<https://blog.csdn.net/y396397735/article/details/78637634>

目录

* [QThreadPool类](https://blog.csdn.net/y396397735/article/details/78637634#qthreadpool%E7%B1%BB)
  + [主要属性](https://blog.csdn.net/y396397735/article/details/78637634#%E4%B8%BB%E8%A6%81%E5%B1%9E%E6%80%A7)
  + [主要成员函数](https://blog.csdn.net/y396397735/article/details/78637634#%E4%B8%BB%E8%A6%81%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0)
* [QRunnable类](https://blog.csdn.net/y396397735/article/details/78637634#qrunnable%E7%B1%BB)
  + [主要成员函数](https://blog.csdn.net/y396397735/article/details/78637634#%E4%B8%BB%E8%A6%81%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0-1)
* [程序实例](https://blog.csdn.net/y396397735/article/details/78637634#%E7%A8%8B%E5%BA%8F%E5%AE%9E%E4%BE%8B)

# QThreadPool类

用来管理 QThreads。此类中的所有函数都是线程安全的.

## 主要属性：

1、activeThreadCount: 此属性表示线程池中的活动线程数，通过activeThreadCount() 调用。   
2、expiryTimeout: 线程活着的时间。没有设置expiryTimeout毫秒的线程会自动退出，此类线程将根据需要重新启动。默认的expiryTimeout为30000毫秒 (30 秒)。如果expiryTimeout为负, 则新创建的线程将不会过期, 在线程池被销毁之前, 它们将不会退出。通过expiryTimeout()调用，通setExpiryTimeout(int expiryTimeout)设置 。   
3、maxThreadCount : int 表示线程池使用的最大线程数。   
通过maxThreadCount() 调用，通过setMaxThreadCount(int maxThreadCount) 设置   
**注意**:即使maxThreadCount限制为零或为负数, 线程池也至少有1个线程。

## 主要成员函数

**QThreadPool \*QThreadPool::globalInstance()**

返回Qt应用程序全局线程池实例。

**void reserveThread()**

预约一个线程，这个函数总是会增加活动线程的数量。这意味着通过使用这个函数，activeThreadCount()可以返回一个大于maxThreadCount()的值。

**void releaseThread()**

释放以前通过调用reserveThread()预约的线程。   
如果不先预约一个线程，调用这个函数会临时增加maxThreadCount()。当线程进入休眠等待时，能够允许其他线程继续。   
要记得在完成等待时调用reserveThread()，以便线程池可以正确控制activeThreadCount()。

**void QThreadPool :: start（QRunnable \* runnable，int priority = 0）**

在任务数量小于maxThreadCount时，为每个runnable任务预约一个线程。超过maxThreadCount时，将任务放入运行队列中。priority 参数用来设置线程运行优先级。

**bool tryStart(QRunnable \*runnable)**

此方法尝试预约一个线程来运行runnable。   
如果在调用的时候没有线程可用，那么这个函数什么都不做，并返回false。否则，将使用一个可用线程立即运行runnable，并返回此函数true。

**void clear()**

用于删除在任务队列中，还没有启动的任务。

**bool tryTake(QRunnable \*runnable)**

如果runnable任务还没开始运行，那么从队列中删除此runable任务，此时函数返回true；如果runnable任务已经运行，返回false。   
只用来删除runnable->autoDelete() == false的runnable任务，否则可能会删错任务.

**bool waitForDone(int msecs = -1)**

等待msecs毫秒, 以便所有线程退出并从线程池中移除所有线程。如果删除了所有线程, 则返回true ，否则, 它将返回false。默认等待时间为-1，即等待最后一个线程退出。

# QRunnable类

QRunnable类是所有runable对象的基类。   
QRunnable类是一个接口, 用于表示需要执行的任务或代码段, 具体任务在run() 函数内部实现。   
可以使用QThreadPool在各个独立的线程中执行代码。如果autoDelete() 返回true (默认值), QThreadPool将自动删除QRunnable 。使用setAutoDelete() 可更改是否自动删除。

## 主要成员函数

**bool autoDelete() const**

获取自动删除是否启用，启用返回true，未启用返回false。

**virtual void run() = 0**

纯虚函数，在QRunnable子类中实现详细任务处理逻辑。

**void setAutoDelete(bool autoDelete)**

如果autoDelete为 true, 则启用自动删除。否则自动删除将被禁用。   
如果启用了自动删除, QThreadPool将在调用 run () 函数返回后自动删除此runable对象。否则, runable对象所有权不属于线程池，由开发人员管理。   
请注意, 必须先设置此标志，(默认构造函数已经将其设置为true)，然后才能调用QThreadPool:: start()。在QThreadPool:: start() 之后调用此函数将导致不可预测后果。

# 程序实例

任务类（runable类）头文件

#ifndef PRINTTASK\_H

#define PRINTTASK\_H

#include <QObject>

#include <QRunnable>

class PrintTask : public QObject, public QRunnable

{

Q\_OBJECT

public:

PrintTask();

~PrintTask();

protected:

void run();

signals:

//注意！要使用信号，采用QObejct 和 QRunnable多继承，记得QObject要放在前面

void mySignal();

};

#endif // PRINTTASK\_H

任务类（runable类）实现文件

#include "printtask.h"

#include <QThread>

#include <iostream>

using std::cout;

using std::endl;

PrintTask::PrintTask()

{

}

PrintTask::~PrintTask()

{

}

//线程真正执行的内容

void PrintTask::run()

{

cout << "PrintTask run 被调用，调用线程ID为：" << QThread::currentThread() << endl;

}

主函数文件：

#include <QCoreApplication>

#include <QThreadPool>

#include "printtask.h"

int main(int argc, char \*argv[])

{

QCoreApplication a(argc, argv);

//设置最大线程数为3的一个线程池

QThreadPool pool;

pool.setMaxThreadCount(3);

for(int i = 0; i < 20; i++)

{

pool.start(new PrintTask());

}

return a.exec();

}

输出：

PrintTask run 被调用，调用线程ID为：00533678

PrintTask run 被调用，调用线程ID为：00533678

PrintTask run 被调用，调用线程ID为：004F3868

PrintTask run 被调用，调用线程ID为：004F3848

PrintTask run 被调用，调用线程ID为：00533678

PrintTask run 被调用，调用线程ID为：00533678

PrintTask run 被调用，调用线程ID为：004F3848

PrintTask run 被调用，调用线程ID为：004F3868

PrintTask run 被调用，调用线程ID为：00533678

PrintTask run 被调用，调用线程ID为：00533678

PrintTask run 被调用，调用线程ID为：004F3848

PrintTask run 被调用，调用线程ID为：00533678

PrintTask run 被调用，调用线程ID为：004F3868

PrintTask run 被调用，调用线程ID为：004F3848

PrintTask run 被调用，调用线程ID为：00533678

PrintTask run 被调用，调用线程ID为：004F3868

PrintTask run 被调用，调用线程ID为：004F3848

PrintTask run 被调用，调用线程ID为：004F3868

...

分析打印结果发现：20个任务，只有3个线程（线程ID分别为00533678、004F3848，004F3868）去执行，符合期望。

版权声明：个人学习之路，若有误，欢迎指正。其中一些博文被证明有错误的地方，最近比较忙，没时间更正，谨慎参考！！ https://blog.csdn.net/y396397735/article/details/78637634