

Объектная модель Qt



Qt расширяет C++

Qt добавляет к C++ следующие возможности:

- Механизм для взаимодействия объектов – сигналы и слоты
- Свойства объектов
- События и фильтры событий
- Контекстный перевод строк для интернационализации

Расширение C++ классов

```
class Counter
{

public:
    Counter() { m_value = 0; }
    int value() const { return m_value; }

    void setValue(int value);

private:
    int m_value;
};
```

```
#include <QObject>
```

```
class Counter : public QObject
{
    Q_OBJECT
    Q_PROPERTY (int value READ value
                WRITE setValue NOTIFY valueChanged)
public:
    Counter() { m_value = 0; }
    int value() const { return m_value; }

public slots:
    void setValue(int value);
signals:
    void valueChanged(int newValue);

private:
    int m_value;
};
```

Структура объявления класса

```
#include <QObject>

class ClassName : public QObject
{
    Q_OBJECT // enable Qt's meta-object system
              // or Q_GADGET if no need in signals and slots
    ... // declaration of properties
public:
    explicit ClassName(QObject *parent = nullptr);
    ... // public methods, fields and types
signals:
    ... // declared like methods but to be connected
public slots:
    ... // public methods to connect to
private:
    ... // private methods, fields and types
private slots:
    ... // private methods to connect to
};
```

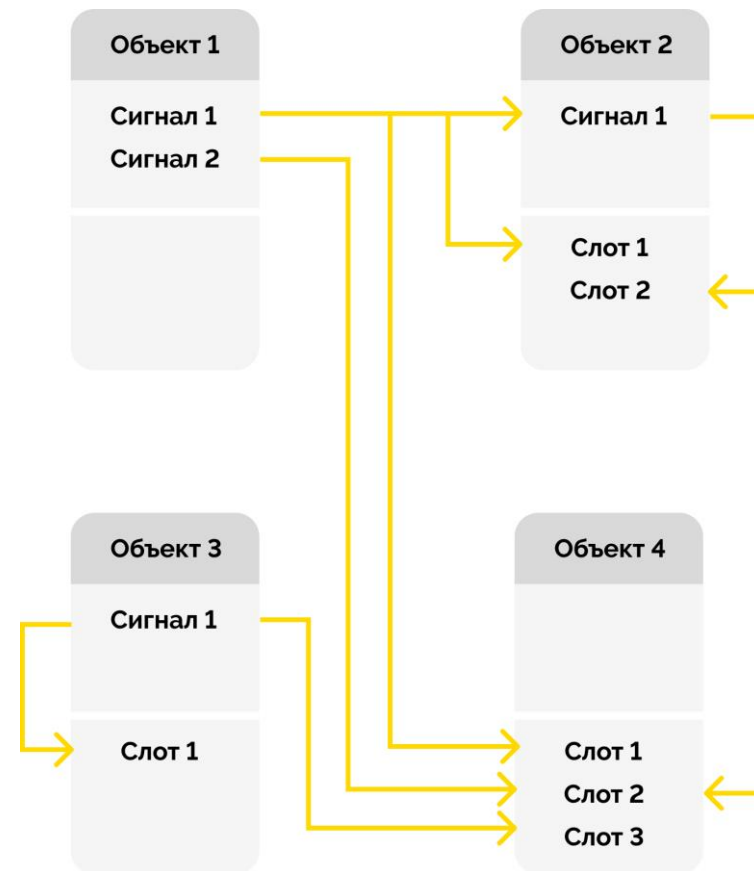
Сигналы и слоты

Сигналы

- Испускаются объектом при изменении его состояния
- Публичные методы, доступны отовсюду (рекомендуется испускать только владельцем)
- Не возвращают значений
- Могут принимать аргументы
- Реализация генерируется с помощью `moc`

Слоты

- Методы, к которым можно подключать сигналы
- Не должны возвращать значение
- Могут быть приватными и публичными
- Могут быть вызваны как обычные методы
- Вызываются при испускании присоединённых сигналов (даже приватные)



Подключение и испускание сигналов

Подключение

```
QObject::connect(senderObject, &ClassName::signalName,  
                 receiverObject, &ClassName::signalOrSlotName,  
                 Qt::ConnectionType type = Qt::AutoConnection);  
QObject::connect(this, &TextProcessor::textChanged,  
                 this, &TextProcessor::processedTextChanged);
```

Отключение

```
QObject::disconnect(senderObject, &ClassName::signalName,  
                    receiverObject, &ClassName::signalOrSlotName);
```

Испускание

```
emit signalName( ... );  
emit textChanged();
```



Q_GADGET

- Облегчённая версия макроса Q_OBJECT
- Для классов, которые не наследуются от QObject
- Находится в приватном разделе определения класса
- Поддерживаются: Q_ENUM, Q_PROPERTY и Q_INVOKABLE
- Не поддерживаются: сигналы или слоты

Регистрация свойств

Синтаксис

```
Q_PROPERTY(type propertyName, READ getterName, [WRITE setterName],  
           [NOTIFY changeSignalName]/[CONSTANT])
```

Примеры

```
Q_PROPERTY(QString text, READ text, WRITE setText, NOTIFY textChanged)  
Q_PROPERTY(QString processedText, READ processedText, NOTIFY  
processedTextChanged)
```


Регистрация перечислений

Регистрация в C++

Синтаксис

```
enum EnumerationName { ... };  
Q_ENUM(EnumerationName)
```

Пример

```
enum Action { NoAction, MakeUppercase, MakeLowercase };  
Q_ENUM(Action)
```

Использование в QML

Синтаксис

```
TypeName.ValueName
```

Примеры

```
TextProcessor.NoAction  
TextProcessor.MakeUppercase  
TextProcessor.MakeLowercase
```

Регистрация методов

Регистрация в C++

Все слоты регистрируются
автоматически

Синтаксис

```
Q_INVOKABLE resultType methodName( ... );
```

Пример

```
Q_INVOKABLE QString prepend(const QString &string);
```

Использование в QML

Синтаксис

```
objectId.methodName( ... )
```

Пример

```
textProcessor.prepend(prependTextField.text)
```

Объявление визуального элемента

```
#include <QQuickItem>

class ClassName : public QQuickItem
{
    Q_OBJECT
    ... // declaration of properties
public:
    explicit ClassName(QQuickItem *parent = nullptr);
    ...

    QSGNode *QQuickItem::updatePaintNode(QSGNode *oldNode,
        QQuickItem::UpdatePaintNodeData *updatePaintNodeData);
signals: ...
public slots: ...
private: ...
private slots: ...
};
```

Пример рендеринга для QSG

```
QSGNode *RedRectangle::updatePaintNode(QSGNode *node, UpdatePaintNodeData *)
{
    QSGSimpleRectNode *rectNode = static_cast<QSGSimpleRectNode *>(node);
    if (!rectNode) {
        rectNode = new QSGSimpleRectNode();
        rectNode->setColor(Qt::red);
    }
    rectNode->setRect(boundingRect());
    return rectNode;
}
```

Элемент на QPainter

```
#include <QQuickPaintedItem>
class ClassName : public QQuickPaintedItem
{
    Q_OBJECT
    ... // declaration of properties
public:
    explicit ClassName(QQuickItem *parent = nullptr);
    ...
    void paint(QPainter *painter);
    signals: ...
    public slots: ...
    private: ...
    private slots: ...
};
```



Контейнеры

- Готовые к использованию структуры данных
- Range-based for
- Итераторы
- Java-Style
- STL-Style
- Совместимы с STL-алгоритмами
- Работа с потоками
- Implicit Sharing
- QVector
- QByteArray
- QList
- QStack
- QQueue
- QMap
- QHash
- QSet
- QString
- QVariant

QVector – динамический массив

- **append(const T &value)** – добавляет элемент в конец вектора
- **insert(int index, const T &value)** – добавляет элемент по индексу
- **T& at(int index)** – получить элемент по индексу
- **int count()** – количество элементов в векторе
- **remove(int index)** – удаляет элемент по индексу
- **int removeAll(const T &value)** – удаляет элементы с таким значением
- **QList<T> toList()** – возвращает QList, созданный из элементов вектора

```
QVector<int> vec;  
vec.append(10);  
vec.append(20);  
vec.append(30);  
QDebug() << vec; // QVector(10,20,30)  
QDebug() << vec[0]; // 10
```

QByteArray – массив байтов

- **toDouble()** – возвращает массив, преобразованный в double
- **toFloat()** – возвращает массив, преобразованный во float
- **QByteArray toHex()** – возвращает копию массива в 16-теричном виде
- **toInt()** – возвращает массив, преобразованный в int
- **toLong()** – возвращает массив, преобразованный в long
- **toShort()** – возвращает массив, преобразованный в short

```
QByteArray x("free");  
QByteArray y("dom");  
x.append(y); // x = "freedom"  
QByteArray byteArray("1234.56");  
double a = byteArray.toDouble(); // a = 1234.56
```


QList - список

- **move(int from, int to)** – перемещает элемент
- **swap(int index1, int index2)** – меняет местами два элемента
- **T takeAt(int index)** – возвращает и удаляет указанный элемент
- **T takeFirst()** – возвращает и удаляет первый элемент
- **T takeLast()** – возвращает и удаляет последний элемент
- **QSet<T> toSet()** – возвращает **QSet** с данными
- **QVector<T> toVector()** – возвращает **QVector** с данными

```
QList<int> list;  
list.append(1);  
list.append(4);  
list.append(2);  
int first = list.takeFirst();    // first = 1  
int last = list.takeLast();     // last = 2
```

QStack - стек

- **push(const T &value)** – добавляет элемент в стек
- **pop()** – извлекает элемент из стека
- **top()** – возвращает ссылку на верхний элемент в стеке
- **swap()** – заменяет элементы стека элементами другого

```
QStack<QString> stack;  
stack.push("Element 1");  
stack.push("Element 2");  
stack.push("Element 3");  
while (!stack.empty()) {  
    qDebug() << stack.pop();  
}
```

QQueue - очередь

- **enqueue(const T &value)** – добавляет значение в конец очереди
- **dequeue()** – извлекает значение из начала очереди
- **head()** – возвращает ссылку на начальный элемент очереди

```
QQueue<QString> queue;  
queue.enqueue("Element 1");  
queue.enqueue("Element 2");  
queue.enqueue("Element 3");  
while (!queue.empty()) {  
    qDebug() << queue.dequeue();  
}
```

QMap - словарь

- **key(const T &value)** – возвращает ключ элемента по значению
- **keys()** – возвращает список ключей
- **value()** – возвращает значение элемента по ключу
- **values()** – список всех значений

```
QMap<QString, QString> mapPhonebook;  
mapPhonebook["Piggy"] = "785 11 11";  
mapPhonebook["Kermit"] = "123 65 56";  
mapPhonebook["Gonzo"] = "631 32 21";  
QDebug() << mapPhonebook.keys();  
// ["Piggy", "Kermit", "Gonzo"]  
QDebug() << mapPhonebook.values();  
// ["785 11 11", "23 65 56", "631 32 21"]
```

QHash – хэш-таблица

Отличия от QMap


- В QMap элементы отсортированы по ключу, в QHash – произвольно.
- QHash предоставляет более быстрый поиск, чем QMap

```
QHash<QString, QString> hashPhonebook;  
hashPhonebook["Piggy"] = "785 11 11";  
hashPhonebook["Kermit"] = "123 65 56";  
hashPhonebook["Gonzo"] = "631 32 21";  
QDebug() << hashPhonebook.keys();  
// ["Piggy", "Kermit", "Gonzo"]  
QDebug() << hashPhonebook.values();  
// ["785 11 11", "23 65 56", "631 32 21"]
```

QSet - МНОЖЕСТВО

- unite(const QSet<T> &other) – объединение
- intersect(const QSet<T> &other) – пересечение
- subtract(const QSet<T> &other) – разность

```
QSet<QString> set1;  
QSet<QString> set2;  
set1 << "Lorem" << "Ipsum";  
set2 << "Amet" << "Lorem";  
QSet<QString> resultSet = set1;  
resultSet.unite(set2);  
qDebug() << "Unite = " << resultSet.toList();  
// Unite = ["Lorem", "Ipsum", "Amet"]
```



QString – строка СИМВОЛОВ UNICODE

- `arg(T &argument)` – определяет параметр внутри строки
- `QStringList split(QString &sep)` – делит строку на подстроки
- `QString trimmed()` – удаляет пробельные символы из строки
- `startsWith()` – начинается ли строка со строки `s`
- `endsWith()` – заканчивается ли строка со строки `s`
- `toLowerCase()` – возвращает строку в нижнем регистре
- `toUpperCase()` – возвращает строку в верхнем регистре

QString - пример

```
QString fileName = "dict.txt";
QString directory = "files/";
QString status = QString("Processing file %1 from %2").arg(fileName).arg(directory);
// status = "Processing file dict.txt from files/"
QStringList list = status.split(" ");
// list = ["Processing", "file", "dict.txt", "from", "files"]
QString trimmed = status.trimmed(); // trimmed = "Processingfiledict.txtfromfiles/"
QString lower = status.toLowerCase(); // lower = "processing file dict.txt from files/"
QString upper = status.toUpperCase(); // upper = "PROCESSING FILE DICT.TXT FROM FILES/"
bool startsWith = status.startsWith("Process"); // startsWith = true
bool endsWith = status.endsWith("Process"); // endsWith = false
```


QVariant – вариативный тип

- setValue(const T &value) – записать значение
- value() – получить значение
- typeName() – тип хранимого значения
- Приведение к базовым типам:
toBool(), toChar(), toInt(), toDouble(), toFloat()
- Приведение к контейнерным типам:
toByteArray(), toList(), toMap(), toHash(), toString()
- Есть другие методы to...()

```
QVariant var;  
var.setValue(5);  
int int_number = var.toInt(); // int_number = 5  
float float_number = var.toFloat(); // float_number = 5.0  
QString str = var.toString(); // str = "5"
```

Модель списка на C++

1. Создать дочерний класс `QAbstractListModel`
2. Реализовать методы `rowCount()` и `data()`
3. Предоставить `headerData()`
4. Перегрузить метод `roleNames()`, если модель использует роли
5. Реализовать методы `setData()` и `flags()`, если модель редактируемая
6. Реализовать методы `insertRows()` и `removeRows()`, если структура модели редактируемая `insertRows()` вызывает `beginInsertRows()` в начале и `endInsertRows()` в конце
`removeRows()` вызывает `beginRemoveRows()` в начале и `endRemoveRows()` в конце
7. Зарегистрировать класс модели как тип QML

Пример QAbstract ListModel

```
#include <QAbstractListModel>

class DemoModel : public QAbstractListModel
{
    Q_OBJECT

public:
    enum DemoRoles {
        NameRole = Qt::UserRole + 1,
    };
    explicit DemoModel(QObject *parent = 0);
    virtual int rowCount(const QModelIndex&) const { return backing.size(); }
    virtual QVariant data(const QModelIndex &index, int role) const;
    QHash<int, QByteArray> roleNames() const;
    Q_INVOKABLE void activate(const int i);
private:
    QVector<QString> backing;
};
```

Пример QAbstract ListModel

```
DemoModel::DemoModel(QObject *parent) : QabstractListModel(parent)
{
    backing << "sea cow" << "platypus" << "axolotl"
              << "quokka" << "pitahui" << "jerboa";
}
QHash<int, QByteArray> DemoModel::roleNames() const
{
    QHash<int, QByteArray> roles;
    roles[NameRole] = "name";
    return roles;
}
QVariant DemoModel::data(const QModelIndex &index, int role) const
{
    if(!index.isValid())
        return QVariant();
    if(role == NameRole)
        return QVariant(backing[index.row()]);
    return QVariant();
}
```

Пример QAbstract ListModel

```
void DemoModel::activate(const int i)
{
    if (i < 0 || i >= backing.size())
        return;

    QString value = backing[i];

    // Remove the value from the old location.
    beginRemoveRows(QModelIndex(), i, i);
    backing.erase(backing.begin() + i);
    endRemoveRows();

    // Add it to the top.
    beginInsertRows(QModelIndex(), 0, 0);
    backing.insert(0, value);
    endInsertRows();
}
```



True Engineering

630128, г. Новосибирск,
ул. Кутателадзе, 4г

(383) 363-33-51, 363-33-50
info@trueengineering.ru
trueengineering.ru

**Новосибирский
Государственный
Университет**