Qt

кросс-платформенный открытый инструментарий разработки программного обеспечения на языке C++

История Qt

- Разработку начали в 1991 году Eirik Chambe-Eng и Haavard Nord (Норвегия)
- 1994 - основана компания Troll Tech, позднее Trolltech
- 2001 - выпущена Qt 3.0, появилась поддержка OS X
- 2005 - выпущена Qt 4.0
- Июнь 2008 - Nokia приобрела Trolltech и переименовала его в Qt Development Frameworks
- Март 2011 - Nokia продала коммерческую часть Qt финской компании Digia
- 3 июля 2013 - выход Qt 5.1
- Сентябрь 2014 – Digia выделила разработку Qt в The Qt Company
- Май 2016 – The Qt Company стала независимой от Digia компанией
- 26 мая 2020 – последняя версия 5.15
- Следующей обещают Qt 6

Лицензии

- Свободные лицензии с января 2009 года, сейчас LGPL v.3, GPL v.2 и GPL v.3
  - Доступен исходный код
  - Исходный код можно модифицировать и распространять модифицированную версию
  - Может быть использована в свободном ПО
  - Может быть использована в коммерческом ПО без обязательного раскрытия исходного кода
- Коммерческая лицензия
  - Доступны дополнительные модули
  - Поддержка

http://www.qt.io/

Кросс-платформенный

- Поддерживаются платформы:
  - Windows
  - Unix/X11
  - OS X
  - WebAssembly
  - Встраиваемые и мобильные платформы
    - Embedded Linux
    - Микроконтроллеры
    - Android
    - iOS
    - BlackBerry 10/QNX
    - Экспериментальная поддержка в сторонних проектах
      - webOS
      - Amazon Kindle

Инструментарий / Framework

- qmake
- Qt Designer
- Qt Assistant
- Qt Linguist
- uic (User Interface Compiler)
- rcc (Resource Compiler)
- moc
- Qt Simulator

- QtCreator

Модули

- **Qt Core – цикл обработки сообщений, сигналы и слоты, строки, регулярные выражения, динамические массивы, файлы, потоки, разделяемая память, настройки программ.**
- **Qt GUI – базовые классы графического интерфейса пользователя, включая поддержку Open GL.**
- Qt Multimedia –классы для взамодействия с аудио, видео устройствами, радио и камерой
- Qt Multimedia Widgets – Классы для работы с мультимедиа с виджетами
- Qt Network – сетевая библиотека
- Qt QML – классы для работы с Javascript и QML
- Qt Quick – инструментарий для разработки динамических интерфейсов.
- Qt Quick Controls – виджеты для Qt Quick
- Qt Quick Layouts – схемы разметки для Qt Quick
- Qt Sql – работа с базами данных
- Qt Test – модульное тестирование
- Qt Web Kit – модуль браузера HTML
- Qt Web Kit Widgets– модуль браузера HTML для виджетов
- **Qt Widgets – виджеты**

- Active Qt – классы для работы с ActiveX и COM
- Qt Concurrent – классы для многопоточных приложений
- Qt D-Bus – поддержка шины D-Bus
- Qt Graphical Effects – графические эффекты для использования с Qt Quick
- Qt Image Formats – поддержка дополнительных форматов графических файлов
- Qt Open GL – аппаратная трехмерная графика (для совместимости с Qt 4)
- Qt Print Support – поддержка печати
- Qt Declarative, Qt Script, Qt Script Tools – поддержка Qt Quick и языка сценариев для совместимости с Qt 4
- Qt Sensors – поддержка аппаратных сенсоров и распознавания жестов.
- Qt Serial Port – поддержка последовательных портов
- QtSvg – работа с векторным форматом SVG
- Qt X11 Extras – поддержка дополнительных функций в X11
- QtXml, QtXmlPatterns – поддержка языка разметки данных XML

- Qt Designer – расширение Qt Designer
- Qt Help – создание справки
- Qt UI Tools – работа с интерфейсом

Поддержка в других языках программирования (Qt 4)

- Ada
- C# & .NET
- D
- Harbour
- Haskell
- Java
- Lisp
- Lua
- Pascal
- Perl
- Python
- R
- Ruby
- Scheme

Программирование с Qt

Hello, world!

Консольная версия

```
#include <QtCore/QCoreApplication>
#include <iostream>

int main(int argc, char *argv[])
{
  QCoreApplication a(argc, argv);

  std::cout<<“Hello, world!”;


  return a.exec();
}
```

GUI версия

```
#include <QtGui>
#include <QMessageBox>

int main(int argc, char *argv[])
{
  QApplication app(argc, argv);

  QMessageBox::information(NULL,
        “GUI”,”Hello, world!");


  return app.exec();
}
```

Файлы проекта

- *.h – заголовочные файлы C++, содержат объявления классов и функций
- *.cpp – исходные файлы C++, содержат определения функций
- *.ui – классы форм Qt, редактируются Qt Designer
- *.qrc – ресурсы Qt
- *.pro – файл проекта Qt
- *.qml – описание интерфейса Qt Quick
- *.pro.user – настройки Qt Creator

.pro - Файлы проекта Qt

```
QT += core gui sql
TARGET = QCOptimizer
TEMPLATE = app
INCLUDEPATH += ../OptLib/

SOURCES += main.cpp\
            mainwindow.cpp \
            plugins.cpp

HEADERS += mainwindow.h \
            InterfacePlugin.h \
           Plugins.h

FORMS += mainwindow.ui

RESOURCES += qcoptimizer.qrc

CONFIG(debug)
{
  DESTDIR=../../bin/debug
}
CONFIG(release)
{
 DESTDIR=../../bin/release
}

LIBS += -lOptLib -L$$DESTDIR
include(../fileWidget/fileWidget.pri)
```

Текстовые строки, юникод и Qt

Где какие кодировки используются

- Windows – «традиционные» 8 бит или UTF-16LE
  - SetWindowTextA(…) – 8-битная кодировка, согласно выбранной кодовой странице
  - SetWindowTextW(…) – юникод, кодировка UTF-16LE
  - Компилятор C/C++
    - “Hello, world!” – текстовая константа, в традиционной 8-битной кодировке.
    - L”Hello, world!” – текстовая константа в UTF-16LE
      - Тип данных: wchar_t
      - Функции для работы: wcslen, wcscat
    - Универсальный вариант: подключите tchar.h:
      - Константы: _T(“Hello, world!”) или _TEXT(“Hello, world!”)
      - Тип данных: _TCHAR
      - Функции: _tcslen, _tcscat
- UNIX – чаще всего UTF-8 (Linux, BSD, Mac OS X)

Как настроить кодировку в Qt

1. В Qt 5 все исходные файлы должны быть сделаны в кодировке UTF-8
2. Текстовые константы передаём через функцию tr():

```
Qlabel label(QObject::tr("Всем привет!"));
label.show();
```

- В классах, унаследованных от QObject, можно писать просто `tr("Тут разный текст")`

- Qt Linguist может найти в тексте программы все константы внутри tr() и обеспечить возможность их перевода на другие языки.

Класс QString

- Текстовые строки в Qt представляются классом **QString**
- Внутри QString текст хранится в UTF-16LE

- Как создать:

```
QString s1= "hello", s2=QObject::tr("привет");
```

- Длина строки:

```
int len=s1.size();          // Длина строки
bool b=s2.isEmpty();        // Пустая или нет
```

- Доступ к элементам строки:

```
s1[0]=QChar('H');           // Чтение и запись
QChar c=s2.at(0);           // Только чтение, быстрее
```

Класс QString - Преобразования

- Из разных кодировок

```
QString s1=QString::fromLocal8bit("Привет");
QString s2=QString::fromUtf8("Hello");
QString s3=QString::fromUtf16(L"Юникод");
```

- В другие кодировки (через QByteArray)

```
char *str1=s1.toLocal8Bit().data();
char *utf8=s2.toUtf8().data();
```

- Цифровые

```
bool ok;
double d=QString("123.45").toDouble(&ok);
int i=QString("17").toInt(&ok);
long hex=QString("FF").toInt(&ok,16);
```

Класс QString - Форматирование

- Функция **arg()**

```
QString i; // current file's number
QString total; // number of files to process
QString fileName; // current file's name

QString status = QString("Processing file %1 of %2: %3")
    .arg(i).arg(total).arg(fileName);
```

- Статическая функция **number()**

```
long a = 63;
QString s = QString::number(a, 16); // s == "3f"
QString t = QString::number(a, 16).toUpper(); // t == "3F"

double d = 6.3;
QString s = QString::number(d, ‘f’ , 3); // s == "6.300"
```

Класс QString – Операции со строками

```
QString str = "and";
str.prepend("rock ");   // str == "rock and"
str.append(" roll");    // str == "rock and roll”
str.replace(5, 3, "&"); // str == "rock & roll"

str=QObject::tr(“rock ”) + QObject::tr(“and “) + QObject::tr(“roll”);

QString str = "Berlin";
str.fill('z’);          // str == "zzzzzz"
str.fill('A', 2);       // str == "AA"

QString str = " lots\t of\nwhitespace\r\n ";
str = str.simplified(); // str == "lots of whitespace";
```

- Сравнения

```
if (str == "auto" || str == "extern" || str == "static") {/* ...*/ }

if(str1 > str2) { /* …  */ }     // быстро, по кодам

if(QString::localeAvareCompare(str1,str2) < 0)
{ /* str1 меньше str2 */ }
```

Класс QString - Поиск

```
QString str = "Peter Pan";
str.contains("peter", Qt::CaseInsensitive); // returns true

QString x = "sticky question";
QString y = "sti";
x.indexOf(y);      // returns 0
x.indexOf(y, 1);   // returns 10
x.indexOf(y, 11);  // returns -1

QString x = "crazy azimuths";
QString y = "az";
x.lastIndexOf(y);    // returns 6
x.lastIndexOf(y, 5); // returns 2

QString str = "the minimum";
str.indexOf(QRegExp("m[aeiou]"), 0);  // returns 4

QString str = "Bananas";
str.endsWith("anas");    // returns true
str.endsWith("pple");    // returns false
```

Класс QString – Разбиение на подстроки

```
QString str;
QString csv = "forename,middlename,surname,phone";
QString path = "/usr/local/bin/myapp"; // первое поле пустое

str = csv.section(',', 2, 2);   // str == "surname"
str = path.section('/', 3, 4);  // str == "bin/myapp"
str = path.section('/', 3, 3, QString::SectionSkipEmpty);
                                // str == "myapp"
str = csv.section(',', -3, -2); // str == "middlename,surname"

QString str = "a,,b,c";
QStringList list1 = str.split(","); // list1: ["a", "", "b", "c"]
QStringList list2 = str.split(",", QString::SkipEmptyParts);
                                    // list2: [ "a", "b", "c" ]
```

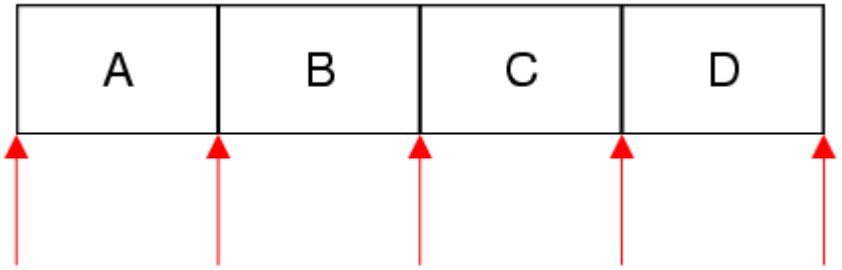
Классы контейнеров

Классы контейнеров Qt

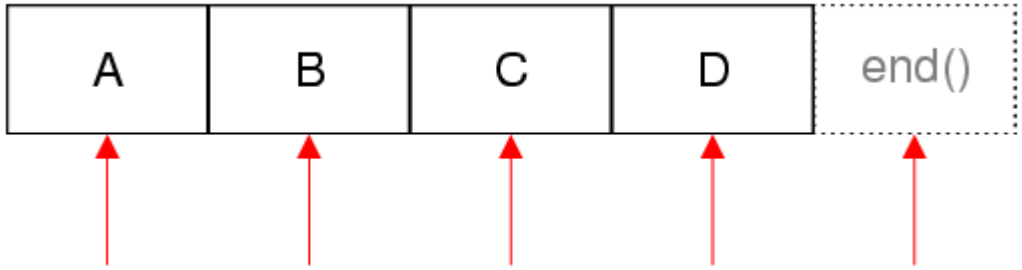
- Массивы
  - QList<T> - список значений, доступных по индексу
    - QQueue<T> - расширение QList - для использования как очередь
  - QLinkedList<T> - список значений, доступных последовательно
  - QVector<T> - массив значений, последовательно расположенных в памяти
    - QStack<T> - расширение QVector - для использования как стек
- Множества и словари
  - QSet<T> - множество элементов (как в математике)
  - QMap<Key, T> - словарь, хранящий отображение ключ-значение
  - QMultiMap<Key, T> - словарь, можно хранить несколько значений для одного ключа
  - QHash<Key, T> - более быстрая версия QMap
  - QMultiHash<Key, T>

Контейнеры – способы доступа

- По индексу
- Итераторы Java-стиля



- Итераторы STL-стиля



- foreach()

Доступ по индексу

```
QList<QString> list;
list << "A" << "B" << "C" << "D";

int i;

for(i=0;i<list.count();i++)
    qDebug() << list[i];
```

Итераторы Java-стиля

- Проход по массиву в прямом порядке:

```
QList<QString> list;
list << "A" << "B" << "C" << "D";

QListIterator<QString> i(list);

while (i.hasNext())
    qDebug() << i.next();
```

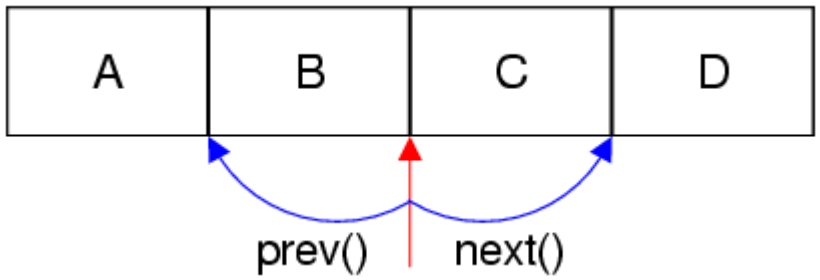
- Проход по массиву в обратном порядке:

```
i.toBack();

while (i.hasPrevious())
    qDebug() << i.previous();
```

Итераторы Java-стиля

Операция	Действие
toFront()	Переводит итератор перед началом списка
toBack()	Переводит итератор за конец списка
hasNext()	Возвращает true, если итератор не за концом списка
next()	Возвращает следующий элемент и переходит вперёд
peekNext()	Возвращает следующий элемент без перехода
hasPrevious()	Возвращает true, если итератор не перед началом
previous()	Возвращает предыдущий элемент и переходит назад
peekPrevious()	Возвращает предыдущий элемент без перехода



Итераторы Java-стиля

- Для изменения элементов нужно использовать **QMutableListIterator**

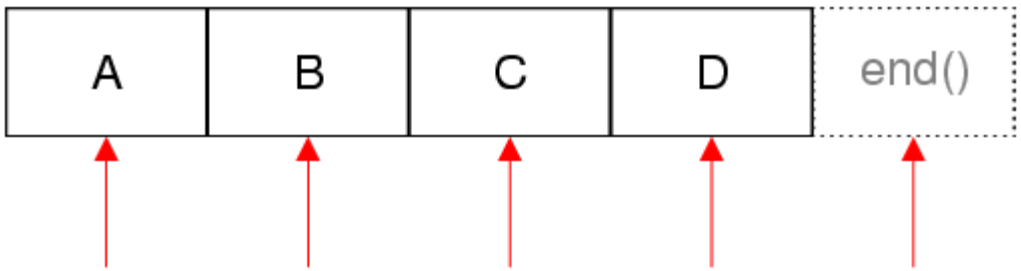
```
QList<int> list;

QMutableListIterator<int> i(list);

while (i.hasNext())
{
   if (i.next() % 2 != 0)
        i.remove();
}
/* ---------------------------------- */
while (i.hasNext())
{
    if (i.next() > 128)
         i.setValue(128);
}
/* ---------------------------------- */
while (i.hasNext())
      i.next() *= 2;
```

Итераторы STL-стиля

Операция	Действие
*i	Возвращает текущий элемент
++i	Переходит на следующий элемент
i+=n	Переходит вперед на n элементов
--i	Переходит на предыдущий элемент
i-=n	Переходит на n элементов назад
i-j	Возвращает число элементов между итераторами



Итераторы STL-стиля

- Проход по массиву в прямом порядке

```
QList<QString> list;
list << "A" << "B" << "C" << "D";

QList<QString>::iterator i;

for (i = list.begin(); i != list.end(); ++i)
    *i = (*i).toLower();
```

- Проход по массиву в обратном порядке

```
i = list.end();
while (i != list.begin())
{
   --i;
   *i = (*i).toLower();
}
```

foreach()

```
QList<QString> list;
list << "A" << "B" << "C" << "D";

QString str;

foreach(str, list)
    qDebug() << str;
```

- **foreach()** делает копию списка, который он обходит, поэтому с его помощью нельзя модифицировать элементы исходного списка.

- Сейчас можно использовать «новый» цикл из C++

```
for(const QString &str: list)
    qDebug() << str;
```

QList

QList<QString> list;

- Добавление

```
list << "one" << "two" << "three";
list.append("four");
list.prepend("zero");
```

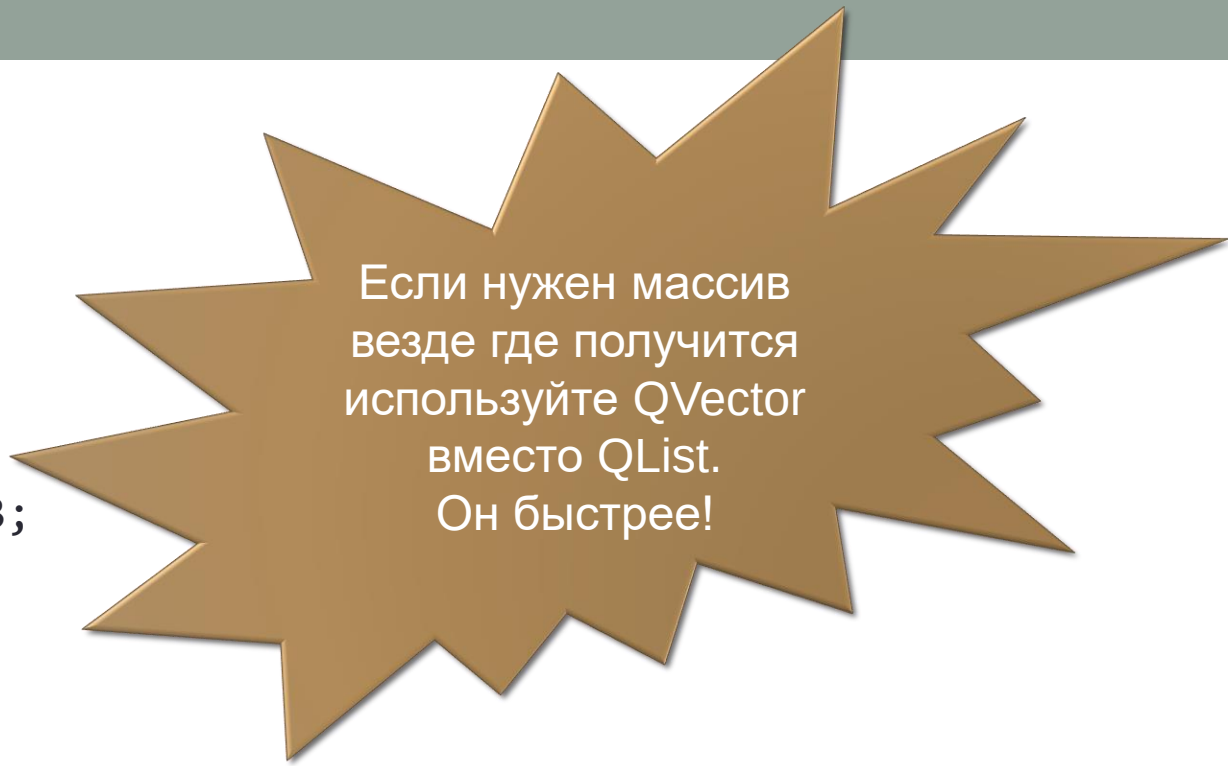
- Поиск

```
int i = list.indexOf("six");
if(i != -1) cout << tr("первое вхождение ‘six’ в позиции") << i;
```

- Модификация
  - `removeAll()`
  - `removeAt(int i)`
  - `swap(int i,int j)`
  - `move(int from,int to)`
  - `reserve(int alloc)`

QVector

QVector<int> vector;



- Добавление

```
vector << 1 << 2 << 3;
vector.append(4);
vector.prepend(5);
```

- Поиск

```
int i = vector.indexOf(6);
if(i != -1) cout << tr("первое вхождение 6 в позиции") << i;
```

- Модификация
  - `removeAll()`
  - `removeAt(int i)`
  - `swap(int i, int j)`
  - `move(int from, int to)`
  - `reserve(int alloc)`

QMap

```
QMap<QString, int> map;
```

- Добавление элементов:

```
map["one"] = 1;
map["seven"] = 7;
map.insert("twelve", 12);
```

- Извлечение элементов:

```
int num1 = map["thirteen"];
if(map.contains("ten")) num1=map.value("ten");
num2 = map.value("three", 3)
```

- Обход элементов:

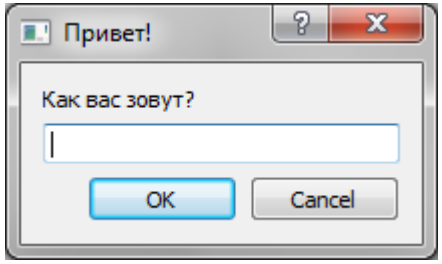
```
QMapIterator<QString, int> i(map);
while (i.hasNext())
{
    i.next();
    cout << i.key() << ": " << i.value() << endl;
}
```

Чуть-чуть ввода

QInputDialog

```
#include <QInputDialog>   // в .pro: QT += widgets

QString text=QInputDialog::getText(
       NULL,                       // родительский виджет
       tr("Привет! "),             // заголовок
       tr("Как Вас зовут?")); // приглашение
```



- Другие функции:
  - `getInt()`
  - `getDouble()`
  - `getItem()`

Файлы

Класс QFile

1. Создаём объект и задаём имя файла:

```
QFile file("in.txt");
```

или

```
QFile file;
file.setFileName("in.txt");
```

2. Проверка на существование:

```
file.exists()
```

3. Удаление:

```
file.remove()
```

4. Переименование:

```
file.rename(const QString & newName);
```

Класс QFile

1. Открыть файл:

```
file.open(OpenMode mode)
   QIODevice::ReadOnly, QIODevice::WriteOnly, QIODevice::ReadWrite, QIODevice::Text
```

2. Чтение:

```
qint64 QIODevice::read ( char * data, qint64 maxSize )
QByteArray QIODevice::read ( qint64 maxSize )
QByteArray QIODevice::readAll ()
QByteArray QIODevice::readLine ( qint64 maxSize = 0 )
```

3. Запись:

```
qint64 QIODevice::write ( const char * data, qint64 maxSize )
qint64 QIODevice::write ( const QByteArray & byteArray )
```

4. Перемещение:

```
bool QFile::atEnd () const
qint64 QFile::size () const
bool QFile::seek ( qint64 pos )
qint64 QFile::pos () const
```

5. Закрыть:

```
void QFile::close ()
```

Класс QFile. Пример

```
QFile file("in.txt");

if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
 return;

while (!file.atEnd())
{
  QByteArray line = file.readLine();
  process_line(line);
}

file.close();
```

Потоки QTextStream

```
QFile file("in.txt");

if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
   return;

QTextStream in(&file);

while (!in.atEnd())
{
   QString line = in.readLine();
   process_line(line);
}
```

```
QFile data("output.txt");
if (!data.open(QFile::WriteOnly | QFile::Truncate))
   return;

QTextStream out(&data);
out << "Result: " << qSetFieldWidth(10) << left << 3.14 << 2.7;
// writes "Result: 3.14      2.7       "
```

QTextStream. Выбор кодировки

```
QFile file(fileName);
if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
    return false;

QTextStream out(&file);

out.setLocale(QLocale::Russian); // формат чисел
out.setCodec("UTF-8");                  // кодировка

out << tr("текст будет выведен в UTF 8") << endl;
```

QFileDialog

```
QString fileName = QFileDialog::getOpenFileName(this,
            tr("Open Image"),
            "/home/jana",
            tr(“PNG (*.png);; Jpeg (*.jpg);; Windows bitmap (*.bmp)"));
```

- Другие функции:

```
QString getExistingDirectory ( QWidget * parent = 0, const QString
& caption = QString(), const QString & dir = QString(),
Options options = ShowDirsOnly )
```

```
QStringList getOpenFileNames ( QWidget * parent = 0, const QString
& caption = QString(), const QString & dir = QString(), const QString
& filter = QString(), QString * selectedFilter = 0, Options options = 0
)
```

```
QString QFileDialog::getSaveFileName ( QWidget * parent = 0,
const QString & caption = QString(), const QString & dir = QString(),
const QString & filter = QString(), QString * selectedFilter = 0,
Options options = 0 )
```

Другие полезные объекты

- Доступ к директориям: QDir
  - Перемещение по директориям
  - Просмотр содержимого директорий
  - Создание директорий
  - Проверка существования файлов и директорий
  - Удаление и переименование файлов
  - Список дисков
  - Работа с текущей директорией
  - Преобразование путей
- Информация о файле: QFileInfo
  - Преобразование путей и доступ к фрагментам имени
  - Определение типа файла
  - Определение размера файла
  - Определение времени создания и модификации файла
  - Проверка прав доступа к файлу
  - Проверка существования файла