sphinx_typesafe Documentation

Release

Richard Gomes

Sep 27, 2017

Contents

Code | Bugs | Forum | Docs | License | Contact

`sphinx_typesafe` is a decorator which enables dynamic type checking on Python method and function calls. It works in conjunction with Sphinx-style docstrings, which makes it particularly convenient for keeping the documentation synchronized with the code actually being executed.

- The decorator can be attached to any function or method.
- Raises `TypeError` if types of arguments do not match the specification.
- Raises `TypeError` if type of return value does not match the specification.
- Performs dynamic type checking.

CHAPTER 1

Python2

Since function annotations are not available in Python2 the way type checking for Python2 is a documentation convention for parameters based on the info field lists of sphinx. So even when you don't use type checking you can use it to generate documentation.

Syntax for Python2 using sphinx style docstrings

This is the preferred way since you will be also documenting your code.

```
from sphinx_typesafe.typesafe import typesafe

@typesafe
def foo(param_a, param_b, param_c):
        """
        :type param_a:  types.StringType
        :type param_b:  types.IntType
        :type param_c:  types.NotImplementedType
        :rtype:         types.BooleanType
        """
        # Do Something
        return True
```

Note: Observe the usage of `rtype` to specify the type returned by the function. When `rtype` is not specified, it is assumed to be `types.NoneType`.

Note: When a parameter specifies `types.NotImplementedType`, the type checking logic simply ignores that parameter, which means that you can pass any type you wish.

Syntax for Python2 using decorator arguments

This is an alternative approach, useful in circunstances where Sphinx-style documentation is not allowed or desired, for whatever reason.

```
from sphinx_typesafe.typesafe import typesafe

@typesafe( { 'param_a' : 'str',
             'param_b' : 'types.IntType',
             'param_c' : 'own_module.OwnType',
             'return'  : 'bool' } )
def foo(param_a, param_b, param_c):
        """ Some Docstring Info          """
        # Do Something
        return True
```

Note: Observe the usage of `return` to specify the type returned by the function.

You can use any Python type

So if you have defined a `Point` class in module `mod1` like below:

```
# File: mod1.py

class Point(object):
    def __init__(self, x = None, y = None):
        """ Initialize the Point. Can be used to give x,y directly."""
        self.x = x
        self.y = y
```

then you can employ this type in your code like this:

```
from mod1 import Point
from sphinx_typesafe.typesafe import typesafe

@typesafe
def foo(afunc):
    """
    :type afunc:     mod1.Point
    :rtype:          types.BooleanType
    """
    return True
```

CHAPTER 2

Python3

Warning: This is a tentative implementation which is not finished at the moment!!

The base technique is the Function Annotations proposed in PEP-3107 which is documented in Python3 What's New (see section New Syntax).

Syntax for Python3

```
from sphinx_typesafe.typesafe import typesafe

@typesafe
def foo(param_a: str, param_b: int) -> bool:
        # Do Something
        return True
```

- The @typesafe decorator will then check all arguments dynamically whenever the foo is called for valid types.
- As a quoting remark from the PEP 3107: "All annotated parameter types can be any python expression.", but for typechecking only types make sense, though.

The idea and parts of the implementation were inspired by the book: Pro Python (Expert's Voice in Open Source)

CHAPTER 3

Building from source

Start from a clean and minimalist virtual environment, for example:

```
$ pip list
pip (1.4)
setuptools (2.1)
wsgiref (0.1.2)
```

Download sources and run test cases

```
$ git clone https://github.com/frgomes/sphinx_typesafe
$ cd sphinx_typesafe
$ python setup.py devtest && py.test
```

CHAPTER 4

FAQ

Why it was called IcanHasTypeCheck ?

IcanHasTypeCheck (ICHTC), refers to the famous lolcats.

Why is now called sphinx_typesafe ?

Because *typesafe* tells immediatelly what it is about. Unfortunately, *typesafe* was already taken on PyPI, so *sphinx_typesafe* seemed to be a good alternative name which also relates to the documentation standard adopted.

CHAPTER 5

Support

Please find links on the top of this page.