layout: post
tags: [PyQt5,python,他山之石]

published : false

学习：

-https://nikolak.com/pyqt-threading-tutorial/

如果GUI app只有一个主线程，那么执行某个耗时较长的任务，会让user有卡顿的感觉，结果没法实时返回，并且也没办法终止执行这个任务。我们可以new一个线程去处理较为耗时的任务，再把处理结果返回给主线程，并在GUI上呈现出来。

just use the main thread:

```
from PyQt5 import QtGui
from PyQt5.QtWidgets import (QApplication,QLabel,QMainWindow,
                             QLineEdit,QDialog,QTextBrowser,QVBoxLayout,
                             QHBoxLayout,QPushButton,QAction,QWidget,QMessageBox)
import sys
import urllib3
import json
import time
urllib3.disable_warnings()

'''
fix :
InsecureRequestWarning
'''




'''
import design
class ThreadingTutorial(QtGui.QMainWindow, design.Ui_MainWindow):

    def __init__(self):
        super(self.__class__, self).__init__()
        self.setupUi(self)
'''
from PyQt5.uic import loadUi
class ThreadingTutorial(QMainWindow):
    #method 2 :
    def __init__(self):
        super(self.__class__, self).__init__()
        loadUi('design.ui',self)
```

```
        #self.setFixedSize(self.sizeHint())
        self.btn_start.clicked.connect(self.start_getting_top_posts)


    def _get_top_post(self,subreddit):
        url = "https://www.reddit.com/r/{}.json?limit=1".format(subreddit)
        headers = {'User-Agent': 'nikolak@outlook.com tutorial code'}
        http = urllib3.PoolManager()
        response = http.request('GET', url,headers=headers)
        data = json.loads(response.data.decode('utf-8'))
        top_post = data['data']['children'][0]['data']
        return "'{title}' by {author} in {subreddit}".format(**top_post)


    def _get_top_from_subreddits(self,subreddits):
        for subreddit in subreddits:
            yield self._get_top_post(subreddit)
            time.sleep(2) #reddit allows only 1 request per 2 seconds to be execute

    def start_getting_top_posts(self):
        subreddit_list = str(self.edit_subreddits.text()).split(',')
        if subreddit_list == ['']:
            QMessageBox.critical(self, "No subreddits",
                                       "You didn't enter any subreddits.",
                                       QMessageBox.Ok)
            return
        #进度条初始化
        self.progressBar.setMaximum(len(subreddit_list))
        self.progressBar.setValue(0)
        for top_post in self._get_top_from_subreddits(subreddit_list):
            self.list_submissions.addItem(top_post)
            self.progressBar.setValue(self.progressBar.value()+1)#更新进度条



def main():
    app = QApplication(sys.argv)
    form = ThreadingTutorial()
    form.show()
    app.exec_()

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

> So the QThread is a simple class that you can pass arguments to when creating a new instance since it has a normal **init** method. Also you don't call the run method itself, but instead you call start - calling run directly in some cases can also freeze your main thread depending on how the run method is implemented in your thread.

原文这里覆盖了**del**,如果不覆盖，有时候调用的线程会试图kill这个thread.

```
from PyQt5.QtCore import QThread
class YourThreadName(QThread):

    def __init__(self):
        QThread.__init__(self)

    def __del__(self):
        self.wait()

    def run(self):
        # your logic here
```

-http://pyqt.sourceforge.net/Docs/PyQt4/qthread.html
原文论文认为应该是这样用：

```
self.myThread = YourThreadName()
self.myThread.start()
```

直接调用run依然会有卡顿现象。start()会自动调用run()，并且不会出现freezing.我写的app里确实出现了这个现象.

python3 PyQt5 urllib3

```
from PyQt5 import QtGui
from PyQt5 import QtCore
from PyQt5.QtCore import QThread
from PyQt5.QtWidgets import (QApplication,QLabel,QMainWindow,
                             QLineEdit,QDialog,QTextBrowser,QVBoxLayout,
                             QHBoxLayout,QPushButton,QAction,QWidget,QMessageBox)
from PyQt5.uic import loadUi
import sys
import urllib3
import json
import time

urllib3.disable_warnings()

class getPostsThread(QThread):
    def __init__(self, subreddits):
        """
        Make a new thread instance with the specified
        subreddits as the first argument. The subreddits argument
        will be stored in an instance variable called subreddits
        which then can be accessed by all other class instance functions

        :param subreddits: A list of subreddit names
        :type subreddits: list
```

```
        """
        QThread.__init__(self)
        self.subreddits = subreddits

    def __del__(self):
        self.wait()

    def _get_top_post(self, subreddit):
        """
        Return a pre-formatted string with top post title, author,
        and subreddit name from the subreddit passed as the only required
        argument.

        :param subreddit: A valid subreddit name
        :type subreddit: str
        :return: A string with top post title, author,
                    and subreddit name from that subreddit.
        :rtype: str
        """
        url = "https://www.reddit.com/r/{}.json?limit=1".format(subreddit)
        headers = {'User-Agent': 'nikolak@outlook.com tutorial code'}
        http = urllib3.PoolManager()
        response = http.request('GET', url,headers=headers)
        data = json.loads(response.data.decode('utf-8'))
        try:
            top_post = data['data']['children'][0]['data']
            #{'message': 'Not Found', 'error': 404}
            #实际上这个爬虫写的有问题，没有考虑到没有找到的情况
            return "'{title}' by {author} in {subreddit}".format(**top_post)
        except Exception as e:
            return "0 result found"

    update_output = QtCore.pyqtSignal(str)
    def run(self):
        """
        Go over every item in the self.subreddits list
        (which was supplied during __init__)
        and for every item assume it's a string with valid subreddit
        name and fetch the top post using the _get_top_post method
        from reddit. Store the result in a local variable named
        top_post and then emit a SIGNAL add_post(QString) where
        QString is equal to the top_post variable that was set by the
        _get_top_post function.

        """
        for subreddit in self.subreddits:
            top_post = self._get_top_post(subreddit)
            self.update_output.emit(top_post.replace('r/',''))
            self.sleep(2)

class Window(QMainWindow):

    def __init__(self):
        super(self.__class__, self).__init__()
        loadUi('design.ui',self)
        #self.setFixedSize(self.sizeHint())
```

```
        self.btn_start.clicked.connect(self.start_getting_top_posts)

    def start_getting_top_posts(self):
        # Get the subreddits user entered into an QLineEdit field
        # this will be equal to '' if there is no text entered
        subreddit_list = str(self.edit_subreddits.text()).split(',')
        if subreddit_list == ['']:  # since ''.split(',') == [''] we use that to ch
                                    # whether there is anything there to fetch from
                                    # and if not show a message and abort
            QMessageBox.critical(self, "No subreddits",
                                       "You didn't enter any subreddits.",
                                       QMessageBox.Ok)
            return
        # Set the maximum value of progress bar, can be any int and it will
        # be automatically converted to x/100% values
        # e.g. max_value = 3, current_value = 1, the progress bar will show 33%
        self.progressBar.setMaximum(len(subreddit_list))
        # Setting the value on every run to 0
        self.progressBar.setValue(0)

        # We have a list of subreddits which we use to create a new getPostsThread
        # instance and we pass that list to the thread
        self.get_thread = getPostsThread(subreddit_list)

        # Next we need to connect the events from that thread to functions we want
        # to be run when those signals get fired

        # Adding post will be handeled in the add_post method and the signal that
        # the thread will emit is  SIGNAL("add_post(QString)")
        # the rest is same as we can use to connect any signal
        self.get_thread.update_output.connect(self.add_post)

        # This is pretty self explanatory
        # regardless of whether the thread finishes or the user terminates it
        # we want to show the notification to the user that adding is done
        # and regardless of whether it was terminated or finished by itself
        # the finished signal will go off. So we don't need to catch the
        # terminated one specifically, but we could if we wanted.
        self.get_thread.finished.connect(self.done)

        # We have all the events we need connected we can start the thread
        self.get_thread.start()
        # At this point we want to allow user to stop/terminate the thread
        # so we enable that button
        self.btn_stop.setEnabled(True)
        # And we connect the click of that button to the built in
        # terminate method that all QThread instances have
        self.btn_stop.clicked.connect(self.get_thread.terminate)
        # We don't want to enable user to start another thread while this one is
        # running so we disable the start button.
        self.btn_start.setEnabled(False)

    def add_post(self, post_text):
        """
        Add the text that's given to this function to the
        list_submissions QListWidget we have in our GUI and
```

```
        increase the current value of progress bar by 1

        :param post_text: text of the item to add to the list
        :type post_text: str
        """
        self.list_submissions.addItem(post_text)
        self.progressBar.setValue(self.progressBar.value()+1)

    def done(self):
        """
        Show the message that fetching posts is done.
        Disable Stop button, enable the Start one and reset progress bar to 0
        """
        self.btn_stop.setEnabled(False)
        self.btn_start.setEnabled(True)
        self.progressBar.setValue(0)
        QMessageBox.information(self, "Done!", "Done fetching posts!")


def main():
    app = QApplication(sys.argv)
    form = Window()
    form.show()
    app.exec_()

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