ООП

Семестр 2.
Qt. Лекция 3

Кафедра ИВТ и ПМ

2019

План

Прошлые темы

Устройство классов Qt
- QObject
- Сигналы и слоты
- Объектная иерархия

qDebug. Отладочный вывод

Outline

Сигналы и слоты

- Что такое сигнал?

Сигналы и слоты

- Что такое сигнал?
  Сигнал метод вызываемый во время события.
  Что такое слот?

Сигналы и слоты

- Что такое сигнал?
  Сигнал метод вызываемый во время события.
  Что такое слот?
  слот - метод, принимающий сигнал.

Outline

QObject

- Все классы в Qt - наследники класса QObject.
- Это виртуальный класс
- Для его использования нужно подключить модуль Qt
- Для этого нужно подключить

  ```
  QT += core
  ```

  ```
  #include <QObject>
  ```

- Вместо <QObject> можно использовать другие файлы Qt, так как классы описанные в них, включают QObject
- QObject поддерживает механизм сигналов и слотов
- Документация:http://doc.qt.io/qt-5/qobject.htmldetails

QObject

Любой класс построенный на основе QObject должен содержать макрос Q_OBJECT.

```
class MtyClass : public QObject{
    Q_OBJECT
    ...
};
```

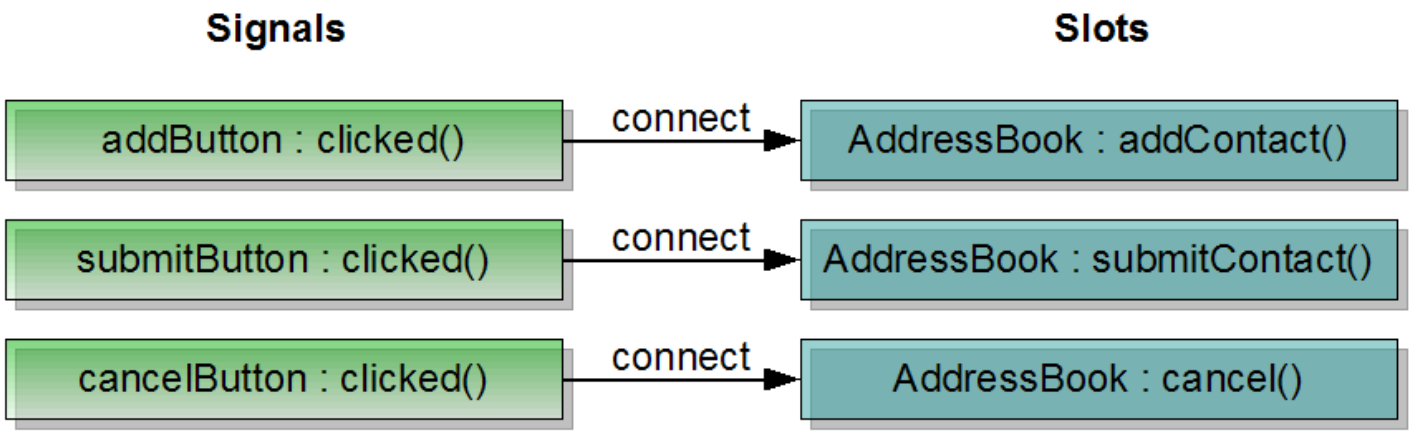
Этот макрос нужен для работы moc.

Если новый класс наследует несколько классов, включая QObject, то QObject должен приводится первым.

Сигналы и слоты

- **Сигнал** (signal) - метод вызываемый во время события.
- **Слот** (slot) - метод принимающий сигнал.

Соединяя сигналы и слоты между собой с помощью специальной функции можно добиться автоматического вызова методов одного объекта (слотов), на вызов методов другого объекта (сигналов)



Сигналы и слоты

```
class A : public QObject{
    Q_OBJECT
public:  explicit A(QObject *parent = nullptr);
signals:void my_signal();
};

class B : public QObject{
    Q_OBJECT
public: explicit B(QObject *parent = nullptr);
public slots: void my_slot(); // должен быть реализован
};

    A *a = new A();
    B *b = new B();
    QObject::connect(a, SIGNAL(my_signal()),
                     b, SLOT(my_slot()));
    a->my_signal(); //  + автоматический вызов my_slot
```

см. пример SignalsAndSlots Basic

Сигналы и слоты

- Сигналы и столы можно объявлять только в классах построенных на основе QObject
- В описании класса должен присутствовать макрос Q_OBJECT
- В конечном итоге, moc (meta object compiler) сгенерирует код на основе описанных классов, который будет обрабатывать соединения сигналов и слотов, а также следить за вызовом сигналов.
- connect соединяет методы отдельный объектов, а не классов.
- Один сигнал может быть соединён с несколькими слотами.
- Сигнал нужно объявить, но не нужно определять.
- Один слот может быть соединён с несколькими сигналами.

Синтаксис соединения сигналов и слотов

- **С использованием макросов SIGNAL и SLOT**

```
connect(sender,   SIGNAL(foo (type1,type2,  ...) ),
        receiver, SLOT  (bar (type3, type4, ...) )   )
```

sender - указатель на объект вызывающий сигнал
reviever - указатель на объект принимающий сигнал (объект со слотом)
foo - сигнал
bar - слот

Указываются только типы формальных параметров сигнала и слота.

Синтаксис соединения сигналов и слотов

Типы и количество параметров сигнала и слота могут не совпадать.

```
connect(button, SIGNAL(clicked(bool)), label, SLOT(clear()))
```

Если же типы параметров совпадают, то слот будут вызван с тем же фактическим параметром что и сигнал.

```
QObject::connect(spinBox, SIGNAL( valueChanged(int)),
                 label,     SLOT( setNum(int))  );
```

При изменении значения в числовом поле ввода spinBox будет вызван сигнал valueChanged, параметр которого содержит новое значение поля ввода.

После вызова сигнала будет вызвал присоединённый к нему слот setNum надписи (QLabel) с таким же фактическим параметром, что и при вызове присоединённого сигнала valueChanged.

Синтаксис соединения сигналов и слотов

- С использованием указателей на методы

```
QObject::connect(button, &QPushButton::pressed,
                 label, &QLabel::hide);
```

Синтаксис соединения сигналов и слотов. Ошибки

При использовании указателей на методы перегруженные методы компилятор сообщает о неоднозначности (unresolved overloaded function type>).

```
QObject::connect(spinBox, &QSpinBox::valueChanged,
                 label,   &QLabel::setNum);
```

Приведённый код призван изменить текст в Label если изменилось значение в числовом поле ввода QSpinBox. Однако из-за того, что в классе QSpinBox существуют перегруженные методы:

```
void valueChanged(int);
void valueChanged(const QString &);
```

Возникает ошибка компиляции:

```
ошибка: no matching function for call to
‘connect(QSpinBox*, <unresolved overloaded function type>,
         QLabel*,   <unresolved overloaded function type>)’
```

Синтаксис соединения сигналов и слотов. Ошибки

Для решений этой проблемы нужно явно указать на отличия методов с помощью указания их типа:

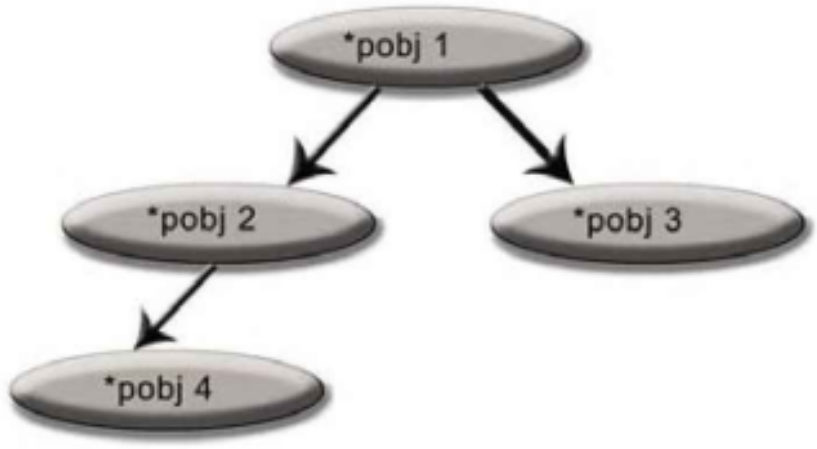
```
connect(spinBox,
 static_cast<void (QSpinBox::*)(int)> (&QSpinBox::valueChanged),
        label,
 static_cast<void (QLabel::*)(int)>   (&QLabel::setNum));
```

QObject

- Содержит метаобъектную информацию (например, объект содержит в себе название класса и информацию о наследовании)
- Механизм объединения объектов в иерархические структуры
  Объекту может быть назначен владелец (параметр parent), это избавит от ручного удаление объекта: он будет удалён владельцем, когда тот сам прекратит существование.
- ...

Объектная иерархия

```
QObject* pobj1 = new QObject;
QObject* pobj2 = new QObject(pobj1);
QObject* pobj4 = new QObject(pobj2);
QObject* pobj3 = new QObject(pobj1);
pobj2->setObjectName("the first child of pobj1");
pobj3->setObjectName("the second child of pobj1");
pobj4->setObjectName("the first child of pobj2");
```



Чтобы удалить все объекты достаточно удалить только владельца - `obj1`

Пример из книги Профессиональное программирование на C++. Макс Шлее. 2015 г

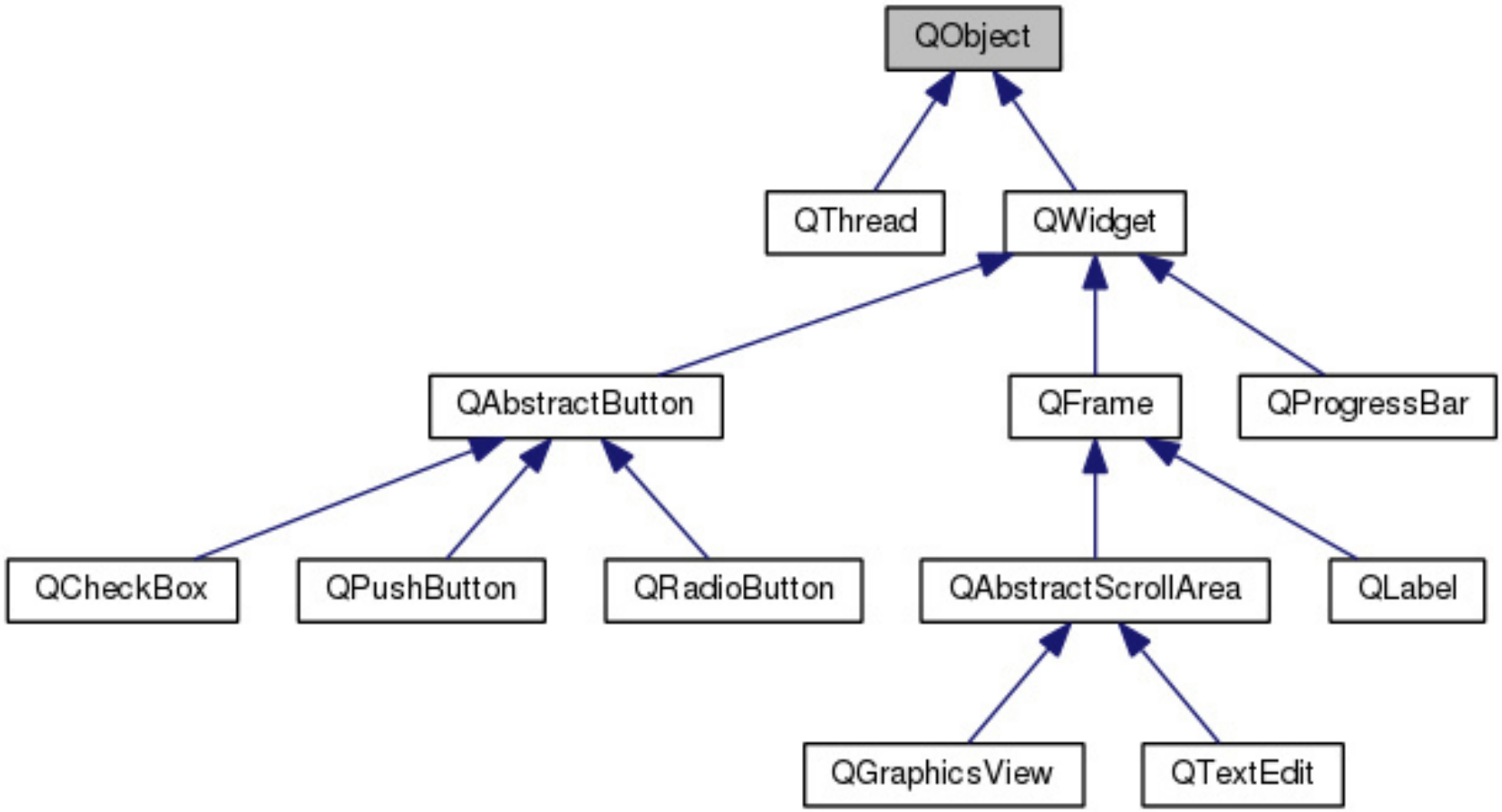
Объектная иерархия

- Организация объектов в иерархию упрощает динамическое управление памятью
- Объекты которые управляют удавлением других объектов называют родительскими (parent)
[1]
- У классов Qt есть конструктор принимающий указатель на родителя
- Следует создавать Qt объекты динамически (с использованием оператора new)
- чтобы не удалять такие объекты вручную назначать им родителя, который сам позаботится об освобождении памяти
- Чем отличается объектная иерархия от иерархии классов?

[1] не стоит путать с наследованием

Иерархия классов

Фрагмент дерева иерархии классов



Некоторые классы

- QApplication - класс взаимодействующий с ОС (обработка событий и т.п.)
- QWidget - базовый класс для элементов интерфейса (пустое окно)
- QMainWindow - основное окно программы
- QLabel - класс "Надпись"
- QSpibBox - класс "Числовое поле ввода"
- QPushButton - класс "Кнопка"
- QTextEdit - класс "Текстовое поле ввода"
- QTableWidget - класс для представления табличных данных"

Outline

Отладочный вывод

В целях изучения Qt полезно использовать так называемый отладочный вывод.
Информация будет напечатана в консоль. Для GUI приложений вывод отладчика можно просмотреть в Qt Creator.

- Для отладочного вывода используется объект класса QDebug.
- Этот объект объявлен в модуле QDebug
- Чтобы получить к нему доступ используется функция qDebug()
- Для вывода используется оператор «

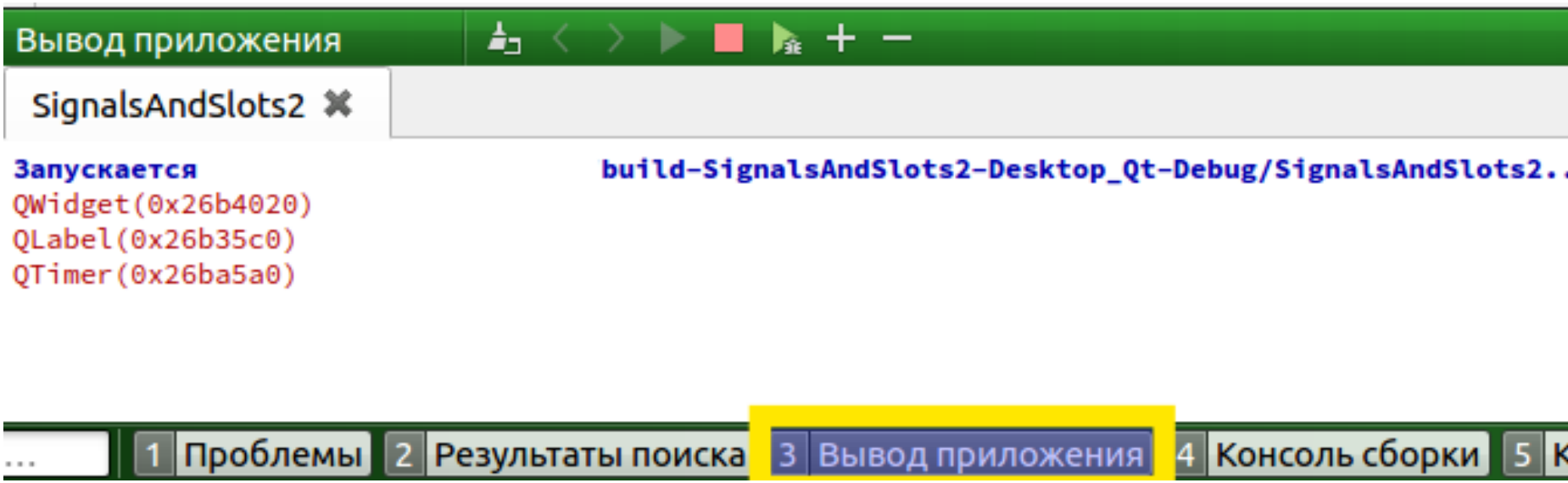
Отладочный вывод

Преимущество вывода через QDebug перд cout:

- QDebug может выводить все объекты унаследованные от QObject
- QDebug может выводить на экран тип объекта при задействованном позднем связывании.

Отладочный вывод

```
QObject * o = new QWidget();
qDebug() << o; // должен быть использован указатель
o = new QLabel();
qDebug() << o;
o = new QTimer();
qDebug() << o;
```



см. также

```
qWarning() << "Uh, oh...";

// something is so screwed we cannot continue.
// Prints a message then aborts.
qFatal( "AAAAAAAAAH!" );

// similar to the above but doesn't abort.
qCritical() << "Oh, noes!";
```

QObject

Соединение и разъединение сигналов и слотов

- connect - соединения сигнала и слота (статический метод)

```
connect(const QObject *sender,
        PointerToMemberFunction signal,
        const QObject *receiver,
        PointerToMemberFunction method)
```

- disconnect - разрыв связи между сигналом и слотом

```
disconnect(const QObject *receiver,
           const char *method = Q_NULLPTR) const
```

QObject

Конструктор

```
QObject(QObject *parent = Q_NULLPTR)
```

В конструкторе QObject можно указать ссылку на владельца данного объекта.
При уничтожении владельца, автоматически будут вызваны деструкторы всех его дочерних[2] объектов.

Это даёт возможность не заботится о уничтожении многих объектов создаваемых динамически.
Например владельцем всех элементов интерфейса будет класс главного окна.

[2] под дочерними объектами понимаются не наследники, а агрегированные объекты, т.е. объекты время жизни которых зависит от времени жизни родительского объекта

QObject

- `const QObjectList &QObject::children() const`
- `QObject *QObject::parent() const`

Так как все классы строятся на основе виртуального QObject то тип возвращаемых элементов будет определён во время выполнения программы.

О документации

В документации в описание каждого класса включены свойства (**properties**). Это в основном закрытые поля класса, доступ к которым возможен только с помощью методов.

Как правило методы возвращающие значения названы так же как и свойства, а методы устанавливающие значения начинаются с префикса get.

objectName

Свойство objectName содержит строку (QString) - имя *объекта*.

Это свойство можно использовать чтобы найти определённый объект по имени.
По умолчанию объект имеет пустое имя.

Методы доступа:

```
QString          objectName() const
void          setObjectName(const QString &name)
```

QMetaObject

В Qt каждый класс содержит метаинформацию о самом себе.

Эта метаинформация содержится в классе QMetaObject. Она включает:

- className() - имя класса
- информацию о классе предоставленную разработчиком с помощью макроса Q_CLASSINFO
- Информацию о методах (их количество, названия, ...)

Q_CLASSINFO

Данная метаинформация о классе представлена в виде пар имя-значение.
В примере приведена информация об авторе класса и ссылка на сайт.

```
class MyClass : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("Author", "Pierre Gendron")
    Q_CLASSINFO("URL", "http://www.my-organization.qc.ca")

public:
    ...
};
```

QMetaObject

```
QLabel label("Hello World!");

qDebug() << label.metaObject()->className();
qDebug() << label.metaObject()->methodCount();
```

Вывод:

QLabel
44

Ссылки и литература

1. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений. 720 с. 2010 г. 700 страниц. Теория. Примеры на C++. Картинки! Вторая половина книги - примеры OOA и OOD с UML диаграммами.
2. MSDN - Microsoft Developer Network
3. Qt 5.X. Профессиональное программирование на C++. Макс Шлее. 2015 и более поздние издания г. 928 с. Книга периодически обновляется с выходом новых версий фреймворка Qt.
4. www.stackowerflow.com - система вопросов и ответов
5. draw.io — создание диаграмм.

Материалы курса

Слайды, вопросы к экзамену, задания, примеры

github.com/VetrovSV/OOP