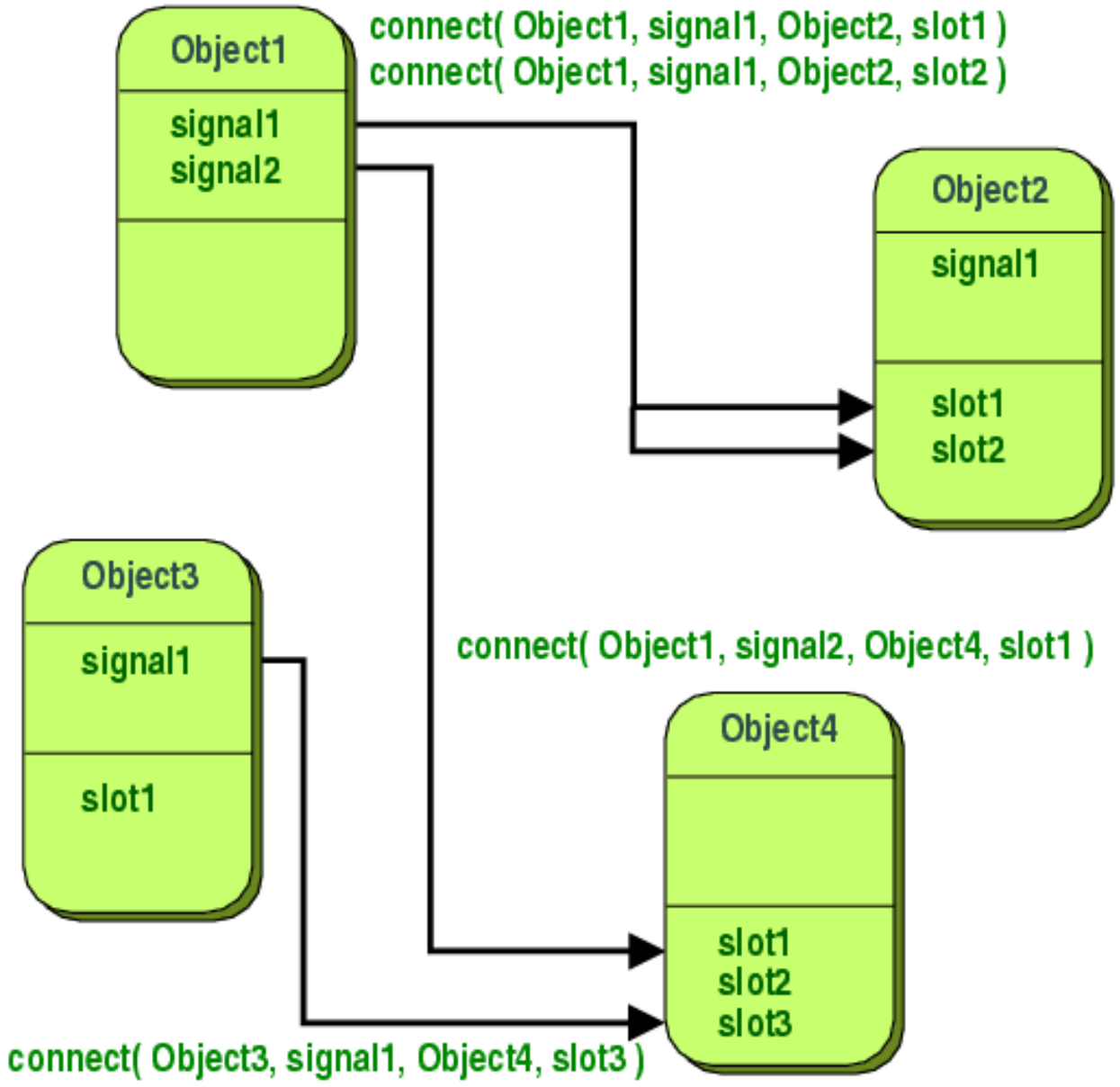
Qt

Сигналы и слоты (signals and slots)

Qt использует методику сигналов и слотов для уведомления об изменении состояния некого объекта и последующем изменении если потребуется.

Сигнал испускается, когда происходит конкретное событие. Слот - это функция, которая вызывается в ответ на конкретный сигнал.



Сигналы и слоты слабо связаны: класс, отправляющий сигнал, не знает и не заботится о том, какие слоты принимают сигнал. Механизм сигналов и слотов Qt гарантирует, что если вы подключите сигнал к слоту, то слот будет вызван с

параметрами сигнала в нужное время. Сигналы и слоты могут принимать любое количество аргументов любого типа.

Все классы, наследуемые от QObject или одного из его подклассов (например, QWidget), могут содержать сигналы и слоты. Сигналы испускаются объектами, когда они изменяют своё состояние таким образом, что это может быть интересно другим объектам. Это всё, что делает объект для взаимодействия. Он не знает и не заботится о том, получает ли что-либо испускаемые им сигналы. Это настоящая инкапсуляция информации, которая позволяет использовать объект как программный компонент.

Слот-методы можно использовать для получения сигналов, но они также являются обычными функциями-членами. Подобно тому, как объект не знает, получает ли что-либо его сигналы, слот-метод не знает, подключены ли к нему какие-либо сигналы. Это позволяет создавать по-настоящему независимые компоненты с помощью Qt.

Сигналы

Сигналы испускаются объектом, когда его внутреннее состояние каким-то образом изменилось, что может быть интересно для клиента или владельца объекта. Сигналы являются функциями общего доступа и могут быть излучены из любого места.

Когда поступает сигнал, подключенные к нему слоты обычно выполняются немедленно, как при обычном вызове функции. Когда это происходит, механизм сигналов и слотов полностью независим от любого цикла обработки событий в графическом интерфейсе. Выполнение кода, следующего за инструкцией emit, произойдет после того, как все слоты будут возвращены.

Если несколько слотов подключены к одному сигналу, они будут выполняться один за другим в порядке подключения, когда будет подан сигнал.

Сигналы автоматически генерируются с помощью компилятора мета объектов moc и не должны быть реализованы в файле `.cpp`. Они никогда не могут иметь возвращаемые типы (т. е. использовать `void`).

Слоты

Слот вызывается, когда генерируется подключённый к нему сигнал. Слоты — это обычные функции C++, которые можно вызывать обычным образом; их

единственная особенность заключается в том, что к ним можно подключать сигналы.

Поскольку слоты являются обычными функциями-членами, при прямом вызове они подчиняются обычным правилам C++. Однако в качестве слотов они могут быть вызваны любым компонентом, независимо от уровня доступа, через соединение «сигнал-слот». Это означает, что сигнал, отправленный экземпляром произвольного класса, может вызвать вызов закрытого слота в экземпляре несвязанного класса.

Вы также можете сделать слоты виртуальными, что оказалось весьма полезным на практике.

Сигналы и слоты работают немного медленнее из-за большей гибкости, которую они обеспечивают, хотя для реальных приложений разница незначительна. В целом, отправка сигнала, подключённого к нескольким слотам, примерно в десять раз медленнее, чем прямой вызов получателей с помощью невиртуальных функций. Это связано с затратами на поиск объекта подключения, безопасную перемотку всех подключений (т. е. проверку того, что последующие получатели не были уничтожены во время отправки) и обобщённую обработку любых параметров. Хотя десять вызовов невиртуальных функций могут показаться большим количеством, это гораздо меньше, чем, например, любая операция `new` или `delete` . Как только вы выполняете операцию со строкой, вектором или списком, которая в глубине требует `new` или `delete` , накладные расходы на сигналы и слоты составляют лишь очень малую долю от полной стоимости вызова функции. То же самое происходит, когда вы выполняете системный вызов в слоте или косвенно вызываете более десяти функций. Простота и гибкость механизма сигналов и слотов вполне оправдывают дополнительные затраты, которые ваши пользователи даже не заметят.

Пример

Минимальное объявление класса на C++ может выглядеть так:

```
class Counter
{
public:
    Counter() { m_value = 0; }

    int value() const { return m_value; }
    void setValue(int value);
```

```
private:
    int m_value;
};
```

Небольшой класс на основе QObject может выглядеть так:'

```
#include <QObject>

class Counter : public QObject
{
 Q_OBJECT

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;
};
```

Версия на основе QObject имеет такое же внутреннее состояние и предоставляет общедоступные методы для доступа к состоянию, но, кроме того, она поддерживает программирование компонентов с использованием сигналов и слотов. Этот класс может сообщать внешнему миру об изменении своего состояния, отправляя сигнал `valueChanged()`, и у него есть слот, в который другие объекты могут отправлять сигналы.

Все классы, содержащие сигналы или слоты, должны содержать Q_OBJECT в верхней части своего объявления. Они также должны быть (прямо или косвенно) производными от QObject.

Слоты реализуются программистом приложения. Вот возможная реализация слота `Counter::setValue()`:

```
void Counter::setValue(int value)
{
    if (value != m_value) {
 m_value = value;
        emit valueChanged(value);
    }
}
```

Строка `emit` посылает объекту сигнал `valueChanged()` с новым значением в качестве аргумента.

В следующем фрагменте кода мы создаём два объекта `Counter` и подключаем сигнал `valueChanged()` первого объекта к слоту `setValue()` второго объекта с помощью QObject::connect():

```
Counter a, b;
QObject::connect(&a, &Counter::valueChanged,
                 &b, &Counter::setValue);

a.setValue(12);     // a.value() == 12, b.value() == 12
b.setValue(48);     // a.value() == 12, b.value() == 48
```

Вызов `a.setValue(12)` приводит к тому, что `a` посылает `valueChanged(12)` сигнал, который `b` получит в своём `setValue()` слоте, то есть `b.setValue(12)` вызывается. Затем `b` посылает тот же `valueChanged()` сигнал, но, поскольку ни один слот не был подключен к `b` сигналу `valueChanged()`, сигнал игнорируется.

Обратите внимание, что функция `setValue()` устанавливает значение и генерирует сигнал только в том случае, если `value != m_value`. Это предотвращает бесконечный цикл в случае циклических соединений (например, если `b.valueChanged()` подключено к `a.setValue()`).

Этот пример показывает, что объекты могут работать вместе, не зная друг о друге никакой информации. Для этого объекты нужно только соединить между собой, что можно сделать с помощью нескольких простых вызовов функции QObject::connect()

MOC (компилятор метаобъектов)

Компилятор метаобъектов `moc` — это программа, которая обрабатывает расширения C++ в Qt.

Инструмент `moc` считывает заголовочный файл C++. Если он находит одно или несколько объявлений классов, содержащих макрос Q_OBJECT, он создает исходный файл C++, содержащий код метаобъектов для этих классов. Помимо прочего, код метаобъектов необходим для механизма сигналов и слотов, информации о типах во время выполнения и системы динамических свойств.

Исходный файл C++, сгенерированный `moc` должен быть скомпилирован и связан с реализацией класса. `moc` анализирует заголовочный файл и **генерирует C++ код**, реализующий вспомогательные структуры (таблицы, вызовы методов, конструкторы сигналов и слотов и т.п.).

Если вы используете qmake для создания своих make-файлов, будут включены правила сборки, которые при необходимости вызывают moc, поэтому вам не нужно будет использовать moc напрямую.

Источники

- Сигналы и слоты
- MOC