Технология ООП

Санкт-Петербургский государственный политехнический университет

25 октября 2011

Qt —- кросс-платформенный инструментарий разработки ПО на языке программирования C++.

Лицензии

- Qt Commercial
- GNU GPL
- GNU LGPL

Преимущества перед другими фреймворками

- Объектно-ориентированная архитектура библиотеки.
- Открытость.
- Кроссплатформенность.
- Отличная документация.
- Обилие примеров.
- Множество готовых классов и методов.

Компоненты

- QtCore
- QtGui
- QtWidgets
- QtNetwork
- QtOpenGL
- QtSql

Дистрибуция

Исходный код:

- git://code.qt.io/qt/qt5.git
- `https://ftp.osuosl.org/pub/blfs/conglomeration/qt5/qt-everywhere-opensource-src-5.15.6.tar.xz`
- https://github.com/qt/qt5

Windows, MacOS X:

- Online-установщик
- Offline-установщик (`http://www.mirrorservice.org/sites/download.qt-project.org/official_releases/qt/5.12/5.12.0/qt-opensource-windows-x86-5.12.0.exe`)
- Offline-установщик (`http://www.mirrorservice.org/sites/download.qt-project.org/official_releases/qt/5.12/5.12.0/qt-opensource-mac-x64-5.12.0.dmg`)

Перед запуском offline-установщика необходимо отключить сеть интернет.

Дистрибуция Windows

Версия 5.12 для Windows:

- `http://www.mirrorservice.org/sites/download.qt-project.org/official_releases/qt/5.12/5.12.0/qt-opensource-windows-x86-5.12.0.exe`
- `https://csspbstu-my.sharepoint.com/:u:/g/personal/glazunov_vv_spbstu_ru/EZP8HI7NDlRJuYPZxPmUuaQBBgyBqNRtcgAG3jo5C12lIg?e=c2PovN`

В установщике можно выбрать сборку либо для разработки в Visual Studio (MSVC 2017), либо для разработки в Qt creator (MinGW). При выборе MinGW — не забудьте поставить компилятор MinGW 7.3.0. При разработке в Visual Studio, через управления расширениями необходимо установить «Qt Visual Studio Tools», где в настройках прописать путь к файлу qmake.exe.

Дистрибуция другие ОС

Ссылки на дистрибутив для MacOSX (Qt creator):

- `http://www.mirrorservice.org/sites/download.qt-project.org/official_releases/qt/5.12/5.12.0/qt-opensource-mac-x64-5.12.0.dmg`
- `https://csspbstu-my.sharepoint.com/:u:/g/personal/glazunov_vv_spbstu_ru/EcSSpwlraCNPvlP0O1QWCrIBg2jhEoE_XiOvgJI-AKTpJA?e=GsCnF9`

В Linux пакет называется в зависимости от дистрибутива, например, в Debian/Ubuntu — это qt5-default или qt6-base-dev.

Компиляция

- Получить исходный код или разархивировать
- Start > Programs > Visual Studio 2019 > Visual Studio Command Prompt.
- cd /d C:\Qt\version
- configure -debug-and-release -opensource
- nmake
- Qt > Qt Options > Qt Versions > Add

Ускорение компиляции

- Download jom https://github.com/qt-labs/jom
- Extract to C:\jom
- Вместо nmake используйте C:\jom\jom.exe -j 4 , где 4 — число ядер процессора.

Документация

- Qt Assistant
- частично-переведенная документация http://doc.crossplatform.ru/qt/
- examples && demos

Пример

Сборка приложений:

- qmake -project
- qmake
- nmake

Или для Visual Studio:

- qmake -project
- qmake -tp vc
- Открыть получившийся .vcproj в VS.

Пример

```
#include <QApplication>
#include <QLabel>


int main(int argc, char **argv)
{
        QApplication app(argc, argv);
        QLabel *label = new QLabel("Hello World!");
        label->show();
        return app.exec();
}
```

Форматирование HTML

```
QLabel *label = new QLabel("<h2><i>Hello</i
    > <font color=red>World!</font></h2>");
```

Сигналы и слоты

```
#include <QApplication>
#include <QPushButton>


int main(int argc, char **argv)
{
        QApplication app(argc, argv);
QPushButton *button = new QPushButton("Quit");
QObject::connect(button, SIGNAL(clicked()),
&app, SLOT(quit()));
button->show();
        return app.exec();
}
```

Механизм сигналов и слотов

Сигналы и слоты — это фундаментальный механизм Qt, позволяющий связывать объекты друг с другом. Связанным объектам нет необходимости что-либо «знать» друг о друге. Сигналы и слоты гораздо удобнее механизма функций обратного вызова (callbacks) и четко вписываются в концепцию ООП.

Механизм сигналов и слотов

Для использования этого механизма объявление класса должно содержать специальный макрос Q_OBJECT на следующей строке после ключевого слова class:

```
class MyClass {
Q_OBJECT

public:
...

};
```

После макроса Q_OBJECT не нужно ставить точку с запятой. Перед выполнением компиляции, Meta Object Compiler (MOC) анализирует такие классы и автоматически внедряет в них всю необходимую информацию.

Сигналы

Сигнал может быть определен следующим образом:

```
class MyClass: public QObject {
Q_OBJECT

public:
...

signals:
    void mySignal();

};
```

Сигналы

Для того чтобы инициировать сигнал (выслать сигнал) нужно ипользовать ключевое слово emit.
Сигналы могут использовать параметры для передачи дополнительной информации.

```
void MyClass::sendMySignal()
{
    emit mySignal();
}
```

Слоты

Слоты практически идентичны обычным членам-методам C++, при их объявлении можно использовать стандартные спецификаторы доступа public, protected или private.

```
class MyClass: public QObject {
Q_OBJECT

public slots:
    void mySlot()
    {
        ...
    }

};
```

Соединение сигналов и слотов

Для соединения сигналов и слотов можно использовать статический метод connect, определенный в классе QObject. Один сигнал может быть соединен со многими слотами, а так же множество сигналов могут быть соединены с единственным слотом.
В общем виде соединение выглядит следующим образом:

```
connect(sender, SIGNAL(signal), receiver, SLOT(slot));
connect(sender, &Sender::signalMethod, receiver, &Receiver::slotMethod);
```

sender и receiver — это указатели на QObject.
signal и slot — сигнатуры сигнала и слота.
Sender и Receiver классы в которых реализованы сигнал и слот с одинаковой сигнатурой.

Соединение сигналов и слотов

Пример соединения:

```
QObject::connect(spinBox, SIGNAL(valueChanged(int))
    , slider, SLOT(setValue(int)));
QObject::connect(spinBox, &QSpinBox::valueChanged,
    slider, &QSlider::setValue);
```

Разъединение сигналов и слотов

Метод disconnect можно использовать для того, чтобы удалить соединение между сигналом и слотом.

```
disconnect(sender0, SIGNAL(overflow()),receiver1,
    SLOT(handleMathError()));
```

На практике прямой вызов disconnect используется редко, так как Qt автоматически удаляет все соединения при удалении объектов.

```
#include <QApplication>
#include <QHBoxLayout>
#include <QSlider>
#include <QSpinbox>


int main(int argc, char **argv)
{
        QApplication app(argc, argv);
QWidget *window = new QWidget;
window->setWindowTitle("Age: ");
QSpinBox *spinBox = new QSpinBox;
QSlider *slider = new QSlider(Qt::Horizontal);
spinBox->setRange(0, 130);
slider->setRange(0, 130);
QObject::connect(spinBox, SIGNAL(valueChanged(int))
    ,
slider, SLOT(setValue(int)));
```

Сигналы и слоты

```
QObject::connect(slider, SIGNAL(valueChanged(int)),
spinBox, SLOT(setValue(int)));
spinBox->setValue(35);
QHBoxLayout *layout = new QHBoxLayout;
layout->addWidget(spinBox);
layout->addWidget(slider);
window->setLayout(layout);
window->show();
        return app.exec();
}
```

Базовый класс QObject

Фундамент Qt — это объектно-ориентированная модель. Абсолютное большинство классов - это наследники QObject. Данный базовый класс содержит поддержку основополагающих механизмов, которые активно используются потомками:

- сигналы и слоты
- метаинформация
- иерархии объектов
- события, фильтры событий
- свойства
- приведение типов
- таймеры

Базовый класс QObject

QObject в качестве предка при организации множественного наследования предполагает два ограничения:

- В перечне предков QObject (или его потомок) должен быть объявлен первым.
- Только один из предков может быть наследником QObject.

Объекты класса QObject (или его потомков) нельзя непосредственно копировать или передавать как значения так как это противоречит многим ключевых особенностям, таким как иерархические структуры, привязки сигналов/слотов. Конструктор копирования и оператор присваивания QObject объявлены пустыми с помощью макроса Q_DISABLE_COPY

Метаинформация в классах Qt

Метаинформация содержит:

- данные о классе
- информацию о наследовании
- информацию о слотах/сигналах

Для хранения метаинформации используется класс QMetaObject, объектом которого владеет QObject.

Метаинформация в классах Qt

Получить указатель на QMetaObject можно с помощью метода metaObject():

```
// obj – объект класса потомка QObject
if( obj->metaObject()->className() == QString("
    FirstClass") )
{
    // obj является объектом класса FirstClass
    ...
}
```

Метаинформация в классах Qt

Для проверки информации о наследовании можно воспользоваться методом QObject::inherits():

```
// obj – объект класса потомка QObject
if( obj->inherits("QAbstractButton") )
{
QAbstractButton* button = static_cast<
    QAbstractButton*>(obj);
     ...
}
```

Метаинформация в классах Qt

Тоже самое можно сделать воспользовавшись qobject_cast<T>:

```
QAbstractButton* button = qobject_cast<
    QAbstractButton*>(obj);
if( button )
{
    ...
}
```

qobject_cast — является аналогом dynamic_cast, но не требует включения RTTI и для наследников QObject класса работает намного быстрее.

Иерархии объектов в Qt

Для построения иерархий необходимо использовать классы, рожденные от QObject. Объекты классов должны создаваться динамически.
Класс QObject содержит реализацию всех необходимых методов для организации иерархий объектов. Конструктор QObject выглядит следующим образом:

```
QObject(QObject *parent=0);
```

Иерархии объектов в Qt

Одно из основных достоинств иерархий объектов — это автоматическое удаление всех дочерних объектов при удалении корневого объекта. Разработчику достаточно удалить только те объекты, которые являются вершинами.

Иерархии объектов в Qt

Методы QObject для работы с иерархиями:

- setParent() — позволяет задать нового родителя.
- parent() — возвращает указатель на текущего родителя.
- children() — возвращает указатель на список дочерних объектов.
- findChildren() — позволяет искать дочерние объекты по имени (поддерживаются регулярные выражения).
- dumpObjectInfo() — отображает отладочную информацию об иерархии.
- setObjectName()/objectName() — задать/получить имя объекта.

Обработка событий в классе QObject

События используются для оповещения объектов о возникновении/изменении каких-либо ситуаций. Основными источниками событий являются элементы пользовательского интерфейса, кроме того существуют события, генерируемые системными объектами (например, таймерами). В общем случае, сразу после возникновения, событие помещается в очередь для дальнейшей обработки.

Таймеры класса QObject

Класс QObject содержит встроенные таймеры, которые позволяют организовать периодическое повторение определенных действий. Для использования таймеров QObject содержит следующие методы:

- int startTimer(int interval)
- virtual void timerEvent(QTimerEvent* event)
- void killTimer(int id)

Таймеры класса QObject

Объект QObject позволяет создавать множество таймеров. При реализации потомка таймеры могут быть запущены следующим образом:

```
void MyObject :: beginPolling()
{
    timer1 = startTimer( 1000 );    // каждую секунду
    timer2 = startTimer( 60000 );   // каждую минуту
}
```

Таймеры класса QObject

При этом виртуальный метод timerEvent() должен действовать в зависимости от идентификатора вызывающего таймера:

```
void MyObject :: timerEvent( QTimerEvent* event )
{
    switch( event->timerId() )
    {
        case timer1:
            // ежесекундное действие
            break;

        case timer2:
            // ежеминутное действие
            break;

        default:
            break;
    }
}
```