



原创 Qt #多线程

Qt中多线程的使用

发表于 2021-03-28 更新于 2024-11-22

📖 字数总计: 2.6k ⌚ 阅读时长: 9分钟 👁 阅读量: 621131 📍 河北

在进行桌面应用程序开发的时候，假设应用程序在某些情况下需要处理比较复杂的逻辑，如果只有一个线程去处理，就会导致窗口卡顿，无法处理用户的相关操作。这种情况下就需要使用多线程，其中一个线程处理窗口事件，其他线程进行逻辑运算，多个线程各司其职，不仅可以提高用户体验还可以提升程序的执行效率。

在qt中使用了多线程，有些事项是需要额外注意的：

- 默认的线程在qt中称之为窗口线程，也叫主线程，负责窗口事件处理或者窗口控件数据的更新
- 子线程负责后台的业务逻辑处理，子线程中不能对窗口对象做任何操作，这些事情需要交给窗口线程处理
- 主线程和子线程之间如果要进行数据的传递，需要使用Qt中的信号槽机制

1. 线程类 QThread

Qt中提供了一个线程类，通过这个类就可以创建子线程了，Qt中一共提供了两种创建子线程的方式，后边会依次介绍其使用方式。先来看一下这个类中提供一些常用API函数：

1.1 常用共用成员函数

```
1 // QThread 类常用 API
2 // 构造函数
3 QThread::QThread(QObject *parent = Q_NULLPTR);
4 // 判断线程中的任务是不是处理完毕了
5 bool QThread::isFinished() const;
6 // 判断子线程是不是在执行任务
7 bool QThread::isRunning() const;
8
9 // Qt中的线程可以设置优先级
10 // 得到当前线程的优先级
11 Priority QThread::priority() const;
12 void QThread::setPriority(Priority priority);
13 优先级:
14     QThread::IdlePriority      --> 最低的优先级
15     QThread::LowestPriority
16     QThread::LowPriority
17     QThread::NormalPriority
18     QThread::HighPriority
19     QThread::HighestPriority
20     QThread::TimeCriticalPriority --> 最高的优先级
21     QThread::InheritPriority   --> 子线程和其父线程的优先级相同，默认是这个
22 // 退出线程，停止底层的事件循环
23 // 退出线程的工作函数
24 void QThread::exit(int returnCode = 0);
25 // 调用线程退出函数之后，线程不会马上退出因为当前任务有可能还没有完成，调回用这个函数
26 // 等待任务完成，然后退出线程，一般情况下会在 exit() 后边调用这个函数
27 bool QThread::wait(unsigned long time = ULONG_MAX);
```



C++

```
1 // 和调用 exit() 效果是一样的
2 // 调用这个函数之后，再调用 wait() 函数
3 [slot] void QThread::quit();
4 // 启动子线程
5 [slot] void QThread::start(Priority priority = InheritPriority);
6 // 线程退出，可能是会马上终止线程，一般情况下不使用这个函数
7 [slot] void QThread::terminate();
8
9 // 线程中执行的任务完成了，发出该信号
10 // 任务函数中的处理逻辑执行完毕了
11 [signal] void QThread::finished();
12 // 开始工作之前发出这个信号，一般不使用
13 [signal] void QThread::started();
```

1.3 静态函数



C++

```
1 // 返回一个指向管理当前执行线程的QThread的指针
2 [static] QThread *QThread::currentThread();
3 // 返回可以在系统上运行的理想线程数 == 和当前电脑的 CPU 核心数相同
4 [static] int QThread::idealThreadCount();
5 // 线程休眠函数
6 [static] void QThread::msleep(unsigned long msecs); // 单位：毫秒
7 [static] void QThread::sleep(unsigned long secs); // 单位：秒
8 [static] void QThread::usleep(unsigned long usecs); // 单位：微秒
```

1.4 任务处理函数



C++

```
1 // 子线程要处理什么任务，需要写到 run() 中
2 [virtual protected] void QThread::run();
```

这个 `run()` 是一个虚函数，如果想让创建的子线程执行某个任务，需要写一个子类让其继承 `QThread`，并且在子类中重写父类的 `run()` 方法，函数体就是对应的任务处理流程。另外，这个函数是一个受保护的成员函数，不能够在类的外部调用，如果想要让线程执行这个函数中的业务流程，需要通过当前线程对象调用槽函数 `start()` 启动子线程，当子线程被启动，这个 `run()` 函数也在线程内部被调用了。

2. 使用方式1



2.1 操作步骤



Qt中提供的多线程的第一种使用方式的特点是：简单。操作步骤如下：

- 1 需要创建一个线程类的子类，让其继承Qt中的线程类 `QThread`，比如：

C++

```
1 class MyThread:public QThread
2 {
3     .....
4 }
```

- 2 重写父类的 `run()` 方法，在该函数内部编写子线程要处理的具体的业务流程

C++

```
1 class MyThread:public QThread
2 {
3     .....
4 protected:
5     void run()
6     {
7         .....
8     }
```

- 3 在主线程中创建子线程对象，new 一个就可以了

C++

```
1 MyThread * subThread = new MyThread;
```

- 4 启动子线程, 调用 start() 方法

C++

```
1 subThread->start();
```

不能在类的外部调用run() 方法启动子线程，在外部调用start()相当于让run()开始运行

当子线程别创建出来之后，父子线程之间的通信可以通过信号槽的方式，注意事项:

- 在Qt中在子线程中不要操作程序中的窗口类型对象, 不允许, 如果操作了程序就挂了
- 只有主线程才能操作程序中的窗口对象, 默认的线程就是主线程, 自己创建的就是子线程

2.2 示例代码

举一个简单的数数的例子，假如只有一个线程，让其一直数数，会发现数字并不会在窗口中时时更新，并且这时候如果用户使用鼠标拖动窗口，就会出现无响应的情况，使用多线程就不会出现这样的现象了。

在下面的窗口中，点击按钮开始在子线程中数数，让后通过信号槽机制将数据传递给UI线程，通过UI线程将数据更新到窗口中。



mythread.h

C++

```
1 #ifndef MYTHREAD_H
2 #define MYTHREAD_H
3
4 #include <QThread>
5
6 class MyThread : public QThread
7 {
8     Q_OBJECT
9 public:
10     explicit MyThread(QObject *parent = nullptr);
11
12 protected:
13     void run();
14
15 signals:
16     // 自定义信号，传递数据
17     void curNumber(int num);
18
19 public slots:
20 };
21
22 #endif // MYTHREAD_H
```

mythread.cpp

```
1 #include "mythread.h"
2 #include <QDebug>
3
4 MyThread::MyThread(QObject *parent) : QThread(parent)
5 {
6
7 }
8
9 void MyThread::run()
10 {
11     qDebug() << "当前线程对象的地址: " << QThread::currentThread();
12
13     int num = 0;
14     while(1)
15     {
16         emit curNumber(num++);
17         if(num == 10000000)
18         {
19             break;
20         }
21         QThread::usleep(1);
22     }
23     qDebug() << "run() 执行完毕, 子线程退出...";
24 }
```

mainwindow.cpp

```
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3 #include "mythread.h"
4 #include <QDebug>
5
6 MainWindow::MainWindow(QWidget *parent) :
7     QMainWindow(parent),
8     ui(new Ui::MainWindow)
9 {
10     ui->setupUi(this);
11
12     qDebug() << "主线程对象地址: " << QThread::currentThread();
13     // 创建子线程
14     MyThread* subThread = new MyThread;
15
16     connect(subThread, &MyThread::curNumber, this, [=](int num)
17     {
18         ui->label->setNum(num);
19     });
20
21     connect(ui->startBtn, &QPushButton::clicked, this, [=]()
22     {
23         // 启动子线程
24         subThread->start();
25     });
26 }
27
28 MainWindow::~MainWindow()
29 {
30     delete ui;
31 }
```

这种在程序中添加子线程的方式是非常简单的, 但是也有弊端, 假设要在一个子线程中处理多个任务, 所有的处理逻辑都需要写到run()函数中, 这样该函数中的处理逻辑就会变得非常混乱, 不太容易维护。

3. 使用方式2

3.1 操作步骤

Qt提供的第二种线程的创建方式弥补了第一种方式的缺点，用起来更加灵活，但是这种方式写起来会相对复杂一些，其具体操作步骤如下：

- 1 创建一个新的类，让这个类从QObject派生

```
1 class MyWork:public QObject
2 {
3     .....
4 }
```

- 2 在这个类中添加一个公共的成员函数，函数体就是我们要子线程中执行的业务逻辑

```
1 class MyWork:public QObject
2 {
3 public:
4     .....
5     // 函数名自己指定，叫什么都可以，参数可以根据实际需求添加
6     void working();
7 }
```

- 3 在主线程中创建一个QThread对象, 这就是子线程的对象

```
1 QThread* sub = new QThread;
```

- 4 在主线程中创建工作的类对象（千万不要指定给创建的对象指定父对象）

```
1 MyWork* work = new MyWork(this); // error
2 MyWork* work = new MyWork;       // ok
```

- 5 将MyWork对象移动到创建的子线程对象中, 需要调用QObject类提供的 `moveToThread()` 方法

```
1 // void QObject::moveToThread(QThread *targetThread);
2 // 如果给work指定了父对象，这个函数调用就失败了
3 // 提示：QObject::moveToThread: Cannot move objects with a parent
4 work->moveToThread(sub); // 移动到子线程中工作
```

- 6 启动子线程，调用 `start()`，这时候线程启动了，但是移动到线程中的对象并没有工作
- 7 调用MyWork类对象的工作函数，让这个函数开始执行，这时候是在移动到的那个子线程中运行的

3.2 示例代码



假设函数处理上面在程序中数数的这个需求，具体的处理代码如下：

mywork.h

```
1 #ifndef MYWORK_H
2 #define MYWORK_H
3
4 #include <QObject>
5
6 class MyWork : public QObject
7 {
8     Q_OBJECT
9 public:
10     explicit MyWork(QObject *parent = nullptr);
11
12     // 工作函数
13     void working();
```

```
14
15 signals:
16     void curNumber(int num);
17
18 public slots:
19 };
20
21 #endif // MYWORK_H
```

mywork.cpp

```
● ● C++
1 #include "mywork.h"
2 #include <QDebug>
3 #include <QThread>
4
5 MyWork::MyWork(QObject *parent) : QObject(parent)
6 {
7
8 }
9
10 void MyWork::working()
11 {
12     qDebug() << "当前线程对象的地址: " << QThread::currentThread();
13
14     int num = 0;
15     while(1)
16     {
17         emit curNumber(num++);
18         if(num == 10000000)
19         {
20             break;
21         }
22         QThread::usleep(1);
23     }
24     qDebug() << "run() 执行完毕, 子线程退出...";
25 }
```

mainwindow.cpp

```
● ● C++
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3 #include <QThread>
4 #include "mywork.h"
5 #include <QDebug>
6
7 MainWindow::MainWindow(QWidget *parent) :
8     QMainWindow(parent),
9     ui(new Ui::MainWindow)
10 {
11     ui->setupUi(this);
12
13     qDebug() << "主线程对象的地址: " << QThread::currentThread();
14
15     // 创建线程对象
16     QThread* sub = new QThread;
17     // 创建工作的类对象
18     // 千万不要指定给创建的对象指定父对象
19     // 如果指定了: QObject::moveToThread: Cannot move objects with a parent
20     MyWork* work = new MyWork;
21     // 将工作的类对象移动到创建的子线程对象中
22     work->moveToThread(sub);
23     // 启动线程
24     sub->start();
25     // 让工作的对象开始工作, 点击开始按钮, 开始工作
26     connect(ui->startBtn, &QPushButton::clicked, work, &MyWork::working);
27     // 显示数据
28     connect(work, &MyWork::curNumber, this, [=](int num)
29     {
```

```
31     });
32 }
33
34 MainWindow::~MainWindow()
35 {
```

使用这种多线程方式，假设有多个不相关的业务流程需要被处理，那么就可以创建多个类似于 `MyWork` 的类，将业务流程放多类的公共成员函数中，然后将这个业务类的实例对象移动到对应的子线程中 `moveToThread()` 就可以了，这样可以让编写的程序更加灵活，可读性更强，更易于维护。



苏丙楹

知识分享

原创 Qt中多线程的使用

打赏作者



本博客所有文章除特别声明外，均采用 CC BY-NC-SA 4.0 许可协议。转载请注明来自 爱编程的大丙！

多线程 14



喜欢这篇文章的人也看了



评论

匿名评论 隐私政策

0/500

昵称 必填

邮箱 必填

网址 选填

发送

1 条评论



热心网友 2024-11-14

2

有错别字“代用这个函数之后, 再调用 wait() 函数”, “代用这个函数”->“但用这个函数”

安徽 Windows 10 Chrome 118.0.0.0