樹莓派, 樹莓派之學習, 樹莓派之教育

ROCK IT 《ML》JUPYTERLAB 【丁】CODE 《七》語義 【五】鴨子！？

2019-03-15 | 懸鉤子 | 發表迴響

畢竟 mypy 是個命令列程式：

```
rock64@rock64:~$ mypy -h
usage: mypy [-h] [-v] [-V] [more options; see below]
            [-m MODULE] [-p PACKAGE] [-c PROGRAM_TEXT] [files ...]

Mypy is a program that will type check your Python code.

Pass in any files or folders you want to type check. Mypy will
recursively traverse any provided folders to find .py files:

    $ mypy my_program.py my_src_folder

For more information on getting started, see:

- http://mypy.readthedocs.io/en/latest/getting_started.html

For more details on both running mypy and using the flags below, see:

- http://mypy.readthedocs.io/en/latest/running_mypy.html
- http://mypy.readthedocs.io/en/latest/command_line.html
...
```

打算深入了解 Python

26.1. `typing` — Support for type hints

New in version 3.5.

Source code: Lib/typing.py

This module supports type hints as specified by **PEP 484**. The most fundamental support consists of the types `Any`,`Union`, `Tuple`, `Callable`, `TypeVar`, and `Generic`. For full specification please see **PEP 484**. For a simplified introduction to type hints see **PEP 483**.

The function below takes and returns a string and is annotated as follows:

```
1 def greeting(name: str) -> str:
2     return 'Hello ' + name
```

In the function `greeting`, the argument `name` is expected to be of type `str` and the return type `str`. Subtypes are accepted as arguments.

或喜多所見聞也：

TYPED PYTHON AST (ABSTRACT SYNTAX TREE) UNPARSER

Mateusz Bysiek 2016-06-16 18:13 Source

> The *typed-astunparse* Python package is to *typed-ast* as *astunparse* is to *ast*. In 10 words: it enables unparsing of Python 3 AST with type comments.
>
> The built-in *ast* module can parse Python source code into AST. It can't, however, generate source code from the AST. That's where *astunparse* comes in. Using a refactored version of an obscure script found in official Python repository, it provides code generation capability for native Python AST.
>
> The *ast* and *astunparse* modules, however, completely ignore type comments introduced in PEP 484. They treat them like all other comments, so when you parse the code using `compile()`, your type comments will be lost. There is no place for them in the AST, so obviously they also cannot be unparsed.
>
> The *typed-ast* module provides an updated AST including type comments defined in PEP 484 and a parser for Python code that contains such comments.

Unfortunately, *typed-ast* doesn't provide any means to go from AST back to source code with type comments. This is where this module, *typed-astunparse*, comes in. It provides unparser for AST defined in *typed-ast*.

links

- *ast*:

  https://docs.python.org/3/library/ast.html

  https://greentreesnakes.readthedocs.io/

- *astunparse*:

  https://pypi.python.org/pypi/astunparse

  https://github.com/simonpercivall/astunparse

  https://astunparse.readthedocs.io/en/latest/

- PEP 483 – The Theory of Type Hints:

  https://www.python.org/dev/peps/pep-0483/

- PEP 484 – Type Hints:

  https://www.python.org/dev/peps/pep-0484/

- PEP 3107 – Function Annotations:

  https://www.python.org/dev/peps/pep-3107/

- PEP 526 – Syntax for Variable Annotations:

  https://www.python.org/dev/peps/pep-0526/

- *typed-ast*:

https://pypi.python.org/pypi/typed-ast

https://github.com/python/typed_ast

也許可藉

mbdevpl/static-typing

Add static type information into Python abstract syntax trees.

static-typing

Attempt to add static type information to Python abstract syntax trees (ASTs).

Works best with ASTs from `typed_ast` module, however it also works with built-in `ast` module.

Be advised that this is an ongoing work, and current implementation is subject to sudden changes.

Contents

想想可能的應用哩☆

※ 註： Python 3.6 前之版本， AST 驗證不可用。

故省略部分範例程式碼。

static-typing examples

Static type information in AST

```
import typing as t
import static_typing as st

code = '''

import typing as t

spam = ['spam', 'spam', 'spam']  # type: t.List[str]

class MyClass:
    def __init__(self):
        self.a = 42  # type: int
        self.b = 42.0  # type: float

def add_one(n: int) -> int:
    temp = n + 1  # type: int
    return temp

'''

module = st.parse(code)
```

```
AST validation not supported for Python < 3.6
```

```
module._module_vars
```

```
{'spam': OrderedSet([typing.List[str]])}
```

```
module._classes
```

```
{'MyClass': <StaticallyTypedClassDefClass@547745807832>}
```

```
cls = module._classes['MyClass']
cls._instance_fields
```

```
{'a': OrderedSet([<class 'int'>]), 'b': OrderedSet([<class 'float'>])}
```

```
cls._methods
```

```
{'__init__': <StaticallyTypedFunctionDefClass@547745808000>}
```

```
method = cls._methods['__init__']
method._kind
```

```
<FunctionKind.Constructor: 7>
```

```
method._nonlocal_assignments
```

```
{<typed_ast._ast3.Attribute at 0x7f8831beb8>: OrderedSet([<class 'int'>]),
 <typed_ast._ast3.Attribute at 0x7f8831bf98>: OrderedSet([<class 'float'>])}
```

```
module._functions
```

```
{'add_one': <StaticallyTypedFunctionDefClass@547745621496>}
```

```
function = module._functions['add_one']
function._params
```

```
{'n': OrderedSet([<class 'int'>])}
```

```
function._returns
```

```
OrderedSet([<class 'int'>])
```

```
function._kind
```

```
<FunctionKind.Function: 1>
```

```
function._local_vars
```

```
{'temp': OrderedSet([<class 'int'>])}
```

Other features

AST validation