第32篇 (网络) 进程和线程

一、进程

在设计一个应用程序时，有时不希望将一个不太相关的功能集成到程序中，或者是因为该功能与当前设计的应用程序联系不大，或者是因为该功能已经可以使用现成的程序很好的实现了，这时就可以在当前的应用程序中调用外部的程序来实现该功能，这就会使用到进程。Qt应用程序可以很容易的启动一个外部应用程序，而且Qt也提供了在多种进程间通信的方法。

Qt的`QProcess`类用来启动一个外部程序并与其进行通信。下面我们来看一下怎么在Qt代码中启动一个进程。

1．首先创建QtGui应用。

工程名称为`myProcess`，其他选项保持默认即可。

2．然后设计界面。

在设计模式往界面上拖入一个`Push Button`部件，修改其显示文本为“启动一个进程”。

3．修改槽。

在按钮上点击鼠标右键，转到其`clicked()`信号对应的槽，更改如下：

```
void MainWindow::on_pushButton_clicked()
{
     myProcess.start("notepad.exe");
}
```

4．进入`mainwindow.h`文件添加代码。

先添加头文件包含：`#include <QProcess>`，然后添加私有对象定义：`QProcess myProcess;`

5．运行程序。

当单击界面上的按钮时就会弹出一个记事本程序。

这里我们使用`QProcess`对象运行了Windows系统下的记事本程序（即`notepad.exe`程序），因为该程序在系统目录中，所以这里不需要指定其路径。大家也可以运行其他任何的程序，只需要指定其具体路径即可。我们看到，可以使用`start()`来启动一个程序，有时启动一个程序时需要指定启动参数，这种情况在命令行启动程序时是很常见的，下面来看一个例子，还在前面的例子的基础上进行更改。

1．在`mainwindow.h`文件中添加代码。

添加私有槽：

```
private slots:
    void showResult();
```

2．在`mainwindow.cpp`文件中添加代码。

（1）先添加头文件包含：`#include <QDebug>`，然后在构造函数中添加如下代码：

```
connect(&myProcess,SIGNAL(readyRead()), this, SLOT(showResult()));
```

（2）然后添加`showResult()`槽的定义：

```
void MainWindow::showResult()
{
    qDebug() << "showResult: " << endl
             << QString(myProcess.readAll());
}
```

（3）最后将前面按钮的单击信号对应的槽更改为：

```
void MainWindow::on_pushButton_clicked()
{
    QString program = "cmd.exe";
    QStringList arguments;
    arguments << "/c dir&pause";
    myProcess.start(program, arguments);
}
```

这里在启动Windows下的命令行提示符程序`cmd.exe`时为其提供了命令作为参数，这样可以直接执行该命令。当命令执行完以后可以执行`showResult()`槽来显示运行的结果。这里为了可以显示结果中的中文字符，使用了`QString()`进行编码转换。这需要在`mian()`函数中添加代码。

3．为了确保可以显示输出的中文字符，在`main.cpp`文件中添加代码。 先添加头文件包含`#include <QTextCodec>`，然后在`main()`函数第一行代码下面，添加如下一行代码：

```
QTextCodec::setCodecForCStrings(QTextCodec::codecForLocale());
```

4．运行程序。

按下界面上的按钮，会在Qt Creator中的应用程序输出栏中输出命令的执行结果。

对于Qt中进程进一步的使用可以参考`QProcess`类的帮助文档。在Qt中还提供了多种进程间通信的方法，大家可以在Qt帮助中查看Inter-ProcessCommunication in Qt关键字对应的文档。

二、线程

Qt提供了对线程的支持，这包括一组与平台无关的线程类，一个线程安全的发送事件的方式，以及跨线程的信号-槽的关联。这些使得可以很容易的开发可移植的多线程Qt应用程序，可以充分利用多处理器的机器。多线程编程也可以有效的解决在不冻结一个应用程序的用户界面的情况下执行一个耗时的操作的问题。关于线程的内容，大家可以在Qt帮助中参考Thread Support in Qt关键字。

（一）启动一个线程

Qt中的`QThread`类提供了平台无关的线程。一个`QThread`代表了一个在应用程序中可以独立控制的线程，它与进程中的其他线程分享数据，但是是独立执行的。相对于一般的程序都是从`main()`函数开始执行，`QThread`从`run()`函数开始执行。默认的，`run()`通过调用`exec()`来开启事件循环。要创建一个线程，需要子类化`QThread`并且重新实现`run()`函数。

每一个线程可以有自己的事件循环，可以通过调用`exec()`函数来启动事件循环，可以通过调用`exit()`或者`quit()`来停止事件循环。在一个线程中拥有一个事件循环，可以使它能够关联其他线程中的信号到本线程的槽上，这使用了队列关联机制，就是在使用`connect()`函数进行信号和槽的关联时，将`Qt::ConnectionType`类型的参数指定为

`Qt::QueuedConnection`。拥有事件循环还可以使该线程能过使用需要事件循环的类，比如`QTimer`和`QTcpSocket`类等。注意，在线程中是无法使用任何的部件类的。

下面来看一个在图形界面程序中启动一个线程的例子，在界面上有两个按钮，一个用于开启一个线程，一个用于关闭该线程。

1．创建项目。

新建Qt Gui应用，名称为`myThread`，类名为`Dialog`，基类选择`QDialog`。

2．设计界面。

完成项目创建后进入设计模式，向界面中放入两个`Push Button`按钮，将第一个按钮的显示文本更改为“启动线程”，将其`objectName`属性更改为`startButton`；将第二个按钮的显示文本更改为“终止线程”，将其`objectName`属性更改为`stopButton`，将其`enabled`属性取消选中。

3．添加自定义线程类。

向项目中添加新的C++类，类名设置为`MyThread`，基类设置为`QThread`，类型信息选择“继承自QObject”。完成后进入`mythread.h`文件，先添加一个公有函数声明：

```
void stop();
```

然后再添加一个函数声明和一个变量的定义：

```
protected:
    void run();
private:
    volatile bool stopped;
```

这里`stopped`变量使用了`volatile`关键字，这样可以使它在任何时候都保持最新的值，从而可以避免在多个线程中访问它时出错。然后进入`mythread.cpp`文件中，先添加头文件`#include <QDebug>`，然后在构造函数中添加如下代码：

```
stopped = false;
```

这里将`stopped`变量初始化为`false`。下面添加`run()`函数的定义：

```
void MyThread::run()
{
    qreal i = 0;
    while (!stopped)
        qDebug() << QString("in MyThread: %1").arg(i++);
    stopped = false;
}
```

这里一直判断`stopped`变量的值，只要它为`false`，那么就一直打印字符串。下面添加`stop()`函数的定义：

```
void MyThread::stop()
{
    stopped = true;
}
```

在`stop()`函数中将`stopped`变量设置为了`true`，这样便可以结束`run()`函数中的循环，从而从`run()`函数中退出，这样整个线程也就结束了。这里使用了`stopped`变量来实现了进程的终止，并没有使用危险的`terminate()`函数。

4．在`Dialog`类中使用自定义的线程。

先到`dialog.h`文件中，添加头文件包含：

```
#include "mythread.h"
```

然后添加私有对象的定义：

```
MyThread thread;
```

下面到设计模式，分别进入两个按钮的单击信号对应的槽，更改如下：

```
// 启动线程按钮
void Dialog::on_startButton_clicked()
{
    thread.start();
    ui->startButton->setEnabled(false);
    ui->stopButton->setEnabled(true);
}
```

```
// 终止线程按钮
void Dialog::on_stopButton_clicked()
{
    if (thread.isRunning()) {
        thread.stop();
        ui->startButton->setEnabled(true);
        ui->stopButton->setEnabled(false);
    }
}
```

在启动线程时调用了`start()`函数，然后设置了两个按钮的状态。在终止线程时，先使用`isRunning()`来判断线程是否在运行，如果是，则调用`stop()`函数来终止线程，并且更改两个按钮的状态。现在运行程序，按下“启动线程”按钮，查看应用程序输出栏的输出，然后再按下“终止线程”按钮，可以看到已经停止输出了。

下面我们接着来优化这个程序，通过信号和槽来将子线程中的字符串显示到主界面上。

1．在`mythread.h`文件中添加信号的定义：

```
signals:
void stringChanged(const QString &);
```

2．然后到`mythread.cpp`文件中更改`run()`函数的定义：

```
void MyThread::run()
{
    long int i = 0;
    while (!stopped) {
       QString str = QString("in MyThread: %1").arg(i);
       emit stringChanged(str);
       msleep(1000);
       i++;
    }
    stopped = false;
}
```

这里每隔1秒就发射一次信号，里面包含了生成的字符串。

3．到`dialog.h`文件中添加槽声明：

```
private slots:
    void changeString(const QString &);
```

4．打开`dialog.ui`，然后向主界面上拖入一个`Label`标签部件。

5．到`dialog.cpp`文件中，在构造函数里面添加信号和槽的关联：

```
// 关联线程中的信号和本类中的槽
connect(&thread, SIGNAL(stringChanged(QString)),
this, SLOT(changeString(QString)));
```

6．然后添加槽的定义：

```
void Dialog::changeString(const QString &str)
{
    ui->label->setText(str);
}
```

这里就是将子线程发送过来的字符串显示到主界面上。现在可以运行程序，查看效果了。

（二）线程同步

Qt中的`QMutex`、`QReadWriteLock`、`QSemaphore`和`QWaitCondition`类提供了同步线程的方法。虽然使用线程的思想是多个线程可以尽可能的并发执行，但是总有一些时刻，一些线程必须停止来等待其他线程。例如，如果两个线程尝试同时访问相同的全局变量，结果通常是不确定的。`QMutex`提供了一个互斥锁（`mutex`）；`QReadWriteLock`即读-写锁；`QSemaphore`即信号量；`QWaitCondition`即条件变量。

（三）可重入与线程安全

在查看Qt的帮助文档时，在很多类的开始都写着“All functions in this class are reentrant”，或者“All functions in this class are thread-safe”。在Qt文档中，术语“可重入（reentrant）”和“线程安全（thread-safe）”用来标记类和函数，来表明怎样在多线程应用程序中使用它们：

一个线程安全的函数可以同时被多个线程调用，即便是这些调用使用了共享数据。因为该共享数据的所有实例都被序列化了。

一个可重入的函数也可以同时被多个线程调用，但是只能是在每个调用使用自己的数据时。