Python2018

compscicenter.ru

aleksey.kladov@gmail.com

Лекция 12: Модули

Модуль

./useless.py

```
"""I'm a (not very) useful module."""

some_variable = "foobar"

def boo():
    return 42


def test_boo():
    assert boo() == 42

print("test_boo")
test_boo()
```

import

./main.py

```
print("before import")
import useless
print([n for n in dir(useless) if not n.startswith("_")])
print(useless.__name__)
print(useless.__doc__)
print(useless.__file__)
```

```
before import
test_boo
['boo', 'some_variable', 'test_boo']
useless
I'm a (not very) useful module.
/home/matklad/code_bits/useless.py
```

import

- `import` это “executable statement” (знакомо)
- возвращает модуль (объект)
- при импорте код модуля исполняется
- `globals()` это `__dict__` модуля

__name__ == __main__

./spam.py

```
def test_foo():
    ...

if __name__ == "__main__":
    print("Running tests ...")
    test_foo()
    print("OK")
```

```
$ python3 spam.py
```

При запуске из командной строки, имя модуля это `__main__`

sys.path

./main.py

```
import sys
print(sys.path)
```

```
$ python3 main.py
```

```
['/home/matklad/code_bits',  # папка с main.py
 # стандартная библиотека и системные модули
 '/nix/store/python3-3.6.7/lib/python36.zip',
 '/nix/store/python3-3.6.7/lib/python3.6',
 '/nix/store/python3-3.6.7/lib/python3.6/lib-dynload',
 '/nix/store/python3-3.6.7/lib/python3.6/site-packages']
```

sys.path

./main.py

```
import collections
print(collections.__file__)
```

```
/nix/store/python3-3.6.7/lib/python3.6/collections/__init__.py
```

sys.path

./main.py

```
import collections
print(collections.__file__)
```

./collections.py

```
print("Hijacked!")
```

```
"Hijacked!"
/home/matklad/code_bits/collections.py
```

sys.path

- Модули ищутся в `sys.path`
- Первое вхождение побеждает

Локальные модули скрывают системные

Формы import

```
from useless import boo as _boo, some_variable

# то же без сахара

import useless
_boo = useless.boo
some_variable = useless.some_variable
del useless  # зачем ?
```

Формы import

```
from useless import *

# то же без сахара

import useless
useless_names = getattr(
    useless, "__all__",
    (n for n in dir(useless) if not n.startswith('_'))
)
for name in useless_names:
    globals()[name] = getattr(useless, name)

del useless_names
del useless
```

Формы import

./useless.py

```
import collections

def public():
    pass

def _private():
    pass
```

./main.py

```
from useless import *

assert collections is not None  # Oops
```

Формы import

./useless.py

```
import collections

__all__ = ['public']

def public():
    pass

def _private():
    pass
```

import *

- Можно импортировать лишнего
- Можно случайно скрыть имя
- Сложно читать код

Вывод: лучше не использовать `import *`

PEP8

Все импорты — в начале файла

```
# No
import sys

def foo():
    from collections import Counter
    # ...

# Yes
import sys
from collections import Counter

def foo():
    ...
```

PEP8

Каждому импорту — своя строка

```
# No
from collections import Counter
import sys, os
from collections import namedtuple

# Yes
import os
import sys
from collections import Counter, namedtuple  # !
```

PEP8

Импорты групируются: stdlib, библиотеки, текущий проект

```
# No
import os
import sys
import click
from superapp import Config
from scipy import stats

# Yes
import os
import sys

import click
from scipy import stats

from superapp import Config
```

PEP8

Импорты сортируются лексикографически

```
# No
from collections import Counter
import sys
import os
from itertools import islice

# Yes
import os
import sys
from collections import Counter
from itertools import islice
```

Вложенные модули

Модуль с подмодулями называется `package`

```
./useless
    __init__.py  # useless
    foo.py       # useless.foo
    bar.py       # useless.bar
```

- `__init__.py`: код модуля-пакета, опционален.

Импорт вложенных модулей

```
import useless

# Attribute error
# useless.foo

assert 'foo' not in dir(useless)

import useless.foo

assert useless.foo is not None
assert 'foo' in dir(useless)
assert 'bar' not in dir(useless)
```

Относительные импорты

./useless/foo.py

```
import useless.bar  # absolute
from . import bar   # relative
```

- `.` — `package` текущего модуля
- `..` — родительский `package`
- Не зависит от `sys.path`

PEP8

> *Absolute imports are recommended, as they are usually more readable and tend to be better behaved (or at least give better error messages) if the import system is incorrectly configured (such as when a directory inside a package ends up on sys.path):*
>
> — *PEP8*

PEP8

> *Absolute imports are recommended, as they are usually more readable and tend to be better behaved (or at least give better error messages) if the import system is incorrectly configured (such as when a directory inside a package ends up on sys.path):*
>
> — *PEP8*

Артефакт Python 2, в Python 3 относительные импорты надёжнее абсолютных.

Относительные импорты vs скрипты

- Текущий пакет определяется значением `__package__`
- `$ python my-tool.py` устанавливает `__package__` в `None`
- `$ python -m my-tool` требует, чтобы `my-tool` был в `sys.path`

Паттерт façade

./useless/init.py

```
from .bar import A, B
from .foo import foo
```

- API маленькое, имплементация большая
- Плоское API лекго использовать
- Разбиение на пакеты — детали имплементации, может меняться

Фасад энергично импортирует дерево модулей.
Большое дерево — долгий импорт.

Паттерн façade

```
# ./useless/foo.py
__all__ = [...]

# ./useless/bar/__init__.py
__all__ = [...]

# ./useless/__init__.py
from .bar import *
from .foo import *

__all__ = bar.__all__ + foo.__all__
```

import изнутри

```
import sys
import useless as u1
import useless as u2

assert u1 is u2 is sys.modules['useless']
```

- импорт создаёт ссылку в `sys.path`
- если модуль уже есть в `sys.path`, то второй раз он не импортируется

import изнутри

```
import sys
import useless as u1
del sys.modules['useless']
import useless as u2

assert u1 is not u2
```

import изнутри

```
import useless

# то же без сахара

useless = __import__('useless', globals(), None, None, 0)
```

`__import__` — внутренняя функция

`importlib.import_module` — публичное API

__import__ изнутри

```
import types

# Создали пустой модуль
mod = types.ModuleType("useless")

# Прочитали код модуля
with open("useless.py") as file:
    source = file.read()

code = compile(source, "useless.py", mode="exec")

# Добавили пустой модуль в `sys.path`
sys.modules["useless"] = mod

# Выполнили тело модуля
exec(code, mod.__dict__)
```

__import__ изнутри

Модуль попадает в `sys.modules` до исполнения

Циклические импорты

```
# ./foo.py
import bar
def do_foo():
    return bar.do_bar()

# ./bar.py
import foo
def do_bar():
    pass
foo.do_foo()

# ./main.py
# работает только в таком порядке!
import bar
import foo
```

Замена модуля

façade тормозит ?

./useless/init.py

```
from .foo import Foo
from .bar import Bar
```

```
_imports = {'.foo': ['Foo'], '.bar': 'Bar'}
_import_from = {n: m for m, ns in _imports.items()
                    for n in ns}

class OnDemandModule(types.ModuleType):
    def __getattr__(self, name):
        modulename = _import_from.get(name)
        if not modulename:
            raise AttributeError('No attribute %s' % name)
        m = importlib.import_module(modulename, __package__)
        value = getattr(m, name)
        setattr(self, name, value)
        return value

newmodule = OnDemandModule(__name__)
newmodule.__dict__.update(globals())
newmodule.__all__ = list(_import_from)
sys.modules[__name__] = newmodule
```

>= 3.7

```
_imports = {'.foo': ['Foo'], '.bar': 'Bar'}
_import_from = {n: m for m, ns in _imports.items()
                    for n in ns}

def __getattr__(module, name):
    modulename = _import_from.get(name)
    if not modulename:
        raise AttributeError('No attribute %s' % name)
    m = importlib.import_module(modulename, __package__)
    value = getattr(m, name)
    setattr(module, name, value)
    return value
```

Больше про syspath

```
['',  # cwd для интерактивных сессий
 # стандартная библиотека
 '/nix/store/python3-3.6.7/lib/python36.zip',
 '/nix/store/python3-3.6.7/lib/python3.6',
 '/nix/store/python3-3.6.7/lib/python3.6/lib-dynload',
 # установленные сторонние библиотеки
 '/nix/store/python3-3.6.7/lib/python3.6/site-packages']
```

- `sys.path` зависит от OS
- `sys.path` изменяемый
- Переменная окружения `PYTHONPATH` добавляет пути в конец `sys.path`

Конфликты имён

- Не может существовать двух модулей с одинаковыми именами
- Случайное совпадение кажется маловероятным
- За исключением одного случая ...

Конфликты имён

- Не может существовать двух модулей с одинаковыми именами
- Случайное совпадение кажется маловероятным
- За исключением одного случая ...
- Две версии одного и того же модуля :-(

PIP

`pip`

утилита для установки сторонних библиотек

`pip install pytest`

установит пакет в глобальный `site-packages` :(:(:(

`pip install pytest --user`

установит пакет в `site-pacakges` пользователя :(

Как избежать dependency hell?

Dependency hell

Опция 1: использовать **только** стандартную библиотеку.

Единственный выход для вспомогательных скриптов :(

venv

`venv`

утилита для создания вируальных окружений — изолированных site-packages

```
$ python3 -m venv my-env
$ source ./my-env/bin/activate
my-env $ which python3
/home/matklad/my-env/bin/python3
my-env $ python3 -c 'import sys; print(sys.path)'
['',
 '/nix/store/python3-3.6.7/lib/python36.zip',
 '/nix/store/python3-3.6.7/lib/python3.6',
 '/nix/store/python3-3.6.7/lib/python3.6/lib-dynload',
 '/home/matklad/my-env/lib/python3.6/site-packages'] # yay!
```

pipenv

`pipenv`

`venv` + `pip` + `Pipfile` + `Pipfile.lock`

`Pipfile`

Список **ограничений** на прямые зависимости. Позволяет добавлять и обновлять зависимости.

`Pipfile.lock`

Список **версий** транзитивных зависимостей. Решает проблему "works on my machine".

Настройка import

- `sys.path_hooks` — настройка интерпретации элементов `sys.path`
- `sys.meta_path` — полный контроль над процессом импорта

```
import sys
def finder_for_path(path):
    for hook in sys.path_hooks:
        try:
            return hook(path)
        except ImportError:
            continue
    return None

for path in sys.path:
    print(finder_for_path(path))
```

```
FileFinder('/home/matklad/code_bits')
<zipimporter object "/nix/store/python3/lib/python36.zip">
FileFinder('/nix/store/python3/lib/python3.6')
FileFinder('/nix/store/python3/lib/python3.6/lib-dynload')
FileFinder('/nix/store/python3/lib/python3.6/site-packages')
```

Плохая идея: модули по HTTP

```
import re
from urllib.request import urlopen

def url_hook(url):
    if not url.startswith(("http", "https")):
        raise ImportError

    with urlopen(url) as response:
        data = response.read().decode()

    filenames = re.findall(r"[a-zA-Z_][a-zA-Z0-9_]*\.py", data)
    modnames = {name[:-3] for name in filenames}
    return URLFinder(url, modnames)

sys.path_hooks.append(url_hook)  # !
```

Плохая идея: модули по HTTP

```
from importlib.abc import PathEntryFinder
from importlib.util import spec_from_loader

class URLFinder(PathEntryFinder):
    def __init__(self, url, modnames):
        self.url = url
        self.modnames = modnames

    def find_spec(self, name, target=None):
        if name not in self.modnames:
            # этот путь не содержит модуля,
            # продолжить поиск в sys.path
            return None
        origin = f"{self.url}/{name}.py"
        loader = URLLoader()
        return spec_from_loader(name, loader, origin=origin)
```

Плохая идея: модули по HTTP

```
from importlib.abc import Loader

class URLLoader(Loader):
    def create_module(self, spec):
        return None

    def exec_module(self, mod):
        with urlopen(mod.__spec__.origin) as response:
            source = response.read()
        code = compile(source, mod.__spec__.origin, mode="exec")
        exec(code, mod.__dict__)
```

sys.path_hooks

- `path_hook` определяет, может ли он обработать строчку из `sys.path`
- `path_hook` возвраящает `PathEntryFinder`
- `PathEntryFinder` определяет, если ли такой модуль и возвращает `ModuleSpec`
- `ModuleSpec` умеет загружать модуль, используя `Loader`

Можно проверить, есть ли модуль, не импортируя его!

Look before you leap

```
try:
    import _useless_speedups as useless
except ImportError:
    import useless
```

```
from importlib.util import find_spec

if find_spec("_useless_speedups"):
    import _useless_speedups as useless
else:
    import useless
```

sys.meta_path

`sys.path_hook` : настройка импорта модуля по пути.

`sys.meta_path` : настройка импорта модуля по имени.

```
import sys
print(sys.meta_path)

<class '_frozen_importlib.BuiltinImporter'>,      # встроенные
<class '_frozen_importlib.FrozenImporter'>,       # замороженные
<class '_frozen_importlib_external.PathFinder'>   # sys.path
```

```
import sys
import subprocess
from importlib.abc import MetaPathFinder
from importlib.util import find_spec

class AutoInstall(MetaPathFinder):
    _loaded = set()

    @classmethod
    def find_spec(cls, name, path=None, target=None):
        if path is not None or name in cls._loaded:
            return None
        cls._loaded.add(name)
        cp = subprocess.run([
            sys.executable, "-m", "pip", "install", name
        ])
        if cp.returncode == 0:
            return find_spec(name)
        print("Failed!", file=sys.stderr)
        return None
```

Что читать в транспорте

- http://dabeaz.com/modulepackage/