

Слоты и сигналы

Дмитрий Фёдоров
Руководитель отдела разработки ПО



Проверка связи





Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включён звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти



Поставьте в чат:

-  если меня видно и слышно
-  если нет

Дмитрий Фёдоров

О спикере:

- более 10 лет в разработке авиационных систем
- возглавляет отдел разработки ПО в НИЦ «ИРТ»



Вспоминаем прошрое занятие

Вопрос: Что такое виджет Qt?



Вспоминаем прошрое занятие

Вопрос: Что такое виджет Qt?

Ответ: Элементарный объект ПИ



Вспоминаем прошрое занятие

Вопрос: Какой базовый класс для всех виджетов?



Вспоминаем прошрое занятие

Вопрос: Какой базовый класс для всех виджетов?

Ответ: QWidget



Вспоминаем прошрое занятие

Вопрос: К каким элементам относится
QLabel?



Вспоминаем прошрое занятие

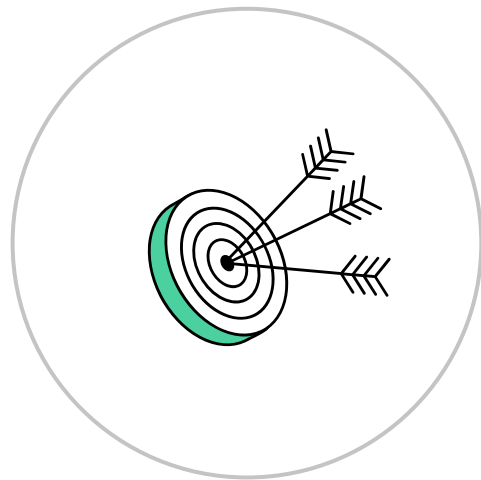
Вопрос: К каким элементам относится QLabel?

Ответ: Элементы отображения



Цели занятия

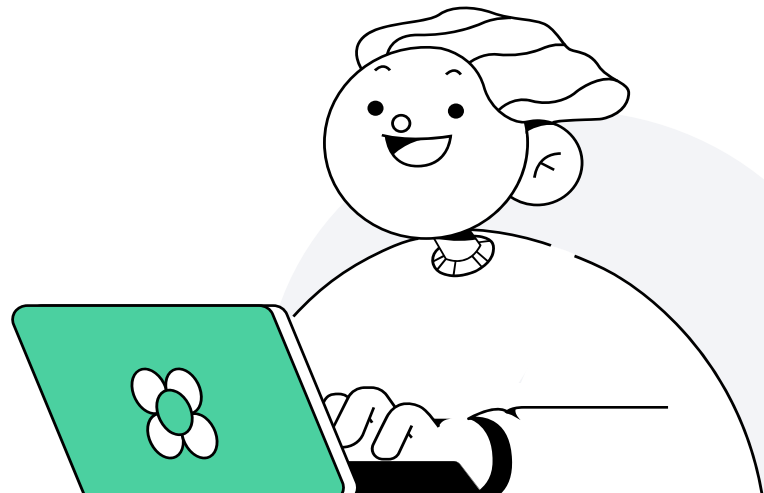
- Изучим механизм сигналов/слотов
- Узнаем как работает механизм сигналов/слотов
- Научимся использовать стандартные сигналы/слоты
- Научимся создавать собственные



План занятия

- | | | | |
|---|---|---|------------------|
| 1 | Объектная философия Qt | 7 | Итоги |
| 2 | До сигналов/слотов | 8 | Домашнее задание |
| 3 | Метаобъектный компилятор | | |
| 4 | Класс QObject | | |
| 5 | Сигналы/слоты | | |
| 6 | Соединение/разъединение сигналов/слотов | | |

*Нажми на нужный раздел для перехода



Объектная философия Qt



1

Объектная философия Qt

Один из принципов ООП заключается в том, что в приложение должно состоять из экземпляров классов(объектов) и функционировать благодаря взаимодействию между ними.

- Стандартные механизмы Qt подразумевают, что любое приложение строится именно по такому принципу.
- Приложение Qt строится на связях между объектами.
- Базовый класс так и называется QObject. Подавляющее большинство классов Qt унаследованы от этого класса.

До сигналов/слотов



2

До сигналов/слотов

Необходимость отслеживания пользовательских событий, которые могут происходить спонтанно.

Основным методом обработки событий (а при программировании встроенных систем в 99% единственным) являются функции обратного вызова.

До сигналов/слотов

Необходимость отслеживания пользовательских событий, которые могут происходить спонтанно.

Основным методом обработки событий (а при программировании встроенных систем в 99% единственным) являются функции обратного вызова.

Указатель на функцию обратного вызова передается в исполняемую функцию в качестве аргумента, когда код доходит до обработки параметра выполняется код функции.

Недостатки функций обратного вызова

- Очень сложный анализ кода, код становится не читаемым

Недостатки функций обратного вызова

- Очень сложный анализ кода, код становится не читаемым
- При использовании сторонних библиотек нет гарантии, что функция ведет себя как задумано

Недостатки функций обратного вызова

- Очень сложный анализ кода, код становится не читаемым
- При использовании сторонних библиотек нет гарантии, что функция ведет себя как задумано
- Поскольку используются указатели, необходима проверка наличия функции.

Недостатки функций обратного вызова

- Очень сложный анализ кода, код становится не читаемым
- При использовании сторонних библиотек нет гарантии, что функция ведет себя как задумано
- Поскольку используются указатели, необходима проверка наличия функции.
- Функции обратного вызова нельзя переопределять.

Callback hell

```
// Constructor
function Person() {
}

Person.prototype.wakeup = function(callback){
    // do wakeup stuff
    callback();
}

Person.prototype.putOnPants = function(callback){
    // do the putting on of pants
    callback();
}

Person.prototype.putOnShirt = function(callback){
    // do the putting on of a shirt
    callback();
}

Person.prototype.putOnShoes = function(callback){
    // put those shoes on, boy
    callback();
}

Person.prototype.goToSchool = function(callback){
    // now go to school!
}

var person = new Person();

person.wakeup(function() {
    person.putOnPants(function() {
        person.putOnShirt(function() {
            person.putOnShoes(function() {
                person.goToSchool();
            });
        });
    });
});

person.wakeup(function(){
    person.putOnPants(function(){
        person.putOnShirt(function(){
            person.putOnShoes(function(){
                person.goToSchool();
            });
        });
    });
});

person.wakeup(function() {
    person.putOnPants(function() {
        person.putOnShirt(function() {
            person.putOnShoes(function() {
                person.goToSchool();
            });
        });
    });
});


person.wakeup(function() {
    person.putOnPants(function() {
        person.putOnShirt(function() {
            person.putOnShoes(function() {
                person.goToSchool();
            });
        });
    });
});

person.wakeup(function() {
    person.putOnPants(function() {
        person.putOnShirt(function() {
            person.putOnShoes(function() {
                person.goToSchool();
            });
        });
    });
});
```

Anonymous Functions stored in the callback variable

Callback hell

```
1 function hell(win) {
2   // for listener purpose
3   return function() {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                      async.eachSeries(SCRIPTS, function(src, callback) {
14                        loadScript(win, BASE_URL+src, callback);
15                      });
16                    });
17                  });
18                });
19              });
20            });
21          });
22        });
23      });
24    });
25  };
26 }
```



```
1
2 var async = require("async");
3
4 User.find(userId, function(err, user){
5   if (err) return errorHandler(err);
6   User.all({where: {id: {$in: user.friends}}}, function(err, friends) {
7     if (err) return errorHandler(err);
8     async.each(friends, function(friend, done){
9       friend.posts = [];
10      Post.all({where: {userId: {$in: friend.id}}}, function(err, posts) {
11        if (err) return errorHandler(err);
12        async.each(posts, function(post, donePosts){
13          friend.push(post);
14          Comments.all({where: post.id}, function(err, comments) {
15            if (err) donePosts(err);
16            post.comments = comments;
17            donePosts();
18          });
19        }, function(err) {
20          if (err) return errorHandler(err);
21          done();
22        });
23      });
24    }, function(err) {
25      if (err) return errorHandler(err);
26      render(user, friends);
27    });
28  }
29 });
30
```

Метаобъектный компилятор



3

Метаобъектный компилятор

- Поскольку механизм сигналов и слотов осуществляет связь между объектами в процессе выполнения приложения, то приложению необходимо знать метаинформацию об обслуживаемых объектах, например о типах данных.
- В Qt для решения этих задач используется метаобъектный компилятор (МОС), который собирает всю необходимую метаинформацию.

Метаобъектный компилятор

- МОС запускается перед стартом основного компилятора и ищет в коде ключевые маркеры (Q_OBJECT, signals, slot и т.д.)
- Результатом работы МОС является сгенерированный код на языке C++
- МОС поставляется вместе с Qt и выполняет свою работу незаметно для пользователя.

Метаобъектный компилятор

```
Q_CONSTINIT static const uint qt_meta_data_MainWindow[] = {  
  
    // content:  
    10,          // revision  
    0,           // classname  
    0,    0,    // classinfo  
    1,    14,   // methods  
    0,    0,   // properties  
    0,    0,   // enums/sets  
    0,    0,   // constructors  
    0,         // flags  
    0,         // signalCount  
  
    // slots: name, argc, parameters, tag, flags, initial metatype offsets  
    1,    1,    20,    2, 0x0a,    1 /* Public */,  
  
    // slots: parameters  
    QMetaType::Void, QMetaType::QString,    3,  
  
    0          // eod  
};
```

Класс QObject



4

Класс QObject

Основным, базовым классом обеспечивающим большинство функционала Qt является класс QObject. Чтобы использовать механизм сигналов и слотов ваши классы также должны быть унаследованы от QObject.

Класс QObject

Основным, базовым классом обеспечивающим большинство функционала Qt является класс QObject. Чтобы использовать механизм сигналов и слотов ваши классы также должны быть унаследованы от QObject.

Обязательно!

1. После открывающей скобки класса сразу должен идти макрос `Q_OBJECT`, без ; в конце. Это необходимо чтобы МОС распознал данный класс.
2. При множественном наследовании, также для правильной работы МОС класс QObject должен стоять на первом месте.
3. При множественном наследовании от класса QObject должен быть унаследован только один из базовых классов. Т.е. нельзя производить наследование от нескольких классов, которые унаследованы от QObject.

Сигналы/слоты



5

Сигналы/слоты

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

Механизм сигналов и слотов полностью замещает старую модель функций обратного вызова, он очень гибок и полностью объектно-ориентирован.

Каждый унаследованный от QObject класс способен отправлять и получать сигналы.

Преимущества механизма сигналов/слотов

- 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. Более того, эта связь 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.

Сигналы

В программировании с использованием Qt под понятием сигнала подразумеваются методы, которые в состоянии осуществлять пересылку сообщений между объектами.

Объект, отправляющий сигналы, ничего не знает об объектах принимающих и обрабатывающих сигналы.

Соединяемые объекты могут быть абсолютно независимы и реализованы отдельно друг от друга.

Сигналы

Сигналы определяются в классе, как и обычные методы, только без реализации.

Методы сигналов не должны возвращать каких-либо значений, и поэтому перед именем метода всегда должен стоять возвращаемый параметр void.

Qt предоставляет большое количество predefined сигналов, которых хватает для решения большинства задач взаимодействия с пользовательским интерфейсом.

```
#ifndef SIGNALS_H
#define SIGNALS_H

#include <QObject>

class Signals : public QObject
{
    Q_OBJECT
public:
    explicit Signals(QObject *parent = nullptr);

signals:
    void sig_Signal(QString rcv);

};

#endif // SIGNALS_H
```

Слоты

Слоты (slots) - это методы, которые присоединяются к сигналам. По сути, они являются обычными методами.

Если необходимо сделать так, чтобы слот мог соединяться только с сигналами сторонних объектов, но не вызываться как обычный метод со стороны, этот слот необходимо объявить как `protected` или `private`.

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

Создание собственных сигналов/слотов

- В заголовочном файле объекта-отправителя, в группе signals определяем сигнал.
- В заголовочном методе объекта-приемника, в группе slots определяем слот и реализуем его в cpp файле.
- При помощи метода QObject::connect соединяем сигнал со слотом.
- В cpp файле, вместе где необходимо вызвать сигнал, вызываем его ***emit signalName***

Отличие слотов

- В слотах нельзя использовать параметры по умолчанию Slot (int n = 1).
- Слоты не могут быть определены как статические.

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

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

public slots:
    void RcvSignal(QString str);

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

Соединение/разъединение сигналов/слотов



6

Соединение

Соединение объектов происходит при помощи статического метода класса QObject “connect”.

```
QObject::connect(const QObject* sender, const QMetaMethod& signal, const QObject* receiver, const QMetaMethod& slot, Qt::ConnectionType type = Qt::AutoConnection ) ;
```

- **sender** - указатель на объект, отправляющий сигнал;
- **signal** - это сигнал, с которым осуществляется соединение.
- **receiver** - указатель на объект, который имеет слот для обработки сигнала.
- **slot** - слот, который вызывается при получении сигнала.
- **type** - управляет режимом обработки.

Соединение

```
connect(sig, &Signals::sig_Signal, this, &MainWindow::RcvSignal, Qt::AutoConnection);
```

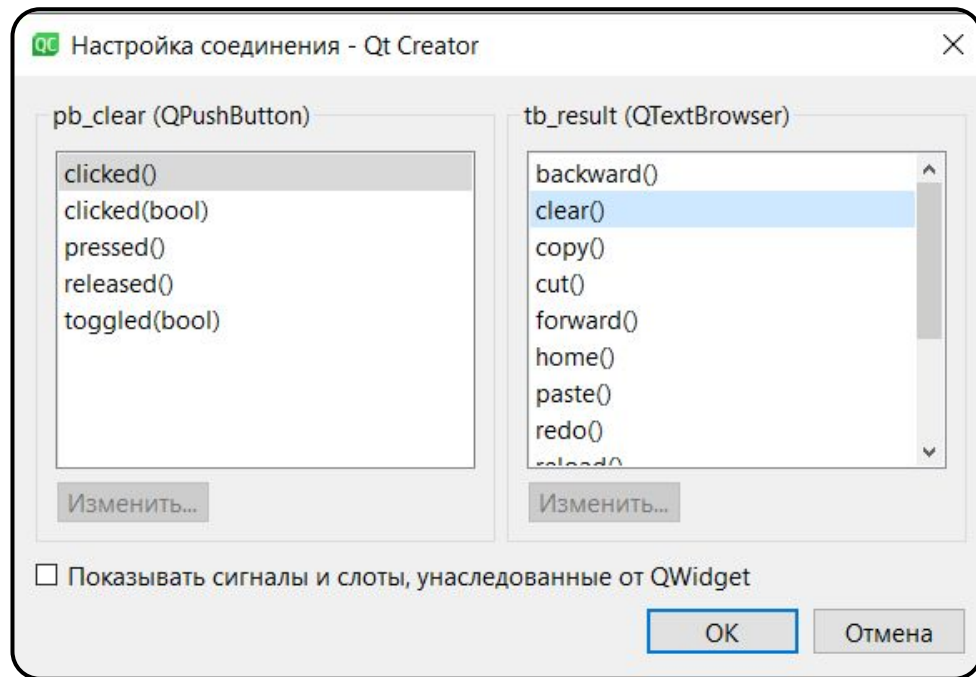
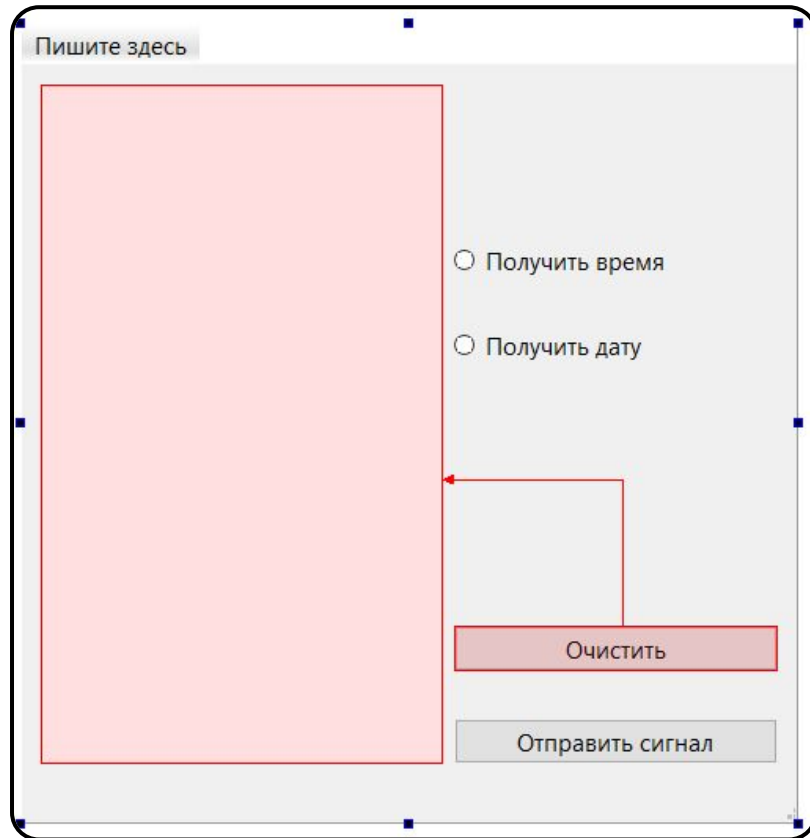
- Qt::DirectConnection – сигнал обрабатывается сразу вызовом соответствующего метода слота.
- Qt::QueuedConnection – сигнал преобразуется в событие и ставится в общую очередь для обработки.
- Qt::AutoConnection – это автоматический режим, который действует следующим образом: если отсылающий сигнал объект находится в одном потоке с принимающим его объектом, то устанавливается режим Qt:: DirectConnection, в противном случае - режим Qt::QueuedConnection.

Важно! Сколько раз будет вызван connect, столько раз будет вызван соответствующий слот!

Соединение с лямбда-функциями

```
connect(ui->pb_sendSignal, &QPushButton::clicked, this, [&]{  
    if(ui->rb_getTime->isChecked()){  
        ui->tb_result->append(QTime::currentTime().toString());  
    }  
    else if(ui->rb_getDate->isChecked()){  
        ui->tb_result->append(QDate::currentDate().toString());  
    }  
});
```

Скрытое соединение



Перегрузка сигналов

Для сокращения кода, сигналы могут быть перегружены. В таком случае необходимо использовать следующий синтаксис при соединении сигналов:

```
connect(sig, qOverload<int>(&Signals::sig_Signal), this, qOverload<int>(&MainWindow::RcvSignal));
```

Разъединение сигналов

```
QObject::disconnect(const QObject* sender, const QMetaMethod& signal, const QObject* receiver,  
const QMetaMethod& slot) ;
```

- **sender** - указатель на объект, отправляющий сигнал;
- **signal** - это сигнал, с которым осуществляется соединение.
- **receiver** - указатель на объект, который имеет слот для обработки сигнала.
- **slot** - слот, который вызывается при получении сигнала.

Итоги



Итоги занятия

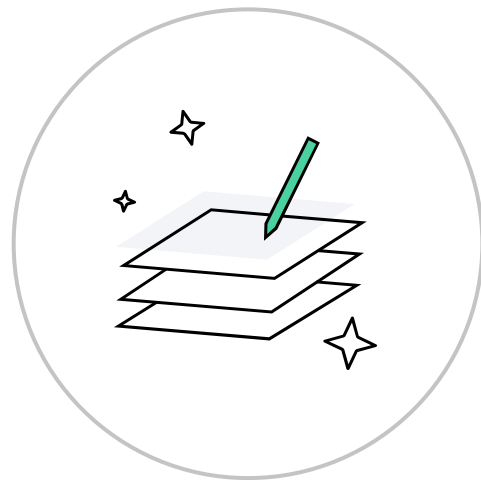
- 1 Узнали про объектную философию Qt.
- 2 Вспомнили про callback функции
- 3 Узнали что такое МОС
- 4 Узнали что такое сигналы/слоты
- 5 Научились пользоваться сигналами/слотами



Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



Дополнительные материалы

- Описание класса [QObject](#)
- Описание механизма [сигналов и слотов](#)
- Статьи на хабре как работают сигналы и слоты [часть 1](#) и [часть 2](#)
- Описание класса [QTimer](#), необходимого для домашней работы.



**Задавайте вопросы
и пишите отзыв о лекции**

