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

'a EvenDriven test module'

__author__ = 'XiMingRI'

#系统模块
from queue import Queue,Empty
from threading import *

class EventManager(object):

    def __init__(self):
        """初始化事件管理器"""
        #事件对象列表
        self.__eventQueue = Queue() #双下划线开头的是私有的外部不能访问,因为
                                    #解释器对外会把它变为_EventManager__eventQueue
        #事件管理器1开关
        self.__active = False

        #事件处理线程
        self.__thread = Thread(target=self.__Run)

        #这里的__handlers是一个字典，用来保存相应的事件的响应函数,其中每个键对应的
        #值是一个列表，列表中保存了对该事件监听的响应函数，一对多
        self.__handlers = {}

    def __Run(self):
        """引擎运行"""
        while self.__active == True:
            try:
                #获取事件阻塞时间为1秒
                #注意event就不是类属性1也不是实例属性，它只是一个临时中间变量
                event = self.__eventQueue.get(block = True,timeout=1)
                self.__EventProcess(event)
            except Empty:
                pass


    #event事件有两个属性：type_、具体的值event,type_属性表明它所属的类型也就
    #是处理它所对应的函数合集，具体值event是传递给函数合集的参数。同一个值event
    #传递给不同的函数所得结果就会不同
    def __EventProcess(self,event):
        """处理事件"""
        #检查是否存在对该事件进行监听的处理函数
        if event.type_ in self.__handlers:
            #若存在，则按顺序将事件传递给处理函数执行
            for handler in self.__handlers[event.type_]:
                handler(event)


    def Start(self):
        """启动"""
        #将事件管理器设置为启动
        self.__active = True
        #启动事件处理线程
```

```
        self.__thread.start()


    def Stop(self):
        """停止"""
        #将事件管理器设置为停止
        self.__active = False
        #等待事件处理线程退出
        self.__thread.join()


    def AddEventListener(self,type_,handler):
        """绑定(包括添加)事件类型type_和监听器处理函数"""
        #如果type_类型的事件已经有了处理函数handler，那么就不用添加
        #如果没有就添加（也就是注册上）


        #尝试获取该事件类型对应的处理函数列表，若无则创建一个空列表
        try:
            handlerList = self.__handlers[type_]
        #handlerList既不是类属性也不是实例属性，只是一个局部的
        #中间变量
        except KeyError:
            handlerList = []
        self.__handlers[type_] = handlerList
        #若要注册的处理函数不在该事件的处理函数列表中，则注册该事件
        if handler not in handlerList:
            handlerList.append(handler)


    def RemoveEventListener(self,type_,handler):
        """移除监听器中对type_类型事件进行处理的函数handler"""
        #也就是解除type_类型的事件与处理函数handler的关系，type_类型的事件
        #不再需要handler函数进行处理（响应）
        try:
            handlerList = self.__handler[type_]
        except KeyError:
            return    #不能是pass而应是return，当handlerList不存在时
                      #就结束，而不能继续向下执行
        if handler in handlerList:
            handlerList.remove(handler)

    #所谓发送事件就是将事件放入事件队列中
    def SendEvent(self,event):
        """发送事件，就是向事件队列中存入事件"""
        self.__eventQueue.put(event)


"""事件对象"""
class Event(object):
    def __init__(self,type_ = None):
        self.type_ = type_  #事件类型
        self.dict = {}     #字典用于保存具体的事件数据



#测试代码
```

```
import sys
from datetime import datetime

#事件名称 新文章
EVENT_ARTICAL = 'Event_Artical'

#事件源 公众号

class PublicAccounts(object):
    def __init__(self,eventManager):
        self.__eventManager = eventManager

    def WriteNewArtical(self):
        #事件对象 写了新文章
        event = Event(EVENT_ARTICAL)
        event.dict['artical'] = u'如何写出更优雅的代码\n'

        #发送事件,所谓发送事件就是将事件放入事件队列中
        self.__eventManager.SendEvent(event)

        print(u'公众号发送新文章\n')


#监听器 订阅者

class Listener(object):
    def __init__(self,username):
        self.__username = username

    #监听器的处理函数 读文章

    def ReadArtical(self,event):
        print(u'%s 收到新文章' %self.__username)
        print(u'正在阅读新文章内： %s' %event.dict['artical'])




    #如果不是事件驱动模型即不将事件源和监听者解耦的话，应该是以下形式：
    #一：事件源（源class的某一个实例）S通过自己的__init__函数将1.事件（事件
#class的实例）event和2.监听者（监听class的实例）listener作为参数
#传入事件源内部：S = PublicAccounts（event,listener）
#self.event =event self.list = listener，同时在事件源内部
#调用监听者（监听class的实例）的处理函数self.list.handler(self.event)处理事件。
#注意以上的self都是事件源类的self。
    #二：直接在事件源中定义事件实例和监听者实例
    #这样事件源与事件、与监听者是直接接触的，每新来一个监听者事件元的定义都需要
#改变；事件与监听者的关系改变也得改变事件元的定义


#而事件驱动模式中，事件源与监听者不直接接触，或者它们根本没有关系。
    #一：事件源的定义只与事件管理器、事件发生关系（而与监听者无关）：
    #1.通过__init__函数将事件管理器eventManager实例传入内部，达成事件源
#与事件管理器的绑定。
    #2.在事件源定义中直接定制（包括创建和指定数据）事件实例。并将事件
#通过事件管理器发送（实质就是将事件加入到绑定的事件管理器的事件队列）
    #二：监听者的定义跟具体事件、事件管理器没有关系；只有自己的属性以及
```

```
#响应函数（响应函数有参数event，但是并不由监听者定义时传入参数，而是等待
#事件管理器将此响应函数注册到队列中并由事件管理器运行时才调用）



'''测试'''

def test():
    listener1 = Listener('thinkroom') #订阅者1
    listener2 = Listener('steve')    #订阅者2

    eventManager = EventManager()
    eventManager1 = EvenManager()

    #绑定事件类型和监听器响应函数（新文章）
    eventManager.AddEventListener(EVENT_ARTICAL,listener1.ReadArtical)
    eventManager.AddEventListener(EVENT_ARTICAL,listener2.ReadArtical)

    #注意绑定事件类型和监听者响应函数不需要eventManager启动起来

    eventManager.Start()

    publicAcc = PublicAccounts(eventManager)
    timer = Timer(2,publicAcc.WriteNewArtical)
    timer.start()

if __name__ =='__main__':
    test()
```