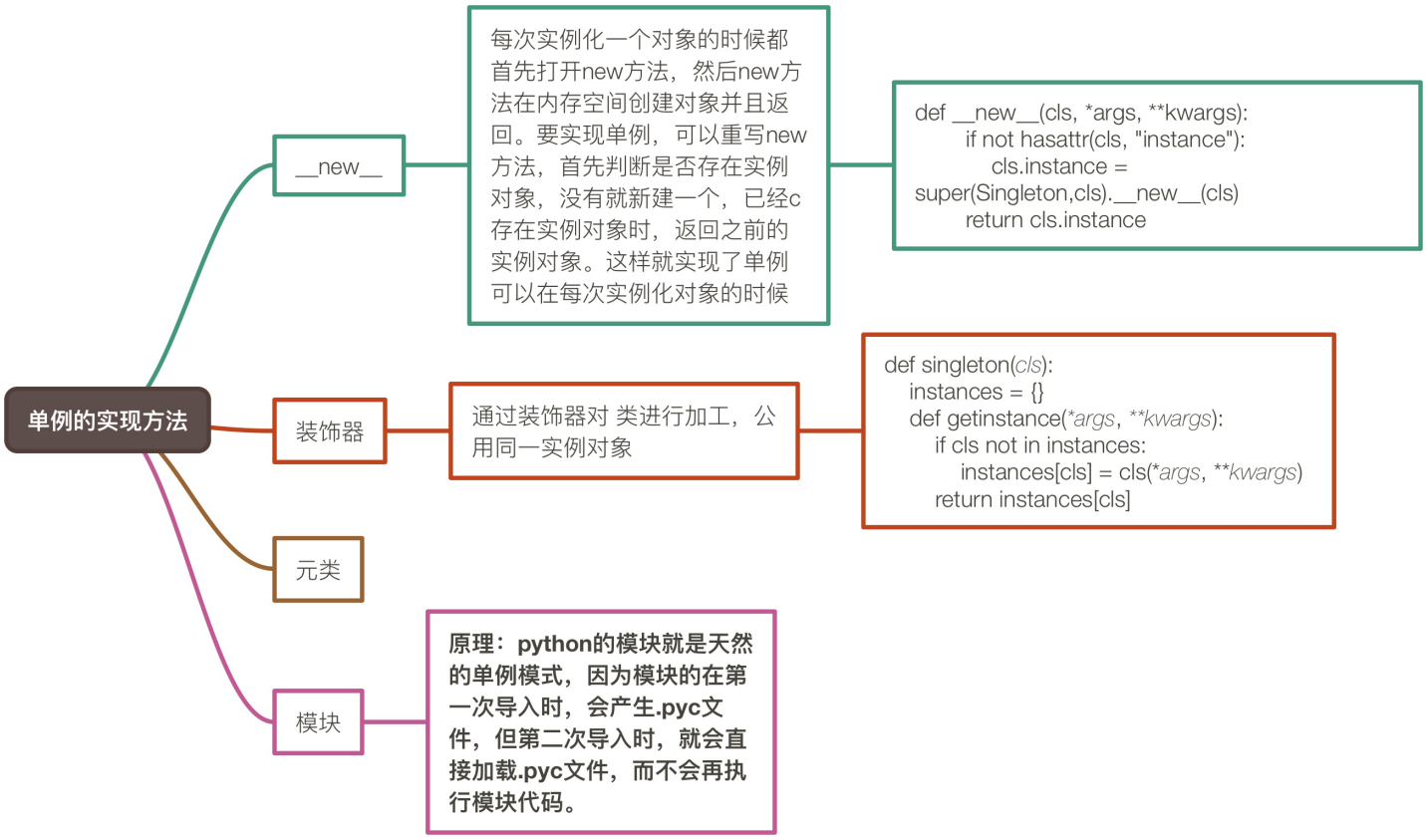
# 设计模式(Python)-单例模式




本系列文章是希望将软件项目中最常见的设计模式用通俗易懂的语言来讲解清楚，并通过Python来实现，每个设计模式都是围绕如下三个问题：

1. 为什么？即为什么要使用这个设计模式，在使用这个模式之前存在什么样的问题？
2. 是什么？通过Python语言来去实现这个设计模式，用于解决为什么中提到的问题。
3. 怎么用？理解了为什么我们也就基本了解了什么情况下使用这个模式，不过在这里还是会细化使用场景，阐述模式的局限和优缺点。

这一篇我们先来看看单例模式。单例模式是设计模式中逻辑最简单，最容易理解的一个模式，简单到只需要一句话就可以理解，即“保证只有一个对象实例的模式”。问题的关键在于实现起来并没有想象的那么简单。不过我们还是先来讨论下为什么需要这个模式吧。

## 为什么

我们首先来看看单例模式的使用场景，然后再来分析为什么需要单例模式。

- Python的logger就是一个单例模式，用以日志记录
- Windows的资源管理器是一个单例模式
- 线程池，数据库连接池等资源池一般也用单例模式
- 网站计数器

从这些使用场景我们可以总结下什么情况下需要单例模式：

1. 当每个实例都会占用资源，而且实例初始化会影响性能，这个时候就可以考虑使用单例模式，它给我们带来的好处是只有一个实例占用资源，并且只需初始化一次；
2. 当有同步需要的时候，可以通过一个实例来进行同步控制，比如对某个共享文件（如日志文件）的控制，对计数器的同步控制等，这种情况下由于只有一个实例，所以不用担心同步问题。

当然所有使用单例模式的前提是我们的确用一个实例就可以搞定要解决的问题，而不需要多个实例，如果每个实例都需要维护自己的状态，这种情况下单例模式肯定是不适用的。
接下来看看如何使用Python来实现一个单例模式。

## 导入模块方法实现单例

**原理：python的模块就是天然的单例模式，因为模块的在第一次导入时，会产生.pyc文件，但第二次导入时，就会直接加载.pyc文件，而不会再执行模块代码。**

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
@content : 模拟模块实现单例
@Time    : 2018/8/9 下午2:20
@Author  : 北冥神君
@File    : card.py
@Software: PyCharm
"""

class Card(object):
    def __init__(self, cardID, passwd, money):
        self.cardID = cardID
        self.passwd = passwd
        self.money = money

c = Card(2222,333,33)

```

```
from card import c
print(c)
```

输出结果为：

```
<card.Card object at 0x11295c048>
```

此方法一般不用，因为只能创建一个实例对象。

## __new__()方法实现单例

```
class Singleton(object):
    def __new__(cls, *args, **kwargs):  # 这里不能使用__init__，因为__init__是在instance已经生成以后才去调用的
        # 每一次实例化的时候，我都只返回instance同一个对象
        if not hasattr(cls, "instance"): #hasattr函数返回对象是否具有给定名称的属性
            cls.instance = super(Card, cls).__new__(cls)
        return cls.instance

s1 = Singleton()
s2 = Singleton()
print(s1)
print(s2)
```

打印结果如下：

```
<__main__.Singleton object at 0x7f3580dbe110>
<__main__.Singleton object at 0x7f3580dbe110>
```

可以看出3次创建对象，结果返回的是同一个对象实例，我们再让我们的例子更接近真实的使用场景来看看

```
class Singleton(object):
    def __new__(cls, *args, **kwargs):
        if not hasattr(cls, "instance"):
            cls.instance = super(Singleton, cls).__new__(cls)
        return cls.instance
    # 返回实例。。因为每次实例化一个对象的时候都首先打开new方法，然后new方法在内存空间创建对象返回，要实现单例，可以在每次实例化对象的时候
    # 没有就创建，已经创建对象时，返回同一个对象。这样就实现了单例
```

```
    def __init__(self, status_number):
        self.status_number = status_number


# 实例化3个对象。
s1 = Singleton(2)
s2 = Singleton(6)
s3 = Singleton(8)
print(s1.status_number, s2.status_number, s3.status_number)
print(s1, s2, s3)
```

这里我们使用了__init__方法，下面是打印结果，可以看出确实是只有一个实例，共享了实例的变量

```
8 8 8
<__main__.Singleton object at 0x10a13f3c8>
<__main__.Singleton object at 0x10a13f3c8>
<__main__.Singleton object at 0x10a13f3c8>

```

不过这个例子中有一个问题我们没有解决，那就是多线程的问题，当有多个线程同时去初始化对象时，就很可能同时判断_instance is None，从而进入初始化instance的代码中。所以为了解决这个问题，我们必须通过同步锁来解决这个问题。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
@content : 通过同步锁来解决这个问题
@Time    : 2018/8/9 下午8:29
@Author  : 北冥神君
@File    : 多线程单例解决方案.py
@Software: PyCharm
"""


import threading
```

```python

# 导入同步锁
try:
    from synchronize import make_synchronized
except ImportError:
    def make_synchronized(func):
        import threading
        func.__lock__ = threading.Lock()

        def synced_func(*args, **kws):
            with func.__lock__:
                return func(*args, **kws)

        return synced_func


class Singleton(object):
    # 同步锁
    @make_synchronized
    def __new__(cls, *args, **kwargs):
        if not hasattr(cls, "instance"):
            cls.instance = object.__new__(cls, *args, **kwarg
s)
        return cls.instance

    def __init__(self):
        self.blog = "www.tencenting.com"

    def go(self):
        pass

#多线程测试
def test_threadingv():
    e = Singleton()
    print(id(e))
    e.go()

# 单线程测试
def test_one():
    e1 = Singleton()
```

```
    e2 = Singleton() # 此时e2为默认www.tencenting.com
    e1.blog = 'Python-BJ-GP-1'      # 此时e1.blog的属性已经改变
    print('打印实例e1', e1.blog)
    print('打印实例e2', e2.blog)
    print('打印实例e1地址', e1)
    print('打印实例e2地址', e2)


if __name__ == "__main__":
    test_one()
    task = []
    # 测试生成30个线程,
    for one in range(30):
        t = threading.Thread(target=test_threadingv)
        task.append(t)
    # 开始启动所有线程
    for one in task:
        one.start()
    #关闭线程
    for one in task:
        one.join()
```

```
打印实例e1 Python-BJ-GP-1
打印实例e2 Python-BJ-GP-1
打印实例e1地址 <__main__.Singleton object at 0x1083ec6a0>
打印实例e2地址 <__main__.Singleton object at 0x1083ec6a0>
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
```

```
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
4433299104
```

至此我们的单例模式实现代码已经接近完美了，不过我们是否可以更简单地使用单例模式呢？答案是有的，接下来就看看如何更简单地使用单例模式。

## 装饰器实现单例

在Python的官方网站给了两个例子是用装饰符来修饰类，从而使得类变成了单例模式，使得我们可以通过更加简单的方式去实现单例模式
例子：（这里只给出一个例子，因为更简单，另外一个大家可以看官网例子。

```python
def singleton(cls):
    instances = {}
    def getinstance(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return getinstance


@singleton
class Card(object):
    pass

```

```
14
15 @singleton
16 class Person(object):
17     pass
18
19
20 c1 = Card()
21 c2 = Card()
22 print(c1 is c2)
23
24 p1 = Person()
25 p2 = Person()
26 print(p1 is p2)
27
```

输出结果为：

```
1 True
2 True
```

```
1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3
4  """
5  @content : python官方提供的
6  @Time    : 2018/8/9 下午9:07
7  @Author  : 北冥神君
8  @File    : python官方用装饰器实现单例.py
9  @Software: PyCharm
10 """
11
12 import functools
13
14
15 def singleton(cls):
16     ''' Use class as singleton. '''
17
18     cls.__new_original__ = cls.__new__
19
```

```
    @functools.wraps(cls.__new__)
    def singleton_new(cls, *args, **kw):
        it = cls.__dict__.get('__it__')
        if it is not None:
            return it

        cls.__it__ = it = cls.__new_original__(cls, *args, **
kw)
        it.__init_original__(*args, **kw)
        return it

    cls.__new__ = singleton_new
    cls.__init_original__ = cls.__init__
    cls.__init__ = object.__init__
    return cls

  # 例子:


@singleton
class Foo:
    def __new__(cls):
        cls.x = 10
        return object.__new__(cls)

    def __init__(self):
        assert self.x == 10
        self.x = 15


assert Foo().x == 15
Foo().x = 20
assert Foo().x == 20
```

## 元类实现单例(未完待续)