# Qt 信号和槽

## 1 简介

信号和槽机制是Qt的核心机制，它可以让编程人员将互不相关的对象绑定在一起，实现对象之间的通信。信号和槽是 QT 自行定义的一种通信机制，它独立于标准的 C/C++ 语言，因此要正确的处理信号和槽，必须借助一个称为 moc（Meta Object Compiler）的 QT 工具，该工具是一个 C++ 预处理程序，它为高层次的事件处理自动生成所需要的附加代码。

**信号**

当对象改变其状态时，信号就由该对象发射(emit)出去，而且对象只负责发送信号，它不知道另一端是谁在接受这个信号(可以一个或者多个槽)。这样就做到了真正的信息封装，能确保对象被当作一个真正的软件组件来使用。

**槽**

用于接收信号，而且槽只是普通的对象成语函数。一个槽并不知道是否有任何信号与自己相连接。而且对象并不了解具体的通信机制。

**信号和槽的连接**

所有从QObject或者其子类（例如QWidget）派生的类都能够包含信号和槽。因为信号与槽的连接是通过QObject的connect（）成员函数来实现的。

connect(sender,SINGAL(singal),receiver,SLOT(slot));

其中sender与receiver是指向对象的指针，SINGAL（）和SLOT（）是转换信号与槽的宏。

## 2 特点

* 一个信号可以连接多个槽

当信号发射时，会以不确定的顺序一个接一个的调用各个槽。

* 多个信号可以连接同一个槽

无论是哪一个信号被发送，都会调用这个槽。

* 信号可以直接相互连接

发射第一个信号时，也可以发射第二个信号。

* 连接可以被移除

这种情况用的比较少，因为在对象被删除时，Qt会自动移除与这个对象相关的所有连接。语法如下：

disconnect(push\_button); //断开push\_button的所有连接

disconnect(sender,SINGAL(singal),receiver,SLOT(slot));

## 3 应注意的问题

* 信号与槽机制与普通函数的调用一样，如果使用不当的话，在程序执行时也有可能产生死循环。因此，在定义槽函数时一定要注意避免间接形成无限循环，即在槽中再次发射所接收到的同样信号。例如 , 在前面给出的例子中如果在 mySlot() 槽函数中加上语句 emit mySignal() 即可形成死循环。
* 如果一个信号与多个槽相联系的话，那么，当这个信号被发射时，与之相关的槽被激活的顺序将是随机的。
* 宏定义不能用在 signal 和 slot 的参数中。
* 信号和槽的参数个数与类型必须一致。

## 4 实现

信号和槽的实现主要依赖于Qt自带的moc元编译器。程序编译之前，先将一些扩展语法去掉，之后生成moc\_文件。moc 全称是 Meta-Object Compiler，也就是“元对象编译器”。Qt 程序在交由标准编译器编译之前，先要使用 moc 分析 C++ 源文件。如果它发现在一个头文件中包含了宏 Q\_OBJECT，则会生成另外一个 C++ 源文件。这个源文件中包含了 Q\_OBJECT 宏的实现代码。这个新的文件名字将会是原文件名前面加上 moc\_ 构成。这个新的文件同样将进入编译系统，最终被链接到二进制代码中去。因此我们可以知道，这个新的文件不是“替换”掉旧的文件，而是与原文件一起参与编译。另外，我们还可以看出一点，moc 的执行是在预处理器之前。因为预处理器执行之后，Q\_OBJECT 宏就不存在了。

## 5 线程间的信号和槽

描述：界面线程MainApp为主线程，工作线程MyThread为一子线程，从工作线程向主线程传递字符串用于在主线程中显示。

Qt的信号与槽机制可以将任何继承自QObject类的对象捆绑在一起，使不同对象之间能够相互通信。

工作线程：

class MyThread:public QThread

{

Q\_OBJECT

...

signals:

void MsgSignal(const QString& tep);//用于向主线程传递字符串

protected:

void run();//run 中的内容才是子线程中执行的内容！

}

void MyThread::run()

{

Sleep(3000);//头文件：windows.h

QString tep("mou-mou-mou");

emit MsgSignal(tep);

Sleep(3000);//各个Sleep的位置可放置具体执行的工作

}

主线程：

class MainApp:public QWidget

{

Q\_OBJECT

...

public:

MainApp();

...

private slots:

void OnMsgSignal(const QString& tep2);//接受子线程传递字符串用于显示

private:

MyThread\* m\_thread;

}

MainApp::MainApp()

{

...

m\_thread= new MyThread();

connect(m\_thread, SIGNAL(MsgSignal(const QString&)),this, SLOT(OnMsgSignal(const QString&))); //此处connect的第五个参数默认变成Qt::QueuedConnection队列方式，而平时使用时用的是Qt::DirectConnection直接连接方式

m\_thread->start();

}

void MainApp::OnMsgSignal(const QString& tep2)

{

//使用子线程传递来的tep2

}

注意：

* connect函数的第五个参数代表信号与槽的连接模式，线程间的信号与槽不能使用Qt::DirectConnection直接连接方式，因为它要求在发信号的线程内执行槽函数。而Qt::QueuedConnection队列方式将信号转换成事件发送到槽函数所在线程的消息队列中让槽函数所在线程来处理，可以实现线程安全的线程间的通信。这样的时效性也不差，上面的实现中，会在子线程“run()”函数中的第二个Sleep之前执行主线程的“OnMsgSignal(constQString& tep2)”。
* 线程间用“信号与槽”传递引用参数的话，一定要加const，因为const文字常量存在常量区中，生命周期与程序一样的长。这样可以避免slot调用的时候参数的运行期已过而使引用无效。

## 6 与C/C++相比较：

* Qt中的信号和槽函数的参数类型必须匹配，避免了C++中传递指针导致的类型匹配的安全性问题。
* Qt中利用信号和槽机制，只需知道发送和接收者，没有回调函数那样，需要知道调用函数的位置。