Система событий Qt

События возникают в операционной системе по мере работы приложения. Они являются результатом действий пользователя, либо работы других приложений.

Событие содержит в себе данные о том, что произошло.

При получении события от операционной системы, приложение Qt преобразует его один из классов, производных от базового QEvent, и помещает его в очередь событий Qt внутри класса QApplication. Оттуда событие попадает к конкретному объекту, где оно должно быть обработано.

Некоторые события могут быть перенаправлены другим объектам, если они не обрабатываются целевым объектом.

При необходимости объекты Qt могут сами порождать события.

Классы событий Qt

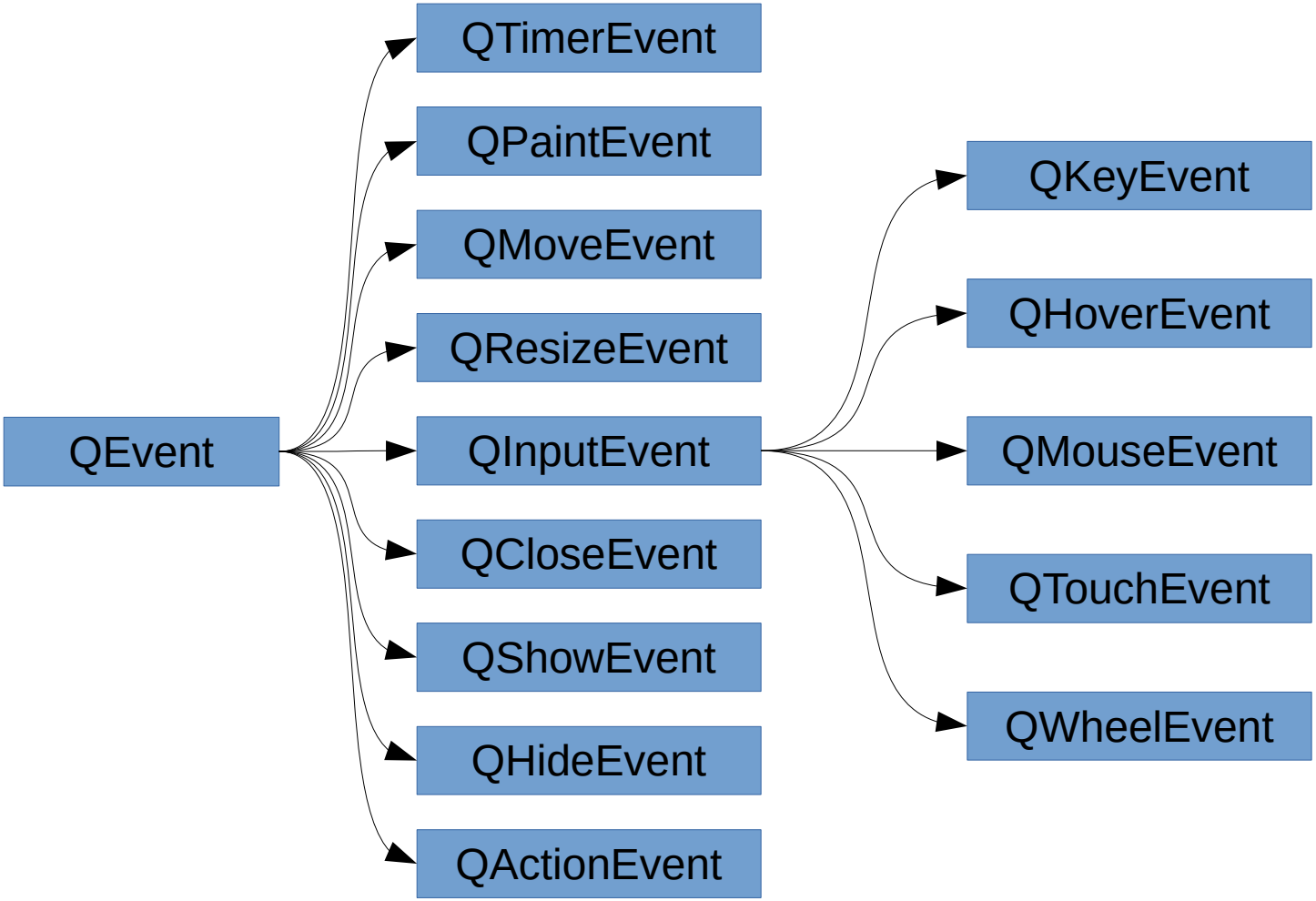
QEvent — базовый класс события. Содержит следующие поля:

- **spontaneous** — показывает, пришло ли событие от операционной системы.
- **accepted** — следует ли прекратить дальнейшее распространение события.
- **type** — тип возникшего события (**QEvent::Type**).

Каждому типу событий соответствует определенный класс, хранящий данные, специфичные для этого события.

Нескольким типам может соответствовать один и тот же класс. Например, типам **QEvent::KeyPress** и **QEvent::KeyRelease** соответствует класс **QKeyEvent**.

Классы событий Qt



Пользовательские события

Тип QEvent::Type является целочисленным типом и содержит стандартные события Qt. Все они уложены в диапазон от 0 до **QEvent::User** (1000). Номера от QEvent::User до **QEvent::MaxUser** (65535) могут использоваться произвольным образом. Для этого нужно:

- Создать новый класс, производный от QEvent, если требуется хранит какие либо данные в событии.
- Вызывать метод **QEvent::registerEventType**, чтобы получить для события уникальный номер от QEvent::User до QEvent::MaxUser.
- Создать экземпляр QEvent, либо производного класса.
- Отправить событие, используя методы QCoreApplication::postEvent (для отложенной обработки) или QCoreApplication::sendEvent (для немедленной обработки).

Обработка событий Qt

Каждый QObject содержит метод **event()** для обработки событий. Этот метод определяет тип события и вызывает конкретный обработчик, передав ему объект события, приведенный к классу, соответствующему типу события.

```
class QWidget : public QObject {
public:
    virtual bool event(QEvent* e) {
        if (e->type() == QEvent::KeyPressEvent) {
            keyPressEvent(dynamic_cast<QKeyEvent*>(e));
            return true;
        }
    }

protected:
    void keyPressEvent(QKeyEvent* ke);
}
```

Обработка событий Qt

При необходимости, производный класс может переопределить метод event для самостоятельной обработки событий. Также можно вызвать родительский вариант метода для обработки остальных событий:

```
class EventParser : public QObject {
public:
    virtual bool event(QEvent* e) {
        if (e->type() == QEvent::User) {
            // < Обработка нестандартного события >
        }
        return QObject::event(e);
    }
}
```

Возвращаемое значение event — переменная типа bool, обозначающая, закончена ли обработка события.

Специализированные обработчики

Специализированные методы-обработчики не имеют возвращаемого значения и предназначены для обработки конкретного события. В большинстве случаев достаточно только переопределить такой обработчик.

Событие может «всплыть», то есть быть передано родительскому объекту для дальнейшей обработки. Для контроля всплытия QEvent содержит два метода:

- **accept** — отмечает событие как полностью обработанное, исключая всплытие.
- **ignore** — отмечает событие как не обработанное, позволяя всплытие.

Перехват событий

При необходимости можно перехватить событие, предназначенное другому объекту. Для перехвата используется метод **eventFilter**. События, предназначенные объекту, будут проходить сначала через eventFilter указанного, если вызвать для объекта метод **installEventFilter**. Дальнейшая обработка аналогична методу event.

```
EventParser::EventParser(QObject* parent) {
    watch->installEventFilter(this);
}

bool EventParser::eventFilter(QObject* o, QEvent* e) {
    if (o == watch && e->type() == QEvent::KeyPress) {
        return true;     // Запрет событий клавиатуры
    }
    return QObject::eventFilter(o, e);
}
```

События клавиатуры

События клавиатуры представлены классом **QKeyEvent** и создаются только для виджета, который находится в фокусе. Неактивный виджет не получает событий клавиатуры.

- **QEvent::KeyPress** — возникает при нажатии на клавишу. Обрабатывается в **keyPressEvent**.
- **QEvent::KeyRelease** — возникает при отпускании клавиши. Обрабатывается в **keyReleaseEvent**.

Событие, не обработанное в дочернем виджете, отправляется родительскому.

События клавиатуры

Методы QKeyEvent:

- **int count()** — возвращает количество одновременно нажатых клавиш, породивших событие.
- **int key()** — возвращает код нажатой клавиши из типа Qt::Key. Например, Qt::Key_Up для кнопки «Стрелка вверх».
- **QString text()** — текст, созданный данным событием.
- **Qt::KeyboardModifiers modifiers()** — возвращает модификаторы клавиатуры, при которых случилось событие. Возвращаемое значение — целое число, полученное битовой комбинацией констант:
  - **Qt::ShiftModifier** — была нажата клавиша Shift.
  - **Qt::ControlModifier** — была нажата клавиша Ctrl.
  - **Qt::AltModifier** — была нажата клавиша Alt.

События мыши

События мыши представлены классом **QMouseEvent** и создаются для любого активного виджета при работе мыши.

- **QEvent::MouseButtonPress** — возникает при нажатии на клавишу. Обрабатывается в **mousePressEvent**.
- **QEvent::MouseButtonRelease** — возникает при отпускании клавиши. Обрабатывается в **mouseReleaseEvent**.
- **QEvent::MouseMove** — возникает при перемещении мыши по виджету. Обрабатывается в **mouseMoveEvent**. Не высылается, если мышь перемещается без нажатых кнопок, а у виджета отключено свойство mouseTracking.
- **QEvent::MouseButtonDblClick** — возникает при двойном нажатии кнопки мыши. Обрабатывается в **mouseDoubleClickEvent**.

Событие, не обработанное в дочернем виджете, отправляется родительскому.

События мыши

Методы QMouseEvent:

- **Qt::MouseButton button()** — возвращает клавишу мыши, нажатие которой породило событие:
  - **Qt::NoButton** — ни одна клавиша не была нажата. Такое значение передается при **QEvent::MouseMove**.
  - **Qt::LeftButton** — была нажата или отпущена левая кнопка.
  - **Qt::RightButton** — то же для правой кнопки.
  - **Qt::MiddleButton** — то же для средней кнопки.
- **Qt::MouseButtons buttons()** — возвращает набор кнопок, при которых было порождено событие, в виде побитовой комбинации предыдущих констант. При **QEvent::MouseButtonPress** и **QEvent::MouseMove** содержит нажатые кнопки, при **QEvent::MouseButtonRelease** — все кроме отпущенных.

События мыши

Методы QMouseEvent:

- **QPoint pos()**, **QPointF& localPos()** — содержит координаты события мыши в координатах виджета.
- **int x()**, **int y()** — содержит отдельные составляющие координат, возвращаемых вызовом pos().
- **QPoint globalPos()**, **int globalX()**, **int globalY()**, **QPointF& windowPos()** — координаты события, пересчитанные относительно корневого виджета (окна верхнего уровня).
- **QPointF& screenPos()** — координаты события, пересчитанные относительно экрана.

События колеса мыши

События колеса мыши представлены классом **QWheelEvent** и создаются для любого активного виджета при вращении колеса мыши.

Тип события — **QEvent::Wheel**.

Обрабатывается в **wheelEvent**.

События колеса мыши

Методы QWheelEvent:

- **Qt::MouseButtons buttons()** — возвращает набор кнопок, при которых было порождено событие, в виде побитовой комбинации констант из Qt::MouseButton.
- **QPoint pos()**, **QPointF posF()** — содержит координаты события мыши в координатах виджета.
- **int x()**, **int y()** — содержит отдельные составляющие координат, возвращаемых вызовом pos().
- **QPoint globalPos()**, **int globalX()**, **int globalY()**, **QPointF& globalPosF()** — координаты события, пересчитанные относительно корневого виджета (окна верхнего уровня).

События колеса мыши

Методы QWheelEvent:

- **QPoint angleDelta()** — возвращает угол поворота колеса по обоим осям в единицах, равных 1/8 градуса. Положительное направление — вращение от пользователя. Колесо типовой мыши останавливается каждые 15 градусов (120 единиц).
- **QPoint pixelDelta()** — возвращает размер прокрутки в пикселях для тех платформ, которые это поддерживают.
- **Qt::MouseEventSource source()** — возвращает источник прокрутки. Одна из констант:
  - **Qt::MouseEventNotSynthesized** — классическое событие мыши, полученное от вращения колеса.
  - **Qt::MouseEventSynthesizedBySystem** — событие колеса синтезировано системой из события касания экрана.
  - **Qt::MouseEventSynthesizedByQt** — то же самое для Qt.

События касания

События касания представлены классом **QTouchEvent**. Они отправляются в виджет только в том случае, если в виджете включен флаг **Qt::WA_AcceptTouchEvents**. В противном случае виджет получает обычные события мыши.

```
ui->widget->setAttribute(Qt::WA_AcceptTouchEvents);
```

События касания не имеют специального обработчика и должны обрабатываться через **event** или **eventFilter**.

- **QEvent::TouchBegin** — возникает при касании пальцем. Событие должно быть обработано виджетом (return true), иначе остальные события будут проигнорированы.
- **QEvent::TouchUpdate** — возникает при перемещении пальца по экрану.
- **QEvent::TouchEnd** — возникает при убирании всех пальцев с экрана.
- **QEvent::TouchCancel** — отмена всех событий касания.

События касания

QTouchEvent содержит один интересующий метод: **QList<QTouchEvent::TouchPoint>& touchPoints()**. Метод возвращает список точек касания, породивших данное событие.

Методы класса QTouchEvent::TouchPoint:

- **QPointF pos()** — возвращает позицию точки касания.
- **double pressure()** — возвращает силу нажатия.
- **double rotation()** — возвращает угол поворота пятна контакта.
- **QSizeF ellipseDiameters()** — возвращает диаметры пятна контакта.
- **QVector2d velocity()** — возвращает скорость перемещения точки.
- **Qt::touchPointState state()** — возвращает состояние точки. По нему можно определить, двигалась ли конкретное пятно контакта.

Обработка событий

Для обработки событий необходимо:

- Создать класс, который должен обрабатывать события особым образом.
- Переопределить в классе методы специализированной обработки (например, keyPressEvent) для получения соответствующих событий. Если необходимо, чтобы событие не обрабатывалось дальше — вызвать у события метод accept.
- При отсутствии специального метода — переопределить метод event и дописать в нем необходимый функционал. В конце метода event вызвать метод event родительского класса.

Либо:

- Определить метод eventFilter в объекте-перехватчике.
- Вызвать метод installEventFilter у того объекта, события которого нужно обработать особым образом. Аргумент метода — указатель на объект-перехватчик.