zope.deprecation Documentation

Release 4.0

Zope Foundation Contributors

Sep 27, 2017

Contents

Contents:

CHAPTER 1

zope.deprecation API

Deprecating objects inside a module

Let's start with a demonstration of deprecating any name inside a module. To demonstrate the functionality, First, let's set up an example module containing fixtures we will use:

```
>>> import os
>>> import tempfile
>>> import zope.deprecation
>>> tmp_d = tempfile.mkdtemp('deprecation')
>>> zope.deprecation.__path__.append(tmp_d)
>>> doctest_ex = '''\
... from . import deprecated
...
... def demo1():
...     return 1
... deprecated('demo1', 'demo1 is no more.')
...
... def demo2():
...     return 2
... deprecated('demo2', 'demo2 is no more.')
...
... def demo3():
...     return 3
... deprecated('demo3', 'demo3 is no more.')
...
... def demo4():
...     return 4
... def deprecatedemo4():
...     """Demonstrate that deprecated() also works in a local scope."""
...     deprecated('demo4', 'demo4 is no more.')
... '''
>>> with open(os.path.join(tmp_d, 'doctest_ex.py'), 'w') as f:
...     _ = f.write(doctest_ex)
```

The first argument to the `deprecated()` function is a list of names that should be declared deprecated. If the first argument is a string, it is interpreted as one name. The second argument is the reason the particular name has been deprecated. It is good practice to also list the version in which the name will be removed completely.

Let's now see how the deprecation warnings are displayed.

```
>>> import warnings
>>> from zope.deprecation import doctest_ex
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     doctest_ex.demo1()
1
>>> print(log[0].category.__name__)
DeprecationWarning
>>> print(log[0].message)
demo1: demo1 is no more.

>>> import zope.deprecation.doctest_ex
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     zope.deprecation.doctest_ex.demo2()
2
>>> print(log[0].message)
demo2: demo2 is no more.
```

You can see that merely importing the affected module or one of its parents does not cause a deprecation warning. Only when we try to access the name in the module, we get a deprecation warning. On the other hand, if we import the name directly, the deprecation warning will be raised immediately.

```
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     from zope.deprecation.doctest_ex import demo3
>>> print(log[0].message)
demo3: demo3 is no more.
```

Deprecation can also happen inside a function. When we first access `demo4`, it can be accessed without problems, then we call a function that sets the deprecation message and we get the message upon the next access:

```
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     doctest_ex.demo4()
4
>>> len(log)
0
>>> doctest_ex.deprecatedemo4()
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     doctest_ex.demo4()
4
>>> print(log[0].message)
demo4: demo4 is no more.
```

Deprecating methods and properties

New let's see how properties and methods can be deprecated. We are going to use the same function as before, except that this time, we do not pass in names as first argument, but the method or attribute itself. The function then returns a

wrapper that sends out a deprecation warning when the attribute or method is accessed.

```
>>> from zope.deprecation import deprecation
>>> class MyComponent(object):
...     foo = property(lambda self: 1)
...     foo = deprecation.deprecated(foo, 'foo is no more.')
...
...     bar = 2
...
...     def blah(self):
...         return 3
...     blah = deprecation.deprecated(blah, 'blah() is no more.')
...
...     def splat(self):
...         return 4
...
...     @deprecation.deprecate("clap() is no more.")
...     def clap(self):
...         return 5
```

And here is the result:

```
>>> my = MyComponent()
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     my.foo
1
>>> print(log[0].message)
foo is no more.
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     my.bar
2
>>> len(log)
0
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     my.blah()
3
>>> print(log[0].message)
blah() is no more.
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     my.splat()
4
>>> len(log)
0
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     my.clap()
5
>>> print(log[0].message)
clap() is no more.
```

Deprecating modules

It is also possible to deprecate whole modules. This is useful when creating module aliases for backward compatibility. Let's imagine, the `zope.deprecation` module used to be called `zope.wanda` and we'd like to retain backward compatibility:

```
>>> import sys
>>> sys.modules['zope.wanda'] = deprecation.deprecated(
...     zope.deprecation, 'A module called Wanda is now zope.deprecation.')
```

Now we can import `wanda`, but when accessing things from it, we get our deprecation message as expected:

```
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     from zope.wanda import deprecated
>>> print(log[0].message)
A module called Wanda is now zope.deprecation.
```

Before we move on, we should clean up:

```
>>> del deprecated
>>> del sys.modules['zope.wanda']
```

Moving modules

When a module is moved, you often want to support importing from the old location for a while, generating a deprecation warning when someone uses the old location. This can be done using the moved function.

To see how this works, we'll use a helper function to create two fake modules in the zope.deprecation package. First will create a module in the "old" location that used the moved function to indicate the a module on the new location should be used:

```
>>> import os
>>> created_modules = []
>>> def create_module(modules=(), **kw): #** highlightfail
...     modules = dict(modules)
...     modules.update(kw)
...     for name, src in sorted(modules.items()):
...         pname = name.split('.')
...         if pname[-1] == '__init__':
...             os.mkdir(os.path.join(tmp_d, *pname[:-1])) #* highlightfail
...             name = '.'.join(pname[:-1])
...         with open(os.path.join(tmp_d, *pname) + '.py', 'w') as f:
...             f.write(src) #* hf
...         created_modules.append(name)
...     import importlib
...     if hasattr(importlib, 'invalidate_caches'):
...         importlib.invalidate_caches()
>>> create_module(old_location=
... '''
... import zope.deprecation
... zope.deprecation.moved('zope.deprecation.new_location', 'version 2')
... ''')
```

and we define the module in the new location:

```
>>> create_module(new_location=
... '''\
... print("new module imported")
... x = 42
... ''')
```

Now, if we import the old location, we'll see the output of importing the old location:

```
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     import zope.deprecation.old_location
new module imported
>>> print(log[0].message)
...
zope.deprecation.old_location has moved to zope.deprecation.new_location.
Import of zope.deprecation.old_location will become unsupported
in version 2
>>> zope.deprecation.old_location.x
42
```

Moving packages

When moving packages, you need to leave placeholders for each module. Let's look at an example:

```
>>> create_module({
... 'new_package.__init__': '''\
... print(__name__ + ' imported')
... x=0
... ''',
... 'new_package.m1': '''\
... print(__name__ + ' imported')
... x=1
... ''',
... 'new_package.m2': '''\
... print(__name__ + ' imported')
... def x():
...     pass
... ''',
... 'new_package.m3': '''\
... print(__name__ + ' imported')
... x=3
... ''',
... 'old_package.__init__': '''\
... import zope.deprecation
... zope.deprecation.moved('zope.deprecation.new_package', 'version 2')
... ''',
... 'old_package.m1': '''\
... import zope.deprecation
... zope.deprecation.moved('zope.deprecation.new_package.m1', 'version 2')
... ''',
... 'old_package.m2': '''\
... import zope.deprecation
... zope.deprecation.moved('zope.deprecation.new_package.m2', 'version 2')
... ''',
... })
```

Now, if we import the old modules, we'll get warnings:

```
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     import zope.deprecation.old_package
zope.deprecation.new_package imported
>>> print(log[0].message)
...
zope.deprecation.old_package has moved to zope.deprecation.new_package.
Import of zope.deprecation.old_package will become unsupported in version 2
>>> zope.deprecation.old_package.x
0

>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     import zope.deprecation.old_package.m1
zope.deprecation.new_package.m1 imported
>>> print(log[0].message)
...
zope.deprecation.old_package.m1 has moved to zope.deprecation.new_package.m1.
Import of zope.deprecation.old_package.m1 will become unsupported in
version 2
>>> zope.deprecation.old_package.m1.x
1

>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     import zope.deprecation.old_package.m2
zope.deprecation.new_package.m2 imported
>>> print(log[0].message)
...
zope.deprecation.old_package.m2 has moved to zope.deprecation.new_package.m2.
Import of zope.deprecation.old_package.m2 will become unsupported in
version 2
>>> zope.deprecation.old_package.m2.x is zope.deprecation.new_package.m2.x
True

>>> (zope.deprecation.old_package.m2.x.__globals__
...  is zope.deprecation.new_package.m2.__dict__)
True

>>> zope.deprecation.old_package.m2.x.__module__
'zope.deprecation.new_package.m2'
```

We'll get an error if we try to import m3, because we didn't create a placeholder for it (Python 3.6 started raising ModuleNotFoundError, a subclass of ImportError with a different error message than earlier releases so we can't see that directly):

```
>>> try:
...     import zope.deprecation.old_package.m3
... except ImportError as e:
...    print("No module named" in str(e))
...    print("m3" in str(e))
True
True
```

Before we move on, let's clean up the temporary modules / packages:

```
>>> zope.deprecation.__path__.remove(tmp_d)
>>> import shutil
>>> shutil.rmtree(tmp_d)
```

Temporarily turning off deprecation warnings

In some cases it is desireable to turn off the deprecation warnings for a short time.

To support such a feature, the `zope.deprecation` package provides a context manager class, `zope.deprecation.Suppressor`. Code running inside the scope of a `Suppressor` will not emit deprecation warnings.

```
>>> from zope.deprecation import Suppressor
>>> class Foo(object):
...     bar = property(lambda self: 1)
...     bar = deprecation.deprecated(bar, 'bar is no more.')
...     blah = property(lambda self: 1)
...     blah = deprecation.deprecated(blah, 'blah is no more.')
>>> foo = Foo()
>>> with Suppressor():
...    foo.blah
1
```

Note that no warning is emitted when `foo.blah` is accessed inside the suppressor's scope.:

The suppressor is implemented in terms of a `__show__` object. One can ask for its status by calling it:

```
>>> from zope.deprecation import __show__
>>> __show__()
True
```

Inside a suppressor's scope, that status is always false:

```
>>> with Suppressor():
...     __show__()
False
```

```
>>> with warnings.catch_warnings(record=True) as log:
...     del warnings.filters[:]
...     foo.bar
1
>>> print(log[0].message)
bar is no more.
```

If needed, your code can manage the depraction warnings manually using the `on()` and `off()` methods of the `__show__` object:

```
>>> __show__.off()
>>> __show__()
False

>>> foo.blah
1
```

Now, you can also nest several turn-offs, so that calling `off()` multiple times is meaningful:

```
>>> __show__.stack
[False]

>>> __show__.off()
>>> __show__.stack
[False, False]

>>> __show__.on()
>>> __show__.stack
[False]
>>> __show__()
False

>>> __show__.on()
>>> __show__.stack
[]
>>> __show__()
True
```

You can also reset `__show__` to `True`:

```
>>> __show__.off()
>>> __show__.off()
>>> __show__()
False

>>> __show__.reset()
>>> __show__()
True
```

Finally, you cannot call `on()` without having called `off()` before:

```
>>> __show__.on()
Traceback (most recent call last):
...
IndexError: pop from empty list
```

CHAPTER 2

Hacking on `zope.deprecation`

Getting the Code

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

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

You can get a read-only checkout from there:

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

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

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

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

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

You can branch the trunk from there using Bazaar:

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

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.deprecation
```

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

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

Running the tests

Run the tests using the build-in `setuptools` testrunner:

```
$ /tmp/hack-zope.deprecation/bin/python setup.py test
running test
.....................................................
----------------------------------------------------------------------
Ran 52 tests in 0.155s

OK
```

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

```
$ /tmp/hack-zope.deprecation/bin/easy_install nose
...
$ /tmp/hack-zope.deprecation/bin/python setup.py nosetests
running nosetests
.....................................................
----------------------------------------------------------------------
Ran 52 tests in 0.155s

OK
```

or:

```
$ /tmp/hack-zope.deprecation/bin/nosetests
.....................................................
----------------------------------------------------------------------
Ran 52 tests in 0.155s

OK
```

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

```
$ /tmp/hack-zope.deprecation/bin/easy_install nose coverage
...
$ /tmp/hack-zope.deprecation/bin/python setup.py nosetests \
    --with coverage --cover-package=zope.deprecation
running nosetests
.....................................................
Name                            Stmts   Miss  Cover   Missing
-------------------------------------------------------------
zope.deprecation                    7      0   100%
zope.deprecation.deprecation      127      0   100%
zope.deprecation.fixture            1      0   100%
-------------------------------------------------------------
TOTAL                             135      0   100%
----------------------------------------------------------------------
Ran 52 tests in 0.155s

OK
```

Building the documentation

`zope.deprecation` 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.deprecation/bin/easy_install Sphinx
...
$ bin/sphinx-build -b html -d docs/_build/doctrees docs docs/_build/html
...
build succeeded.
```

You can also test the code snippets in the documentation:

```
$ bin/sphinx-build -b doctest -d docs/_build/doctrees docs docs/_build/doctest
...

Doctest summary
===============
   89 tests
    0 failures in tests
    0 failures in setup code
build succeeded.
Testing of doctests in the sources finished, look at the  \
    results in _build/doctest/output.txt.
```

Using `zc.buildout`

Setting up the buildout

`zope.deprecation` 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/deprecation/.'
...
Generated script '.../bin/sphinx-quickstart'.
Generated script '.../bin/sphinx-build'.
```

Running the tests using

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 52 tests with 0 failures and 0 errors in 0.366 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.deprecation 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.deprecation and dependencies, and runs the tests via python setup.py test -q.
- The coverage environment builds a virtualenv with python2.6, installs zope.deprecation, installs nose and coverage, and runs nosetests with statement coverage.
- The docs environment builds a virtualenv with python2.6, installs zope.deprecation, 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.interface/setup.py
py26 sdist-reinst: .../zope.interface/.tox/dist/zope.interface-4.0.2dev.zip
py26 runtests: commands[0]
..........
----------------------------------------------------------------------
Ran 52 tests in 0.155s

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.interface/setup.py
py26 sdist-reinst: .../zope.interface/.tox/dist/zope.interface-4.0.2dev.zip
py26 runtests: commands[0]
...
Doctest summary
===============
  89 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.deprecation

Submitting a Bug Report

zope.deprecation tracks its bugs on Github:

https://github.com/zopefoundation/zope.deprecation/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.deprecation/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.deprecation/cool_feature
```

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

CHAPTER 3

Indices and tables

- genindex
- modindex
- search