PyQt5

PyQt5

```
Base of Qt features
   My Fist Application
   My Awesome Application
   Signals and Slots
   Toolbars 、Statusbar and Menus
   Actions
   Widgets
       CheckBox Widget
       ComboBox Widget
       Label Widget
       LineEdit Widget
       List Widget
       Tab Widget
   Layouts
       Horizontal Layout
       Vertical Layout
       Nested Layout
       Margins and Spacing
       Grid Layout
       Stacked Layout
    Dialog
       Custom Dialog
```

Base of Qt features

My Fist Application

• 导入必要模块

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *
```

• 创建应用实例

```
1 # 创建应用实例
2 app = QApplication(sys.argv)
```

• 创建窗口实例

```
1 # 创建窗口实例
2 window = QMainWindow()
```

• 显示窗口

```
1 # 显示窗口
2 window.show()
```

• 应用事件循环

```
1  # 应用事件循环
2  app.exec_()
```

运行程序会出现下面窗口:



My Awesome Application

下面我们为上面的窗口添加窗口标题和在窗口上显示文字标签。

• 设置窗口标题

```
1 # 设置窗口标题
2 self.setWindowTitle('My Awesome App')
```

• 创建文字标签

```
1 | lbl = QLabel('This is awesome!!!', self)
```

• 设置标签对齐方式,设置为居中对齐

```
1 # 设置标签对齐方式,Qt.AlignCenter是中心对齐
2 lbl.setAlignment(Qt.AlignCenter)
```

• 在窗口中心显示标签

```
1 # 在窗口中心设置标签
2 self.setCentralWidget(lbl)
```

```
1 import sys
2 from PyQt5.QtWidgets import *
 3 from PyQt5.QtCore import *
  from PyQt5.QtGui import *
 5
6
7
   class MainWindow(QMainWindow):
8
      def __init__(self, *args, **kwargs):
9
           # 继承QMainWindow的__init___方法
           super().__init__(*args, **kwargs)
10
11
           # 设置窗口标题
12
13
          self.setWindowTitle('My Awesome App')
14
           lbl = QLabel('This is awesome!!!', self)
15
16
           # 设置标签对齐方式,Qt.AlignCenter是中心对齐
           lbl.setAlignment(Qt.AlignCenter)
17
           # 在窗口中心设置标签
18
19
          self.setCentralWidget(lbl)
20
21
    # 创建应用实例,传入命令参数列表
22
   app = QApplication(sys.argv)
23
24
   # 创建窗口实例
25
26 | window = MainWindow()
    # 显示窗口
27
   window.show()
28
29
```



Signals and Slots

下面让我们来了解一下信号与槽的概念。信号可以理解为我们用户用键盘或者鼠标的任何动作,可以将这个信号捕捉,交给程序处理也就是交给槽,再反馈给用户想看到的内容,这就形成了人与电脑交互。

• 将窗口标题改变的信号与槽(我们创建的函数)相连

```
1 # 信号: self.windowTitleChanged
2 # 连接函数方法: .connect()
3 # 槽: self.onWindowTitleChanged
4 self.windowTitleChanged.connect(self.onWindowTitleChanged)
```

• 重写文本菜单事件

```
      1
      # 当在窗口右击时,执行以下函数

      2
      def contextMenuEvent(self, event):

      3
      print('Context menu event.')

      4
      继承原有文本菜单事件的功能

      5
      super().contextMenuEvent(event)
```

```
import sys
 2
   from PyQt5.QtWidgets import *
    from PyQt5.QtCore import *
    from PyQt5.QtGui import *
 4
 5
 6
    class MainWindow(QMainWindow):
 7
 8
        def __init__(self, *args, **kwargs):
 9
            super().__init__(*args, **kwargs)
10
11
            self.windowTitleChanged.connect(self.onWindowTitleChanged)
12
13
            self.windowTitleChanged.connect(lambda x: self.my_custom_fn())
14
15
            self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x))
16
17
            self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x, 25))
18
            self.setWindowTitle('My Awesome App')
19
20
            lbl = QLabel('This is Awesome!!!')
21
            lbl.setAlignment(Qt.AlignCenter)
22
            self.setCentralWidget(lbl)
23
        def onWindowTitleChanged(self, s):
24
25
            print(s)
26
27
        def my_custom_fn(self, a='Hello', b=5):
28
            print(a, b)
29
30
        def contextMenuEvent(self, event):
31
            print('Context menu event.')
32
            super().contextMenuEvent(event)
33
```

```
34
35   app = QApplication(sys.argv)
36   window = MainWindow()
37   window.show()
38   app.exec_()
```



终端出现以下情况:

My Awesome App Hello 5 My Awesome App 5 My Awesome App 25

在窗口中右击,则出现以下情况:

Context menu event.

Toolbars 、**Statusbar and Menus**

下面让我们来了解一下窗口的必要的元素:工具栏、状态栏和菜单栏。

• 创建工具栏实例

```
1 | toolbar = QToolBar('My Main Toolbar')
```

• 设置工具栏里面工具的图标大小

```
1 | self.setIconSize(QSize(16, 16))
```

• 设置工具栏里面工具图标的显示风格

```
1 | self.setToolButtonStyle(Qt.ToolButtonTextBesideIcon)
```

• 在窗口上添加工具栏

```
1 | self.addToolBar(toolbar)
```

• 创建状态栏实例

```
1 | statusbar = QStatusBar(self)
```

• 在窗口上添加状态栏

```
1 | self.setStatusBar(statusbar)
```

• 创建菜单栏

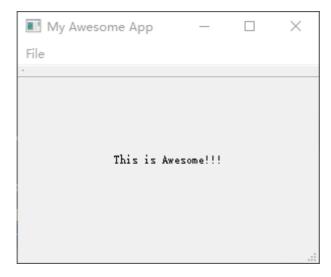
```
1 | menu = self.menuBar()
```

• 在菜单栏上添加第一个菜单

```
1 | file_menu = menu.addMenu('&File')
```

```
1 | import sys
 2 from PyQt5.QtWidgets import *
 3 from PyQt5.QtCore import *
    from PyQt5.QtGui import *
 4
 5
 6
 7
    class MainWindow(QMainWindow):
        def __init__(self, *args, **kwargs):
 8
 9
            super().__init__(*args, **kwargs)
10
11
            self.windowTitleChanged.connect(self.onWindowTitleChanged)
12
13
            self.windowTitleChanged.connect(lambda x: self.my_custom_fn())
```

```
14
15
            self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x))
16
            self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x, 25))
17
18
19
            self.setWindowTitle('My Awesome App')
20
            self.setGeometry(300, 300, 300, 220)
21
22
            1b1 = QLabel('This is Awesome!!!')
23
            lbl.setAlignment(Qt.AlignCenter)
            self.setCentralWidget(lbl)
24
25
26
            toolbar = QToolBar('My Main Toolbar')
27
            # 设置工具栏图标大小
28
            self.setIconSize(QSize(16, 16))
29
            # 设置工具栏按钮风格
30
            self.setToolButtonStyle(Qt.ToolButtonTextBesideIcon)
31
            self.addToolBar(toolbar)
32
33
            statusbar = QStatusBar(self)
            self.setStatusBar(statusbar)
34
35
36
            # add menuBar and add menu on menuBar
            menu = self.menuBar()
37
38
            file_menu = menu.addMenu('&File')
39
        def onWindowTitleChanged(self, s):
40
41
            print(s)
42
43
        def my_custom_fn(self, a='Hello', b=5):
            print(a, b)
44
45
        def contextMenuEvent(self, event):
46
47
            print('Context menu event.')
48
            super().contextMenuEvent(event)
49
50
    app = QApplication(sys.argv)
51
    window = MainWindow()
52
53
    window.show()
    app.exec_()
```



Actions

下面让我们来看一下动作按钮,这是在工具栏和菜单中的按钮。

• 添加第一个动作按钮

```
1 # 创建动作按钮实例,第一个参数传入按钮图标,第二个参数传入按钮名称,第三个参数传入窗口实例
2
   button_action = QAction(QIcon('../icons/16/045.png'), 'Home', self)
3
   # 设置按钮状态提示, 当鼠标悬浮在按钮上时, 状态栏会出现输入文字
4 button_action.setStatusTip('This is action.')
   # 将动作按钮触发信号连接槽(自己所写函数)
6 | button_action.triggered.connect(self.onMyToolBarButtonClick)
7
   # 设置动作按钮为可检查模式,即点击为True,再次点击为False。如果没有此行,点击此动作按钮会
   出现终端显示的按钮状态一直是False
8 button_action.setCheckable(True)
   # 设置动作按钮快捷键,尽量使用QKeySequence(快捷键组合)方法作为参数传入
10 button_action.setShortcut(QKeySequence('Ctrl+h'))
11 # 将此动作按钮加入工具栏中
12 toolbar.addAction(button_action)
```

• 在菜单中加入动作按钮

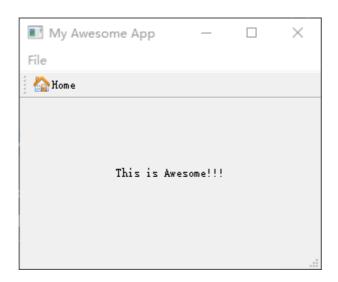
```
1 | file_menu.addAction(button_action)
```

• 创建与动作按钮连接的槽函数

```
1 def onMyToolBarButtonClick(self, s):
2 print('click', s)
```

```
1 | import sys
    from PyQt5.QtWidgets import *
 2
    from PyQt5.QtCore import *
 4
    from PyQt5.QtGui import *
 5
 6
 7
    class MainWindow(QMainWindow):
        def __init__(self, *args, **kwargs):
 8
 9
            super().__init__(*args, **kwargs)
10
11
            self.windowTitleChanged.connect(self.onWindowTitleChanged)
12
13
            self.windowTitleChanged.connect(lambda x: self.my_custom_fn())
14
15
            self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x))
16
17
            self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x, 25))
18
19
            self.setWindowTitle('My Awesome App')
            self.setGeometry(300, 300, 300, 220)
20
21
            lbl = QLabel('This is Awesome!!!')
22
23
            lbl.setAlignment(Qt.AlignCenter)
24
            self.setCentralWidget(lbl)
```

```
25
26
            toolbar = QToolBar('My Main Toolbar')
27
            # 设置工具栏图标大小
28
            self.setIconSize(QSize(16, 16))
29
            # 设置工具栏按钮风格
30
            self.setToolButtonStyle(Qt.ToolButtonTextBesideIcon)
31
            self.addToolBar(toolbar)
32
33
            statusbar = QStatusBar(self)
34
            self.setStatusBar(statusbar)
35
36
            button_action = QAction(QIcon('../icons/16/045.png'), 'Home', self)
37
            button_action.setStatusTip('This is action.')
            button_action.triggered.connect(self.onMyToolBarButtonClick)
38
39
            button_action.setCheckable(True)
40
            button_action.setShortcut(QKeySequence('Ctrl+h'))
41
            toolbar.addAction(button_action)
42
            # add menuBar and add menu on menuBar
43
44
            menu = self.menuBar()
45
            file_menu = menu.addMenu('&File')
            file_menu.addAction(button_action)
46
47
48
49
        def onMyToolBarButtonClick(self, s):
            print('click', s)
50
51
52
        def onWindowTitleChanged(self, s):
53
            print(s)
        def my_custom_fn(self, a='Hello', b=5):
55
56
            print(a, b)
57
        def contextMenuEvent(self, event):
58
59
            print('Context menu event.')
60
            super().contextMenuEvent(event)
61
62
63
    app = QApplication(sys.argv)
64
    window = MainWindow()
    window.show()
65
    app.exec_()
```



点击工具栏上的Home或者点击菜单栏上File里面的Home,在或者按 Ctr1+h 终端会出现以下情况:

click True

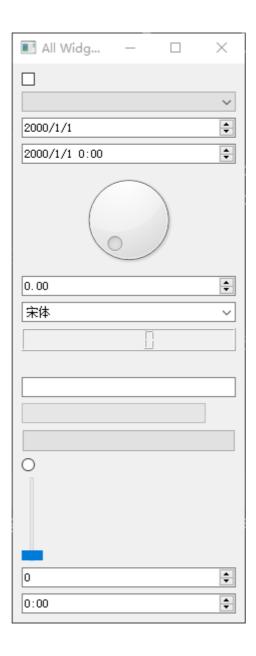
再次点击,则终端会出现以下情况:

click True click False

Widgets

• 下面代码展示了Qt里大多数部件

```
1 import sys
2
    from PyQt5.QtWidgets import *
 3
    from PyQt5.QtCore import *
4
    from PyQt5.QtGui import *
 5
 6
7
    class MainWindow(QMainWindow):
8
        def __init__(self, *args, **kwargs):
9
           super().__init__(*args, **kwargs)
10
11
           self.init_ui()
12
13
       def init_ui(self):
14
           # self.setGeometry(300, 300, 300, 220)
15
            self.setWindowTitle('All Widgets')
16
17
            layout = QVBoxLayout()
18
           widgets = [QCheckBox, # 检查框
19
                      QComboBox, # 下拉列表框
                      QDateEdit, # 日期编辑框
20
21
                      QDateTimeEdit, # 日期与时间编辑框
22
                      QDial, #刻度盘
23
                      QDoubleSpinBox, # 双精度选值框
24
                      QFontComboBox, # 字体下拉列表框
25
                      QLCDNumber, # 数字显示屏
26
                      QLabel, #标签
27
                      QLineEdit, # 行编辑框
28
                      QProgressBar, # 进度条
29
                      QPushButton, # 提交按钮
                      QRadioButton, # 单选按钮
30
31
                      QSlider, # 滑动条
32
                      QSpinBox, # 选值框
33
                      QTimeEdit] # 时间编辑框
34
35
            for w in widgets:
36
               layout.addWidget(w())
37
38
           widget = QWidget()
39
           widget.setLayout(layout)
40
            self.setCentralWidget(widget)
41
42
43
    def main():
        app = QApplication(sys.argv)
44
        window = MainWindow()
45
46
        window.show()
47
        app.exec_()
48
49
50
    if __name__ == '__main__':
51
        main()
```



CheckBox Widget

下面让我们来看一下检查框部件的一些基本使用。

• 导入需要的模块

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *
```

• 创建应用与窗口框架

```
class MainWindow(QMainWindow):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

self.init_ui()

def init_ui(self):
    self.setWindowTitle('CheckBox')
```

```
10
11
    def main():
        app = QApplication(sys.argv)
12
13
        window = MainWindow()
14
        window.show()
15
        app.exec_()
16
17
    if __name__ == '__main__':
18
19
        main()
```

• 创建检查框实例

```
1 checkbox = QCheckBox()
```

• 设置检查框状态

• 将检查框状态改变信号与槽(我们自己创建的函数)相连

```
1 checkbox.stateChanged.connect(self.show_state)
```

• 在窗口中心显示检查框

```
1 | self.setCentralWidget(checkbox)
```

• 创建槽函数

```
def show_state(self, state):
    print('state:', state == Qt.Checked)
    print(state)
```

```
1 import sys
    from PyQt5.QtWidgets import *
 3
   from PyQt5.QtCore import *
   from PyQt5.QtGui import *
4
 5
6
7
    class MainWindow(QMainWindow):
        def __init__(self, *args, **kwargs):
8
9
            super().__init__(*args, **kwargs)
10
            self.init_ui()
11
12
        def init_ui(self):
13
```

```
14
           self.setWindowTitle('CheckBox')
15
16
            checkbox = QCheckBox()
            # 两种设置检查框状态方式(.setChecked() and .setCheckState())
17
18
            # 如需设置三态,则也有两种方式(.setTristate() and
    .setCheckState(Qt.PartiallyChecked))
19
           checkbox.setChecked(True)
            checkbox.setTristate(True)
20
21
            # checkbox.setCheckState(Qt.PartiallyChecked) # 检查框有三种状态
    (Qt.Checked, Qt.Unchecked, Qt.PartiallyChecked)
22
23
            checkbox.stateChanged.connect(self.show_state)
24
25
            self.setCentralWidget(checkbox)
26
27
        def show_state(self, state):
28
            print('state:', state == Qt.Checked)
29
            print(state)
30
31
32
    def main():
33
        app = QApplication(sys.argv)
34
        window = MainWindow()
35
        window.show()
36
        app.exec_()
37
38
39
    if __name__ == '__main__':
40
        main()
```



点击检查框,会取消勾选,同时终端会显示以下情况:



再次点击检查框,检查框为中间态,终端出现以下情况:



再次点击检查框,会出现以下情况:



ComboBox Widget

- 导入模块与框架与CheckBox代码一样
- 创建下拉选项框实例

```
1 | combobox = QComboBox(self)
```

• 在下拉列表中添加选项,此处添加三个选项

```
1 | combobox.addItems(['One', 'Two', 'Three'])
```

• 可设置可编辑模式,即在窗口下拉选项框中输入一个选项,按回车则可将这个选项添加进入下拉选项中

```
1 | combobox.setEditable(True)
```

- 设置输入选项的插入位置,默认是末尾插入,这里我们设置顶部插入
- 1 combobox.setInsertPolicy(QComboBox.InsertAtTop)
- 设置选项的最大数目

```
1 combobox.setMaxCount(5)
```

• 将下拉选项框的指数变化信号与槽(我们自己创建的函数)相连

```
1 combobox.currentIndexChanged.connect(self.index_changed)
```

• 也可以将文本信号传输给槽函数

```
1 | combobox.currentIndexChanged[str].connect(self.text_changed)
```

• 创建处理指数信号的槽函数

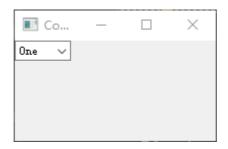
```
1 def index_changed(self, index):
2 print(index)
```

• 创建处理文本信号的槽函数

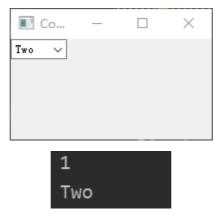
```
1 def text_changed(self, text):
2 print(text)
```

```
import sys
 2
   from PyQt5.QtWidgets import *
    from PyQt5.QtCore import *
 4
    from PyQt5.QtGui import *
 5
 6
 7
    class MainWindow(QMainWindow):
 8
        def __init__(self, *args, **kwargs):
9
            super().__init__(*args, **kwargs)
10
11
            self.init_ui()
12
13
        def init_ui(self):
            self.setWindowTitle('ComboBox')
14
15
16
            combobox = QComboBox(self)
17
            # 在下拉列表框中添加选项
            combobox.addItems(['One', 'Two', 'Three'])
18
            # 设置可编辑
19
20
            combobox.setEditable(True)
21
22
            # 设置插入选项模式,默认是在末尾插入
23
            combobox.setInsertPolicy(QComboBox.InsertAtTop)
24
            # 设置选项最大数目
25
26
            combobox.setMaxCount(5)
27
28
            # 默认发送的是index信号
29
            combobox.currentIndexChanged.connect(self.index_changed)
30
31
            # 可以发送文本信号
            combobox.currentIndexChanged[str].connect(self.text_changed)
32
33
34
            # self.setCentralWidget(combobox)
35
36
        def index_changed(self, index):
37
            print(index)
38
        def text_changed(self, text):
39
40
            print(text)
41
```

```
42
43
   def main():
        app = QApplication(sys.argv)
44
45
        window = MainWindow()
46
        window.show()
47
        app.exec_()
48
49
50 if __name__ == '__main__':
51
        main()
```



点击窗口中向下的箭头则可展开下拉选项, 当我选择Two时, 终端出现以下情况:



现在我们添加一个选项First, 下拉选项中最上面会出现First的选项, 同时终端会出现以下情况:



Label Widget

- 导入模块与框架与CheckBox代码一样
- 创建标签实例,有两种设置标签文字方法

• 设置标签文字字体大小三步走,获取当前字体,更新字体,应用字体

```
1  # 获取当前字体
2  font = 1b1.font()
3  # 更新字体,设置字体大小
4  font.setPointSize(30)
5  # 应用字体
6  lb1.setFont(font)
```

• 设置标签对齐方式

```
      1 # 设置标签对齐方式,使用'|'进行对齐方式的组合

      2 lbl.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
```

• 将标签在窗口中心显示

```
1 | self.setCentralwidget(lbl)
```

```
1 import sys
2 from PyQt5.QtWidgets import *
   from PyQt5.QtCore import *
   from PyQt5.QtGui import *
4
 5
6
7
    class MainWindow(QMainWindow):
8
       def __init__(self, *args, **kwargs):
9
           super().__init__(*args, **kwargs)
10
           self.init_ui()
11
12
       def init_ui(self):
13
           self.setWindowTitle('Label')
14
15
16
           lbl = QLabel('Yes')
17
           # 1b1 = QLabe1()
           # lbl.setText('Yes') # 两种设置标签文本的方法
18
19
           # 获取当前字体
20
           font = 1b1.font()
21
           # 更新字体,设置字体大小
22
23
           font.setPointSize(30)
           # 应用字体
24
           lbl.setFont(font)
25
26
           # 设置标签对齐方式,使用'|'进行对齐方式的组合
27
           lbl.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
28
           self.setCentralWidget(lbl)
29
```

```
30
31
    def main():
32
        app = QApplication(sys.argv)
        window = MainWindow()
33
34
        window.show()
35
        app.exec_()
36
37
38 if __name__ == '__main__':
39
        main()
```



如果我想显示图片标签,怎么办呢,下面代码将带你实现设置图片标签。在上面代码基础上添加一下代码:

```
1 # 在标签上设置位图
2 lbl.setPixmap(QPixmap('../../icons/named/heart.png'))
```

重新运行程序出现以下窗口:



如果想要图片占满窗口,则添加以下代码:

```
1 | lbl.setScaledContents(True) # 缩放标签,使标签填充满窗口
```

重新运行程序出现以下窗口:



LineEdit Widget

- 导入模块与框架与CheckBox代码一样
- 创建行编辑实例

```
1 | line_edit = QLineEdit(self)
```

• 设置行编辑的最大字符长度

```
1 | line_edit.setMaxLength(10)
```

• 设置提示词

```
1 | line_edit.setPlaceholderText('Enter your Text')
```

• 将回车信号与槽函数相连

```
1 | line_edit.returnPressed.connect(self.return_pressed)
```

• 将选择信号与槽函数相连

```
1 | line_edit.selectionChanged.connect(self.selection_changed)
```

• 将文本改变信号与槽函数相连

```
1 | line_edit.textChanged.connect(self.text_changed)
```

• 将文本编辑信号与槽函数相连

```
1 | line_edit.textEdited.connect(self.text_edit)
```

• 创建回车信号的槽函数

```
def return_pressed(self):
print('Return Pressed')
# 设置行编辑部件文本
self.centralWidget().setText('BOOM!')
```

• 创建选择信号的槽函数

```
def selection_changed(self):
    print('Selection Changed')
    print(self.centralwidget().selectedText())
```

• 创建文本改变信号的槽函数

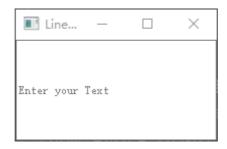
```
def text_changed(self, text):
    print('Text changed...')
    print(text)
```

• 创建文本编辑信号的槽函数

```
def text_edit(self, text):
    print('Text edited...')
    print(text)
```

```
import sys
 2
    from PyQt5.QtWidgets import *
 3
    from PyQt5.QtCore import *
    from PyQt5.QtGui import *
 4
 5
 6
 7
    class MainWindow(QMainWindow):
8
        def __init__(self, *args, **kwargs):
9
            super().__init__(*args, **kwargs)
10
11
            self.init_ui()
12
        def init_ui(self):
13
            self.setWindowTitle('Line Edit')
14
15
            line_edit = QLineEdit(self)
16
17
            # 设置行编辑的最大字符长度
18
            line_edit.setMaxLength(10)
19
            # 设置提示词
20
            line_edit.setPlaceholderText('Enter your Text')
21
22
            # 设置行编辑为只读
23
            # line_edit.setReadOnly(True)
24
            # 信号.returnPressed是接收回车信号
25
            line_edit.returnPressed.connect(self.return_pressed)
26
27
            # 信号.selectionChanged是接收鼠标选中的文本信号
28
            line_edit.selectionChanged.connect(self.selection_changed)
29
            # 信号.textChanged是接收文本改变信号
            line_edit.textChanged.connect(self.text_changed)
            # 信号。textEdited是接收文本编辑信号
31
32
            line_edit.textEdited.connect(self.text_edit)
33
34
            self.setCentralWidget(line_edit)
35
            # print(self.centralwidget())
36
37
        def return_pressed(self):
38
            print('Return Pressed')
39
            # 设置行编辑部件文本
40
            self.centralWidget().setText('BOOM!')
41
42
        def selection_changed(self):
43
            print('Selection Changed')
44
            print(self.centralWidget().selectedText())
45
46
        def text_changed(self, text):
47
            print('Text changed...')
48
            print(text)
49
50
        def text_edit(self, text):
```

```
print('Text edited...')
51
52
            print(text)
53
54
55 def main():
56
       app = QApplication(sys.argv)
57
        window = MainWindow()
58
       window.show()
59
        app.exec_()
60
61
   if __name__ == '__main__':
62
63
        main()
```



输入你好,则会出现以下情况:



Text edited... 你好 Text changed... 你好

如果按键盘回车键,则会出现以下情况:



Return Pressed Text changed... BOOM!

当选中BOOM!时,会出现以下情况:

BOOM!

Selection Changed BOOM!

List Widget

这个部件与ComboBox Widget很像。

- 导入模块与框架与CheckBox代码一样
- 创建列表部件实例

```
1 | list_widget = QListWidget(self)
```

• 添加列表选项

```
1 | list_widget.addItems(['One', 'Two', 'Three'])
```

• 将列表部件的选项变化信号与槽函数相连

```
1 | list_widget.currentItemChanged.connect(self.index_changed)
```

• 将列表部件的选项文本变化信号与槽函数相连

```
1 list_widget.currentTextChanged.connect(self.text_changed)
```

• 创建选项变化信号的槽函数

```
def index_changed(self, index):
    print(index) # QListWidgetItem object
    print(index.text())
```

• 创建选项文本变化信号的槽函数

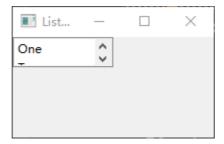
```
1 def text_changed(self, text):
2 print(text)
```

```
import sys
from PyQt5.Qtwidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *

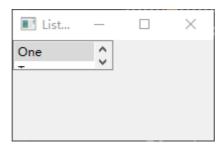
class MainWindow(QMainWindow):
```

```
8
        def __init__(self, *args, **kwargs):
            super().__init__(*args, ** kwargs)
 9
10
            self.init_ui()
11
12
        def init_ui(self):
13
14
            self.setWindowTitle('ListBox')
15
16
            list_widget = QListWidget(self)
17
            list_widget.addItems(['One', 'Two', 'Three'])
18
19
            list_widget.currentItemChanged.connect(self.index_changed)
            list_widget.currentTextChanged.connect(self.text_changed)
20
21
        def index_changed(self, index):
22
            print(index) # QListWidgetItem object
23
24
            print(index.text())
25
26
        def text_changed(self, text):
27
            print(text)
28
29
30
    def main():
31
        app = QApplication(sys.argv)
32
        window = MainWindow()
        window.show()
33
34
        app.exec_()
35
36
37
    if __name__ == '__main__':
38
        main()
```

运行程序会显示以下窗口:



选择One这一项会出现以下情况:



<PyQt5.QtWidgets.QListWidgetItem object at 0x00000202D66E4310>
One
One

Tab Widget

- 导入模块与框架与CheckBox代码一样
- 自定义一个颜色类

```
class Color(QWidget):
2
       def __init__(self, color, *args, **kwargs):
3
           super(Color, self).__init__(*args, **kwargs)
4
           self.setAutoFillBackground(True)
5
           # 获取当前调色板
6
           palette = self.palette()
8
           # 为窗口调色板设置传入参数的颜色
9
           palette.setColor(QPalette.Window, QColor(color))
10
           # 应用调色板颜色
11
           self.setPalette(palette)
```

• 创建标签集实例

```
1 | tabs = QTabwidget()
```

• 设置标签集部件位置

```
1 tabs.setTabPosition(QTabWidget.North)
```

• 设置可方向键切换标签

```
1 | tabs.setMovable(True)
```

• 在标签集中加入标签,并给每个标签设置一种颜色

```
colors = ['red', 'green', 'blue', 'purple', 'pink']
for n, color in enumerate(colors):
   tabs.addTab(Color(color), color)
```

```
import sys
    from PyQt5.QtWidgets import *
 3
    from PyQt5.QtCore import *
4
    from PyQt5.QtGui import *
 5
 6
 7
    class Color(QWidget):
8
        def __init__(self, color, *args, **kwargs):
            super(Color, self).__init__(*args, **kwargs)
9
10
            self.setAutoFillBackground(True)
11
            palette = self.palette()
12
            palette.setColor(QPalette.Window, QColor(color))
13
            self.setPalette(palette)
14
15
16
    class MainWindow(QMainWindow):
17
```

```
18
        def __init__(self, *args, **kwargs):
19
            super().__init__(*args, **kwargs)
20
            self.init_ui()
21
22
        def init_ui(self):
23
            self.setWindowTitle('Tabs Widget')
24
25
26
            tabs = QTabWidget()
27
            # 设置标签集部件位置
            tabs.setTabPosition(QTabWidget.North)
28
29
            # 方向键可切换标签
            tabs.setMovable(True)
30
31
            colors = ['red', 'green', 'blue', 'purple', 'pink']
32
            for n, color in enumerate(colors):
33
34
                tabs.addTab(Color(color), color)
35
36
            self.setCentralWidget(tabs)
37
38
39
    def main():
40
        app = QApplication(sys.argv)
41
        window = MainWindow()
42
        window.show()
43
        app.exec_()
44
45
   if __name__ == '__main__':
46
        main()
```

运行程序会有以下窗口:



按左右方向键或者按上面标签按钮可切换背景颜色。

Layouts

下面来讲以下窗口中的布局。

Horizontal Layout

- 导入模块、创建框架和自定义颜色类与Tab Widget代码一样
- 创建水平布局实例

```
1 | h_layout = QHBoxLayout()
```

• 在布局中添加颜色部件

```
h_layout.addwidget(Color('red'))
h_layout.addwidget(Color('green'))
h_layout.addwidget(Color('blue'))
```

• 创建部件实例

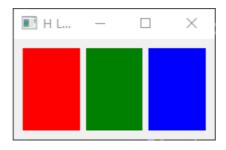
```
1 | widget = QWidget()
```

• 在部件上设置布局

```
1 | widget.setLayout(h_layout)
```

```
import sys
    from PyQt5.QtWidgets import *
 2
   from PyQt5.QtCore import *
    from PyQt5.QtGui import *
 4
 5
 6
 7
    class Color(QWidget):
 8
        def __init__(self, color, *args, **kwargs):
 9
            super(Color, self).__init__(*args, **kwargs)
            self.setAutoFillBackground(True)
10
11
12
            palette = self.palette()
            palette.setColor(QPalette.Window, QColor(color))
13
14
            self.setPalette(palette)
15
16
    class MainWindow(QMainWindow):
17
18
        def __init__(self, *args, **kwargs):
19
            super().__init__(*args, **kwargs)
20
21
            self.init_ui()
22
23
        def init_ui(self):
24
            self.setWindowTitle('H Layout')
25
26
            h_layout = QHBoxLayout()
27
```

```
28
            h_layout.addwidget(Color('red'))
29
            h_layout.addWidget(Color('green'))
30
            h_layout.addwidget(Color('blue'))
31
32
            widget = QWidget()
            widget.setLayout(h_layout)
33
34
            self.setCentralWidget(widget)
35
36
37
    def main():
38
39
        app = QApplication(sys.argv)
        window = MainWindow()
40
41
        window.show()
42
        app.exec_()
43
44
45
    if __name__ == '__main__':
46
        main()
```



Vertical Layout

- 导入模块、创建框架和自定义颜色类与Tab Widget代码一样
- 创建垂直布局实例

```
1 | v_layout = QHBoxLayout()
```

• 在布局中添加颜色部件

```
v_layout.addwidget(Color('red'))
v_layout.addwidget(Color('green'))
v_layout.addwidget(Color('blue'))
```

• 创建部件实例

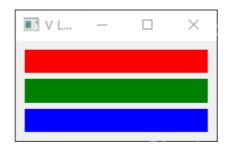
```
1 | widget = QWidget()
```

• 在部件上设置布局

```
1 | widget.setLayout(v_layout)
```

```
1 import sys
```

```
from PyQt5.QtWidgets import *
 3
    from PyQt5.QtCore import *
    from PyQt5.QtGui import *
 5
 6
 7
    class Color(QWidget):
 8
        def __init__(self, color, *args, **kwargs):
 9
            super(Color, self).__init__(*args, **kwargs)
10
            # 填充背景
11
            self.setAutoFillBackground(True)
12
13
            # 获取当前调色板
14
            palette = self.palette()
            # 对窗口调色板设置颜色
15
16
            palette.setColor(QPalette.Window, QColor(color))
17
            # 应用更新的调色板
            self.setPalette(palette)
18
19
20
21
    class MainWindow(QMainWindow):
        def __init__(self, *args, **kwargs):
22
23
            super().__init__(*args, **kwargs)
24
            self.init_ui()
25
26
        def init_ui(self):
27
28
            self.setWindowTitle('V Layout')
29
30
            v_layout = QVBoxLayout()
31
            color_widget = Color('red')
32
33
            v_layout.addwidget(color_widget)
            v_layout.addWidget(Color('green'))
34
35
            v_layout.addwidget(Color('blue'))
36
37
            widget = QWidget()
38
            widget.setLayout(v_layout)
39
            self.setCentralWidget(widget)
40
41
42
43
    def main():
44
        app = QApplication(sys.argv)
45
        window = MainWindow()
46
        window.show()
47
        app.exec_()
48
49
    if __name__ == '__main__':
50
51
        main()
```



Nested Layout

- 导入模块、创建框架和自定义颜色类与Tab Widget代码一样
- 创建一个水平布局和两个垂直布局实例

```
h_layout = QHBoxLayout()
v_layout1 = QVBoxLayout()
v_layout2 = QVBoxLayout()
```

• 在第一个垂直布局中添加颜色部件

```
v_layout1.addwidget(Color('red'))
v_layout1.addwidget(Color('green'))
v_layout1.addwidget(Color('blue'))
```

• 在第二个垂直布局中添加颜色部件

```
1  v_layout2.addwidget(Color('yellow'))
2  v_layout2.addwidget(Color('purple'))
```

• 在水平布局中加入这两个垂直布局和一个颜色部件

```
h_layout.addLayout(v_layout1)
h_layout.addWidget(Color('pink'))
h_layout.addLayout(v_layout2)
```

• 创建部件实例

```
1 | widget = QWidget()
```

• 在部件上设置布局

```
1 | widget.setLayout(h_layout)
```

```
import sys
from PyQt5.Qtwidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *

class Color(Qwidget):
    def __init__(self, color, *args, **kwargs):
    super(Color, self).__init__(*args, **kwargs)
```

```
10
            self.setAutoFillBackground(True)
11
12
            palette = self.palette()
13
            palette.setColor(QPalette.Window, QColor(color))
14
            self.setPalette(palette)
15
16
    class MainWindow(QMainWindow):
17
18
        def __init__(self, *args, **kwargs):
19
            super().__init__(*args, **kwargs)
20
21
            self.init_ui()
22
23
        def init_ui(self):
24
            self.setWindowTitle('Nested Layout')
25
26
            h_layout = QHBoxLayout()
27
            v_layout1 = QVBoxLayout()
            v_1ayout2 = QVBoxLayout()
28
29
            v_layout1.addWidget(Color('red'))
30
31
            v_layout1.addwidget(Color('green'))
32
            v_layout1.addWidget(Color('blue'))
33
34
            v_layout2.addwidget(Color('yellow'))
            v_layout2.addwidget(Color('purple'))
35
36
37
            h_layout.addLayout(v_layout1)
            h_layout.addwidget(Color('pink'))
38
39
            h_layout.addLayout(v_layout2)
40
41
            widget = QWidget()
            widget.setLayout(h_layout)
42
43
44
            self.setCentralWidget(widget)
45
46
47
    def main():
48
        app = QApplication(sys.argv)
49
        window = MainWindow()
50
        window.show()
51
        app.exec_()
52
53
54
    if __name__ == '__main__':
55
        main()
```

运行程序会有以下窗口:



Margins and Spacing

在Nested Layout的代码基础上添加代码,使得改变布局边距和布局间隔。

• 给水平布局设置上下左右边距大小

```
1 | h_layout.setContentsMargins(0, 0, 0, 0)
```

• 给水平布局设置间隔大小

```
1 h_layout.setSpacing(20)
```

• 给第一个垂直布局设置间隔大小

```
1 v_layout1.setSpacing(10)
```

• 给第二个垂直布局设置间隔大小

```
1 v_layout2.setSpacing(5)
```

• 完整代码

```
import sys
 2 from PyQt5.QtWidgets import *
 3
    from PyQt5.QtCore import *
    from PyQt5.QtGui import *
 5
 6
 7
    class Color(QWidget):
 8
        def __init__(self, color, *args, **kwargs):
 9
            super(Color, self).__init__(*args, **kwargs)
10
            self.setAutoFillBackground(True)
11
            palette = self.palette()
12
13
            palette.setColor(QPalette.Window, QColor(color))
            self.setPalette(palette)
14
15
16
    class MainWindow(QMainWindow):
17
18
        def __init__(self, *args, **kwargs):
19
            super().__init__(*args, **kwargs)
            self.init_ui()
21
22
23
        def init_ui(self):
24
            self.setWindowTitle('Margins and Spacing')
25
26
            h_layout = QHBoxLayout()
            h_layout.setContentsMargins(0, 0, 0, 0)
27
28
            h_layout.setSpacing(20)
29
30
            v_layout1 = QVBoxLayout()
31
            v_layout1.setSpacing(10)
32
33
            v_layout2 = QVBoxLayout()
```

```
34
            v_layout2.setSpacing(5)
35
36
            v_layout1.addWidget(Color('red'))
            v_layout1.addwidget(Color('green'))
37
38
            v_layout1.addwidget(Color('blue'))
39
40
            v_layout2.addWidget(Color('yellow'))
            v_layout2.addWidget(Color('purple'))
41
42
43
            h_layout.addLayout(v_layout1)
            h_layout.addwidget(Color('pink'))
44
45
            h_layout.addLayout(v_layout2)
46
47
            widget = QWidget()
48
            widget.setLayout(h_layout)
49
50
            self.setCentralWidget(widget)
51
52
53
    def main():
        app = QApplication(sys.argv)
54
55
        window = MainWindow()
56
        window.show()
57
        app.exec_()
58
59
60
    if __name__ == '__main__':
61
        main()
```



Grid Layout

- 导入模块、创建框架和自定义颜色类与Tab Widget代码一样
- 创建网格布局实例

```
1 | grid_layout = QGridLayout()
```

• 在网格特定位置添加特定颜色部件

```
grid_layout.addwidget(Color('red'), 0, 0)
grid_layout.addwidget(Color('green'), 1, 0)
grid_layout.addwidget(Color('blue'), 1, 1)
grid_layout.addwidget(Color('purple'), 2, 1)
```

```
1 import sys
```

```
2
    from PyQt5.QtWidgets import *
 3
    from PyQt5.QtCore import *
    from PyQt5.QtGui import *
 5
 6
 7
    class Color(QWidget):
 8
        def __init__(self, color, *args, **kwargs):
 9
            super(Color, self).__init__(*args, **kwargs)
10
            self.setAutoFillBackground(True)
11
12
            palette = self.palette()
13
            palette.setColor(QPalette.Window, QColor(color))
14
            self.setPalette(palette)
15
16
17
    class MainWindow(QMainWindow):
18
        def __init__(self, *args, **kwargs):
19
            super().__init__(*args, **kwargs)
20
21
            self.init_ui()
22
23
        def init_ui(self):
24
            self.setWindowTitle('Grid Layout')
25
26
            grid_layout = QGridLayout()
27
28
            grid_layout.addwidget(Color('red'), 0, 0)
29
            grid_layout.addwidget(Color('green'), 1, 0)
            grid_layout.addWidget(Color('blue'), 1, 1)
30
31
            grid_layout.addwidget(Color('purple'), 2, 1)
32
33
            widget = QWidget()
34
            widget.setLayout(grid_layout)
35
36
            self.setCentralWidget(widget)
37
38
39
    def main():
40
        app = QApplication(sys.argv)
41
        window = MainWindow()
42
        window.show()
43
        app.exec_()
44
45
46
    if __name__ == '__main__':
47
        main()
```



Stacked Layout

下面来看看堆叠布局。

• 创建堆叠布局实例

```
1 | stacked_layout = QStackedLayout()
```

• 创建一个水平布局和一个垂直布局

```
button_layout = QHBoxLayout()
page_layout = QVBoxLayout()
```

• 在垂直布局中加入水平布局和堆叠布局

```
page_layout.addLayout(button_layout)
page_layout.addLayout(stacked_layout)
```

在堆叠布局中加入各个颜色部件,同时为每个颜色部件创建一个按钮,将按钮按下信号与堆叠布局中当前颜色部件指数相连

```
colors = ['red', 'green', 'blue', 'purple', 'pink']
for n, color in enumerate(colors):
    stacked_layout.addwidget(Color(color))

btn = QPushButton(color)
    btn.pressed.connect(lambda i=n: stacked_layout.setCurrentIndex(i))
button_layout.addwidget(btn)
```

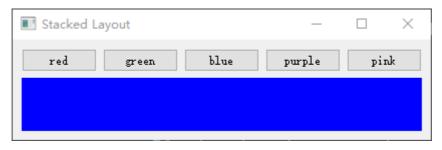
```
import sys
 1
 2
    from PyQt5.QtWidgets import *
 3
    from PyQt5.QtCore import *
    from PyQt5.QtGui import *
 4
 5
 6
 7
    class Color(QWidget):
 8
        def __init__(self, color, *args, **kwargs):
 9
            super(Color, self).__init__(*args, **kwargs)
            self.setAutoFillBackground(True)
10
11
12
            palette = self.palette()
            palette.setColor(QPalette.Window, QColor(color))
13
14
            self.setPalette(palette)
15
16
17
    class MainWindow(QMainWindow):
        def __init__(self, *args, **kwargs):
18
19
            super().__init__(*args, **kwargs)
20
21
            self.init_ui()
22
        def init_ui(self):
23
            self.setWindowTitle('Stacked Layout')
24
```

```
25
26
            stacked_layout = QStackedLayout()
27
            button_layout = QHBoxLayout()
28
            page_layout = QVBoxLayout()
29
30
            # stacked_layout.addWidget(Color('red'))
31
            # stacked_layout.addwidget(Color('green'))
            # stacked_layout.addwidget(Color('blue'))
32
33
            # stacked_layout.addWidget(Color('purple'))
34
35
            # stacked_layout.setCurrentIndex(1)
36
37
            page_layout.addLayout(button_layout)
38
            page_layout.addLayout(stacked_layout)
39
            colors = ['red', 'green', 'blue', 'purple', 'pink']
40
41
            for n, color in enumerate(colors):
42
                 stacked_layout.addwidget(Color(color))
43
44
                 btn = QPushButton(color)
45
                 btn.pressed.connect(lambda i=n:
    stacked_layout.setCurrentIndex(i))
46
                 button_layout.addWidget(btn)
47
48
            widget = QWidget()
            widget.setLayout(page_layout)
49
50
51
            self.setCentralWidget(widget)
52
53
    def main():
54
55
        app = QApplication(sys.argv)
        window = MainWindow()
56
        window.show()
57
58
        app.exec_()
59
60
    if __name__ == '__main__':
61
62
        main()
```

运行程序会有以下窗口:



按哪个颜色按钮下面就会显示什么颜色, 比如按蓝色按钮:



实现功能与标签集部件类似。

Dialog

Custom Dialog

• 自定义对话框类

```
class CustomDialog(QDialog):
 2
        def __init__(self, *args, **kwargs):
 3
            super(CustomDialog, self).__init__(*args, **kwargs)
 4
            self.setWindowTitle('Hello')
 6
            # 创建按钮
 7
8
            btn = QDialogButtonBox.Ok | QDialogButtonBox.Cancel
9
10
            # 将按钮放在按钮盒中
11
            self.buttonbox = QDialogButtonBox(btn)
            # 将信号与槽相连
12
            self.buttonbox.accepted.connect(self.accept)
13
            self.buttonbox.rejected.connect(self.reject)
14
15
16
            self.layout = QVBoxLayout()
            self.layout.addwidget(self.buttonbox)
17
            self.setLayout(self.layout)
18
```

```
import sys
    from PyQt5.QtWidgets import *
    from PyQt5.QtCore import *
 4
    from PyQt5.QtGui import *
 5
 6
 7
    class CustomDialog(QDialog):
 8
        def __init__(self, *args, **kwargs):
 9
            super(CustomDialog, self).__init__(*args, **kwargs)
10
11
            self.setWindowTitle('Hello')
12
            # 创建按钮
13
14
            btn = QDialogButtonBox.Ok | QDialogButtonBox.Cancel
15
16
            # 将按钮放在按钮盒中
            self.buttonbox = QDialogButtonBox(btn)
17
18
            # 将信号与槽相连
19
            self.buttonbox.accepted.connect(self.accept)
20
            self.buttonbox.rejected.connect(self.reject)
21
            self.layout = QVBoxLayout()
22
23
            self.layout.addWidget(self.buttonbox)
            self.setLayout(self.layout)
24
25
26
27
    class MainWindow(QMainWindow):
28
        def __init__(self, *args, **kwargs):
29
            super(MainWindow, self).__init__(*args, **kwargs)
```

```
30
31
            self.init_ui()
32
        def init_ui(self):
33
34
            self.setWindowTitle('Custom Dialog')
35
36
            action = QAction(QIcon('../../icons/named/compass.png'), 'Dialog',
    self)
37
            action.setStatusTip('Create Dialog Box')
38
            action.triggered.connect(self.showDialog)
39
40
            toolbar = QToolBar('My Toolbar')
41
            self.setIconSize(QSize(16, 16))
            self.setToolButtonStyle(Qt.ToolButtonTextBesideIcon)
42
            self.addToolBar(toolbar)
43
44
45
            toolbar.addAction(action)
46
        def showDialog(self, s):
47
48
            print('Click', s)
49
            dialog = CustomDialog()
50
51
            if dialog.exec_():
                print('Success!')
52
53
            else:
                print('Cancel!')
54
55
56
    def main():
57
58
        app = QApplication(sys.argv)
        window = MainWindow()
59
60
        window.show()
61
        app.exec_()
62
63
64
    if __name__ == '__main__':
65
        main()
```

运行程序会有以下窗口:



点击工具栏上的Dialog按钮则会弹出一个标题是Hello的对话框,里面有两个按钮,一个OK和一个Cancel。点击OK则会在终端上显示Success!,点击Cancel则会在终端上显示Cancel!。