<https://blog.csdn.net/qinpanke/article/details/51705283>

**第一部分：QT线程池的构建与使用**

    网上关于QT线程池QThreadPool的文章很多，而且大都千篇一律，基本上都是参考QT的帮助文档介绍QT全局线程池的用法。这样就往往会使人产生误解，QT是不是推荐大家使用其全局线程池，而不推荐使用自定义构造的线程池？ 实际情况并不是这样的。而且在实际的项目当中我们通常并不希望仅仅使用一个全局的线程池，而是在需要线程池的工程中都构建和维护自己一个小小的线程池（我们知道一个良好架构的项目通常是由多个工程组成的）。综上，我们来分析以下两个问题：

**（1）    非全局的线程池如果构建与使用呢？**

1. #include <QObject>
2. #include <QRunnable>
3. #include <QThread>
4. #include <QThreadPool>
5. #include <QDebug>
6. class HelloWorldTask : public QRunnable
7. {
8. *// 线程执行任务：每间隔1s打印出线程的信息*
9. void run()
10. {
11. for (int nCount = 0; nCount < 5; nCount++)
12. {
13. qDebug() << QThread::currentThread();
14. QThread::msleep(1000);
15. }
16. }
17. };
18. int main(int argc, char \*argv[])
19. {
20. QCoreApplication a(argc, argv);
22. QThreadPool threadpool; *// 构建一个本地的线程池*
23. threadpool.setMaxThreadCount(3); *// 线程池中最大的线程数*
24. for (int nNum = 0; nNum < 100; nNum++)
25. {
26. HelloWorldTask \*task; *// 循环构建可在线程池中运行的任务*
27. threadpool.start(task); *//线程池分配一个线程运行该任务*
28. QThread::msleep(1000);
29. }
30. return a.exec();
31. }

    上述程序，构建了一个线程最大数量为3的本地线程池。每间隔1s的时间创建一个线程任务并置入到线程池的任务队列中（QT内部机制实现该队列，我们只需要调用QThreadPool的start函数置入即可）。每个线程任务的持续时间为5s。

**（2）    程序当中QRunnable是以指针的形式创建的，该指针是需要程序员去释放，还是QThreadPool在运行完线程后自动释放？**

    解答：在上述例子当中，我们创建的QRunnable类型的指针 QRunnable \*task 是不需要我们手动去回收内存的，QThreadPool在结束该任务的执行后会将对该内存进行清空。

    上述解答并不是凭空猜测，一方面根据是QT文档中的一句话：

**QThreadPool takes ownership and deletes 'hello'automatically**

     用直白的话说就是：QThreadPool会占有这个指针的句柄并在运行结束后释放指针所占的内存。

    另一方面，我们也通过改进上面的例子进行验证。

1. #include <QObject>
2. #include <QRunnable>
3. #include <QThread>
4. #include <QThreadPool>
5. #include <QDebug>
6. class HelloWorldTask : public QRunnable
7. {
8. *// 线程执行任务：每间隔1s打印出线程的信息*
9. void run()
10. {
11. int m\_dataMem[256\*1000]; *// 占约 1MB内存空间*
12. for (int nCount = 0; nCount < 5; nCount++)
13. {
14. qDebug() << QThread::currentThread();
15. QThread::msleep(1000);
16. }
17. }
18. };
19. int main(int argc, char \*argv[])
20. {
21. QCoreApplication a(argc, argv);
23. QThreadPool threadpool;
24. threadpool.setMaxThreadCount(1);
25. for (int nNum = 0; nNum < 100; nNum++)
26. {
27. HelloWorldTask \*task;
28. threadpool.start(task);
29. QThread::msleep(1000);
30. }
31. return a.exec();
32. }

     在程序运行过程中，我们观察发现程序的进程一直仅占用约1MB的内存空间。如果在main函数中所创建的100个HelloWorldTask 指针对象没有被QThreadPool释放的话，随着程序的运行该程序所占内存空间应该逐步攀升到约100MB。然而实际情况是，该程序最高仅占用1MB的内存空间。

    综上两个方面可以得出以下结论：QRunnable创建的对象QThreadPool在执行完该对象后会帮助我们来清空内存，不需要我们手动回收内存。

**第二部分：QThread 与QRunnable + QThreadPool适用的应用场景**

    QThread是QT的线程类，通过继承QThread然后重写run函数即可实现一个线程类。QThreadPool+ QRunnable配合构建线程池的方法也可以实现线程。我们通过以下问题对上述两种构建线程的方法进行分析和说明。

**（1）既然QThread这么简单我们为什么还要使用QThreadPool + QRunnable创建线程池的方法来使用线程机制呢？**

主要原因：当线程任务量非常大的时候，如果频繁的创建和释放QThread会带来比较大的内存开销，而线程池则可以有效避免该问题，相关的基础支持可以自行百度线程池的优点。

**（2）QThread与 QThreadPool + QRunnable分别适用于哪种应用场景？**

     QThread适用于那些常驻内存的任务。而且QThread可以通过信号/槽的方式与外界进行通信。而QRunnable则适用于那些不常驻内存，任务数量比较多的情况。

**第三部分：QRunnable 如何与外界进行通信**

       方法1：QRunnable并不继承自QObject类，因此无法使用信号/槽的方式与外界进行通信。我们就必须的使用其他方法，这里给大家介绍的是使用：QMetaObject::invokeMethod()函数。

       方法2：使用多重继承的方法，任务子类同时继承自QRunnable和QObject。

      欢迎交流讨论

      QQ 33845995

      Email : qinpanke@hpu.edu.cn

阿科

2016年6月18日 于森林半岛