理解 Python 的上下文管理器

作者：OldPanda
原文链接：https://ld246.com/article/1556387074482
来源网站：链滴

原文载于 https://old-panda.com/2019/04/17/python-context-manager/

任何 Python 教程，必然会讲解如何打开一个文件。而任何提到打开文件的地方，都必然会推荐用 with 来操作文件的读写。比如说这里有一篇非常优秀的教程，文中提到

> 在 Python 中，文件读写是通过 open() 函数打开的文件对象完成的。使用 with 语句操作文件 IO 是好习惯。

并且给出了详细的代码示例。但为什么 with 关键字能在结束这个 block 的时候自动调用 close() 呢让我们去一探究竟。

知

首先来看一下 with 是怎么来的。 with 被提出是在 PEP 343 ，其中有段对于 with 操作的详细说明。单来说，执行下面代码的前提是，要求 EXPR 的类实现了 __enter__ 和 __exit__ 方法。

```
with EXPR as VAR:
    BLOCK
```

进入 with block 之后，第一件事就是把 __enter__ 的返回值赋给 VAR ，然后执行 BLOCK 的内容，论能否顺利执行，最终都会执行 __exit__ 方法来“收拾残局”，在我们的情况下，即关闭文件。

了解了 with 的原理，那么 open 函数又是如何提供 __enter__ 和 __exit__ 方法的呢？让我们去源码找找蛛丝马迹。当我们写下 open() 某个文件时， Python 实际上调用的是这里，这个函数实质上返的是类 [TextIOWrapper](https://github.com/python/cpython/blob/3.7/Lib/_pyio.py#L1908) 实例。我们可以通过 Python shell 来证实。

```
>>> open("test.txt")
<_io.TextIOWrapper name='test.txt' mode='r' encoding='UTF-8'>
```

这个类继承自 [TextIOBase](https://github.com/python/cpython/blob/3.7/Lib/_pyio.py#L1756) ，而 TextIOBase 又继承自 [IOBase](https://github.com/python/cpython/blob/3.7/Lib/_pyio.py L281) ，这个类就是所有文件 IO 类的基类，所有文件读写的类都继承了它。通过阅读这个基类，我发现其中有这么一段代码

```
...
    ### Context manager ###

    def __enter__(self):  # That's a forward reference
        """Context management protocol.  Returns self (an instance of IOBase)."""
        self._checkClosed()
        return self

    def __exit__(self, *args):
        """Context management protocol.  Calls close()"""
        self.close()
...
```

于是我们找到了原因，当通过 with 来打开一个文件时，我们得到的是一个 IOBase 子类的实例，这实例提供各种读写文件的方法，当退出 with 代码块时，调用文件关闭方法。这样我们就不用编写大的 try...except...finally 来确保文件安全关闭了。

行

PEP 343 说了，只要一个类能提供 __enter__ 和 __exit__ 方法，我们就能用 with 来保证某个操作在码执行完毕后能继续执行，作为收尾。我们试着写一个简单的类，来证明我们的理解是对的。要求在 with 结束后，打印出一句我最喜欢的诗句。

```
>>> class Foo(object):
...     def __init__(self, bar):
...         self.bar = bar
...     def __enter__(self):
...         return self
...     def __exit__(self, *args):
...         print("苟利国家生死以，岂因祸福避趋之")
...
>>> with Foo("naive") as foo:
...     print(foo.bar)
...
naive
苟利国家生死以，岂因祸福避趋之
```

非常简单的一个类，就保证了每次代码执行之后，都能在屏幕上打印出这样一句话。但这样仍然稍显烦，毕竟我想的是能少些两句就少些两句，每次专门写这样一个类，还必须实现那两个方法，想想就啰嗦，有没有更好的办法？办法就在这句“上下文管理器”的注释里 ### Context manager ### ，微搜索一下，我们就能找到这样一篇文档，参照代码示例，我们不难写出一段简单的代码来实现同样功能。

```
>>> from contextlib import contextmanager
>>>
>>> @contextmanager
... def foo(bar):
...     yield bar
...     print("苟利国家生死以，岂因祸福避趋之")
...
>>> with foo("naive") as f:
...     print(f)
...
naive
苟利国家生死以，岂因祸福避趋之
```

这里 yield 抛出的是我们真正感兴趣的内容，后续则是 with 块结束后我们必须进行的操作。