

Qt

Обзор классов

Лекция 2

Кафедра ИВТ и ПМ

2019

Outline

Прошлые темы

Основные элементы интерфейса пользователя
QWidget

Core classes

Развёртывание. Кратко

Прошлые темы

- ▶ Что такое фреймворк?
- ▶ Для чего код организуют именно в фреймворки?
- ▶ Назовите примеры фреймворков и их назначение.
- ▶ Охарактеризуйте фреймворк Qt.

Прошлые темы

- ▶ Что такое API?
- ▶ Что такое сигналы и слоты?
- ▶ Для чего они используются?
- ▶ Что такое парадигма программирования?
- ▶ Что такое событийно-ориентированное программирование?
- ▶ Для чего предназначены классы `QCoreApplication` и `QApplication`?

Outline

Прошлые темы

Основные элементы интерфейса пользователя
QWidget

Core classes

Развёртывание. Кратко

Outline

Прошлые темы

Основные элементы интерфейса пользователя
QWidget

Core classes

Развёртывание. Кратко

QWidget - основной класс для всех элементов интерфейса пользователя.

Он принимает события мыши и клавиатуры, рисует самого себя на экране.

Все классы виджетов наследуются от **QWidget**, поэтому они имеют много общих методов и полей.

Виджеты

При создании окна с несколькими элементами интерфейса один из виджетов должен быть главным.

Такая иерархия достигается за счёт агрегации подчинённых виджетов в главный.

Если виджеты создаётся в дизайнера форм Qt Creator'a, то эта связь устанавливается автоматически во время генерации `cpp` файла из `ui` файла формы.

Виджеты

Если виджеты создаются вручную, то может понадобится у подчинённых виджетов при вызове конструктора передать параметром указатель на основной виджет.

```
QWidget::QWidget(QWidget *parent = Q_NULLPTR)
```

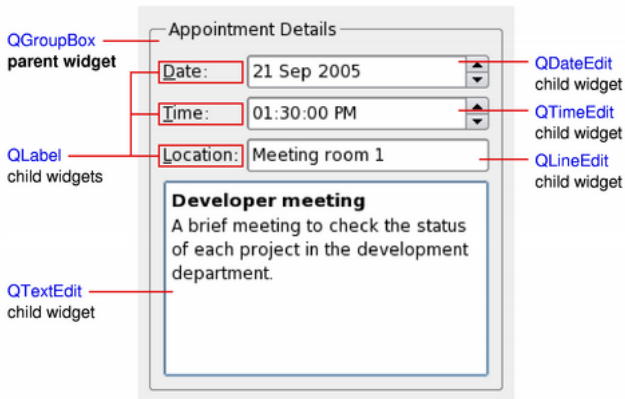
Если подчинённые виджеты добавляются сначала на компоновщик, а уже потом на основной виджет. То они автоматически становятся дочерними виджетами по отношению к основному.

Таким образом все виджеты находящиеся в окне, в конечном итоге агрегируются в основной виджет.

Виджеты

```
QWidget w;  
QVBoxLayout *layoyt = new QVBoxLayout();  
  
QPushButton *b = new QPushButton("PushMe");  
QPushButton *b2 = new QPushButton("Show Label");  
QLabel *l = new QLabel("I am Label");  
  
layoyt->addWidget(b);  
layoyt->addWidget(l);  
layoyt->addWidget(b2);  
w.setLayout(layoyt);  
  
qDebug() << w.children();  
// (QVBoxLayout(0xa8f4f0), QPushButton(0xd3e7c0),  
// QLabel(0xc70c70), QPushButton(0xc728d0))
```

Основным виджетом может выступать либо пустой виджет, либо другие виджеты-контейнеры. Например QTabWidget, QGroupBox и другие.



QWidget содержит свойства отвечающие за размер и положение элемента интерфейса пользователя, однако ручная работа с ними в большинстве случаев не рекомендуется.


За изменение размеров элемента интерфейса пользователя (виджетов) должен отвечать отдельный класс - компоновщик (layout), который будет автоматически менять ширину, высоту и положение виджета в зависимости от размеров окна.

В дизайнера форм можно задавать ограничения размера в контекстном меню виджета.

Поля класса лучше всего изменять в дизайнере форм QtCreator'a.

При изменении свойств основного виджета (на котором расположены другие виджеты), аналогично изменяются и свойства всех дочерних. Так например можно изменить шрифт одновременно во всём окне.

По каждому из свойств доступна справка:
выделить свойство -> F1

Свойство	Значение
▼ QObject	
objectName	centralWidget
▼ QWidget	
enabled	<input checked="" type="checkbox"/>
▶ geometry	[(0, 39), 736 x 487]
▶ sizePolicy	[Preferred, Preferred, 0, 0]
▶ minimumSize	0 x 0
▶ maximumSize	16777215 x 16777215
▶ sizeIncrement	0 x 0
▶ baseSize	0 x 0
palette	Унаследованная
▶ font	A [Ubuntu, 11]
cursor	 Arrow
mouseTracking	<input type="checkbox"/>
tabletTracking	<input type="checkbox"/>
focusPolicy	NoFocus
contextMenuPolicy	DefaultContextMenu
acceptDrops	<input type="checkbox"/>
▶ tooltip	
tooltipDuration	-1
▶ statusTip	
▶ whatsThis	
▶ accessibleName	
▶ accessibleDescription	
layoutDirection	LeftToRight

События QWidget

При изменении некоторых свойств виджета вызываются *обработчики* соответствующего события.

Например при изменении размера (вызов `resize(w, h)`) окна вызывается слот

```
resizeEvent(QResizeEvent *event);
```

Некоторые обработчики событий

```
void QWidget::showEvent(QShowEvent *event)
```

```
void QWidget::hideEvent(QHideEvent *event)
```

```
// перерисовка виджета
```

```
void QWidget::paintEvent(QPaintEvent *event)
```

```
void QWidget::closeEvent(QCloseEvent *event)
```

По умолчанию обработчики не имеют реализации или не выполняют никакой работы, но их можно определить внутри класса.

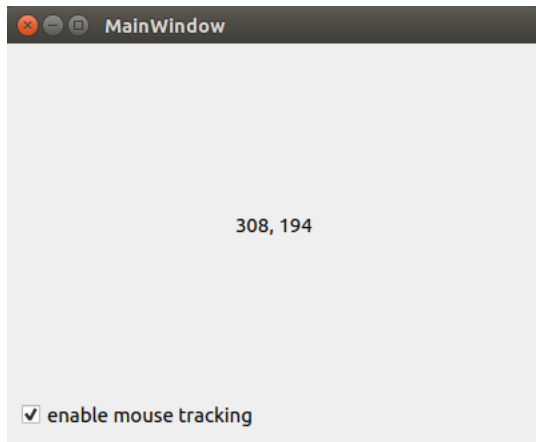
Пример

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent), ui(new Ui::MainWindow){
    ui->setupUi(this);
    // Включить отслеживание мыши для данного класса (главного окна)
    setMouseTracking(true);
    // Если это не сделать, то обработчик движения мыши будет
    // вызываться только если нажата одна из кнопок мыши
    // Включить отслеживание мыши виджетом,
    // хранящим всё содержимое главного окна
    ui->centralWidget->setMouseTracking(true);
    ui->checkBox->setChecked(
        ui->centralWidget->hasMouseTracking() );}

void MainWindow::mouseMoveEvent(QMouseEvent *e){
    ui->label->setText( QString::number( e->x() ) + ", "
        + QString::number( e->y() ));}

void MainWindow::on_checkBox_stateChanged(int arg1){
    setMouseTracking(arg1);}
```


Пример



События QWidget

Аналогично для событий генерируемых пользователем (клик или движение мыши, нажатие клавиши и т.д.) QWidget содержит виртуальные методы.

```
void QWidget::mouseMoveEvent(QMouseEvent *event)
void QWidget::mousePressEvent(QMouseEvent *event)
void QWidget::mouseDoubleClickEvent(QMouseEvent *event)

void QWidget::keyPressEvent(QKeyEvent *event)

void QWidget::contextMenuEvent(QContextMenuEvent *event)

void QWidget::dragEnterEvent(QDragEnterEvent *event)
```

При вызове обработчика ему передаётся в параметр объект описывающий соответствующее событие. Например координаты мыши или код нажатой клавиши.

События QWidget

Эти не специфические события для виджета нужно определять в производном от него классе вручную.

Обработчики для событий *других* виджетов, расположенных на данном окне создаются из контекстного меню конкретного виджета в дизайнера форм (go to slot...).

Основные свойства QWidget

► Размер

```
width();    // ширина в пикселях
```

```
height();   // высота в пикселях
```

```
resize( int w, int h); // задать размер виджета
```

```
// задать и зафиксировать размер
```

```
setFixedSize(w, h)
```

Основные свойства QWidget

- ▶ **enabled** : bool

Свойство отвечающее за "включение/выключение" элемента интерфейса.

```
bool isEnabled() const
```

```
void setEnabled(bool)
```

- ▶ **visible** : bool

- виден ли элемент интерфейса пользователю. bool

```
isVisible() const
```

```
virtual void setVisible(bool visible)
```

Основные свойства QWidget

- Фокус ввода с клавиатуры
focus : bool

```
bool hasFocus() const
```

```
void setFocus()
```

```
// Можно ли устанавливать фокус ввода?
```

```
// Каким способом устанавливать фокус ввода?
```

```
focusPolicy() const
```

```
void setFocusPolicy(Qt::FocusPolicy policy)
```

Основные элементы интерфейса пользователя

Часто используемые элементы интерфейса пользователя представлены классами:

- ▶ QLabel - надпись, также может отображать картинку;
- ▶ QLineEdit - однострочное текстовое поле ввода;
- ▶ QTextEdit - многострочное поле ввода;
- ▶ QSpinBox, QDoubleSpinBox - числовое поле ввода для целых и вещественных чисел соответственно
- ▶ QRadioButton - переключатель (позволяет выбор одного из нескольких вариантов)
- ▶ QCheckBox - флажок (галочка)
- ▶ QComboBox - Комбинированный список

Все эти элементы интерфейса могут быть использованы в Дизайнере форм Qt Creator.

Основные элементы интерфейса пользователя

- ▶ QListWidget - список
- ▶ QTableWidget - таблица
- ▶ QChartView - компонент для отображения графиков
- ▶ QGraphicsView - компонент для отображения графики
- ▶ QOpenGLWidget - компонент для рисования с помощью OpenGL
- ▶ QTextBrowser - текстовый браузер

Основные элементы интерфейса пользователя

- ▶ QWidget - таблица
- ▶ QChartView - компонент для отображения графиков
- ▶ QGraphicsView - компонент для отображения графики
- ▶ QOpenGLWidget - компонент для рисования с помощью OpenGL
- ▶ QTextBrowser - текстовый браузер

Иерархия наследования

Свойство	Значение
▼ QObject	
objectName	spinBox_n
▸ QWidget	
▼ QAbstractSpinBox	
wrapping	<input type="checkbox"/>
frame	<input checked="" type="checkbox"/>
▸ alignment	AlignLeft, AlignVCenter
readOnly	<input type="checkbox"/>
buttonSymbols	UpDownArrows
▸ specialValueText	
accelerated	<input type="checkbox"/>
correctionMode	CorrectToPreviousValue
keyboardTracking	<input checked="" type="checkbox"/>
showGroupSeparator	<input type="checkbox"/>
▼ QSpinBox	
▸ suffix	
▸ prefix	
minimum	0
maximum	99
singleStep	1
value	3
displayIntegerBase	10

Все классы имеющие отношение к графическому интерфейсу построены на основе QWidget.

Это хорошо видно в разделе свойств объекта в дизайнере QtCreator.

QWidget

▼ QWidget	
enabled	<input checked="" type="checkbox"/>
▶ geometry	[(1, 1), 48 x 26]
▶ sizePolicy	[Minimum, Fixed, 0, 0]
▶ minimumSize	0 x 0
▼ maximumSize	16777215 x 16777215
Ширина	16777215
Высота	16777215
▶ sizeIncrement	0 x 0
▶ baseSize	0 x 0
palette	Унаследованная
▶ font	A [Ubuntu, 11]
cursor	Arrow
mouseTracking	<input type="checkbox"/>
tabletTracking	<input type="checkbox"/>
focusPolicy	WheelFocus
contextMenuPolicy	DefaultContextMenu
acceptDrops	<input type="checkbox"/>
▶ tooltip	
tooltipDuration	-1
▶ statusTip	
▶ whatsThis	
▶ accessibleName	
▶ accessibleDescription	
layoutDirection	LeftToRight
autoFillBackground	<input type="checkbox"/>
styleSheet	
▶ locale	Russian, Russia
▶ inputMethodHints	ImhDigitsOnly

Большинство методов QWidget (так и остальных классов Qt) задающих свойства имеют названия `setProperty` и `property` для задания и получения свойства соответственно.

Сигналы и обработчики событий

- ▶ Виджеты могут вызывать свои сигналы в ответ на некоторые события.
- ▶ Например в ответ на клик мышью, позиционирование курсора, появление, нажатие клавиши или изменение содержимого (если их содержимое может изменять пользователь)
- ▶ Эти сигналы вызываются автоматически
- ▶ При создании слотов (обработчиков событий) в дизайнерах форм они автоматически соединятся с соответствующими сигналами
- ▶ Поэтому при вызове сигнала вызывается и слот

QLabel

```
QString text() const; // Получение текста QLabel  
void setText(const QString &); // Получение текста QLabel
```

QLabel

Добавление изображения в QLabel

```
// Загрузка изображения  
// здесь изображение прикреплено к проекту  
QPixmap pixmapTarget = QPixmap(":/image/icon/target.png");  
  
// Масштабирование изображения  
pixmapTarget = pixmapTarget.scaled(size-5, size-5,  
    Qt::KeepAspectRatio, Qt::SmoothTransformation);  
  
ui->label_image_power->setPixmap(pixmapTarget);
```

QPushButton

Сигналы

- ▶ `void clicked()`
- ▶ `void pressed()`
- ▶ `void released()`

Outline

Прошлые темы

Основные элементы интерфейса пользователя
QWidget

Core classes

Развёртывание. Кратко

Основные классы

- ▶ Qt содержит множество классов как для создания элементов интерфейса пользователя, так и для хранения данных, работы с сетью, изображениями и т.п.
- ▶ Классы используемые для хранения данных совместимы с аналогичными из STL и во много похожи на них
- ▶ Во многом классы из Qt удобнее для программиста, чем классы из STL

Рекомендации

- ▶ Перед решением задачи и написанием собственного кода следует проверить документацию на наличие подходящих классов.
- ▶ Перед использованием класса следует познакомиться с его документацией.
- ▶ Если не существует готовых решений, то следует изучить лучшие практики (best practice).

Справка Qt

F1 - вызов справки по классу (или функции), на который установлен курсор.

Справка по классу обычно состоит из

- ▶ общего описания класса
- ▶ Properties - списка свойств (полей класса и методов доступа к ним),
- ▶ Public Functions - открытых методов,
- ▶ Public Slots - открытых методов, которые вызываются в ответ на события.
- ▶ Закрытых методов
- ▶ Detailed Description - подробного описания класса, в котором могут быть приведены примеры его использования.

Основные классы Qt

Когда использовать STL, а когда аналогичные классы Qt?

Проблема бананов, обезьян и джунглей

Проблема с ОО-языками заключается в том, что они тянут за собой всё своё окружение. Вы хотели всего лишь банан, но в результате получаете гориллу, держащую этот банан, и все джунгли в придачу.

—Джо Армстронг, создатель Erlang

Когда использовать STL, а когда аналогичные классы Qt?

- ▶ В модулях приложения, которое и так использует Qt
Например класс главного окна
- ▶ В модулях, которые потенциально не будут использованы вне Qt приложений
- ▶ Везде, где выгода от использования именно Qt классов превосходит недостатки из-за проблемы бананов, обезьян и джунглей
- ▶ При использовании классов Qt конечный размер приложения может сильно вырасти из-за необходимости распространять его с dll (so файлами на Linux) файлами Qt.

Core classes

Файл проекта:

```
QT += core
```

Почти все Qt классы (не только основные) содержатся в одноимённых заголовочных файлах.

Например QPoint:

```
##include <QPoint>
```

Заголовочные файлы в некоторых других классов (по большей части это разного рода виджеты) могут находиться в отдельных каталогах фреймворка:

```
##include <QtWidgets/QLabel>
```

Заголовочный файл и модуль Qt указывается для каждого класса в документации.

Core classes

Классы предназначены для работы с данными, файловой системой, временем, исключением и т.п. содержатся в ядре фреймворка - модуле core.

Некоторые из **core classes**

QSize QRect QPoint

QString QVector QStringList QStack QSet QPair QMap QList

QTime QDate QTimer

QException

QRegExp

QUrl

QTextStream QFile

QMessageLogger

QString

Класс для хранения строк в Unicode кодировке.

```
QString str;
```

```
// Число -> строка
```

```
str.setNum(1234);           // str == "1234"
```

```
// Строка -> число
```

```
float x = str.toFloat();
```

```
// Число -> строка. Статический метод
```

```
QString s = QString::number(42.05)
```

```
// Возвращает строку без пробелов в начале и конце
```

```
str = str.trimmed();
```

```
// в std::string
```

```
std::string s = str.toStdString()
```

Информация о каталогах и их структуре.

```
// Упрощает работу с путями к файлам
QDir directory("Documents/Letters");
QString path = directory.filePath("contents.txt");
QString name = directory.dirName();
QString absPath = directory.absoluteFilePath("contents.txt");

// текущая папка
QDir cdir = QDir::current();
QString cdir_path = QDir::currentPath();
// Папка пользователя
QDir home = QDir::home();
```

QDir

```
// проверка на существование
if ( dir.exists() ){... }

// Список имён файлов
QStringList dd = d.entryList();

// сменить папку (текущая папка для прогр. не меняется)
home.cd("another-dir");

// изменить текущую папку программы
if ( QDir::setCurrent("another-dir") ){
    // папка поменялась
}
```

QFile

Чтение и запись данных в файлы.

```
QFile f("myfile");

if ( !f.open(QFile::WriteOnly) )
    // не удалось открыть файл ;
;
char *data = "some data 12345 \n";
f.write(data, strlen(data));
f.close();

if ( !f.open(QFile::ReadOnly) )
    // не удалось открыть файл ;

char buf[1024];
qint64 r = f.readLine(buf, 1024);
if ( r!=-1 )
    // прочитано r байт
```

QTextStream

Упрощает работу с текстовыми файлами.

```
// создаём класс-файл
QFile data("output.txt");
// Открываем файл для записи
if (data.open(QFile::WriteOnly)) {

    // Для удобства записи разных типов данных
    // в текстовый файл
    // используем этот класс
    QTextStream out(&data);

    out << "Result: " << qSetFieldWidth(10) << left << 3.14
    // writes "Result: 3.14          2.7          "
}
```

QTextStream

Упрощает работу с текстовыми файлами.

```
QFile data("output.txt");
if (data.open(QFile::ReadOnly)) {
    QTextStream in(&data);
    QString str = in.readLine();
    // уберём лишние пробелы в начале и в конце
    str = str.trimmed();
    // заменим повторяющиеся пробелы на один
    unsigned n;
    do{ n = str.length();
        str = str.replace("  ", " ");
    } while (n!=str.length());

    // разделим строку по пробелам
    QStringList sl = str.split(" ");
    x = sl[1].toFloat();
    x = sl[3].toFloat();
```

QTimer

```
QTimer *timer = new QTimer;  
QObject::connect(timer, &QTimer::timeout,  
                 [](){qDebug() << ".";});  
timer->setInterval(1500); // в миллисекундах  
timer->start();
```

```
// Таймер с одиночным срабатыванием  
QTimer::singleShot(200, объект, SLOT(метод));  
// после срабатывания будет вызван  
// метод указанного объекта
```

В примере использована лямбда функция, но соединить сигнал таймера timeout можно с любой функцией или методом.

Outline

Прошлые темы

Основные элементы интерфейса пользователя
QWidget

Core classes

Развёртывание. Кратко

Развёртывание Qt приложений

Для работы Qt приложения кроме исполняемого файла необходим набор библиотек Qt, так как в исполняемый файл помещается преимущественно код разработчика приложения, а не весь необходимый код фреймворка.

В некоторых дистрибутивах Linux (например Ubuntu) набор библиотек Qt уже установлен в системе и приложения (если версии библиотек совпадают) можно распространять без них.

В ОС Windows часто библиотеки либо не установлены, либо отличаются версией или присутствуют не в полном составе. Поэтому приложение часто распространяют уже с набором dll файлов фреймворка Qt. Для создание такого набора используется программа windeployqt.

Qt for Windows - Deployment Развёртывание приложений Qt в Windows

Развёртывание Qt приложений

Qt for Windows - Deployment

Развёртывание приложений Qt в Windows

Ссылки и литература

- ▶ Qt Википедия
- ▶ OpenSource версия
- ▶ Qt wiki

Книги:

- ▶ Qt 5.X. Профессиональное программирование на C++. Макс Шлее. 2015 г. 928 с. Книга периодически обновляется с выходом новых версий фреймворка Qt.
- ▶ Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++. Марк Саммерфилд