Introduzione al mondo Qt

Qt Creator, il framework e una prima applicazione

Politecnico Open unix Labs

Corso programmazione Qt e KDE

Indice

Perché Qt?

1
Multipiattaforma

Multipiattaforma

Qt è ufficialmente supportato da Nokia[1] su:

- Embedded Linux[2]
- Mac OS X
- Windows
- Linux (X11)
- Windows CE/Mobile
- Symbian
- MeeGo

[1] http://qt.nokia.com/products/platform/platforms

[2] Qt offre un proprio window manager indipendente da X11

Porting non ufficiali

Esistono port non supportati da Nokia su altre piattaforme[3]:

- OpenSolaris
- Haiku
- OS/2
- Amiga OS4
- iPhone
- webOS
- Amazon Kindle DX
- Android
- BlackBerry

[3] https://en.wikipedia.org/wiki/Qt_(framework)#External_ports

Non ti piace C++?

- Esistono binding per moltissimi altri linguaggi
- Java, C#, Ruby, PHP, Perl...
- Sono spesso di scarsa qualità
- Eccetto i binding per Python (PyQt e PySide)

Non ti piace C++?

- Esistono binding per moltissimi altri linguaggi
- Java, C#, Ruby, PHP, Perl...
- Sono spesso di scarsa qualità
- Eccetto i binding per Python (PyQt e PySide)

Non ti piace C++?

- Esistono binding per moltissimi altri linguaggi
- Java, C#, Ruby, PHP, Perl...
- Sono spesso di scarsa qualità
- Eccetto i binding per Python (PyQt e PySide)

Non ti piace C++?

- Esistono binding per moltissimi altri linguaggi
- Java, C#, Ruby, PHP, Perl...
- Sono spesso di scarsa qualità
- Eccetto i binding per Python (PyQt e PySide)

2
Free Software

LGPL

- Qt è rilasciato sotto licenza LGPL 2.1
- Esiste anche una licenza commerciale

3
Ricchezza

Non solo GUI

- QtGui è solo uno dei moduli del framework Qt
- Con Qt, C++ raggiunge un livello superiore:
  - Semplicità e usabilità
  - Uniformità sulle varie piattaforme
  - E poi metacall, segnali, slot...

Non solo GUI

- QtGui è solo uno dei moduli del framework Qt
- Con Qt, C++ raggiunge un livello superiore:
  - Semplicità e usabilità
  - Uniformità sulle varie piattaforme
  - E poi metacall, segnali, slot...

I moduli

- QtCore: il cuore di Qt
- QtGui: tutto quello che riguarda le GUI
- QtMultimedia: riproduzione video e audio
- QtNetwork: QTcpSocket, QFtp, QNetworkRequest...
- QtOpenGL: interfaccia alle librerie OpenGL
- QtScript: scripting per le applicazioni Qt
- QtSql: interfaccia ai database basati su SQL
- QtSvg: supporto per grafica vettoriale SVG
- QtWebKit: interfaccia al motore di rendering HTML WebKit
- QtXml: interfaccia per XML

I moduli (cont.)

- QtXmlPatterns: XPath, XQuery, XSLT, XSD...
- QtOpenVG: grafica 2D per embedded
- QtDeclarative: interfacce dinamiche fluide
- Phonon: interfaccia avanzata per riproduzione audio/video
- Qt3Support: modulo di compatibilità con Qt 3
- QtHelp: supporto per la realizzazione di guide
- QtTest: supporto per effettuare test d'unità
- QAxContainer: integrazione ActiveX (Windows only)
- QtDBus: IPC tramite D-Bus (Unix only)

Install time!

- Tutto il necessario per sviluppare su Qt:

http://qt.nokia.com/downloads/

Qt Creator

- Qt Creator è la scelta ideale per sviluppare con Qt:
- Qt Designer
- Refactoring
- Debugger ottimizzato per Qt
- Supporto Git, CVS, Mercurial, SVN

Demo

Indice

Indice

Tipi di dato interi

- Qt offre tipi di dato interi di lunghezza fissata
- Ad esempio qint8[4], qint16, qint32, qint64
- Niente problemi di portabilità tra piattaforme diverse
- **sizeof**(qint8) == 1, sempre

[4] http://qt-project.org/doc/qt-4.8/qtglobal.html#qint8-typedef

QChar e QString

- QChar è un carattere Unicode a 16 bit[5]
- QStringè un array di QChar
- Una maniera unificata di maneggiare stringhe non-ASCII
- Le QString sono modificabili ma copy-on-write

[5] UCS-2 per essere precisi

Modificare QString

```
QString stringa = "Salve a tutti !!!";

// Salve a tutti!
stringa.chop(2);

// Salve a tutti! Come
stringa += " Come";

// Salve a tutti! Come va?
stringa.append(" va?");
```

Modificare QString (cont.)

```
// Salve a tutti voi! Come va?
stringa.insert(13, "␣voi");

// Salve a tutti! Come va?
stringa.remove(14, 4);

// Ciao a tutti! Come va?
stringa.replace("Salve", "Ciao");
```

Parti di QString

```
// ao
stringa.mid(2, 2);

// Ciao
stringa.left(4);

// va?
stringa.right(3);

// Ciao a tutti! Come va? Ciao a tutti! Come
   va?
stringa.repeated(2);
```

Altri metodi di QString

```
stringa.indexOf("a");          // 2

stringa.at(1);                 // i

stringa[1];                    // i

stringa.count("a");            // 3
```

Altri metodi di QString (cont.)

```
stringa.length();                // 22

stringa.contains("utti");        // vero

stringa.endsWith("?");           // vero

stringa.startsWith("Salve");     // falso
```

Conversione da numero a stringa

```
// 123
QString::number(123);
```

Il metodo QString::arg()

```
// Siamo 3 amici e ci piace mangiare.
QString("Siamo %1 amici, e ci piace %2.")
    .arg(3)
    .arg("mangiare");
```

QString e **char***

- QString supportano caratteri non-ASCII
- Per convertire nel classico array di **char**:

```
QString miaStringa = "testo";
char *miaStringaClassica =
    miaStringa.toLocal8Bit().data();
cout << miaStringaClassica << endl;
```

Indice

I contenitori

- Qt offre una nutrita serie di contenitori
- Permettono di soddisfare le esigenze più comuni
- Fanno uso dei template: QList<QString>
- Anche contenitori chiave-valore: QMap<QString,**int**>

Contenitori semplici

- QList<T>: array di puntatori, accesso via indice rapido
- QLinkedList<T>: lista concatenata, inserimenti rapidi
- QVector<T>: array vero e proprio, inserimenti lenti
- QStack<T>: pila LIFO, basato su QVector<T>
- QQueue<T>: coda FIFO, basato su QList<T>
- QSet<T>: senza ordinato né duplicati, basato su QHash

Esempio

- Hanno interfacce abbastanza simili

```
QList<QString> lista;
lista.append("primo");
lista += "secondo";
lista.push_back("terzo");

lista.removeLast();
```

Il ciclo foreach

- Qt introduce un nuovo costrutto[6]: il foreach

```
foreach (QString elemento, lista)
    stampa(elemento);
```

[6] Tramite macro, il tutto è standard-compliant

Contenitori chiave-valore

- QMap<Key,T>: array associativo ordinato per chiave
- QMultiMap<Key,T>: più di un valore per chiave
- QHash<Key,T>: non ordinato, lookup rapidi
- QMultiHash<Key,T>

Esempio

```
QHash<QString, int> punteggio;

punteggio.insert("Mario", 123);
punteggio.insert("Carlo", 3);
punteggio.insert("Paolo", 2);

cout << punteggio["Mario"] << endl;
```

Indice

QtGrep

- Realizziamo una semplice applicazione console
- Effettuiamo una ricerca nei file della cartella corrente
- Per la ricerca useremo le espressioni regolari

Alcune classi che ci serviranno

- QDir: agisce su una cartella (lista file, cambiare cartella...)
- QFile: permette di agire su un file (aprirlo, eliminarlo...)
- QTextStream: permette di leggere e scrivere su un flusso
- QRegExp: effettua ricerche e sostituzioni tramite regex

Demo

Indice

Spingiamoci oltre C++

- Finora abbiamo visto come Qt amplia le possibilità di C++
- Tutte le feature usate sono compatibili con il compilatore
- QObject richiede invece un altro passaggio, moc
- moc è un generatore di codice C++ della SDK di Qt
- Qt Creator si occupa di eseguirlo prima di compilare

QObject

- QObject è la classe alla base del modello a oggetti di Qt
- Offre le seguenti funzionalità:
  - Gerarchia di oggetti ad albero
  - Segnali e slot
  - Proprietà
  - Introspezione

Un QObject minimale

```
class TestClass : public QObject
{
    Q_OBJECT

public:
    TestClass(QObject *parent = 0);

};
```

```
TestClass::TestClass(QObject *parent) :
   QObject(parent)
{
}
```

Caratteristiche

- Eredita la classe QObject
- Q_OBJECT serve per arricchire la classe di nuovi metodi
- Il costruttore prende un riferimento ad un QObject
- moc genererà moc_testclass.cpp

Indice

Albero di oggetti

- Il costruttore prende un QObject, che sarà il suo genitore
- In questo modo si crea un albero di oggetti
- QObject::children() permette di avere i figli dell'oggetto
- Usato ad esempio per collegare una finestra ai suoi widget

Esempio di albero

```
TestClass nonno(NULL);
TestClass figlio1(&nonno);
TestClass figlio2(&nonno);
TestClass nipote1_1(&figlio1);
TestClass nipote1_2(&figlio1);
TestClass nipote1_3(&figlio1);
TestClass nipote2_1(&figlio2);
```

Stampiamo l'albero

```
int node_index = 0;
int depth = -1;

void printTree(QObject* radix)
{
  node_index++;
  depth++;
  cout << QString("\t").repeated(depth)
          .toLocal8Bit().data()
       << node_index << endl;
  foreach (QObject* child, radix->children())
    printTree(child);
  depth--;
}
```

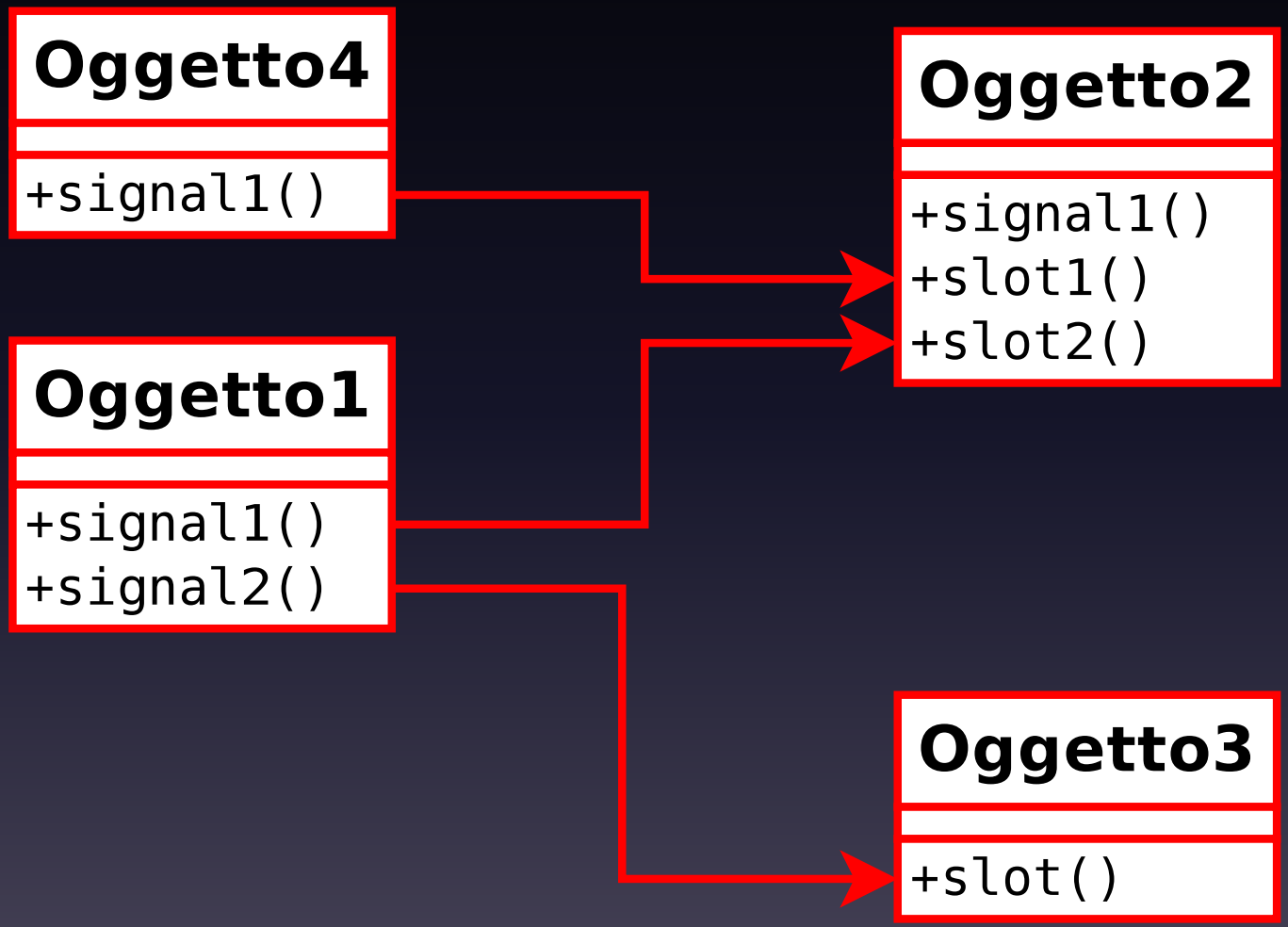
Stampiamo l’albero (cont.)

Indice

Notifiche e comunicazione tra classi

- Supponiamo che l'utente clicchi su un bottone
- Vogliamo che la classe "bottone" venga notificata
- Callback tramite puntatori a funzione sono una soluzione

L’ approccio Qt: segnali e slot



L’ approccio Qt: segnali e slot (cont.)

- Una classe espone slot e segnali
- Un evento (pressione di un bottone) è un segnale
- Un segnale può essere associato ad uno o più slot

Il click su un bottone

- Osserviamo la classe QAbstractButton:

```
class QAbstractButton : public QWidget
{
    Q_OBJECT

// ...

signals:
    void pressed();
    void released();
    void clicked(bool checked = false);
    void toggled(bool checked);

// ...

}
```

Il click su un bottone (cont.)

- QAbstractButton ha 4 segnali che rappresentano gli eventi
- Useremo QPushButton, che eredita da QAbstractButton
- I segnali non vanno implementati, sono solo dichiarazioni
- Proviamo a collegare il segnale clicked ad un nostro slot

Aggiungiamo uno slot

- Espandiamo la nostra classe con uno slot:

```
class TestClass : public QObject
{
    Q_OBJECT
public:
    TestClass(QObject *parent = 0);

public slots:
    void mybutton_clicked(bool checked);
};
```

```
void TestClass::mybutton_clicked(bool checked)
{
    // Do something
}
```

Creiamo il collegamento segnale-slot

- Gli slot vanno implementati!
- Ora colleghiamo il nuovo slot al segnale di un bottone:

```
TestClass::TestClass(QObject *parent) :
    QObject(parent)
{
     QPushButton bottone;
     QObject::connect(
          &bottone,
          SIGNAL(clicked(bool checked)),
          this,
          SLOT(mybutton_clicked(bool checked)));
}
```

Connessioni automatiche

- A volte si hanno molti segnali e slot da collegare
- Usando delle convenzioni il collegamento è automatico
- Questo si realizza con QMetaObject::connectSlotsByName
  - Prendi tutti gli oggetti figli (children) dell'oggetto
  - Prendi il nome e i segnali di ciascun figlio
  - Associa i segnali dei figli allo slot corrispondente:

  void on_<nome figlio>_<nome segnale>(<parametri>)

Modifichiamo la nostra classe

```
class TestClass : public QWidget
{
    Q_OBJECT
public:
    TestClass(QWidget *parent = 0);

public slots:
    void on_mybutton_clicked(bool checked);
};
```

Aggiungiamo uno slot

- Connettiamo il segnale a on_mybutton_clicked:

```
TestClass::TestClass(QWidget *parent) :
    QWidget(parent)
{
     QPushButton bottone(this);
     bottone.setObjectName("mybutton");
     QMetaObject::connectSlotsByName(this);
}
```

Indice

L'introspezione[8]

- È un'alternativa all'introspezione di C++ (RTTI)
- QObject offre informazioni sulla struttura di una classe:
  - Nome della classe
  - Classe da cui eredita
  - Metodi e costruttori
  - Segnali, slot e proprietà
- Si richiama tramite il metodo QObject::metaObject()[7]

[7] http://qt-project.org/doc/qt-4.8/qobject.html#metaObject

[8] altresì nota come reflection

Esempio

```
TestClass istanza(NULL);
const QMetaObject *istanza_info
  = istanza.metaObject();

cout << istanza_info->className() << endl;

for (int i = 0;
     i < istanza_info->methodCount();
     i++)
  cout << istanza_info->method(i).signature()
       << endl;
```

Chiamata di metodi per nome

- È anche possibile chiamare un metodo per nome:

```
bool QMetaObject::invokeMethod(
    QObject * obj,
    const char * member, ...)
```

- E creare nuove istanze della classe:

QObject * QMetaObject::newInstance(...)

Indice

Il main

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Struttura di un’applicazione grafica

- Nel main viene creata una QApplication
- Viene creata e mostrata un’istanza di MainWindow
- exec passa la gestione al ciclo degli eventi di Qt

Il form designer

- Il form designer lavora sui file .ui
- Genera un file C++: ui_mainwindow.h
- Contiene tutte le informazioni sulla GUI

La classe MainWindow

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};
```

La classe MainWindow (cont.)

- Eredita da QMainWindow, che eredita da QWidget
- QWidget è la classe base di tutti i componenti grafici Qt
- Il codice generato dal designer è accessibile tramite ui

Creare gli slot tramite il form designer

- Creiamo una casella di testo chiamata textEditor
- Click destro su un componente ▷ Go to slot...
- Viene generato uno slot apposito un MainWindow

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = 0);
    ~MainWindow();
private slots:
    void on_textEditor_textChanged();
private:
    Ui::MainWindow *ui;
};
```

Demo

Indice

Threading in Qt

- Qt offre vari metodi per l'esecuzione parallela di codice
- Diamo un occhio rapido a QThread e a QSemaphore

QThread

- Per creare un thread è sufficiente ereditare da QThread
- Il metodo principale è QThread::run()
- Per lanciare un thread si usa QThread::start()
- QThread::wait() permette di attendere la terminazione

QSemaphore

- Semplice meccanismo di sincronizzazione a semaforo
- Alla creazione fissiamo il numero di thread concorrenti
- Nel caso semplice:

```
QSemaphore semaforo(1);

semaforo.acquire();
// Qui l'esecuzione sarà possibile
// solo un thread alla volta
semaforo.release();
```

Demo