

JUL 12TH, 2015

## Useful Hack: Lazy module attribute

昨天晚上师兄在 qq 上和我诉苦，说我们的代码测试起来太不方便了。问题大概出在这段代码：

```
# motorclient.py
import motor
from settings import mongo_machines, REPLICASET_NAME
from pymongo import ReadPreference

_motorclient = motor.MotorReplicaSetClient(
    ','.join(mongo_machines),
    replicaSet=REPLICASET_NAME,
    readPreference=ReadPreference.NEAREST)

fbt = _motorclient.fbt
reward = _motorclient.fbt_reward
fbt_log = _motorclient.fbt_log
```

这是我之前写的对 motor 的简单封装，无关代码已经去掉。目的很简单，他们每次要访问数据库只要先 `import motorclient`，然后用 `motorclient.dbname` 操作各个数据库就行了。问题在哪里呢？

发现 motorclient 好蛋疼，只要一 `import` 就必须连接数据库，本地测试每次都要打 mock

这是他的原话，他想在没有配置副本集的本机进行测试，然而 motorclient 只要一 `import` 就会初始化一个 `MotorReplicaSetClient`，于是只能 Mock。于是现在有如下需求：

1. 希望保证现有接口不变
2. 不要一 `import` 就初始化
3. 能非常容易地变成只连接本地数据库而不是副本集

怎么办呢？

我突然想到，David Beazley 的演讲里好像提到了这个概念（关于他的演讲请参考 [PyCon2015 笔记](#)）。在 [slide](#) 的 150-152 页，他当时想实现的是，`import` 某个 package 的时候，不直接把 submodules/subpackage 都给 `import` 进来（因为很耗

时间，相当于把所有文件执行一次），而是**按需 import**，他把这个技巧叫 "Lazy Module Assembly"。我面临的需求和他类似，也要用 "lazy" 的方式加载，只不过针对的是一个 module 里的变量。

上网搜了搜，参考了 [SO 上的某答案](#)，完成了 lazy 版，我把它叫做 **lazy module attribute**。

```
# motorclient.py
mode = None
def set_mode(m):
    global mode
    mode = m

class Wrapper:

    localhost = '127.0.0.1'
    port = 27017
    dbs = {
        'fbt': None,
        'reward': None,
        'fbt_log': None
    }

    def __init__(self, module):
        self.module = module
        self._motorclient = None

    def __getattr__(self, item):
        if item in self.dbs and self._motorclient is None:
            if mode == 'test':
                self._motorclient = motor.MotorClient(host=self.localhost,
                                                         port=self.port)
            else:
                self._motorclient = motor.MotorReplicaSetClient(
                    ','.join(mongo_machines),
                    replicaSet=REPLICASET_NAME,
                    readPreference=ReadPreference.NEAREST)

            self.dbs['fbt'] = self._motorclient.fbt
            self.dbs['reward'] = self._motorclient.fbt_reward
            self.dbs['fbt_log'] = self._motorclient.fbt_log
            self.module.__dict__.update(self.dbs)

        return getattr(self.module, item)

sys.modules[__name__] = Wrapper(sys.modules[__name__])
```

它的工作流程是这样：

1. 在 `import motorclient`，会创建一个 `Wrapper` 类的实例替换掉这个 module 本身，并且把原来的 module object 赋给 `self.module`，别的什么都不做。

2. 然后我们在别的文件中访问 `motorclient.fbt`，进入 `Wrapper` 实例的 `__getattr__` 方法，`item='fbt'`。因为是初次访问，`self._motorclient is None` 的条件满足，这时开始初始化变量。

3. 根据全局变量 `mode` 的值，我们会创建 `MotorClient` 或是 `MotorReplicaSetClient`，然后把那几个数据库变量也赋值，并且更新 `self.module.__dict__.update(self.dbs)`。这个效果就和我们的老版本初始化完全一样了，相当于直接把变量定义写在文件里。

4. 然后调用 `getattr(self.module, item)`，因为我们已经更新过 `self.module` 的 `__dict__`，所以能够正常返回属性值。第一次访问至此结束

5. OK，下一次再访问 `motorclient.fbt`，因为 `self._motorclient` 已经有值了，所以我们就不再初始化，直接把活交给 `self.module` 就好了。

下面的内容比较 `internal`，看不下去的同学就不要看了。。。

本来说到这里就差不多了，不过对 Python import 机制比较了解的同学可能会看出代码中的一个潜在问题。就是这句话：

```
python sys.modules[__name__] = Wrapper(sys.modules[__name__])
```

为什么 `sys.modules` 的 key 一定就是 `__name__` 呢？下面将追根溯源，证明这一点。

据 Brett Cannon 在《How Import Works》演讲的 [slide](#) 第 27 页对 `load_module` 函数的描述，`sys.modules` 所用的 key 是 `fullname`：

`module = sys.modules.setdefault(fullname, imp.new_module(fullname))`

`source_path = self.get_filename(fullname)`

`bytecode_path = imp.cache_from_source(source_path)`

`stats = self.path_stats(source_path)`

`data = self.get_data(bytecode_path)`

如果标准库中有类似 `module.__name__ = fullname` 这种东西，那么我们可以断定 `__name__` 就是 `fullname`。于是苦逼地翻了半天源码，好在终于找到了，有这么一句话：`python module.__name__ = spec.name`

那么这个 Spec 又是什么呢？它实际上是 Python3.4 里才引入的一个类，官方的描述是 "A specification for a module's import-system-related state"。好，就差最后一步了！我又找到了一句代码：

```
spec = spec_from_loader(fullname, self)
```

没错，Spec 初始化的第一个参数，传入的是 `fullname`，而它恰恰被赋给了 `spec.name`。至此，我们的推理终于完成，现在可以肯定地说，`sys.modules` 的 key 就是这个 module 的 `__name__`。（实际上这个证明针对的是 Python3.4，不过这种接口肯定是向前兼容的，对于所有版本都成立）

Q.E.D.

当初写 ezcf 的时候其实就遇到过这个问题，只不过没深究。我的代码中有这么一段：

```
class BaseLoader(_BaseClass):  
    def __init__(self, *args, **kwargs):
```

```
pass

def load_module(self, fullname):
    if fullname in sys.modules:
        mod = sys.modules[fullname]
    else:
        mod = sys.modules.setdefault(fullname, imp.new_module(fullname))

    mod.__file__ = self.cfg_file
    mod.__name__ = fullname # notice this !!
    mod.__loader__ = self
    mod.__package__ = '.'.join(fullname.split('.')[::-1])

    return mod
```

当时看到别人都写 `self.__name__ = fullname`，于是就跟着这么写了，并不明白其中原理。现在终于弄清了，这是 **convention**，必须得这么写。

不是后记：

感觉好久都没有写文章了啊。。空余时间基本都去刷题了啊。。。跪求 Offer  
以及如果读者中有能内推的请联系我（然而并没有什么读者(π\_π)