neilux

博客访问： 158277
博文数量： 102
博客积分： 2500
博客等级： 少校
技术积分： 555
用 户 组： 普通用户
注册时间： 2005-12-13 18:27

加关注 短消息
论坛 加好友

QT的信号与槽机制介绍

2005-12-21 17:30:50

分类： C/C++

信号与槽作为QT的核心机制在QT编程中有着广泛的应用，本文介绍了信号与槽的一些基本概念、元对象工具以及在实际使用过程中应注意的一些问题。

QT是一个跨平台的C++ GUI应用构架，它提供了丰富的窗口部件集，具有面向对象、易于扩展、真正的组件编程等特点，更为引人注目的是目前Linux上最为流行的KDE桌面环境就是建立在QT库的基础之上。QT支持下列平台：MS/WINDOWS-95、98、NT和2000；UNIX/X11-Linux、Sun Solaris、HP-UX、Digital Unix、IBM AIX、SGI IRIX；EMBEDDED-支持framebuffer的Linux平台。伴随着KDE的快速发展和普及，QT很可能成为Linux窗口平台上进行软件开发时的GUI首选。

一、概述

信号和槽机制是QT的核心机制，要精通QT编程就必须对信号和槽有所了解。信号和槽是一种高级接口，应用于对象之间的通信，它是QT的核心特性，也是QT区别于其它工具包的重要地方。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。

在我们所熟知的很多GUI工具包中，窗口小部件(widget)都有一个回调函数用于响应它们能触发的每个动作，这个回调函数通常是一个指向某个函数的指针。但是，在QT中信号和槽取代了这些凌乱的函数指针，使得我们编写这些通信程序更为简洁明了。 信号和槽能携带任意数量和任意类型的参数，他们是类型完全安全的，不会像回调函数那样产生core dumps。

所有从QObject或其子类(例如Qwidget)派生的类都能够包含信号和槽。当对象改变其状态时，信号就由该对象发射(emit)出去，这就是对象所要做的全部事情，它不知道另一端是谁在接收这个信号。这就是真正的信息封装，它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，但它们是普通的对象成员函数。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。

你可以将很多信号与单个的槽进行连接，也可以将单个的信号与很多的槽进行连接，甚至于将一个信号与另外一个信号相连接也是可能的，这时无论第一个信号什么时候发射系统都将立刻发射第二个信号。总之，信号与槽构造了一个强大的部件编程机制。

二、信号

当某个信号对其客户或所有者发生的内部状态发生改变，信号被一个对象发射。只有 定义过这个信号的类及其派生类能够发射这个信号。当一个信号被发射时，与其相关联的槽将被立刻执行，就象一个正常的函数调用一样。信号-槽机制完全独立于任何GUI事件循环。只有当所有的槽返回以后发射函数（emit）才返回。 如果存在多个槽与某个信号相关联，那么，当这个信号被发射时，这些槽将会一个接一个地 执行，但是它们执行的顺序将会是随机的、不确定的，我们不能人为地指定哪个先执行、哪 个后执行。

信号的声明是在头文件中进行的，QT的signals关键字指出进入了信号声明区，随后即可 声明自己的信号。例如，下面定义了三个信号：

```
signals:
    void mySignal();
    void mySignal(int x);
    void mySignalParam(int x,int y);
```

在上面的定义中，signals是QT的关键字，而非C/C++的。接下来的一行void mySignal() 定义了信号mySignal，这个信号没有携带参数；接下来的一行void mySignal(int x)定义 了重名信号mySignal，但是它携带一个整形参数，这有点类似于C++中的虚函数。从形式上 讲信号的声明与普通的C++函数是一样的，但是信号却没有函数体定义，另外，信号的返回 类型都是void，不要指望能从信号返回什么有用信息。

信号由moc自动产生，它们不应该在.cpp文件中实现。

三、槽

槽是普通的C++成员函数，可以被正常调用，它们唯一的特殊性就是很多信号可以与其相关联。当与其关联的信号被发射时，这个槽就会被调用。槽可以有参数，但槽的参数不能有缺省值。

既然槽是普通的成员函数，因此与其它的函数一样，它们也有存取权限。槽的存取权限决定了谁能够与其相关联。同普通的C++成员函数一样，槽函数也分为三种类型，即public slots、private slots和protected slots。

1. public slots：在这个区内声明的槽意味着任何对象都可将信号与之相连接。这对于组件编程非常有用，你可以创建彼此互不了解的对象，将它们的信号与槽进行连接以便信息能够正确的传递。
2. protected slots：在这个区内声明的槽意味着当前类及其子类可以将信号与之相连接。这适用于那些槽，它们是类实现的一部分，但是其界面接口却面向外部。
3. private slots：在这个区内声明的槽意味着只有类自己可以将信号与之相连接。这适用于联系非常紧密的类。

槽也能够声明为虚函数，这也是非常有用的。

槽的声明也是在头文件中进行的。例如，下面声明了三个槽：

```
public slots:
    void mySlot();
    void mySlot(int x);
    void mySignalParam(int x,int y);
```

四、信号与槽的关联

通过调用QObject对象的connect函数来将某个对象的信号与另外一个对象的槽函数相关联，这样当发射者发射信号时，接收者的槽函数将被调用。该函数的定义如下：

```
bool QObject::connect ( const QObject * sender, const char * signal,
    const QObject * receiver, const char * member ) [static]
```

这个函数的作用就是将发射者sender对象中的信号signal与接收者receiver中的member槽函数联系起来。当指定信号signal时必须使用QT的宏SIGNAL()，当指定槽函数时必须使用宏SLOT()。如果发射者与接收者属于同一个对象的话，那么在connect调用中接收者参数可以省略。

例如，下面定义了两个对象：标签对象label和滚动条对象scroll，并将valueChanged()信号与标签对象的setNum()相关联，另外信号还携带了一个整形参数，这样标签总是显示滚动条所处位置的值。

```
QLabel     *label  = new QLabel;
QScrollBar *scroll = new QScrollBar;
QObject::connect( scroll, SIGNAL(valueChanged(int)),
                  label,  SLOT(setNum(int)) );
```

一个信号甚至能够与另一个信号相关联，看下面的例子：

```
class MyWidget : public QWidget
{
public:
    MyWidget();
...
signals:
    void aSignal();
...
private:
...
    QPushButton *aButton;
};

MyWidget::MyWidget()
{
    aButton = new QPushButton( this );
    connect( aButton, SIGNAL(clicked()), SIGNAL(aSignal()) );
}
```

在上面的构造函数中，MyWidget创建了一个私有的按钮aButton，按钮的单击事件产生的信号clicked()与另外一个信号aSignal()进行了关联。这样一来，当信号clicked()被发射时，信号aSignal()也接着被发射。当然，你也可以直接将单击事件与某个私有的槽函数相关联，然后在槽中发射aSignal()信号，这样的话似乎有点多余。

当信号与槽没有必要继续保持关联时，我们可以使用disconnect函数来断开连接。其定义如下：

```
bool QObject::disconnect ( const QObject * sender, const char * signal,
        const Object * receiver, const char * member ) [static]
```

这个函数断开发射者中的信号与接收者中的槽函数之间的关联。

有三种情况必须使用disconnect()函数：

- 断开与某个对象相关联的任何对象。这似乎有点不可理解，事实上，当我们在某个对象中定义了一个或者多个信号，这些信号与另外若干个对象中的槽相关联，如果我们要切断这些关联的话，就可以利用这个方法，非常之简洁。

```
disconnect( myObject, 0, 0, 0 )
或者
myObject->disconnect()
```

- 断开与某个特定信号的任何关联。

```
disconnect( myObject, SIGNAL(mySignal()), 0, 0 )
或者
myObject->disconnect( SIGNAL(mySignal()) )
```

- 断开两个对象之间的关联。

```
disconnect( myObject, 0, myReceiver, 0 )
或者
myObject->disconnect(  myReceiver )
```

在disconnect函数中0可以用作一个通配符，分别表示任何信号、任何接收对象、接收对象中的任何槽函数。但是发射者sender不能

为0，其它三个参数的值可以等于0。

五、元对象工具

元对象编译器moc（meta object compiler）对C++文件中的类声明进行分析并产生用于初始化元对象的C++代码，元对象包含全部信号和槽的名字以及指向这些函数的指针。

moc读C++源文件，如果发现有Q_OBJECT宏声明的类，它就会生成另外一个C++源文件，这个新生成的文件中包含有该类的元对象代码。例如，假设我们有一个头文件mysignal.h，在这个文件中包含有信号或槽的声明，那么在编译之前 moc 工具就会根据该文件自动生成一个名为mysignal.moc.h的C++源文件并将其提交给编译器；类似地，对应于mysignal.cpp文件moc工具将自动生成一个名为mysignal.moc.cpp文件提交给编译器。

元对象代码是signal/slot机制所必须的。用moc产生的C++源文件必须与类实现一起进行编译和连接，或者用#include语句将其包含到类的源文件中。moc并不扩展#include或者#define宏定义,它只是简单的跳过所遇到的任何预处理指令。

六、程序样例

这里给出了一个简单的样例程序，程序中定义了三个信号、三个槽函数，然后将信号与槽进行了关联，每个槽函数只是简单的弹出一个对话框窗口。读者可以用kdevelop生成一个简单的QT应用程序，然后将下面的代码添加到相应的程序中去。

信号和槽函数的声明一般位于头文件中，同时在类声明的开始位置必须加上Q_OBJECT语句，这条语句是不可缺少的，它将告诉编译器在编译之前必须先应用moc工具进行扩展。关键字signals指出随后开始信号的声明，这里signals用的是复数形式而非单数，siganls没有public、private、protected等属性，这点不同于slots。另外，signals、slots关键字是QT自己定义的，不是C++中的关键字。

信号的声明类似于函数的声明而非变量的声明，左边要有类型，右边要有括号，如果要向槽中传递参数的话，在括号中指定每个形式参数的类型，当然，形式参数的个数可以多于一个。

关键字slots指出随后开始槽的声明，这里slots用的也是复数形式。

槽的声明与普通函数的声明一样，可以携带零或多个形式参数。既然信号的声明类似于普通C++函数的声明，那么，信号也可采用C++中虚函数的形式进行声明，即同名但参数不同。例如，第一次定义的void mySignal()没有带参数，而第二次定义的却带有参数，从这里我们可以看到QT的信号机制是非常灵活的。

信号与槽之间的联系必须事先用connect函数进行指定。如果要断开二者之间的联系，可以使用函数disconnect。

```
                                                //tsignal.h
...
class TsignalApp:public QMainWindow
{
        Q_OBJECT
        ...
        //信号声明区
        signals:
                //声明信号mySignal()
                void mySignal();
                //声明信号mySignal(int)
                void mySignal(int x);
                //声明信号mySignalParam(int,int)
                void mySignalParam(int x,int y);

        //槽声明区
        public slots:
                //声明槽函数mySlot()
                void mySlot();
                //声明槽函数mySlot(int)
                void mySlot(int x);
                //声明槽函数mySignalParam (int, int)
                void mySignalParam(int x,int y);
}
...

//tsignal.cpp
...
TsignalApp::TsignalApp()
{
        ...
        //将信号mySignal()与槽mySlot()相关联
        connect(this,SIGNAL(mySignal()),SLOT(mySlot()));
        //将信号mySignal(int)与槽mySlot(int)相关联
        connect(this,SIGNAL(mySignal(int)),SLOT(mySlot(int)));
        //将信号mySignalParam(int,int)与槽mySlotParam(int,int)相关联
        connect(this,SIGNAL(mySignalParam(int,int)),SLOT(mySlotParam(int,int)));
}

// 定义槽函数mySlot()
```

```
void TsignalApp::mySlot()
{
        QMessageBox::about(this,"Tsignal", "This is a signal/slot sample without
parameter.");
}

// 定义槽函数mySlot(int)
void TsignalApp::mySlot(int x)
{
        QMessageBox::about(this,"Tsignal", "This is a signal/slot sample with one
parameter.");
}

// 定义槽函数mySlotParam(int,int)
void TsignalApp::mySlotParam(int x,int y)
{
        char s[256];
        sprintf(s,"x:%d y:%d",x,y);
        QMessageBox::about(this,"Tsignal", s);
}
void TsignalApp::slotFileNew()
{
        //发射信号mySignal()
        emit mySignal();
        //发射信号mySignal(int)
        emit mySignal(5);
        //发射信号mySignalParam(5, 100)
        emit mySignalParam(5,100);
}
```

七、应注意的问题

信号与槽机制是比较灵活的，但有些局限性我们必须了解，这样在实际的使用过程中做到有的放矢，避免产生一些错误。下面就介绍一下这方面的情况。

1．信号与槽的效率是非常高的，但是同真正的回调函数比较起来，由于增加了灵活性，因此在速度上还是有所损失，当然这种损失相对来说是比较小的，通过在一台i586-133的机器上测试是10微秒（运行Linux），可见这种机制所提供的简洁性、灵活性还是值得的。但如果我们要追求高效率的话，比如在实时系统中就要尽可能的少用这种机制。

2．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。

3．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。

4. 宏定义不能用在signal和slot的参数中。

既然moc工具不扩展#define，因此，在signals和slots中携带参数的宏就不能正确地工作，如果不带参数是可以的。例如，下面的例子中将带有参数的宏SIGNEDNESS(a)作为信号的参数是不合语法的：

```
                                        #ifdef ultrix
#define SIGNEDNESS(a) unsigned a
#else
#define SIGNEDNESS(a) a
#endif

class Whatever : public QObject
{

[...]

signals:

    void someSignal( SIGNEDNESS(a) );

[...]

};
```

5. 构造函数不能用在signals或者slots声明区域内。

的确，将一个构造函数放在signals或者slots区内有点不可理解，无论如何，不能将它们放在private slots、protected slots或者public slots区内。下面的用法是不合语法要求的：

```
                                        class SomeClass : public QObject
{
    Q_OBJECT
public slots:
    SomeClass( QObject *parent, const char *name )
        : QObject( parent, name ) {}  // 在槽声明区内声明构造函数不合语法
[...]
};
```

6. 函数指针不能作为信号或槽的参数。

例如，下面的例子中将void (*applyFunction)(QList*, void*)作为参数是不合语法的：

```
                                        class someClass : public QObject
{
    Q_OBJECT
[...]
public slots:
    void apply(void (*applyFunction)(QList*, void*), char*); // 不合语法
};
```

你可以采用下面的方法绕过这个限制：

```
                                        typedef void (*ApplyFunctionType)(QList*, void*);


class someClass : public QObject
{
    Q_OBJECT
[...]
public slots:
    void apply( ApplyFunctionType, char *);
};
```

7.信号与槽不能有缺省参数。

既然signal->slot绑定是发生在运行时刻，那么，从概念上讲使用缺省参数是困难的。下面的用法是不合理的：

```
                                        class SomeClass : public QObject
{
    Q_OBJECT
public slots:
    void someSlot(int x=100); // 将x的缺省值定义成100，在槽函数声明中使用是错误的
};
```

8.信号与槽也不能携带模板类参数。

如果将信号、槽声明为模板类参数的话，即使moc工具不报告错误，也不可能得到预期的结果。 例如，下面的例子中当信号发射时，槽函数不会被正确调用：

```
                                        [...]
public slots:
    void MyWidget::setLocation (pair<int,int> location);

[...]
public signals:
    void MyObject::moved (pair<int,int> location);
```

但是，你可以使用typedef语句来绕过这个限制。如下所示：

```
                                        typedef pair<int,int> IntPair;
[...]
public slots:
    void MyWidget::setLocation (IntPair location);

[...]
public signals:
    void MyObject::moved (IntPair location);
```

这样使用的话，你就可以得到正确的结果。

9.嵌套的类不能位于信号或槽区域内，也不能有信号或者槽。

例如，下面的例子中，在class B中声明槽b()是不合语法的，在信号区内声明槽b()也是不合语法的。

```
                                        class A
{
    Q_OBJECT
public:
```

```
    class B
{
    public slots:   // 在嵌套类中声明槽不合语法
        void b();
    [....]
    };
signals:
    class B
{
    // 在信号区内声明嵌套类不合语法
    void b();

    [....]
    }:
};
```

10.友元声明不能位于信号或者槽声明区内。相反，它们应该在普通C++的private、protected或者public区内进行声明。下面的例子是不合语法规范的：

```
                                        class someClass : public QObject
{
    Q_OBJECT
[...]
signals: //信号定义区
    friend class ClassTemplate; // 此处定义不合语法
};
```

相关资源

-
- KDevelop

作者简介

唐新华 软件工程师。Email: xhsmart@263.netQT

上一篇：基于 Linux 的动态电源管理:使嵌入式设备更节能
下一篇：Qt 的内部进程通信机制

0