前言

在使用wxPython做GUI开发的时候，经常需要在UI中起线程来完成特定的任务，且该线程通常需要与UI交互。

在wxPython中，提供了`wxCallAfter`、`wxCallLater`和`wxPostEvent`这三个**线程安全**方法来交互。

现在设想一个场景：在UI界面中实时绘制图表(一个曲线图等)，同时UI界面用户事件响应不受到影响(比如，用户拖动窗口，不产生卡顿等阻塞用户输入响应的现象)。

基于上述场景，我们分析三种方式。

1、采用`wxCallAfter`。

首先`wxCallAfter`函数，是wxPython提供的用于在非GUI线程中调用GUI函数使用的。另外，官方解释是**"Call the specified function after the current and pending eventhandlers have been completed."**，意为在当前及等待的事件处理完成之后执行指定函数。

由此可见，如果我们使用`wxCallAfter`函数，那么当子线程与用户输入同时发生，二者处于**同步**处理状态。假设还是前言中的场景，那么当用户拖动窗口的同时子线程在实时绘制窗口图表，就会导致拖动的卡顿现象发生。

2、采用`wxCallLater`。

首先`wxCallLater`函数，是wxPython提供的用于在非GUI线程中调用GUI函数使用的。它需要一个定时器，意为在一定时间之后执行指定函数。其他与`wxCallAfter`一样。

3、采用`wxPostEvent`

首先`wxPostEvent`是wxPython中用于发送特定事件的，需要自定义事件才能配合使用。接下来，咱们基于wxPython的事件自定义一个工作线程，提供与UI交互的接口。

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

import threading
import wx
from wx.lib.newevent import NewEvent


class _BaseMetaClass(type):

    def __init__(cls, name, bases, _dict):
        super().__init__(name, bases, _dict)
        cls.POST, cls.EVT_POST = NewEvent()
        cls.FINISHED, cls.EVT_FINISHED = NewEvent()
        cls.ERROR, cls.EVT_ERROR = NewEvent()


class WorkThread(BaseThread, metaclass=_BaseMetaClass):

    def __init__(self, win, context=None, **kwargs):
        """
        @param win: the wxPython GUI Window Object
```

```
        @param data: user data（parameters for task）
        @param kwargs: deliver this to threading.Thread.__init__
        """
        self.win = win
        self.context = context
        super().__init__(**kwargs)

    def run(self):
        try:
            rv = self.execute()
        except Exception as e:
            wx.PostEvent(self.win, self.ERROR(exc=e))
        else:
            wx.PostEvent(self.win, self.FINISHED(result=rv))

    def execute(self):
        raise NotImplemented('method *run* must be implemented in sub class.')

    def post(self, data):
        """
        post an Event with data to self.win, will be handled by `handler` we bind.
        @param data: user data post to self.win, accept by bind handler
        @return: None
        """
        wx.PostEvent(self.win, self.POST(data=data))

    @staticmethod
    def _handlerWrapper(func_or_method):

        if not callable(func_or_method):
            raise TypeError('{} is not a callable object.'.format(func_or_method))

        def wrapper(event):
            return func_or_method(**event._getAttrDict())
        return wrapper

    def _bind(self, evt, handler):
        """
        bind Event to self.win with a `handler`
        @param evt: wxPython Event Binder object
        @param handler: callable, a wxPython GUI Event handler. usually be a method of self.win
        @return: None
        """
        handler = self._handlerWrapper(handler)
        self.win.Bind(evt, handler)

    def setPostHandler(self, handler):
        self._bind(self.EVT_POST, handler)

    def setFinishHandler(self, handler):
        self._bind(self.EVT_FINISHED, handler)

    def setErrorHandler(self, handler):
        self._bind(self.EVT_ERROR, handler)
```

> 1、`_BaseMetaClass`是一个元类，其主要目的在于用户继承基类`WorkThread`的时候保证每个新类都有属于自己的 事件，而不是共享一种事件类型，这是有意为之。
>
> 2、分别通过`setPostHandler`、`setFinishHandler`和`setErrorHandler`绑定，交互接口(可以是窗口对象方法也可以是普通函数)。都是通过`wx.PostEvent`实现。