sphinxcontrib-asyncio

Release 0.3.0-

Andrew Svetlov

December 04, 2020

CONTENTS

Add coroutine markup support to sphinx-based docs.

CHAPTER

ONE

INSTALLATION

1. Install from PyPI:

```
$ pip install sphinxcontrib-asyncio
```

2. Enable `sphinxcontrib-asyncio` extension in your `conf.py`:

```
extensions = ['sphinxcontrib.asyncio']
```

CHAPTER

TWO

USAGE IN DOCUMENTS

Use `cofunction` instead of `function`:

```
.. cofunction:: coro(a, b)

   Simple coroutine function.
```

coroutine coro(*a*, *b*)
Simple coroutine function.

and `comethod` instead of `method`:

```
.. class:: A

   .. comethod:: meth(self, param)

      Coroutine method.
```

class A

coroutine meth(*self*, *param*)
Coroutine method.

For more complex markup use *directive options*, e.g. `async-with` for *asynchronous context managers factories*:

```
.. cofunction:: open_url(param)
   :async-with:

   A function that returns asynchronous context manager.
```

async-with open_url(*param*)
A function that returns asynchronous context manager.

That means `open_url` can be used as:

```
async with open_url(arg) as cm:
    pass
```

`async-for` may be used for *asynchronous generator* markup:

```
.. cofunction:: iter_vals(arg)
   :async-for:

   A function the returns asynchronous generator.
```

async-for iter_vals(*arg*)

A function the returns asynchronous generator.

`iter_vals()` can be used as:

```
async for item in iter_args(arg):
    pass
```

By default `async-for` and `async-with` suppresses `coroutine`.

If both `await func()` and `async with func():` are allowed (func is both *coroutine* and *asynchronous context manager*) explicit **coroutine** flag:

```
.. cofunction:: get(url)
   :async-with:
   :coroutine:

   A function can be used in ``async with`` and ``await`` context.
```

coroutine async-with get(*url*)

A function can be used in `async with` and `await` context.

`comethod` also may be used with **staticmethod** and **classmethod** optional specifiers, e.g.:

```
.. class:: A

   .. comethod:: f(cls, arg)
      :classmethod:

      This is classmethod
```

class A

classmethod coroutine f(*cls*, *arg*)

This is classmethod

CHAPTER

THREE

USAGE IN *SPHINX.EXT.AUTODOC* EXTENSION

`sphinxcontrib-asyncio` add special documenters for autodocs, which will use *cofunction* and *comethod* directives if the function is an `async def` or is marked with `coroutine` decorator.

For example this source:

```
import asyncio


class MyClass:

    def my_func(self):
        """ Normal function """

    @asyncio.coroutine
    def my_coro(self):
        """ This is my coroutine """


@asyncio.coroutine
def coro(param):
    """ Module level async function """
```

Using this simple configuration in your *.rst* file:

```
.. autocofunction:: coro

.. autoclass:: MyClass
   :members:
```

Will yield next documentation:

coroutine coro(*param*)
Module level async function

class MyClass

my_func()
Normal function

coroutine my_coro()
This is my coroutine

You can set directive options by adding it to *autocofunction* and *autocomethod* directives:

```
.. autocofunction:: coro
   :async-for:
   :coroutine:
```

coroutine async-for coro(*param*)
Module level async function

You can also force *coroutine* prefix on not-coroutine method by overriding it as *autocomethod* directive:

```
.. autoclass:: MyClass
   :members:
   :exclude-members: my_func

   .. autocomethod:: my_func()
```

class MyClass

coroutine my_func()
Normal function

coroutine my_coro()
This is my coroutine

CHAPTER

FOUR

DISCUSSION LIST

aio-libs google group: https://groups.google.com/forum/#!forum/aio-libs

Please post your questions and ideas here.

CHAPTER

FIVE

AUTHORS AND LICENSE

The `sphinxcontrib-asyncio` package is written by Andrew Svetlov.

It's *Apache 2* licensed and freely available.

Feel free to improve this package and send a pull request to GitHub.

INDEX