Примитивы синхронизации Qt

Анна Субботина

26 Октября 2015

QThread

```
class MyThread : public QThread
{
   Q_OBJECT

protected:
   void run();
};

void MyThread::run()
{
   ...
}


int main()
{
…
   MyThread *mt = new MyThread();
   mt->start();
…
}
```

QThread

```
class MyThread : public QThread
{
   Q_OBJECT
public:
   void stop() { stopped = true; };
   MyThread() : stopped(false){};
protected:
   void run()
   {
      while(!stopped)
          cout << “QThread is working” << endl;
      stopped = false;
   };
private:
   volatile bool stopped;    // доступ из разных потоков
};
int main()
{
   MyThread *mt = new MyThread();
   mt->start();
   mt->stop(); // mt->wait(); mt->terminate();
…
}
```

QMutex

```
class MyThread : public QThread
{
   Q_OBJECT
public:
   void stop() { mutex.lock(); stopped = true; mutex.unlock(); };
   MyThread() : stopped(false){};
protected:
   void run()
   { while(1) {
             mutex.lock();
             if (stopped) { stopped = false; mutex.unlock(); break; }
             mutex.unlock();
             cout << "QThread is working" << endl;
   }
};
private:
   bool stopped; QMutex mutex;
};
int main()
{
   MyThread *mt = new MyThread();
   mt->start(); mt->stop();
…
}
```

QMutexLocker

```
class MyThread : public QThread
{
   Q_OBJECT
public:
   void stop() { QMutexLocker locker(&mutex); stopped = true; };
   MyThread() : stopped(false){};
protected:
   void run() { while(run_func) {};
};
private:
   bool stopped; QMutexLocker mutex;
   bool run_func() {
            QMutexLocker locker(&mutex);
            if (stopped) { stopped = false; return false; }
            cout << “QThread is working” << endl; return true;
   }
};
int main()
{
   MyThread *mt = new MyThread();
   mt->start(); mt->stop();
…
}
```

QWaitCondition

```
class MyThread : public QThread
{
   Q_OBJECT
public:
   void stop() { condition.wakeAll(); };
   MyThread() {};
protected:
   void run()
   {       mutex.lock();
           cout << “QThread is waiting” << endl;
           condition.wait(&mutex);
};
private:
   QWaitCondition condition; QMutex mutex;
};
int main()
{
   MyThread *mt = new MyThread();
   mt->start(); mt->stop();
…
}
```

QThreadPool

```
class MyThread : public QRunnable
{
public:
        MyThread() {};
        virtual ~ MyThread() {};
protected:
        void run() { cout << "QThread is working" << endl; };
};

int main()
{
    MyThread *h1 = new MyThread();
    MyThread *h2 = new MyThread();
//  QThreadPool::globalInstance()->setMaxThreadCount(1);
    QThreadPool::globalInstance()->start(h1);
    QThreadPool::globalInstance()->start(h2);
    QThreadPool::globalInstance()->waitForDone();
    return 0;
}
```

emit signal

```
class MyObject : public QObject
{ Q_OBJECT
public slots:
	void MySlot() { cout << "slot called" << endl; }
};
class Thread1 : public QThread
{ Q_OBJECT
public:
	void run() {
		cout << "thread 1 started" << endl;
		for (int i = 0; i < 5; i++) { sleep(1); emit MySignal(); }
	}
signals:
	void MySignal();
};

class Thread2 : public QThread
{ Q_OBJECT
public:
	void run() { cout << "thread 2 started" << endl; exec(); }
};
```

emit signal

```
int main(int argc, char **argv)
{
	QCoreApplication a(argc, argv);
	Thread1 th1;
	Thread2 th2;
	MyObject ob;
	QObject::connect(&th1, SIGNAL(MySignal()), &ob, SLOT(MySlot()));
	th2.start();
	ob.moveToThread(&th2);
	th1.start();
	th1.wait();
	th2.quit();
	th2.wait();

	return 0;
}
```

ЗАДАНИЕ

- Решить задачу читателей и писателей
- Решить задачу “эстафета”

Вопросы?