Объектная модель

Объектная модель Qt подразумевает, что все построено на объектах. Фактически, класс QObject— основной, базовый класс. Подавляющее большинство классов Qt являются его наследниками. Классы, имеющие сигналы и слоты, должны быть унаследованы от этого класса.

Класс QObject содержит в себе поддержку:

♦ сигналов и слотов (signal/slot);

♦ механизма объединения объектов в иерархии;

♦ метаобъектной информации;

♦ приведения типов;

♦ свойств.

Сигналы и слоты — это средства, позволяющие эффективно производить обмен информацией о событиях, вырабатываемых объектами.

Механизм объединения объектов в иерархические структуры позволяет резко сократить временные затраты при разработке приложений, не заботясь об освобождении памяти создаваемых объектов, поскольку объекты-предки сами отвечают за уничтожение своих потомков.

Метаобъектная информация включает в себя информацию о наследовании классов, что позволяет определять, являются ли классы непосредственными наследниками, а также узнать имя класса.

Приведение типов. Для приведения типов Qt предоставляет шаблонную функцию qobject_cast<T>(), базирующуюся на метаинформации, создаваемой метаобъектным компилятором MOC для классов, унаследованных от QObject.

Свойства — это поля, для которых обязательно должны существовать методы чтения. С их помощью можно получать доступ к атрибутам объектов извне. Свойства также широко задействованы в визуальной среде разработки пользовательского интерфейса Qt Designer. Задается свойство использованием макроса q_property. Определение свойства в общем виде выглядит следующим образом:

```
Q_PROPERTY(type name
           READ getFunction [WRITE setFunction]
           [RESET resetFunction]
           [DESIGNABLE bool]
           [SCRIPTABLE bool]
```

[STORED bool]

)

Первыми задаются тип и имя свойства, вторым — имя метода чтения (read). Определение остальных параметров не является обязательным. Третий параметр задает имя метода записи (write), четвертый — имя метода сброса значения (reset), пятый (designable) является логическим (булевым) значением, говорящим, должно ли свойство появляться в инспекторе свойств Qt Designer. Шестой параметр (scriptable) — также логическое значение, которое управляет тем, будет ли свойство доступно для языка сценариев Qt Script. Последний, седьмой параметр (stored) управляет сериализацией, то есть тем, будет ли свойство запоминаться во время сохранения объекта.

Механизм сигналов и слотов

Элементы графического интерфейса определенным образом реагируют на действия пользователя и посылают сообщения. Существует несколько вариантов такого решения.

Старая концепция функций обратного вызова (callback functions), лежащая в основе X Window System, основана на использовании обычных функций, которые должны вызываться в результате действий пользователя. Применение такой концепции значительно усложняет исходный код программы, делая его менее понятным. Кроме того, отсутствует возможность производить проверку типов возвращаемых значений, потому что во всех случаях возвращается указатель на пустой тип void. Например, для того чтобы сопоставить код с кнопкой, необходимо передать в функцию указатель на кнопку. Если пользователь нажимает на кнопку, функция будет вызвана. Сами библиотеки не проверяют, были ли аргументы, переданные в функцию, требуемого типа, а это часто является причиной сбоев. Другой недостаток функций обратного вызова заключается в том, что элементы графического интерфейса пользователя тесно связаны с функциональными частями программы и это, в свою очередь, заметно усложняет разработку классов независимо друг от друга.

Часть вины скрыта в самом языке C++. Дело в том, что C++ не создавался как средство для написания пользовательского интерфейса, и поэтому он не предоставляет соответствующей поддержки, делающей программирование в этой области более удобным. Qt расширяет язык C++ дополнительными ключевыми словами для выполнения этой задачи.

Механизм сигналов и слотов полностью замещает старую модель функций обратного вызова, он очень гибок и полностью объектно-ориентирован. Сигналы и слоты — это краеугольный концепт программирования с использованием Qt, позволяющий соединить вместе несвязанные друг с другом объекты. Каждый унаследованный от QObject класс способен отправлять и получать сигналы. Эта особенность идеально вписывается в концепцию объектной ориентации и не противоречит человеческому восприятию. Представьте себе ситуацию: у вас звонит телефон, и вы реагируете на это снятием трубки. На языке сигналов и слотов подобную ситуацию можно описать следующим образом: объект «телефон» выслал сигнал «звонок», на который объект «человек» отреагировал слотом «снятия трубки».

Использование механизма сигналов и слотов дает программисту следующие преимущества:

♦ каждый класс, унаследованный от QObject, может иметь любое количество сигналов и слотов;

♦ сообщения, посылаемые посредством сигналов, могут иметь множество аргументов любого типа;

♦ сигнал можно соединять с различным количеством слотов. Отправляемый сигнал поступит ко всем подсоединенным слотам;

♦ слот может принимать сообщения от многих сигналов, принадлежащих разным объектам;

♦ соединение сигналов и слотов можно производить в любой точке приложения;

♦ сигналы и слоты являются механизмами, обеспечивающими связь между объектами. ♦ при уничтожении объекта происходит автоматическое разъединение всех сигнально-слотовых связей. Это гарантирует, что сигналы не будут отправляться к несуществующим объектам.

Нельзя не упомянуть и о недостатках, связанных с применением сигналов и слотов:

♦ сигналы и слоты не являются частью языка C++, поэтому требуется запуск дополнительного препроцессора перед компиляцией программы;

♦ отправка сигналов происходит немного медленнее, чем обычный вызов функции, который осуществляется при использовании механизма функций обратного вызова;

♦ существует необходимость в наследовании класса QObject;

♦ в процессе компиляции не производится никаких проверок: имеется ли сигнал или слот в соответствующих классах или нет; совместимы ли сигнал и слот друг с другом и могут ли они быть соединены вместе. Об ошибке станет известно лишь тогда, когда приложение будет запущено в отладчике или на консоли. Вся эта информация выводится на консоль.

Сигналы

Сигналы (signals) окружают нас в повседневной жизни везде: звонок будильника, жест регулировщика, а также и в не повседневной— например, индейский сигнальный костер и т. д. В программировании с использованием Qt под этим понятием подразумеваются методы, которые в состоянии осуществлять пересылку сообщений. Причиной для появления сигнала может быть сообщение об изменении состояния управляющего элемента — например, перемещение ползунка. На подобные изменения присоединенный объект, отслеживающий такие сигналы, может соответственно отреагировать, что, впрочем, и не обязательно. Это очень важный момент — он говорит о том, что соединяемые объекты могут быть абсолютно независимы и реализованы отдельно друг от друга.

Сигналы определяются в классе, как и обычные методы, только без реализации. С точки зрения программиста они являются лишь прототипами методов, содержащихся в заголовочном файле определения класса. Методы сигналов не должны возвращать каких-либо значений, и поэтому перед именем метода всегда должен стоять возвращаемый параметр void.

Пример сигнала:

```
class MySignal {
Q_OBJECT
signals:
        void dolt();
};
```

Выслать сигнал можно при помощи ключевого слова emit. Ввиду того, что сигналы играют роль вызывающих методов, конструкция отправки сигнала emit doit () приведет к обычному вызову метода doit ().Чтобы иметь возможность отослать сигнал программно из объекта этого класса, следует добавить метод sendsignal(), вызов которого заставит объект класса MySignal отправлять сигнал doit ().

Реализация сигнала

```
class MySignal {
Q_OBJECT
public:
void sendSignal()
{
        emit dolt ();
}
signals:
        void dolt();
};
```

Сигналы также имеют возможность высылать информацию, передаваемую в параметре. Например, если возникла необходимость передать в сигнале строку текста, то можно реализовать это, как показано ниже.

Реализация сигнала с параметром

```
class MySignal : public QObject {
Q_OBJECT
public:
void sendSignal()
{
        emit sendString("Information");
}
signals:
        void sendString(const QString);
};
```

Слоты

Слоты (slots) — это методы, которые присоединяются к сигналам. По сути, они являются обычными методами. Самое большое их отличие состоит в возможности принимать сигналы. Как и обычные методы, они определяются в классе как public, private или protected. Если необходимо сделать так, чтобы слот мог соединяться только с сигналами сторонних объектов, но не вызываться как обычный метод со стороны, то тогда слот нужно объявить как protected или private. Во всех других случаях

объявляйте их как public. В объявлениях перед каждой группой слотов должно стоять соответственно: private slots:, protected slots: или public slots:.

Правда, есть небольшие ограничения, отличающие обычные методы от слотов. В слотах нельзя использовать параметры по умолчанию— например: slotMethod(int n = 8), или определять слоты как static.

Реализация слота

```
class MySlot : public QObject {
Q_OBJECT
public:
        MySlot();
public slots:
        void slot()
        {
                qDebug() << "I'm a slot";
        }
};
```

Соединение объектов

Соединение объектов осуществляется при помощи статического метода connect (), который определен в классе QObject. В общем виде вызов метода connect () выглядит следующим образом:

QObject::connect(const QObject* sender, const char* signal, const QObject* receiver, const char* slot, Qt::ConnectionType type = Qt:Autoconnection);

Ему передаются пять следующих параметров:

♦ sender — указатель на объект, отправляющий сигнал;

♦ signal — это сигнал, с которым осуществляется соединение. Прототип (имя и аргументы) метода сигнала должен быть заключен в специальный макрос signal (method ());

♦ receiver — указатель на объект, который имеет слот для обработки сигнала;

♦ slot — слот, который вызывается при получении сигнала. Прототип слота должен быть заключен в специальный макрос slot (method ());

♦ type — управляет режимом обработки. Имеется три возможных значения:

Qt::DirectConnection— сигнал обрабатывается сразу вызовом соответствующего метода слота, Qt::QueuedConnection— сигнал преобразуется в событие и ставится в общую очередь для обработки, Qt::Autoconnection— это автоматический режим, который действует следующим образом: если отсылающий сигнал объект находится в одном потоке с принимающим его объектом, то устанавливается режим Qt:: DirectConnection, В противном случае — режим Qt::QueuedConnection. Этот режим (Qt::Autoconnection) определен в методе connection () по умолчанию. Вам вряд ли придется изменять режимы «вручную», но полезно знать, что такая возможность есть.

Окна программы, показанные на рис. 1, демонстрируют механизм сигналов и слотов в действии. В этом примере создается приложение, в первом окне которого (справа) находится кнопка нажатия, а во втором (слева)— виджет надписи. При щелчке в правом окне на кнопке ADD (Добавить) происходит увеличение отображаемого в левом окне значения на единицу. Как только значение достигнет пяти, произойдет выход из приложения.

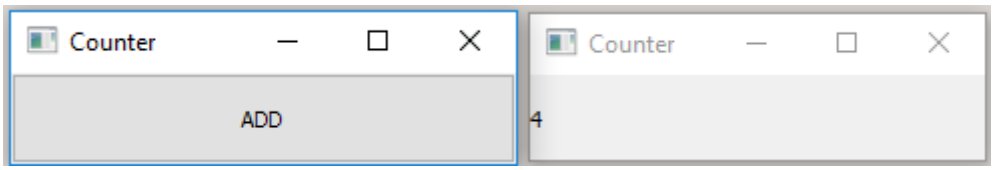

Рис. 1. Программа-счетчик. Демонстрация работы механизма сигналов и слотов

Файл Counter.pro

```
TEMPLATE = app
QT += widgets
windows:TARGET = ../Counter
HEADERS += Counter.h
SOURCES += Counter.cpp main.cpp
```

Файл main.cpp

```
#include <QtWidgets>
#include "Counter.h"

int main (int argc, char** argv)
{
   QApplication app(argc, argv);
   QLabel      lbl("0");
```

```
    QPushButton cmd("ADD");
    Counter     counter;
    lbl.show();
    cmd.show();
    QObject::connect(&cmd, SIGNAL(clicked()),
                     &counter, SLOT(slotInc())
                    );
    QObject::connect(&counter, SIGNAL(counterChanged(int)),
                     &lbl, SLOT(setNum(int))
                    );
    QObject::connect(&counter, SIGNAL(goodbye()),
                     &app, SLOT(quit())
                    );
    return app.exec();
}
```

В основной программе приложения создается объект надписи lbl, нажимающаяся кнопка cmd и объект счетчика counter (описание которого приведено ниже). Далее сигнал clicked() соединяется со слотом slotinc(). При каждом нажатии на кнопку вызывается метод slotinc(), увеличивающий значение счетчика на 1. Он должен быть в состоянии сообщать о подобных изменениях, чтобы элемент надписи отображал всегда только действующее значение. Для этого сигнал counterChanged (int), передающий в параметре актуальное значение счетчика, соединяется со слотом setNum(int), способным принимать это значение.

Наконец, сигнал goodbye(), символизирующий конец работы счетчика, соединяется со слотом объекта приложения quit(), который осуществляет завершение работы приложения, после нажатия кнопки ADD в пятый раз. Наше приложение состоит из двух окон, и после закрытия последнего окна его работа автоматически завершится.

Файл Counter.h

```
#pragma once
#include <QObject>
class Counter : public QObject {
   Q_OBJECT
private:
```

```
    int m_nValue;
public:
    Counter();

public slots:
    void slotInc();

signals:
    void goodbye       (   );
    void counterChanged(int);
};
```

Файл counter.cpp

```
#include "Counter.h"
Counter::Counter() : QObject()
                   , m_nValue(0)
{
}
void Counter::slotInc()
{
    emit counterChanged(++m_nValue);

    if (m_nValue == 5) {
        emit goodbye();
    }
}
```

Выводы:

Сигналы и слоты могут быть соединены друг с другом, причем сигнал может быть соединен с большим количеством слотов. Слот, в свою очередь, тоже может быть соединен со многими сигналами. Методы сигналов должны быть обозначены в определении класса специальным словом signals, а слоты — slots. При этом слоты являются обыкновенными методами языка C++ и в их определении могут присутствовать модификаторы public, protected, private. Отправка сигнала

производится при помощи ключевого слова emit. Класс, содержащий сигналы и слоты, должен быть унаследован от класса QObject или от класса, унаследованного от этого класса. Соединение объектов производится при помощи статического метода QObject::connect().

Задания для самостоятельной работы:

1. Изменить пример таким образом, чтобы при нажатии на кнопку происходило уменьшение значения на 2. Если достигает -10, то выход из программы.
2. Добавить еще одну кнопку, которая увеличивает на 2, если достигло 10, то выход из программы.
3. Написать приложение с кнопкой и текстовым полем QLineEdit. При нажатии на кнопку текст, введенный в поле, должен появиться в заголовке программы.

Примечания: QLineEdit имеет метод text(), которое возвращает текст в нем. Для установки заголовки необходимо написать this->setWindowTitle (QString).