Лекция 04. Сигналы и слоты

Читать в книге Шлее по Qt 5.10: гл. 2

В Windows API или библиотеке классов MFC (Visual Studio) для того, чтобы сопоставить программный код с интерфейсным элементом приложения, например, с кнопкой, необходимо передать в некую функцию-обработчик указатель на эту кнопку. Элементы графического интерфейса пользователя оказываются тесно связаны с функциональными частями программы. Для обеспечения связей сообщений и методов их обработки при этом часто используются макросы — карты сообщений.
Схожий, но усовершенствованный принцип управления используется и в Qt.
Так, препроцессор Qt вставляет дополнительную информацию на место метки Q_OBJECT в описании класса. Внедрять макрос в определение класса имеет смысл в тех случаях, когда созданный класс использует механизм сигналов и слотов или если ему необходима информация о свойствах.
В Qt реализована концепция *функций обратного вызова* (callback functions) - в результате действий пользователя вызываются обычные методы класса типа void.
Однако в Qt используется альтернативный callback-функциям механизм обмена сообщениями - сигналы и слоты.

Механизм сигналов и слотов основан на следующих принципах:

- 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. Связь также может выполняться между объектами, которые находятся в различных потоках;
- при уничтожении объекта происходит автоматическое разъединение всех сигнально-слотовых связей. Это гарантирует, что сигналы не будут отправляться к несуществующим объектам.

Особенности работы механизма сигналов и слотов следующие:

- сигналы и слоты не являются частью языка C++, поэтому требуется запуск дополнительного препроцессора перед компиляцией программы;
- отправка сигналов происходит медленнее, чем обычный вызов функции, который производится 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 можно будет узнать лишь тогда, когда приложение будет запущено. Вся эта информация выводится в консоль приложения, поэтому, чтобы увидеть её при отладке проекта в Windows, в проектном файле нужно в секции CONFIG добавить опцию console.

Сигналы - это методы, которые в состоянии осуществлять пересылку сообщений.
Сигналы определяются в классе, как обычные методы, но без реализации. Они являются прототипами методов, содержащихся в заголовочном файле определения класса. Всю дальнейшую заботу о реализации кода для этих методов берет на себя препроцессор. Методы сигналов не должны возвращать каких-либо значений, поэтому перед именем метода всегда должно стоять void.

Сигнал не обязательно соединять со слотом. Если соединения не произошло, то сигнал просто не будет обрабатываться. Подобное разделение отправляющих и получающих объектов исключает возможность того, что один из подсоединенных слотов каким-то образом сможет помешать объекту, отправившему сигналы.
Стандартная библиотека Qt предоставляет большое количество уже готовых сигналов для существующих элементов управления. В основном, для решения поставленных задач хватает этих сигналов, но иногда возникает необходимость реализации новых сигналов в своих классах.

Пример 4.1. Простое приложение с классами MySignal и MySlot (MySignal.zip).

```
#include <QObject>
class MySignal : public QObject {
 Q_OBJECT
 //...
public:
 MySignal();
signals:
  void doIt();
 //...
};
```

Препроцессор обеспечит примерно такую реализацию сигнала:

```
void MySignal::doIt() {
 QMetaObject::activate(this, &staticMetaObject, 0, 0);
}
```

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(). Сигналы могут отправляться из классов, которые их содержат. Например, в листинге выше сигнал doIt() может отсылаться только объектами класса MySignal, и никакими другими. Чтобы иметь возможность отослать сигнал программно из объекта этого класса, следует добавить метод sendSignal(), вызов которого заставит объект класса MySignal отправлять сигнал doIt():

```
class MySignal : public QObject {
 Q_OBJECT
 //...
public:
 MySignal();
 void sendSignal() {
  emit doIt();
 }
signals:
  void doIt();
 //...
};
```

Сигналы также имеют возможность высылать информацию, передаваемую в аргументе функции.

```
#include <QObject>

class MySignal : public QObject {
 Q_OBJECT
 //...
public:
 MySignal();
 void sendSignal() {
  emit sendString("Information");
  emit doIt();
 }

signals:
 void doIt();
 void sendString(const QString&);
};
```

Слоты (slots) — это методы, которые присоединяются к сигналам. По сути, они являются обычными методами. Основное их отличие состоит в возможности принимать сигналы. Как и обычные методы, они определяются в классе с атрибутами доступа public, private или protected. Соответственно, перед каждой группой слотов должно стоять одно из ключевых слов private slots: protected slots: или public slots:

В слотах нельзя использовать параметры по умолчанию, например slotMethod (int n = 8) или определять слоты как static.

Классы библиотеки содержат целый ряд уже реализованных слотов. Но определение слотов для своих классов — это частая процедура.

```
#include <QObject>
#include <QtCore>

class MySlot : public QObject {
 Q_OBJECT
 public:
  MySlot();
 public slots:
  void slot() {
   qDebug() << sender()->objectName();
  }
};
```

Внутри слота вызовом метода sender() можно узнать, от какого объекта был выслан сигнал. Он возвращает указатель на объект типа QObject. Например, в этом случае в консоль будет выведено имя объекта, выславшего сигнал.
Примечание. Это сработает, только если вызывался метод sender()->setObjectName (QString);, так как свойство по умолчанию пусто.
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 аргументов:
1) sender — указатель на объект, отправляющий сигнал;
2) signal — это сигнал, с которым осуществляется соединение. Прототип (имя и аргументы) метода сигнала должен быть заключен в специальный макрос SIGNAL(method());
3) receiver — указатель на объект, который имеет слот для обработки сигнала;
4) slot — слот, который вызывается при получении сигнала. Прототип слота должен быть заключен в специальном макросе SLOT(method());
5) 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.

Как может быть осуществлено соединение объектов в программе:

```
int main() {
 QObject::connect(pSender, SIGNAL(signalMethod()),pReceiver, SLOT(slotMethod()));
 return 0;
}
```

Если вызов происходит из класса, унаследованного от QObject, тогда префикс QObject:: можно опустить:

```
MyClass::MyClass() : QObject() {
 connect(pSender, SIGNAL(signalMethod()),pReceiver, SLOT(slotMethod()));
}
```

В случае если слот содержится в классе, из которого производится соединение, то можно воспользоваться сокращенной формой метода connect(), опустив третий параметр

(pReceiver), указывающий на объект-получатель. Другими словами, если в качестве объекта-получателя должен стоять указатель this, его можно просто не указывать:

```
MyClass::MyClass() : QObject() {
 connect(pSender, SIGNAL(signalMethod()), SLOT(slot()));
}

void MyClass::slot() {
 qDebug() << "I'm a slot";
}
```

Иногда возникают ситуации, когда объект не обрабатывает сигнал, а просто передает его дальше. Для этого необязательно определять слот, который в ответ на получение сигнала (при помощи emit) отсылает свой собственный. Можно просто соединить сигналы друг с другом. Отправляемый сигнал должен содержаться в определении класса:

```
MyClass::MyClass() : QObject() {
 connect(pSender, SIGNAL(signalMethod()), SIGNAL(mySignal()));
}
```

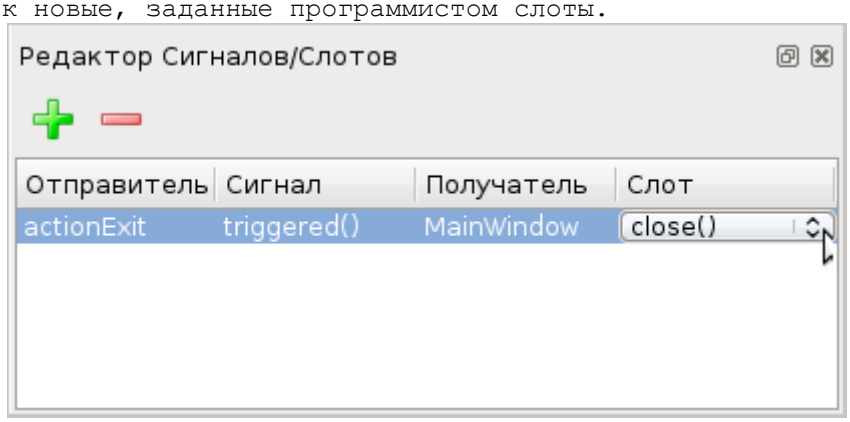
Отправку сигналов заблокировать можно на некоторое время, вызвав метод blockSignals() с параметром true. Объект будет "молчать", пока блокировка не будет снята тем же методом blockSignals() с параметром false.
При помощи метода signalsBlocked() можно узнать текущее состояние блокировки сигналов.

Параметры в слот передаются из сигнала, если количество, порядок и типы этих параметров в сигнале и слоте совпадают (или в слоте их может быть меньше).

Приведём код main.cpp для проекта с подключёнными классами MySignal и MySlot:

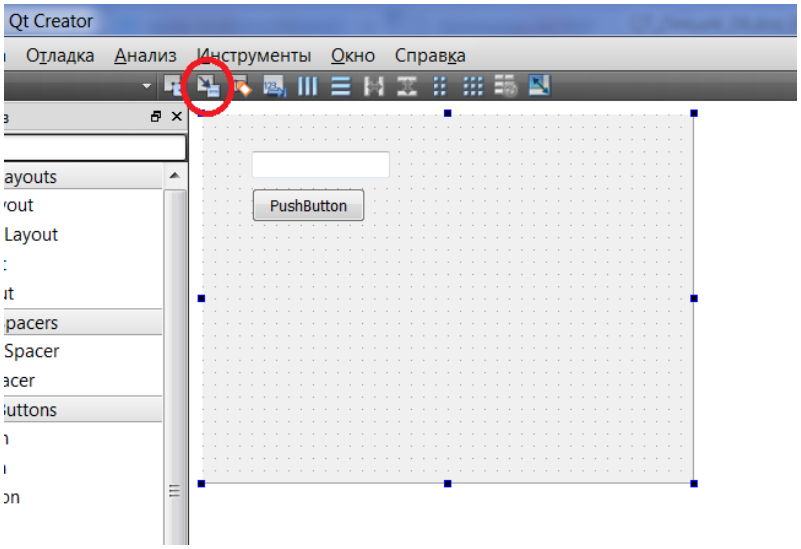
```
#include "mysignal.h"
#include "myslot.h"

#include <QObject>
#include <QApplication>

int main(int argc, char *argv[]) {
  QApplication a(argc, argv);
  MySignal w;
  MySlot s;
  QObject::connect(&w, SIGNAL(doIt()), &s, SLOT(slot()));
  w.sendSignal();
  return 0;
}
```

Соединять сигналы со слотами не обязательно программно. В режиме дизайна формы нажмите клавишу F4 для доступа к интерфейсу управления сигналами и слотами. Там же можно добавить в список новые, заданные программистом слоты.



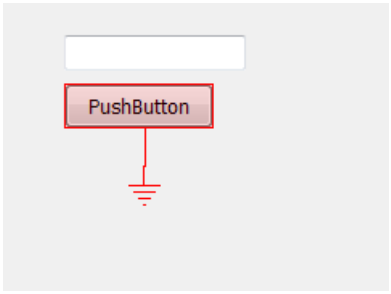
встроенный редактор сигналов и слотов

Проиллюстрируем соединение сигналов со слотами на примере обработки текстового поля QLineEdit и кнопки QPushButton, размещённых на форме виджета:



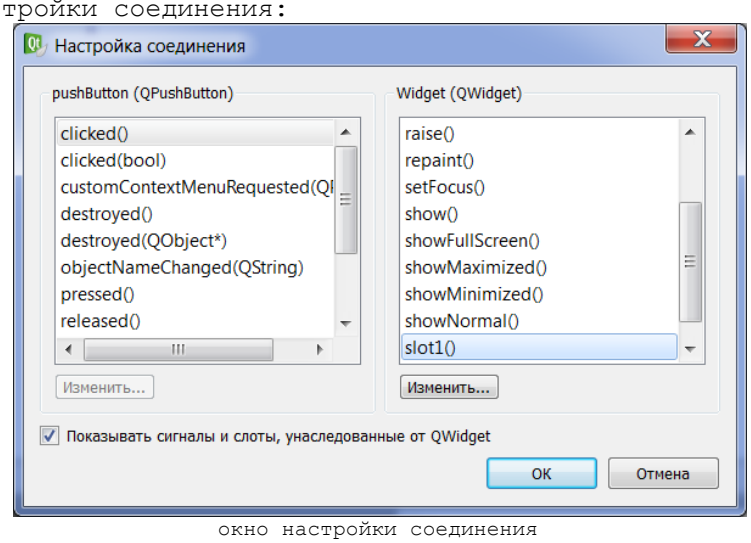
вид формы в режиме дизайна

Нажмём в режиме дизайна формы клавишу F4 или обведённую на рисунке кнопку "Изменение сигналов/слотов", затем зажмём левую кнопку мыши на поверхности PushButton и протянем красную линию в окно виджета:



вызов настройки соединения

После отпускания кнопки мыши появилось окно "Настройка соединения", слева выберем сигнал clicked(), а справа нажмём кнопку изменить, затем в новом окне Сигналы/Слоты кнопку "+" под списком слотов. К виджету добавился слот slot1(), после нажатия OK он появился в окне настройки соединения:



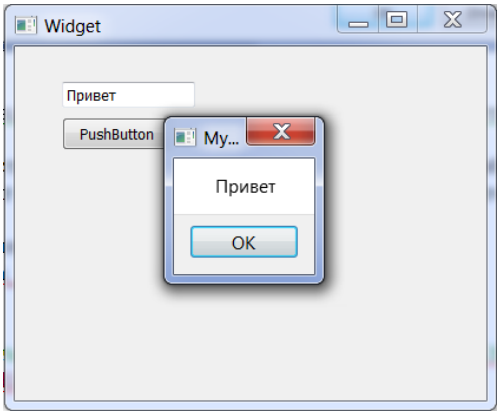
окно настройки соединения

После нажатия OK связь создана и отображена на форме, вернуться к обычном виду можно нажатием клавиши F3.
Пустая функция-слот создастся только после того, как мы нажмём правой кнопкой мыши на PushButton и выберем пункт меню "Перейти к слоту...", а затем сигнал clicked(), для которого создавали слот.
В добавленной в модуль функции можно писать код, например:

```
void Widget::on_pushButton_clicked() {
 QMessageBox msg;
 //Не забудьте добавить #include <QMessageBox> или
 //#include <QtWidgets> в widget.h!
```

```
 msg.setText(ui->lineEdit->text());
 msg.exec();
}
```

Приложение готово к работе, по нажатию кнопки выполняется этот код:



приложение в работе

Существенно и то, что в пятой версии Qt стало можно применять запись соединения, основанную на указателях.
Запись с макросами:

```
connect(button, SIGNAL(clicked()), this, SLOT(slotButton()));
```

Запись на основе указателей:

```
connect(button, &QPushButton::clicked, this, &MainWindow::slotButton);
```

Например:

```
QObject::connect(&w, &MySignal::doIt, &s, &MySlot::slot);
```

Преимущество второго варианта заключается в том, что имеется возможность определить несоответствие сигнатур и неверное наименование слота или сигнала ещё на стадии компиляции проекта, а не в процессе тестирования приложения.

Пример 4.2. Проект Counter (класс-наследник QObject, для добавления щёлкнуть правой кнопкой мыши на выделенном жирным заголовке проекта, Добавить новый…, Класс C++, указать базовый класс QObject).
В этом проекте демонстрируется программное соединение и рассоединение сигналов и слотов.

Файл counter.h

```
#ifndef COUNTER_H
#define COUNTER_H

#include <QObject>
#include <QLabel>
#include <QPushButton>

class Counter : public QObject {
    Q_OBJECT
private:
 int Value;
 bool Connected;
public:
 Counter(QObject *parent=0);
 QLabel lbl;
 QPushButton cmd,cmd2;
public slots:
 void slotInc();
 void disconnector();
signals:
 void goodbye ();
 void counterChanged(int);
};

#endif // COUNTER_H
```

Файл counter.cpp

```
#include "counter.h"

Counter::Counter (QObject *parent) :  QObject(parent), Value(0) {
 QObject::connect(&cmd2, SIGNAL(clicked()),this, SLOT(disconnector()) );
 this->Connected = false;
 this->disconnector();
}

void Counter::slotInc() {
 if (this->Connected==true) {
  emit counterChanged(++this->Value);
  if (this->Value == 10) { emit goodbye(); } //ограничиваемся 10 нажатиями
 }
}

void Counter::disconnector() {
 if (this->Connected == true) {
  QObject::disconnect(&cmd, SIGNAL(clicked()),this, SLOT(slotInc()) );
  QObject::disconnect(this, SIGNAL(counterChanged(int)), &lbl, SLOT(setNum(int)));
  this->Connected = false;
  cmd2.setText("CONNECT");
 }
 else {
  QObject::connect(&cmd, SIGNAL(clicked()),this, SLOT(slotInc()) );
  QObject::connect(this, SIGNAL(counterChanged(int)), &lbl,
    SLOT(setNum(int)), Qt::DirectConnection );
  this->Connected = true;
  cmd2.setText("DISCONNECT");
 }
}
```

Файл main.cpp

```
#include <QApplication>
#include "counter.h"

int main(int argc, char *argv[]) {
 QApplication a(argc, argv);
 Counter counter;
 counter.lbl.setText("0");
 counter.lbl.move(100,100);
 counter.cmd.setText("ADD");
 counter.cmd.move(100,200);
 counter.cmd2.setText("DISCONNECT");
 counter.cmd2.move(100,300);

 counter.lbl.show();
 counter.cmd.show();
 counter.cmd2.show();
 QObject::connect(&counter, SIGNAL(goodbye()), &a, SLOT(quit()) );
 return a.exec();
}
```

Задание к лабораторной работе 4: добавить в проект из лабораторных 2-3 отправку приложением сигнала (например, "Вычислено"), который обрабатывается виджетом, логирующим сделанные вычисления (например, "История вычислений").