Programação gráfica com QT - Parte 3

Nesta parte veremos os sinais e slots (se alguém achar uma palavra em português para slot, vai ajudar bastante).

É dessa forma que a Qt trata os eventos: o componete dispara um sinal e você tem que agregar esse sinal a um slot.

Na parte anterior fizemos uma pequena janela que continha um botão "Fechar", que não funcionava. Nosso objetivo agora é fazer esse botão funcionar da maneira apropriada.

Vejamos o código (mywidget.cpp). Novamente, só será destacado o que for diferente do código anterior. Por isso, é importante que você já tenha visto as duas partes anteriores:

```
 1: /****************************************************
 2:  * Programação gráfica com Qt
 3:  *
 4:  * Implementação de MyWidget
 5:  *
 6:  * Programa: mywidget.cpp
 7:  ****************************************************/
 8:
 9: #include "mywidget.h"
10: #include <qpushbutton.h>
11: #include <qlineedit.h>
12: #include <qlabel.h>
13:
14: MyWidget::MyWidget(QWidget* parent, const char* name, WFlags fl)
15:         : QWidget(parent, name, fl)
16: {
17:     setGeometry(100, 100, 300, 200);
18:
19:     c_campo=new QLabel("Campo:", this);
20:     c_campo->move(20, 20);
21:
22:     campo=new QLineEdit(this);
23:     campo->setGeometry(20, 45, 260, 25);
24:
25:     fechar=new QPushButton("&Fechar", this);
26:     fechar->setGeometry(230, 150, 50, 30);
27:
28:     connect(fechar, SIGNAL(clicked()), this, SLOT(close()));
29: }
```

A única diferença é o connect() na linha 28.

28: connect(fechar, SIGNAL(clicked()), this, SLOT(close()));

Esse conect liga o sinal "clicked()" de botão ao slot "close()"

de this.

Vamos compilar e testar:

g++ -I $QTDIR/include -c main.cpp
g++ -I $QTDIR/include -c mywidget.cpp
g++ -L $QTDIR/lib -lqt -o mywidget main.o mywidget.o
./mywidget

Obviamente, você poderia escrever rotinas antes de fechar a janela, mas para isso, você mesmo teria que criar um slot.

Vejamos. Primeiro vamos declará-lo no header (.h):

```
 1: /*************************************************
 2:  * Programação gráfica com Qt
 3:  *
 4:  * Interface de MyWidget
 5:  *
 6:  * Programa: mywidget.h
 7:  *************************************************/
 8:
 9: #ifndef MYWIDGET_H
10: #define MYWIDGET_H
11:
12: #include <qwidget.h>
13:
14: class QPushButton;
15: class QLineEdit;
16: class QLabel;
17:
18: class MyWidget : public QWidget {
19:     Q_OBJECT
20: public:
21:     QLabel* c_campo;
22:     QLineEdit* campo;
23:     QPushButton* fechar;
24:
25:     MyWidget(QWidget* parent=0, const char* name=0, WFlags fl=0);
26: public slots:
27:     void fecharJanela();
28: };
29:
30: #endif // MYWIDGET_H
```

No .h temos as seguintes (novas) linhas:

19: Q_OBJECT

E

26: public slots:
27: void fecharJanela();

O Q_OBJECT é necessário para gerarmos o arquivo MOC (veremos mais à frente), e abaixo de public slots: declaramos todos os slots da nossa classe.

Agora na implementação:

```
 1: /*************************************************
 2:  * Programação gráfica com Qt
 3:  *
 4:  * Implementação de MyWidget
 5:  *
 6:  * Programa: mywidget.cpp
 7:  *************************************************/
 8:
 9: #include "mywidget.h"
10: #include <qpushbutton.h>
11: #include <qlineedit.h>
12: #include <qlabel.h>
13:
14: MyWidget::MyWidget(QWidget* parent, const char* name, WFlags fl)
15:         : QWidget(parent, name, fl)
16: {
17:     setGeometry(100, 100, 300, 200);
18:
19:     c_campo=new QLabel("Campo:", this);
20:     c_campo->move(20, 20);
21:
22:     campo=new QLineEdit(this);
23:     campo->setGeometry(20, 45, 260, 25);
24:
25:     fechar=new QPushButton("&Fechar", this);
26:     fechar->setGeometry(230, 150, 50, 30);
27:
28:     connect(fechar, SIGNAL(clicked()), this, SLOT(fecharJanela()));
29: }
30:
31: void MyWidget::fecharJanela()
32: {
33:     qWarning("Adeus, mundo cruel!!!!");
34:     close();
35: }
```

O connect() passou a ser:

connect(botao, SIGNAL(clicked()), this, SLOT(fecharJanela()));

Temos a implementação de fecharJanela():

31: void MyWidget::fecharJanela()

```
31: void MyWidget::fecharJanela()
32: {
33: qWarning("Adeus, mundo cruel!!!!");
34: close();
35: }
```

Na linha 33, usamos um qWarning para escever uma mensagem no terminal, e na linha 34, chamamos o close() (que foi usado no início deste artigo) para fechar a janela

Além dessas alterações no código, existe mais um passo para compilarmos o programa: antes de gerarmos o binário (executável), precisamos gerar um arquivo MOC (gerado pelo Meta Object Compiler da Qt). Esse arquivo (na verdade um programa .cpp) contém instruções para o funcionamento dos sinais e slots, e deve sempre ser criado quando se tem sinais e slots (e deve ser recriado sempre que o .h da classe é alterado).

Para criar o MOC, digite:

```
moc -o mywidget.moc.cpp mywidget.h
```

E agora vamos para a seqüencia de compilação (não esqueça do MOC):

```
g++ -c -I $QTDIR/include mywidget.cpp
g++ -c -I $QTDIR/include mywidget.moc.cpp
g++ -c -I $QTDIR/include main.cpp
g++ -o mywidget -L $QTDIR/lib -lqt mywidget.o mywidget.moc.o main.o
```

E vamos ao teste

```
$ ./mywidget
Adeus, mundo cruel!
$
```

Na próxima parte veremos como facilitar o design das janelas com o Qt Designer. Mas sugiro que antes disso, você faça alguns exercícios com o que já vimos. Utilize a documentação da Qt (normalmente em usr/lib/qt/doc ou http://doc.trolltech.com) para ver os sinais e slots de cada componente e crie os seus próprios slots. A intenção destes artigos sobre Qt não é a de tranformar alguém em um expert na programação gráfica, mas apenas indicar o "caminho das pedras".

Até a próxima!

Autor: Ricardo Vaz Mannrich
Email: husk-le@comlinux.com.br