

Разработка на C++

Слоты и сигналы

План

- Что такое макросы в QT?
- Для чего используют макросы?
- Понятие сигналов и слотов в Qt.
- Создание пользовательских сигналов и слотов.
- Соединение сигналов и слотов с помощью метода connect.
- Практический пример: создание класса с кнопкой, генерирующей пользовательский сигнал.
- Обработка стандартных событий с помощью слотов и сигналов.
- Пример: реакция на события мыши и клавиатуры с использованием сигналов и слотов.

Что такое макросы в QT?

Макросы в Qt - это особые команды, которые предоставляются библиотекой Qt для упрощения и расширения кода. Они выполняются компилятором во время предварительной обработки.

Если вы когда-нибудь фотографировались на документы, вы наверняка видели такую картину: вас сфотографировали, выровняли голову по какому-то шаблону, убрали прыщи, а потом нажали какую-то кнопку — и за секунду у вас на листе шесть фотографий с логотипом фотомастерской, и всё выводится на печать. Это поработал макрос.

В программировании это работает примерно также :)

Для чего используют макросы?

Создание констант: Макросы могут определять константы, которые используются в вашем коде. Например, `Q_PI` - это макрос, который определяет значение числа Пи. *(работает на версии QT 6 и выше)*

```
#include <QtCore/qmath.h> // Включаем заголовочный файл с определением Q_PI
#include <QDebug>
int main()
{
    qreal radius = 5.0;
    qreal area = Q_PI * radius * radius; // Вычисление площади круга
    qDebug() << "Площадь круга с радиусом" << radius << "равна" << area;
    return 0;
}
```

Для чего используют макросы?

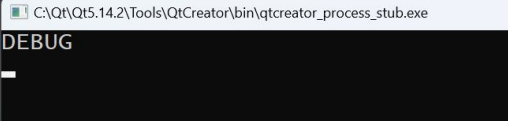
Управление компиляцией: Макросы могут включать или исключать куски кода из компиляции в зависимости от условий. Например, макрос **#ifdef** может быть использован для условной компиляции кода только при определенных условиях.

```
#include <iostream>

#define DEBUG // Определение макроса DEBUG

int main() {
    #ifdef DEBUG
        std::cout << "DEBUG" << std::endl;
    #else
        std::cout << "Default" << std::endl;
    #endif

    return 0;
}
```

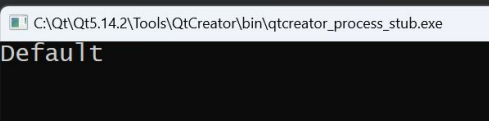
A screenshot of a Qt Creator console window. The title bar shows the file path 'C:\Qt\Qt5.14.2\Tools\QtCreator\bin\qtcreator_process_stub.exe'. The console output displays the word 'DEBUG' on a single line.

```
#include <iostream>

// #define DEBUG // Определение макроса DEBUG

int main() {
    #ifdef DEBUG
        std::cout << "DEBUG" << std::endl;
    #else
        std::cout << "Default" << std::endl;
    #endif

    return 0;
}
```

A screenshot of a Qt Creator console window. The title bar shows the file path 'C:\Qt\Qt5.14.2\Tools\QtCreator\bin\qtcreator_process_stub.exe'. The console output displays the word 'Default' on a single line.

Для чего используют макросы?

Облегчение чтения кода: Макросы могут создавать краткие и понятные имена для длинных команд или выражений. Например, макрос **Q_FOREACH** упрощает итерацию по контейнерам.

Обратите внимание, что начиная с Qt 5.0, Q_FOREACH является устаревшим в пользу стандартных C++ циклов, таких как for и range-based for, но он все еще поддерживается для обратной совместимости и может использоваться в коде, написанном для более ранних версий Qt.

Привели этот пример для ознакомления!

```
#include <QList>
#include <QDebug>

int main() {
    QList<int> numbers;
    numbers << 1 << 2 << 3 << 4 << 5;

    // Используем Q_FOREACH для итерации по списку numbers
    Q_FOREACH(int number, numbers) {
        qDebug() << "element:" << number;
    }

    return 0;
}
```

C:\Qt\Qt5.14.2\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
element: 1
element: 2
element: 3
element: 4
element: 5
```

Для чего используют макросы?

Расширение функциональности языка: макросы Qt добавляют поддержку сигналов и слотов, которых нет в стандартном C++.

Генерация кода: Макросы могут создавать код автоматически. Например, макрос **Q_OBJECT** генерирует код, который связывает класс с системой метаобъектов Qt, что позволяет использовать сигналы и слоты.

Q_PROPERTY макрос: Используется для объявления свойств (properties) класса, что позволяет использовать систему метаобъектов и интегрировать свойства с механизмами Qt, такими как сигналы и слоты, а также с графическим интерфейсом.

О том что такое *сигналы и слоты* поговорим далее:

Сигналы и слоты

Это механизм в библиотеке Qt, который позволяет объектам взаимодействовать друг с другом и реагировать на события или действия. Давайте объясним это простыми словами:

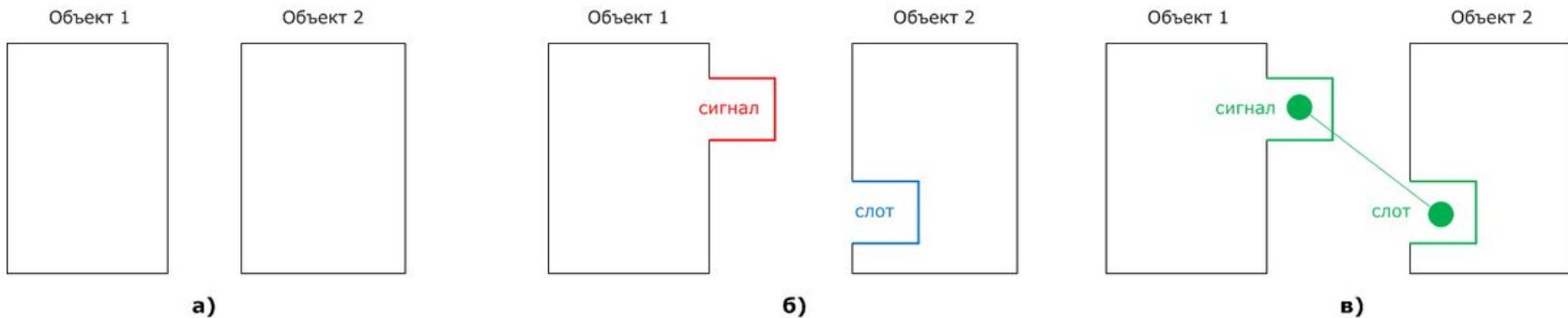
Сигналы (signals) - это как сообщения, которые объект отправляет, когда что-то происходит. *Например, объект может отправить сигнал, когда кнопка на экране была нажата, или когда данные были обновлены.*

Сигналы сообщают другим объектам, что что-то произошло, но они сами по себе не делают ничего. Они просто оповещают.

Сигналы и слоты

Слоты (Slots) - это как функции, которые готовы **реагировать на сигналы**. Когда объект получает сигнал, он может вызывать свой слот, чтобы выполнить какие-то действия.

Например, если объект получил сигнал о нажатии кнопки, его слот может открыть новое окно или изменить текст на экране.



Сигналы и слоты

Связь между объектами устанавливается следующим образом: у одного объекта должен быть **сигнал**, а у второго - **слот**. Сигнал объявляется однажды и на этом всё, ему не нужна реализация. Слот же, в общем-то, представляет собой функцию, и потому кроме объявления должен иметь реализацию, как и обычная функция.

Потому, соединив сигнал первого объекта и слот второго, мы получаем следующее: *каждый раз, когда первый объект посылает свой сигнал, второй объект принимает его в свой слот и выполняет его функцию.*

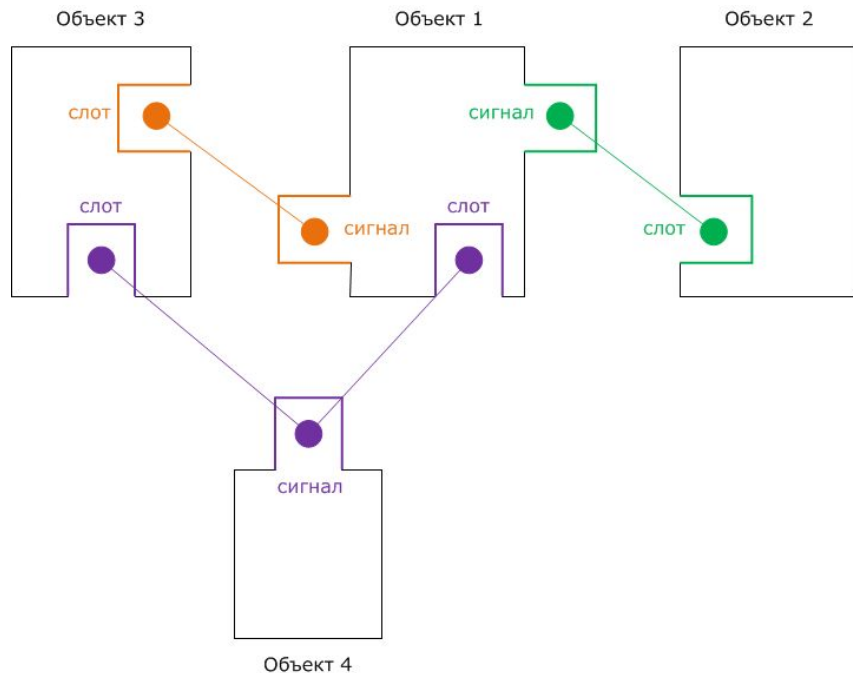
Таким образом, чтобы соединить два объекта, нужно:

- создать у одного сигнал, а у второго слот;
- соединить сигнал первого и слот второго.

Сигналы и слоты

Каждый объект может иметь больше одного сигнала и больше одного слота. Соединяться могут также более двух объектов:

Как видно, кроме очевидных соединений, при отправке Объектом 4 сигнала выполняется слот Объекта 3 и слот Объекта 1.



Пример, где на каждый сигнал есть свой слот

Представьте, что у вас есть **кофемашина**. Ваша кофемашина имеет несколько компонентов: кнопку "Включить", бак для воды, мельницу для кофейных зерен и кофейный выход. Вы хотите, чтобы все эти компоненты могли взаимодействовать друг с другом с помощью сигналов и слотов.

Вы нажимаете кнопку "Включить", кофемашина отправляет **сигнал** "Система включена".



При получении сигнала "Система включена", срабатывает **слот** и бак для воды начинает работу, мельница начинает измельчать зерна и кофейный выход готовится к приему кофе.



Когда бак для воды заполняется, он отправляет **сигнал** "Бак заполнен".



При получении сигнала "Бак заполнен", срабатывает **слот** и мельница и кофейный выход начинают работать.



Когда мельница измельчает кофейные зерна, она отправляет **сигнал** "Зерна измельчены".



При получении сигнала "Зерна измельчены", срабатывает **слот** и кофейный выход начинает приготовление кофе.



Когда кофе приготовлен, кофемашина отправляет **сигнал** "Кофе готов".



при получении сигнала "Кофе готов", срабатывает **слот** и вы можете получить свой кофе.

Пример одного сигнала и нескольких слотов

*Представьте, что у вас есть **чат-приложение**, и вы хотите, чтобы сообщения, отправленные одним пользователем, автоматически обновлялись и на стороне других пользователей без явной пересылки. В этом случае пользователи должны быть связаны так, чтобы при отправке сообщения одним объектом, другие объекты автоматически обновлялись.*

Сигнал: При отправке сообщения одним пользователем объект отправляет сигнал "Сообщение отправлено" со всеми деталями сообщения, такими как текст и отправитель.

Слот: Когда один пользователь генерирует сигнал "Сообщение отправлено", другие пользователи вызывают свои слоты "Обновить чат", и в результате чаты всех объектов автоматически обновляются с новыми сообщениями.



Что такое Q_OBJECT?

Q_OBJECT - это макрос в Qt, который добавляет специальную функциональность к вашему классу.

- Позволяет вам определять **сигналы и слоты** в вашем классе.
- Добавляет **метаинформацию** о вашем классе во время компиляции. Это позволяет например получить список его методов и свойств.
- Позволяет вашему классу **обрабатывать события**, например клики мыши или нажатия клавиш.
- Классы с Q_OBJECT могут использовать механизм **сериализации** Qt для сохранения и восстановления своего состояния в файлы.
- Упрощает интеграцию классов с элементами управления и **виджетами GUI**.
- Qt может **автоматически управлять памятью** и жизненным циклом объектов с Q_OBJECT, что снижает риск утечек памяти.

Давайте создадим приложение QT Widgets и посмотрим на файл mainwindow.h:

mainwindow.h

QT_BEGIN_NAMESPACE и QT_END_NAMESPACE - это макросы пространства имен QT. Весь код размещенный между этими макросами, помогает избежать конфликтов имен с другими частями вашего приложения.

namespace Ui { class MainWindow; } - это объявление пространства имен Ui , которое содержит класс MainWindow.

Это используется в сгенерированных Qt Designer-ом файлах интерфейса пользовательского интерфейса (UI файлы). Класс MainWindow в этом контексте представляет собой пользовательский интерфейс вашего приложения.

```
#pragma once
#include <QMainWindow>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

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

mainwindow.h

`class MainWindow : public QMainWindow` - реализуем свое окно на основе базового функционала QT.

`Q_OBJECT` - эта строка подключает нам все преимущества, о которых мы говорили на предыдущих слайдах

```
MainWindow(QWidget *parent = nullptr);  
~MainWindow();
```

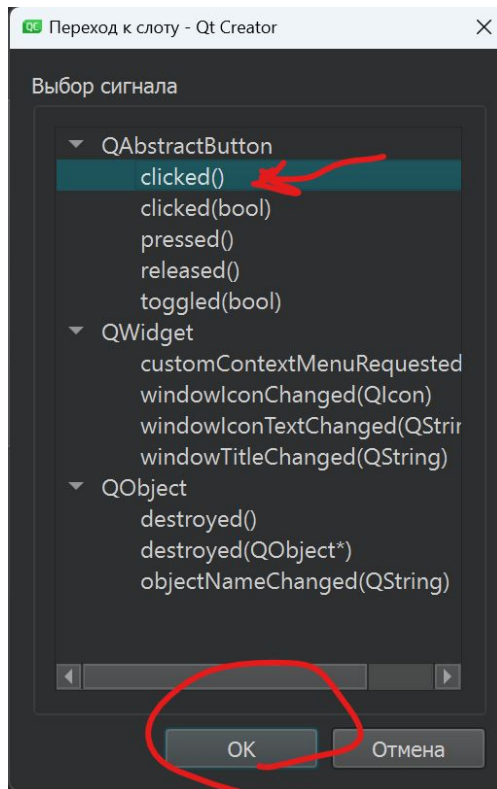
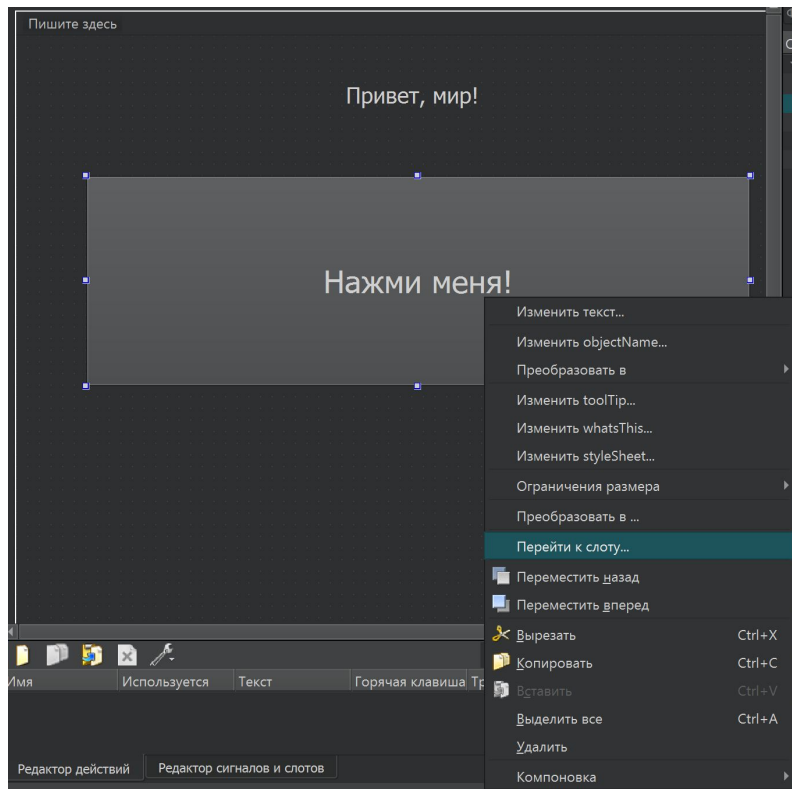
Конструктор может создавать дочерние от основного окна, в деструкторе мы уничтожаем указатели объявленные в классе

`Ui::MainWindow *ui;` - здесь хранятся все объекты размещенные в дизайнерае

```
#pragma once  
#include <QMainWindow>  
QT_BEGIN_NAMESPACE  
namespace Ui { class MainWindow; }  
QT_END_NAMESPACE
```

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


Создание слотов с помощью графического редактора



.cpp

```
void MainWindow::on_pushButton_clicked()
{
}
}
```

.h

```
private slots:
void on_pushButton_clicked();
```

Создание сигналов и слотов

Давайте напишем программу, в которой при нажатии на кнопку “Нажми меня!” выводится сообщение “Привет, мир!”.

Для этого нам нужны кнопка и текстовое поле в файле mainwindow.h:

`QPushButton* hello_button`

`QLabel* label`

Не забываем подключить библиотеки для этих виджетов

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPushButton>
#include <QLabel>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private:
    QPushButton* hello_button;
    QLabel* label;
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Создание сигналов и слотов

Для того чтобы связать кнопку и текстовое поле напишем следующий код в файле `mainwindow.cpp`

В этом файле уже будет автоматически сгенерированный код, но для работы нашей программы нужно дописать следующие блоки выделенные на скриншоте

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow) {
    ui->setupUi(this);

    hello_button = new QPushButton("Нажми меня", this); // Создание кнопки
    hello_button->setGeometry(150, 100, 100, 30); // Установка позиции и размеров кнопки

    label = new QLabel("Привет, мир!", this); // Создание метки
    label->setGeometry(150, 150, 150, 30); // Установка позиции и размеров метки
    label->hide(); // Делаем виджет невидимым

    connect(hello_button, &QPushButton::clicked, label, &QLabel::show);
}

MainWindow::~MainWindow() {
    delete ui;
    delete hello_button;
    delete label;
}
```

Создание сигналов и слотов

Давайте разберем что делает следующая строка:

```
connect(hello_button, &QPushButton::clicked, label, &QLabel::show);
```

Метод connect предназначен для связи сигнала со слотом между объектами, в данном случае первый и третий аргумент это объекты, второй это сигнал, а третий это слот.

`&QPushButton::clicked` - это сигнал, который отправляется если пользователь нажимает на кнопку

`&QLabel::show` - это слот, который показывает текстовое поле, которое мы заранее спрятали методом `hide()`

Складывается следующая картина:

```
connect (источник_сигнала, ожидаемый_сигнал, объект_со_слотом, слот_для_этого_сигнала)
```

пример:

```
connect (кнопка,          клик,          текстовое поле,      показать текст      )
```

Создание сигналов и слотов

```
#include "mainwindow.h"
#include <QPushButton>
#include <QApplication>
#include <QDebug>
#include <QTextCodec> // Добавлен этот заголовочный файл для работы с кодировкой

int main(int argc, char *argv[]){
    QApplication a(argc, argv);
    QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8")); // Устанавливаем кодировку
    MainWindow w;

    QPushButton *button = new QPushButton("Нажми меня!", &w);
    button->setGeometry(10, 10, 150, 30);

    QObject::connect(button, &QPushButton::clicked, &w, [=]() {
        qDebug() << "Кнопка была нажата!";
        //тут может быть ещё код
    });

    w.show();
    return a.exec();
}
```

Создание сигналов и слотов

QObject::connect: Это статическая функция класса QObject, предоставляемая библиотекой Qt для создания соединений между сигналами и слотами (или функциями). Сигналы и слоты - это основной механизм взаимодействия между объектами в Qt.

button: Это указатель или объект, представляющий кнопку, с которой мы хотим установить соединение.

&QPushButton::clicked: Это сигнал, который мы хотим связать с какой-то функцией или слотом. В данном случае, сигнал "clicked" - это сигнал, который генерируется при щелчке на кнопке. QPushButton::clicked указывает на этот сигнал.

&w: Это объект w, с которым мы хотим связать сигнал. Обычно w - это объект окна (например, главного окна приложения), с которым мы хотим взаимодействовать.

[=](): Это лямбда-выражение, которое будет выполнено при возникновении сигнала "clicked" от кнопки. В данном случае, оно выглядит как [=](), что означает, что лямбда-функция не принимает аргументов. Лямбда-функция представляет собой блок кода, который будет выполнен при каждом щелчке на кнопке.

Реакция на щелчки мыши или нажатие клавиши

```
#pragma once
#include <QMainWindow>
#include <QLabel>
#include <QPushButton>
#include <QKeyEvent>
```

```
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
```

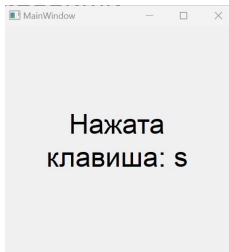
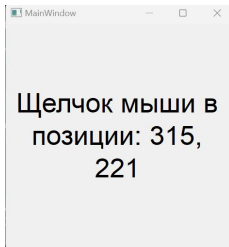
public:

```
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

```
    void keyPressEvent(QKeyEvent *event) override;
    void mousePressEvent(QMouseEvent *event) override;
```

private:

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



```
#include "mainwindow.h"
#include "ui_mainwindow.h"
```

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow){
    ui->setupUi(this);
    setMouseTracking(true);
}
```

```
MainWindow::~MainWindow(){ delete ui; }
```

```
void MainWindow::keyPressEvent(QKeyEvent *event){
    // Выводим сообщение при нажатии клавиши
    ui->label->setText(QString("Нажата клавиша: %1").arg(event->text()));
}
```

```
void MainWindow::mousePressEvent(QMouseEvent *event){
    // Выводим сообщение при щелчке мыши
    ui->label->setText(QString("Щелчок мыши в позиции: %1, %2").arg(event->x()).arg(event->y()));
}
```