Тема 2. Событийно-ориентированное программирование.

Лекция 4. Обработка событий средствами Qt.

Обо мне

- ФОТО

- Харченко Владислав Алексеевич, 26 лет.

- 2012 – 2017 гг. ВГТУ. Факультет Радиотехники и Электроники.
- 2014 – 2018 гг. Фриланс.
- 2017 – 2018 гг. АО «НИИ СВТ». Программист.
- 2018 – 2021 гг. АО «НИИ СВТ». Старший программист.

Учебные вопросы

1. Классы событий
2. Перехват событий
3. Фильтр событий

Источники

- Официальная документация: https://doc.qt.io/qtforpython
- Прохоренок Н. А., Дронов В. А. Python 3 и PyQt 5. Разработка приложений. 2019 г.

Используемые в курсе инструменты для разработки		
IDE	PyCharm CE	https://www.jetbrains.com/pycharm/download
Окружение	Virtualenv	https://docs.python.org/3/library/venv.html
VSC (рекомендовано)	GIT	https://git-scm.com
Фреймворк	PySide2	https://doc.qt.io/qtforpython/

1. Классы событий

Основной цикл приложения

- `app = QtWidgets.QApplication()`
  `app.exec_()` данная конструкция запускает основной цикл приложения и в процессе выполнения извлекает системные события собственного окна из очереди событий.
- События как правило поступают автоматически при их вызове. Однако существуют методы для их отправки вручную (postEvent() и sendEvent())
- Получает события стандартный метод event(). Для перехвата событий метод event() может быть переопределён.

Классы QEvent

- Список всех классов QEvent:

QActionEvent, QChildEvent, QCloseEvent, QDragLeaveEvent, QDropEvent, QDynamicPropertyChangeEvent, QEnterEvent, QExposeEvent, QFileOpenEvent, QFocusEvent, QGestureEvent, QGraphicsSceneEvent, QHelpEvent, QHideEvent, QIconDragEvent, QInputEvent, QInputMethodEvent, QInputMethodQueryEvent, QMoveEvent, QPaintEvent, QPlatformSurfaceEvent, QResizeEvent, QScrollEvent, QScrollPrepareEvent, QShortcutEvent, QShowEvent, QStateMachine:: SignalEvent, QStateMachine:: WrappedEvent, QStatusTipEvent, QTimerEvent, QWhatsThisClickedEvent, QWhatsThisClickedEvent и QWhatsThisClickedEvent

2. Перехват событий

Методы класса QEvent

- **accept()** – разрешает дальнейшую обработку события;
- **ignore()** – запрещает дальнейшую обработку события;
- **setAccepted(bool)** – разрешает дальнейшую обработку;
- **isAccepted()** – возвращает состояние события;
- **spontaneous()** – возвращает True, если событие сгенерировано системой, False, если внутри программы;
- **type()** – возвращает тип события;

Переопределение метода event()

```
def event(self, event: QtCore.QEvent) -> bool:
    print(event.type())
```

```
PySide2.QtCore.QEvent.Type.Move
PySide2.QtCore.QEvent.Type.PlatformSurface
PySide2.QtCore.QEvent.Type.WinIdChange
PySide2.QtCore.QEvent.Type.WindowIconChange
PySide2.QtCore.QEvent.Type.Polish
PySide2.QtCore.QEvent.Type.Move
PySide2.QtCore.QEvent.Type.Resize
PySide2.QtCore.QEvent.Type.Show
PySide2.QtCore.QEvent.Type.CursorChange
PySide2.QtCore.QEvent.Type.ShowToParent
PySide2.QtCore.QEvent.Type.PolishRequest
PySide2.QtCore.QEvent.Type.UpdateLater
PySide2.QtCore.QEvent.Type.UpdateRequest
PySide2.QtCore.QEvent.Type.WindowActivate
PySide2.QtCore.QEvent.Type.ActivationChange
PySide2.QtCore.QEvent.Type.InputMethodQuery
PySide2.QtCore.QEvent.Type.Paint
```

События при запуске простого приложения

Примечание:
Метод event() является более приоритетным при перехвате событий.

Пример:
В случае переопределения метода event() и closeEvent() в коде приложения, метод closeEvent() срабатывать не будет, а событие будет перехвачено методом event()

```
def closeEvent(self, event):
    event.ignore()
```
вызываться не будет

```
def event(self, event: QtCore.QEvent) -> bool:
    print(event.type())
```

*выход из ситуации:

```
def event(self, event: QtCore.QEvent) -> bool:
    print(event.type())
    if event.type() == QtCore.QEvent.Type.Close:
        event.ignore()
```

Переопределение метода event() (продолжение)

Правильный вариант, если хотим переопределить не только метод event(), но и задать дополнительное поведение, переопределив метод другого конкретного события.

```
def closeEvent(self, event):
    print("work closeEvent")
    event.accept()

def event(self, event: QtCore.QEvent) -> bool:
    print(event.type())
    if event.type() == QtCore.QEvent.Type.Close:
        event.setAccepted(False)
    return QtWidgets.QWidget.event(self, event)
```

Какой будет результат выполнения данного фрагмента кода?

3. Фильтр событий

Метод eventFilter()

- Данный метод предоставляет возможность одного экземпляра QObject отслеживать события, предназначенные для другого экземпляра QObject до того как последний получит их.
- Так же позволяет определять нестандартное поведение для конкретного экземпляра QObject путем отбора необходимых событий и экземпляров.

Пример:

```
def eventFilter(self, watched: QtCore.QObject, event: QtCore.QEvent) -> bool:

    return super(MyEventHandler, self).eventFilter(watched, event)
```

Чем eventFilter() отличается от event()?

Установка фильтра событий

1. Регистрируем фильтр событий, вызовом метода installEventFilter() у того объекта, которому предназначены события:

```
self.someButton.installEventFilter(self)
```

2. Создаём обработчик перехваченных событий eventFilter():

```
def eventFilter(self, watched: QtCore.QObject, event: QtCore.QEvent) -> bool:
    if watched == self.button1 and event.type() == QtCore.QEvent.MouseButtonDblClick:
        print("DblMouseClck")
    elif watched == self.button2 and event.type() == QtCore.QEvent.KeyPress:
        print("Key pressed")

    return super(MyEventHandler, self).eventFilter(watched, event)
```

Примечание:
Если событие не было обработано по пути к объекту назначения, или самим объектом, то процесс обработки события повторяется, но на этот раз объектом назначения становится виджет-владелец. Так продолжается до тех пор, пока событие не будет обработано, либо пока событие не достигнет виджет самого верхнего уровня. (Пример: событие наведения мыши или активации фокуса ввода)

Спасибо за внимание!