProxiedObject, getProxiedObject

>>> class C(object):
...     pass

>>> c1 = C()
>>> c2 = C()

>>> p = ProxyBase(c1)
```

setProxiedObject() allows us to change the object a proxy refers to, returning the previous referent:

```
>>> old = setProxiedObject(p, c2)
>>> old is c1
True

>>> getProxiedObject(p) is c2
True
```

The first argument to *setProxiedObject()* must be a proxy; other objects cause it to raise an exception:

```
>>> try:
...     setProxiedObject(c1, None)
... except TypeError:
...     print("TypeError raised")
... else:
...     print("Expected TypeError not raised")
TypeError raised
```

CHAPTER 2

zope.proxy API

zope.proxy.interfaces

zope.proxy

zope.proxy.decorator

CHAPTER 3

Hacking on `zope.proxy`

Getting the Code

The main repository for `zope.proxy` is in the Zope Foundation Github repository:

https://github.com/zopefoundation/zope.proxy

You can get a read-only checkout from there:

```
$ git clone https://github.com/zopefoundation/zope.proxy.git
```

or fork it and get a writeable checkout of your fork:

```
$ git clone git@github.com/jrandom/zope.proxy.git
```

The project also mirrors the trunk from the Github repository as a Bazaar branch on Launchpad:

https://code.launchpad.net/zope.proxy

You can branch the trunk from there using Bazaar:

```
$ bzr branch lp:zope.proxy
```

Working in a `virtualenv`

Installing

If you use the `virtualenv` package to create lightweight Python development environments, you can run the tests using nothing more than the `python` binary in a virtualenv. First, create a scratch environment:

```
$ /path/to/virtualenv --no-site-packages /tmp/hack-zope.proxy
```

Next, get this package registered as a "development egg" in the environment:

```
$ /tmp/hack-zope.proxy/bin/python setup.py develop
```

Running the tests

Then, you canrun the tests using the build-in `setuptools` testrunner:

```
$ /tmp/hack-zope.proxy/bin/python setup.py test -q
.....................................................................................
 ↪.....................................................................
----------------------------------------------------------------------
Ran 147 tests in 0.010s

OK
```

If you have the `nose` package installed in the virtualenv, you can use its testrunner too:

```
$ /tmp/hack-zope.proxy/bin/easy_install nose
...
$ /tmp/hack-zope.proxy/bin/nosetests
.....................................................................................
 ↪.....................................................................
----------------------------------------------------------------------
Ran 149 tests in 0.107s

OK
```

If you have the `coverage` pacakge installed in the virtualenv, you can see how well the tests cover the code:

```
$ /tmp/hack-zope.proxy/bin/easy_install nose coverage
...
$ /tmp/hack-zope.proxy/bin/nosetests --with coverage
.....................................................................................
 ↪.....................................................................
Name                    Stmts   Miss  Cover   Missing
-----------------------------------------------------
zope.proxy                271      0   100%
zope.proxy._compat          2      0   100%
zope.proxy.decorator       18      0   100%
zope.proxy.interfaces      10      0   100%
-----------------------------------------------------
TOTAL                     301      0   100%
----------------------------------------------------------------------
Ran 149 tests in 0.148s

OK
```

Building the documentation

`zope.proxy` uses the nifty `Sphinx` documentation system for building its docs. Using the same virtualenv you set up to run the tests, you can build the docs:

```
$ /tmp/hack-zope.proxy/bin/easy_install Sphinx
...
$ cd docs
```

```
$ /tmp/hack-zope.proxy/bin/sphinx-build \
  -b html -d _build/doctrees   . _build/html
...
build succeeded.
```

You can also test the code snippets in the documentation:

```
$ /tmp/hack-zope.proxy/bin/sphinx-build \
 -b doctest -d _build/doctrees   . _build/doctest
...
running tests...

Document: api
-------------
1 items passed all tests:
  23 tests in default
23 tests in 1 items.
23 passed and 0 failed.
Test passed.

Document: narr
--------------
1 items passed all tests:
  37 tests in default
37 tests in 1 items.
37 passed and 0 failed.
Test passed.

Doctest summary
===============
   60 tests
    0 failures in tests
    0 failures in setup code
    0 failures in cleanup code
build succeeded.
```

Using `zc.buildout`

Setting up the buildout

`zope.proxy` ships with its own `buildout.cfg` file and `bootstrap.py` for setting up a development buildout:

```
$ /path/to/python2.6 bootstrap.py
...
Generated script '.../bin/buildout'
$ bin/buildout
Develop: '/home/jrandom/projects/Zope/BTK/event/.'
...
Generated script '.../bin/test'.
```

Running the tests

You can now run the tests:

```
$ bin/test --all
Running zope.testing.testrunner.layer.UnitTests tests:
  Set up zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
  Ran 147 tests with 0 failures and 0 errors in 0.000 seconds.
Tearing down left over layers:
  Tear down zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
```

Using `tox`

Running Tests on Multiple Python Versions

tox is a Python-based test automation tool designed to run tests against multiple Python versions. It creates a `virtualenv` for each configured version, installs the current package and configured dependencies into each `virtualenv`, and then runs the configured commands.

`zope.proxy` configures the following `tox` environments via its `tox.ini` file:

- The `py26`, `py27`, `py33`, `py34`, and `pypy` environments builds a `virtualenv` with `pypy`, installs `zope.proxy` and dependencies, and runs the tests via `python setup.py test -q`.
- The `coverage` environment builds a `virtualenv` with `python2.6`, installs `zope.proxy`, installs `nose` and `coverage`, and runs `nosetests` with statement coverage.
- The `docs` environment builds a virtualenv with `python2.6`, installs `zope.proxy`, installs `Sphinx` and dependencies, and then builds the docs and exercises the doctest snippets.

This example requires that you have a working `python2.6` on your path, as well as installing `tox`:

```
$ tox -e py26
GLOB sdist-make: .../zope.proxy/setup.py
py26 sdist-reinst: .../zope.proxy/.tox/dist/zope.proxy-4.0.2dev.zip
py26 runtests: commands[0]
...
----------------------------------------------------------------------
Ran 147 tests in 0.000s

OK
_____________________________________ summary ______________________________________
py26: commands succeeded
congratulations :)
```

Running `tox` with no arguments runs all the configured environments, including building the docs and testing their snippets:

```
$ tox
GLOB sdist-make: .../zope.proxy/setup.py
py26 sdist-reinst: .../zope.proxy/.tox/dist/zope.proxy-4.0.2dev.zip
py26 runtests: commands[0]
...
Doctest summary
===============
 60 tests
   0 failures in tests
   0 failures in setup code
   0 failures in cleanup code
build succeeded.
```

```
_____________________________________ summary ______________________________________
py26: commands succeeded
py27: commands succeeded
py32: commands succeeded
pypy: commands succeeded
coverage: commands succeeded
docs: commands succeeded
congratulations :)
```

Contributing to `zope.proxy`

Submitting a Bug Report

`zope.proxy` tracks its bugs on Github:

https://github.com/zopefoundation/zope.proxy/issues

Please submit bug reports and feature requests there.

Sharing Your Changes

Note: Please ensure that all tests are passing before you submit your code. If possible, your submission should include new tests for new features or bug fixes, although it is possible that you may have tested your new code by updating existing tests.

If have made a change you would like to share, the best route is to fork the Githb repository, check out your fork, make your changes on a branch in your fork, and push it. You can then submit a pull request from your branch:

https://github.com/zopefoundation/zope.proxy/pulls

If you branched the code from Launchpad using Bazaar, you have another option: you can "push" your branch to Launchpad:

```
$ bzr push lp:~jrandom/zope.proxy/cool_feature
```

After pushing your branch, you can link it to a bug report on Github, or request that the maintainers merge your branch using the Launchpad "merge request" feature.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search