**单例模式**

**定义**：单例对象的类只能允许一个实例存在

**实现思路**：

1. 将该类的构造方法定义为私有方法，这样其他处的代码就无法通过调用该类的构造方法来实例化该类的对象，只有通过该类提供的静态方法来得到该类的唯一实例
2. 在该类内提供一个静态方法，当我们调用这个方法时，如果该类持有的引用不为空就返回这个引用，如果类保持的引用为空就创建该类的实例并将实例的引用赋予该类保持的引用

**单例的几种方式**：

1. 使用模块

Class Singleton(object):

Def foo(self):

Pass

singleton = Singleton()

from a import singleton

1. 使用装饰器

def Singleton(cls):

\_instance = {}

Def \_singleton(\*args, \*\*kargs):

If cls not in \_instance:

\_instance[cls] = cls(\*args, \*\*kargs)

return \_instance[cls]

return \_singleton

@Singleton

class A(object):

a = 1

def \_\_init\_\_(self, x=0):

self.x = x

a1 = A(2)

a2 = A(3)

1. 使用类

class Singleton(object):

def \_\_init\_\_(self):

pass

@classmethod

def instance(cls, \*args, \*\*kwargs):

if not hasattr(Singleton, “\_instance”):

Singleton.\_instance = Singleton(\*args, \*\*kwargs)

Return Singleton.\_instance

import threading

def task(arg):

obj = Singleton.instance()

print(obj)

for i in range(10):

t = threading.Thread(target=task,arg=[I,])

t.start()