purplesky

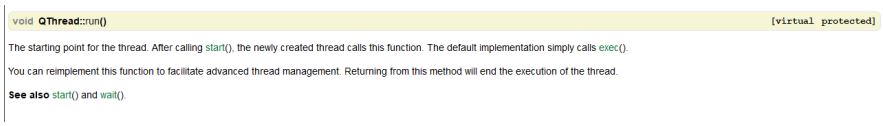
博客园 首页 新随笔 联系 订阅 管理

QT多线程的使用

Qt中提供了对于线程的支持，它提供了一些独立于平台的线程类，要进行多线程方法，可以有两种方式。

1. 第一种方式

qt提供QThread类，在QThread类中有一个virtual函数QThread::run()。



void QThread::run() [virtual protected]

The starting point for the thread. After calling start(), the newly created thread calls this function. The default implementation simply calls exec().

You can reimplement this function to facilitate advanced thread management. Returning from this method will end the execution of the thread.

See also start() and wait().

要创建一个新的线程，我们只需定义一个MyThread类，让其继承QThread，然后重新实现QThread::run()。

```
#ifndef MYTHREAD_H
#define MYTHREAD_H

#include <QThread>

class MyThread : public QThread
{
    Q_OBJECT
public:
    explicit MyThread(QObject *parent = 0);
    ~MyThread();

protected:
    //QThread的虚函数
    //线程处理函数
    //不能直接调用，通过start()间接调用
    void run();

signals:
    void isDone();  //处理完成信号

signals:

public slots:
};

#endif // MYTHREAD_H

};
```

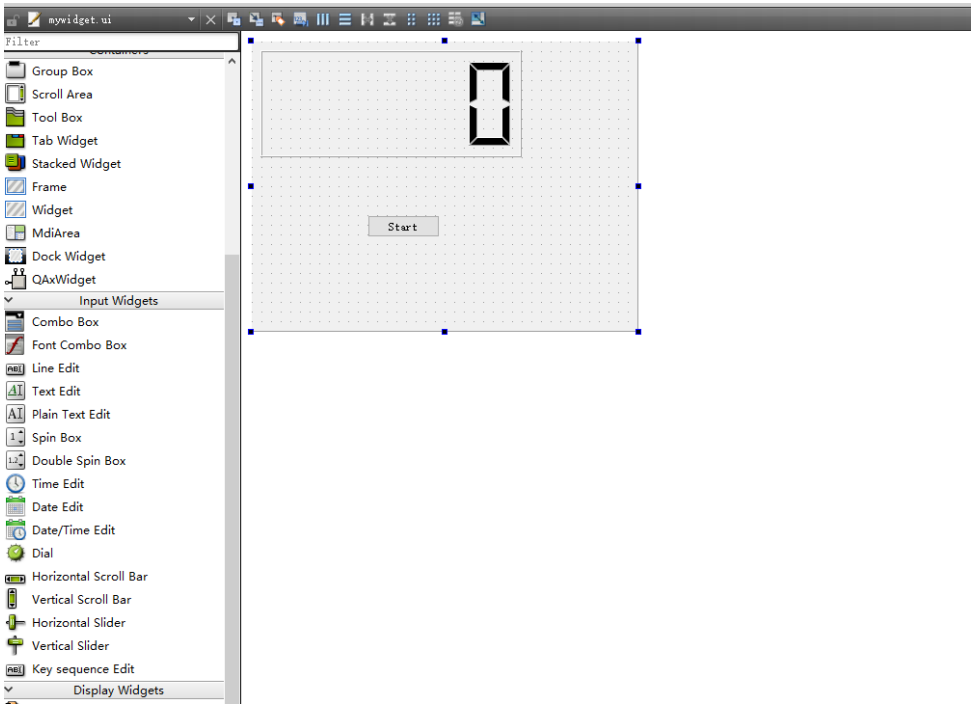
然后可以在run中写入要进行的操作，比如可以让其等待5秒。若不是多线程，在运行时我们单击窗口，窗口会出现无响应的状态。

那如何通知线程结束？这就可以用qt的信号和槽机制了，我们可以在操作完成时发出一个完成信号，完成信号我们在声明文件里已经定义了。

```
void MyThread::run()
{
    //很复杂的数据处理
    //需要耗时5秒
    sleep(5);

    emit isDone();  //发送完成信号
}
```

这样，我们就把线程的操作写完了。

现在，我们先来布一个简单的ui，只用到了一个LcdNumber和一个PushButton。



在当前widget的头文件中定义一些需要用到的操作。并加入我们定义的线程文件。

```
#ifndef MYWIDGET_H
#define MYWIDGET_H

#include <QWidget>
#include <QTimer>
#include "mythread.h"  //线程头文件


namespace Ui {
class MyWidget;
}

class MyWidget : public QWidget
{
    Q_OBJECT

public:
    explicit MyWidget(QWidget *parent = 0);
    ~MyWidget();

    void dealTimeout();  //定时器槽函数
    void dealDone();    //线程槽函数
    void stopThread();  //停止线程

private slots:
    void on_pushButton_clicked();

private:
    Ui::MyWidget *ui;

    QTimer *myTimer;  //声明变量
    MyThread *thread;  //线程对象
};

#endif // MYWIDGET_H
```

我们用一个timer定时器来让Lcd控件按指定时间更新数字，当我们点击开始按钮时，定时器启动，自动触发timerout信号。捕获timerout信号，在dealTimerout()函数中写入需要进行的操作，当捕获到timerout时，自动使用dealTimerout槽函数。

dealTimerout()我们可以这样写：

```
void MyWidget::dealTimeout()
{
    static int i = 0;
    i++;
    //设置lcd的值
    ui->lcdNumber->display(i);
}
```

4. 大整数加法计算(1471)
5. 大整数乘法运算(1079)

评论排行榜

1. 右值引用与移动语义(1)

在widget的构造函数中，先创建一个定时器并为线程函数分配空间：

```
myTimer = new QTimer(this);

thread = new QThread(thread);  //分配空间
```

然后我们关联信号和槽：

```
//只要定时器启动，自动触发timeout
connect(myTimer, &QTimer::timeout, this, &MyWidget::dealTimeout);
```

在开始按钮的槽函数中，启动定时器，并开启线程，需要注意的是，我们不能直接调用run函数，要通过start()间接调用线程函数。

```
void MyWidget::on_pushButton_clicked()
{
    //若定时器没有工作
    if(myTimer->isActive() == false)
    {
        myTimer->start(100);
    }

    //启动线程，处理数据
    thread->start();
}
```

线程结束时我们接收到isDone信号，我们在其中关闭定时器：

```
void MyWidget::dealDone()
{
    qDebug() << "it is over";  //打印线程结束信息
    myTimer->stop();  //关闭定时器
}
```

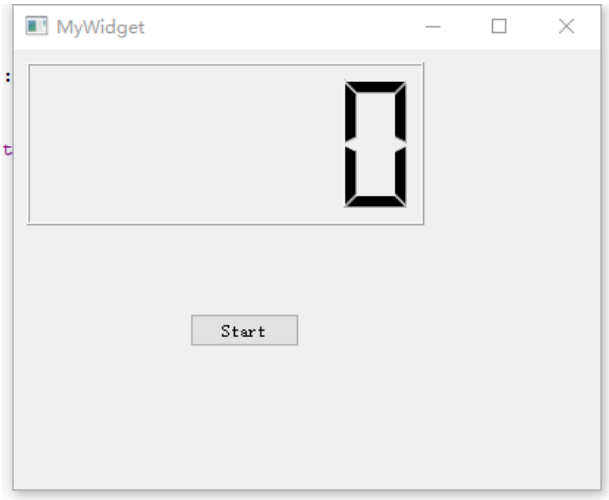
我们选择在退出窗口时关闭线程，退出窗口时会触发destroyed信号，线程关闭的槽函数实现如下：

```
void MyWidget::stopThread()
{
    //停止线程
    thread->quit();

    //等待线程处理完手头工作
    thread->wait();
}
```

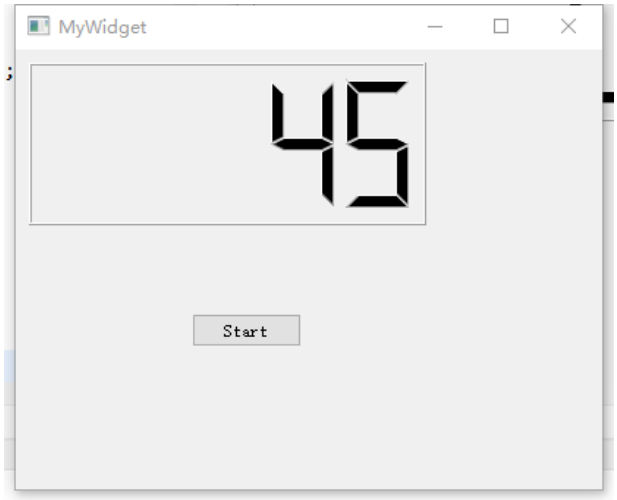
最后，在widget主窗口的构造函数中加入线程的关联信号和槽：

```
connect(thread, &MyThread::isDone, this, &MyWidget::dealDone);

//当按窗口右上角x时，触发destroyed信号
connect(this, &MyWidget::destroyed, this, &MyWidget::stopThread);
```

现在，当我们启动程序时，窗口如下：



当我们点击Start按钮时，触发定时器，开启线程，每100ms更新一次Lcd中的数值：



5秒后，线程停止，发出isDone信号，执行dealDone槽函数，显示it is over并关闭计时器：

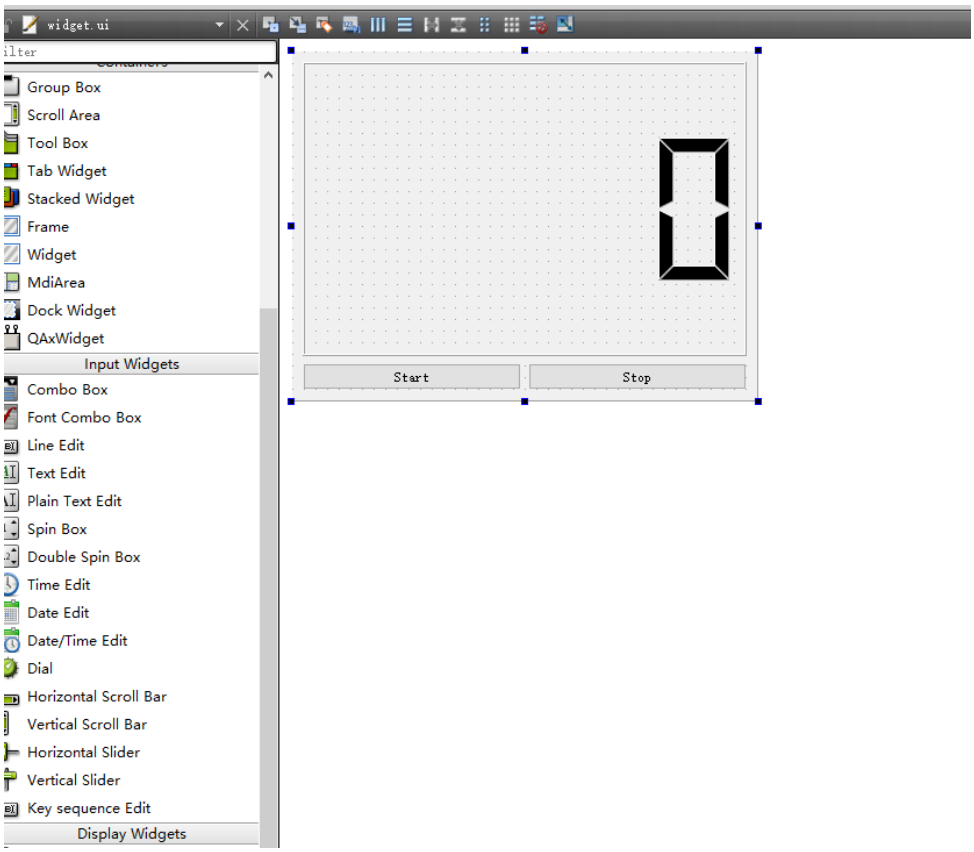
```
Qt: Untested Windows version 10.0 detected!
it is over
```

再次点击，再次启动定时器，继续累加数字并设置到Lcd中，点击x，程序退出，停止线程。

---

2. 第二种方式

新建一个工程，ui如图：



新建一个类，继承自QObject，然后在类中设置一个线程函数。

```
#ifndef MYTHREAD_H
#define MYTHREAD_H

#include <QObject>

class MyThread : public QObject
{
    Q_OBJECT
public:
    explicit MyThread(QObject *parent = 0);
    ~MyThread();
```

```
    //线程处理函数
    void MyTimeout();

    void setFlag(bool flag = true);


signals:
    void mySignal();

public slots:

private:
    bool isStop;
};

#endif // MYTHREAD_H
```

通过一个bool变量来控制线程结束，通过发出mySignal()信号来调用处理槽函数。

在widget中定义如下：

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include "mythread.h"
#include <QThread>

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

    void dealSignal();
    void dealClose();

signals:
    void startThread();  //启动子线程的信号

private slots:
    void on_pushButtonStart_clicked();

    void on_pushButtonStop_clicked();

private:
    Ui::Widget *ui;

    MyThread *myT;
    QThread *thread;
};

#endif // WIDGET_H
```

两个槽函数dealSignal()和dealClose()分别关联着mySignal()和destroyed()信号。

信号startThread()用于启动子线程。

在widget的实现中，

- 首先创建一个线程对象，需要注意不能指定父对象。
  myT = new MyThread;
- 创建一个Qthread子线程对象
  QThread *thread = new QThread(this);
- 把我们的自定义线程类，加入到子线程（若是myT指定了父对象，此处就会出错。）
  my->moveToThread(thread);
- 启动子线程，只是把线程开启了，并没有启动线程处理函数
  thread.start();
- 线程的启动，必须通过signal - slot的方式。

各种函数实现如下：

```
void Widget::dealSignal()
{
    static int i = 0;
    i++;
    ui->lcdNumber->display(i);
}

//Start按钮
void Widget::on_pushButtonStart_clicked()
{
    if(thread->isRunning() == true)
    {
        return;
    }

    //启动线程，但是没有启动线程处理函数
    thread->start();
    myT->setFlag(false);

    //不能直接调用线程处理函数，直接调用导致线程处理函数和主线程在同一个线程
    //myT->MyTimeout();

    //只能通过 signal - slot方式
    emit startThread();
}

//Stop按钮
void Widget::on_pushButtonStop_clicked()
{
    if(thread->isRunning() == false)
    {
        return;
    }

    myT->setFlag(true);
    thread->quit();
    thread->wait();
}

void Widget::dealClose()
{
    on_pushButtonStop_clicked();
    delete myT;
}
```

dealSignal中，使Lcd数字递增。

当按下Start按钮，启动线程，设置线程标志为flase,通过发出startThread()信号来调用真正的线程函数MyThread::MyTimeout。

当按下Stop按钮时，设置线程标志为true，关闭线程。

由于未给myT指定父对象，所以需要我们手动来释放内存，当点击x时，关闭线程，delete释放。

运行如图：

```
void Widget::on_pushButtonStart_clicked()
{
    if(thread->isRunning() == true)
    {
        return;
    }

    //启动线程，但是没有启动线程处理函数
    thread->start();
    myT->setFlag(false);

    //不能直接调用线程处理函数，直接调用导致线程处理函数和主线程在同一个线程
    //myT->MyTimeout();

    //只能通过 signal - slot方式
    emit startThread();
}

//Stop按钮
void Widget::on_pushButtonStop_clicked()
{
    if(thread->isRunning() == false)
    {
        return;
    }

    myT->setFlag(true);
    thread->quit();
    thread->wait();
}

void Widget::dealClose()
{
    on_pushButtonStop_clicked();
    delete myT;
}
```


```
应用程序输出
CarManager  QThread  ThreadPro
Starting F:\Project\QT\day5\build-ThreadPro-Desktop_Qt_5_4_1_MinGW_32bit-Debug\debug\ThreadPro.exe...
Qt: Untested Windows version 10.0 detected!
主线程号:  QThread(0x14636e20)
子线程号:  QThread(0x16199358)
子线程号:  QThread(0x16199358)
子线程号:  QThread(0x16199358)
子线程号:  QThread(0x16199358)
子线程号:  QThread(0x16199358)
子线程号:  QThread(0x16199358)
子线程号:  QThread(0x16199358)
子线程号:  QThread(0x16199358)
子线程号:  QThread(0x16199358)
```

分类: QT

标签: 多线程

好文要顶 关注我 收藏该文

觅影
关注 - 0
粉丝 - 0

0 0

+加关注

« 上一篇: 指针与引用的区别
» 下一篇: QT之SQL的使用

posted @ 2018-07-22 12:46 觅影 阅读(13460) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册， 访问 网站首页。

相关博文:
· 在Qt（C++）中使用QThread实现多线程
· Qt多线程（二）
· PyQt5信号、定时器及多线程

· 界面编程之QT的线程20180731
· Qt中的多线程编程(转自IBM中国)
» 更多推荐...

最新 IT 新闻:
· 11 月全球 Web 服务器调查报告：nginx 表现最佳
· DeepMind联合创始人加入谷歌 负责AI政策方面工作
· 长鑫存储：收获大量原奇梦达内存专利
· 高通做笔电芯片：主打长续航和低价格 挑战英特尔
· 高通谷歌联手：安卓手机最快明年可取代身份证和驾照
» 更多新闻...