PyQt5

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

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

- 创建窗口实例

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

- 显示窗口

```
# 显示窗口
window.show()
```

- 应用事件循环

```
# 应用事件循环
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

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

- 创建文字标签

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

- 设置标签对齐方式，设置为居中对齐

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

- 在窗口中心显示标签

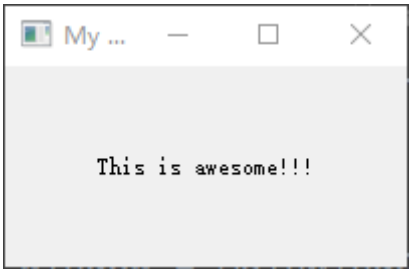
```
# 在窗口中心设置标签
self.setCentralWidget(lbl)
```

- 完整代码如下

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


class MainWindow(QMainWindow):
    def __init__(self, *args, **kwargs):
        # 继承QMainWindow的__init__方法
        super().__init__(*args, **kwargs)

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

        lbl = QLabel('This is awesome!!!', self)
        # 设置标签对齐方式，Qt.AlignCenter是中心对齐
        lbl.setAlignment(Qt.AlignCenter)
        # 在窗口中心设置标签
        self.setCentralWidget(lbl)


# 创建应用实例，传入命令参数列表
app = QApplication(sys.argv)

# 创建窗口实例
window = MainWindow()
# 显示窗口
window.show()

```

```
30 # 应用事件循环
31 app.exec_()
```

运行程序会出现以下窗口：



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

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

- 重写文本菜单事件

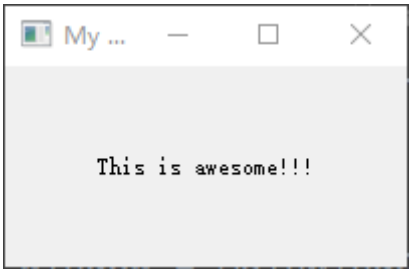
```
# 当在窗口右击时，执行以下函数
def contextMenuEvent(self, event):
    print('Context menu event.')
    # 继承原有文本菜单事件的功能
    super().contextMenuEvent(event)
```

- 完整代码如下

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


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

        self.windowTitleChanged.connect(self.onWindowTitleChanged)

        self.windowTitleChanged.connect(lambda x: self.my_custom_fn())

        self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x))

        self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x, 25))

        self.setWindowTitle('My Awesome App')
        lbl = QLabel('This is Awesome!!!')
        lbl.setAlignment(Qt.AlignCenter)
        self.setCentralWidget(lbl)

    def onWindowTitleChanged(self, s):
        print(s)

    def my_custom_fn(self, a='Hello', b=5):
        print(a, b)

    def contextMenuEvent(self, event):
        print('Context menu event.')
        super().contextMenuEvent(event)

```

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

运行程序会出现以下窗口：



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

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

- 设置工具栏里面工具的图标大小

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

- 设置工具栏里面工具图标的显示风格

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

- 在窗口上添加工具栏

```
self.addToolBar(toolbar)
```

- 创建状态栏实例

```
statusbar = QStatusBar(self)
```

- 在窗口上添加状态栏

```
self.setStatusBar(statusbar)
```

- 创建菜单栏

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

- 在菜单栏上添加第一个菜单

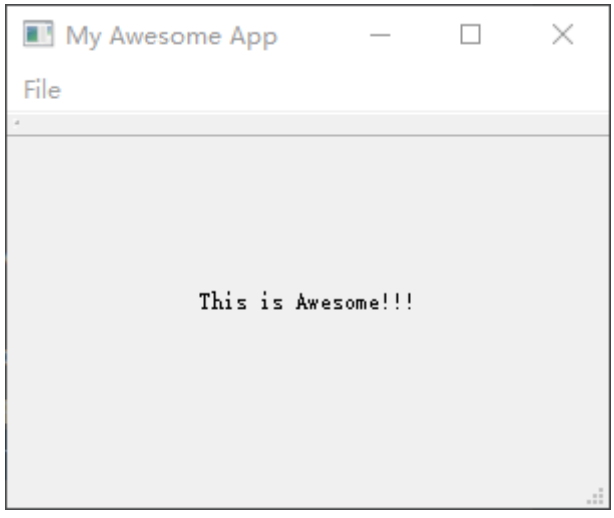
```
file_menu = menu.addMenu('&File')
```

- 完整代码如下

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


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

        self.windowTitleChanged.connect(self.onWindowTitleChanged)

        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')
20          self.setGeometry(300, 300, 300, 220)
21
22          lbl = QLabel('This is Awesome!!!')
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          # add menuBar and add menu on menuBar
37          menu = self.menuBar()
38          file_menu = menu.addMenu('&File')
39
40      def onWindowTitleChanged(self, s):
41          print(s)
42
43      def my_custom_fn(self, a='Hello', b=5):
44          print(a, b)
45
46      def contextMenuEvent(self, event):
47          print('Context menu event.')
48          super().contextMenuEvent(event)
49
50
51  app = QApplication(sys.argv)
52  window = MainWindow()
53  window.show()
54  app.exec_()
```

运行程序会出现以下窗口：



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

```
# 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
button_action = QAction(QIcon('../icons/16/045.png'),  'Home', self)
# 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
button_action.setStatusTip('This is action.')
# 将动作按钮触发信号连接槽(自己所写函数)
button_action.triggered.connect(self.onMyToolBarButtonClick)
# 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
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
button_action.setShortcut(QKeySequence('Ctrl+h'))
# 将此动作按钮加入工具栏中
toolbar.addAction(button_action)
```

- 在菜单中加入动作按钮

```
file_menu.addAction(button_action)
```

- 创建与动作按钮连接的槽函数

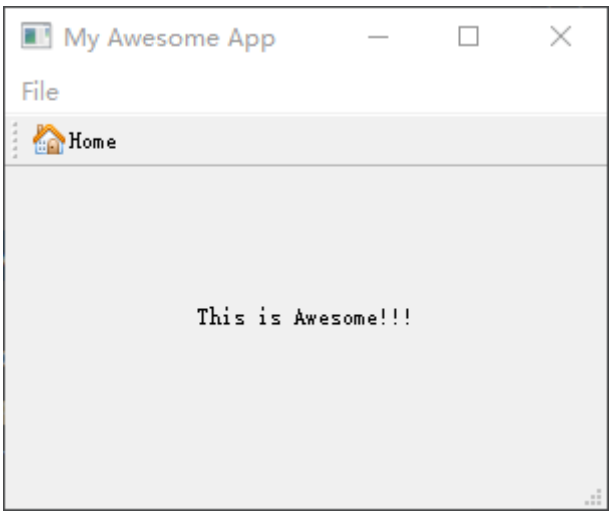
```
def onMyToolBarButtonClick(self, s):
    print('click', s)
```

- 完整代码如下

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


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

        self.windowTitleChanged.connect(self.onWindowTitleChanged)

        self.windowTitleChanged.connect(lambda x: self.my_custom_fn())

        self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x))

        self.windowTitleChanged.connect(lambda x: self.my_custom_fn(x, 25))

        self.setWindowTitle('My Awesome App')
        self.setGeometry(300, 300, 300, 220)

        lbl = QLabel('This is Awesome!!!')
        lbl.setAlignment(Qt.AlignCenter)
        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.')
38          button_action.triggered.connect(self.onMyToolBarButtonClick)
39          button_action.setCheckable(True)
40          button_action.setShortcut(QKeySequence('Ctrl+h'))
41          toolbar.addAction(button_action)
42
43          # add menuBar and add menu on menuBar
44          menu = self.menuBar()
45          file_menu = menu.addMenu('&File')
46          file_menu.addAction(button_action)
47
48
49      def onMyToolBarButtonClick(self, s):
50          print('click', s)
51
52      def onWindowTitleChanged(self, s):
53          print(s)
54
55      def my_custom_fn(self, a='Hello', b=5):
56          print(a, b)
57
58      def contextMenuEvent(self, event):
59          print('Context menu event.')
60          super().contextMenuEvent(event)
61
62
63  app = QApplication(sys.argv)
64  window = MainWindow()
65  window.show()
66  app.exec_()
```

运行程序会出现以下窗口：



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`Ctrl+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：

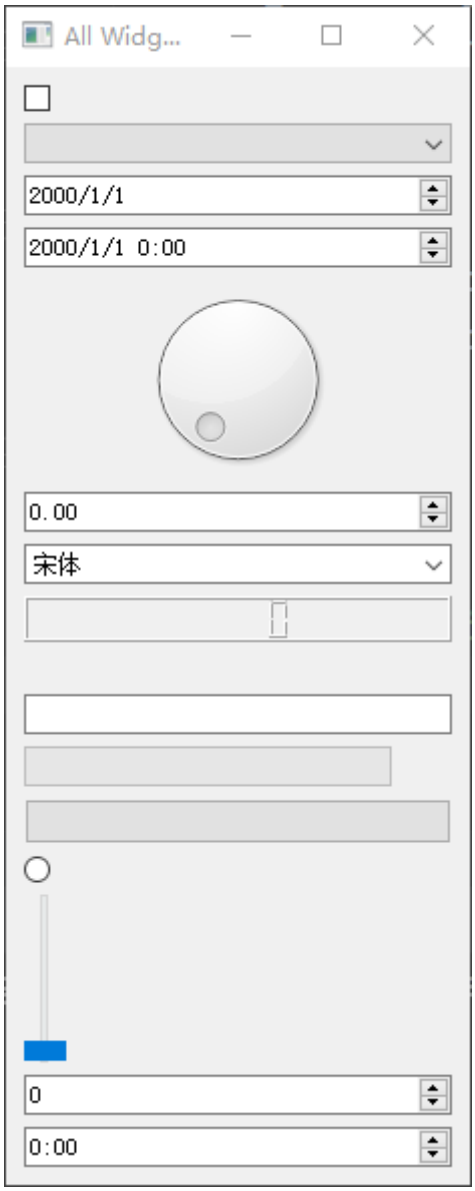
```
click True
click False
```

Widgets

- 下面代码展示了Qt里大多数部件

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


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

        self.init_ui()

    def init_ui(self):
        # self.setGeometry(300, 300, 300, 220)
        self.setWindowTitle('All Widgets')

        layout = QVBoxLayout()
        widgets = [QCheckBox,  # 检查框
                   QComboBox,  # 下拉列表框
                   QDateEdit,  # 日期编辑框
                   QDateTimeEdit,  # 日期与时间编辑框
                   QDial,  # 刻度盘
                   QDoubleSpinBox,  # 双精度选值框
                   QFontComboBox,  # 字体下拉列表框
                   QLCDNumber,  # 数字显示屏
                   QLabel,  # 标签
                   QLineEdit,  # 行编辑框
                   QProgressBar,  # 进度条
                   QPushButton,  # 提交按钮
                   QRadioButton,  # 单选按钮
                   QSlider,  # 滑动条
                   QSpinBox,  # 选值框
                   QTimeEdit]  # 时间编辑框

        for w in widgets:
            layout.addWidget(w())

        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    main()
```

运行程序会出现以下窗口：



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

- 创建检查框实例

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

- 设置检查框状态

```
1  # 两种设置复选框状态方式(.setChecked() and .setCheckState())
2  # 如需设置三态，则也有两种方式(.setTristate() and
   .setCheckState(Qt.PartiallyChecked))
3  checkbox.setChecked(True)
4  checkbox.setTristate(True)
5  # checkbox.setCheckState(Qt.PartiallyChecked)  # 复选框有三种状态(Qt.Checked,
   Qt.Unchecked, Qt.PartiallyChecked)
```

- 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

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

- 完整代码如下

```
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):
```

```
        self.setWindowTitle('CheckBox')

        checkbox = QCheckBox()
        # 两种设置检查框状态方式(.setChecked() and .setCheckState())
        # 如需设置三态，则也有两种方式(.setTristate() and
.setCheckState(Qt.PartiallyChecked))
        checkbox.setChecked(True)
        checkbox.setTristate(True)
        # checkbox.setCheckState(Qt.PartiallyChecked)  # 检查框有三种状态
(Qt.Checked, Qt.Unchecked, Qt.PartiallyChecked)

        checkbox.stateChanged.connect(self.show_state)

        self.setCentralWidget(checkbox)

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


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    main()
```

运行程序会出现以下窗口：

点击检查框，会取消勾选，同时终端会显示以下情况：


```
state: False
0
```

再次点击检查框，检查框为中间态，终端出现以下情况：


```
state: False
1
```

再次点击检查框，会出现以下情况：


```
state: True
2
```

ComboBox Widget

- 导入模块与框架与CheckBox代码一样
- 创建下拉选项框实例

```
combobox = QComboBox(self)
```

- 在下拉列表中添加选项，此处添加三个选项

```
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

```
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

```
combobox.setInsertPolicy(QComboBox.InsertAtTop)
```

- 设置选项的最大数目

```
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

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

- 也可以将文本信号传输给槽函数

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

- 创建处理指数信号的槽函数

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

- 创建处理文本信号的槽函数

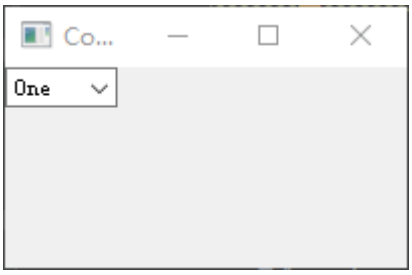
```
def text_changed(self, text):
    print(text)
```

- 完整代码如下

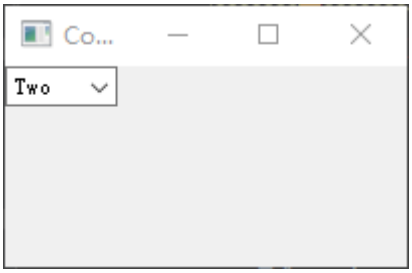
```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *


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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('ComboBox')

        combobox = QComboBox(self)
        # 在下拉列表框中添加选项
        combobox.addItems(['One', 'Two', 'Three'])
        # 设置可编辑
        combobox.setEditable(True)

        # 设置插入选项模式，默认是在末尾插入
        combobox.setInsertPolicy(QComboBox.InsertAtTop)

        # 设置选项最大数目
        combobox.setMaxCount(5)

        # 默认发送的是index信号
        combobox.currentIndexChanged.connect(self.index_changed)

        # 可以发送文本信号
        combobox.currentIndexChanged[str].connect(self.text_changed)

        # self.setCentralWidget(combobox)

    def index_changed(self, index):
        print(index)

    def text_changed(self, text):
        print(text)

```

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

运行程序会出现以下窗口：

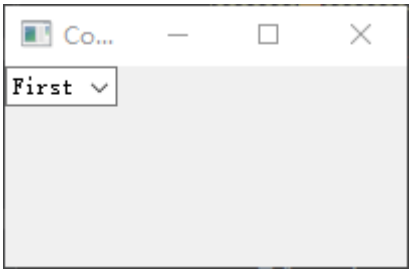


点击窗口中向下的箭头则可展开下拉选项，当我选择Two时，终端出现以下情况：




```
1
Two
```

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：




```
2
Two
0
First
```

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

```
lbl = QLabel('Yes')

# lbl = QLabel()
# lbl.setText('Yes')
```

- 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

```
# 获取当前字体
font = lbl.font()
# 更新字体，设置字体大小
font.setPointSize(30)
# 应用字体
lbl.setFont(font)
```

- 设置标签对齐方式

```
# 设置标签对齐方式，使用'|'进行对齐方式的组合
lbl.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
```

- 将标签在窗口中心显示

```
self.setCentralWidget(lbl)
```

- 完整代码如下

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


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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('Label')

        lbl = QLabel('Yes')
        # lbl = QLabel()
        # lbl.setText('Yes')  # 两种设置标签文本的方法

        # 获取当前字体
        font = lbl.font()
        # 更新字体，设置字体大小
        font.setPointSize(30)
        # 应用字体
        lbl.setFont(font)
        # 设置标签对齐方式，使用'|'进行对齐方式的组合
        lbl.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
        self.setCentralWidget(lbl)

```

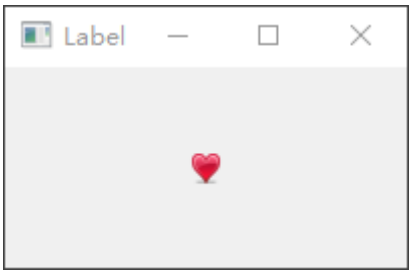
```
30
31 def main():
32     app = QApplication(sys.argv)
33     window = MainWindow()
34     window.show()
35     app.exec_()
36
37
38 if __name__ == '__main__':
39     main()
```

运行程序会出现以下窗口：

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

```
line_edit = QLineEdit(self)
```

- 设置行编辑的最大字符长度

```
line_edit.setMaxLength(10)
```

- 设置提示词

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

- 将回车信号与槽函数相连

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

- 将选择信号与槽函数相连

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

- 将文本改变信号与槽函数相连

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

- 将文本编辑信号与槽函数相连

```
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

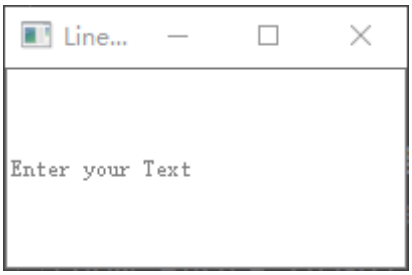
```
def text_edit(self, text):
    print('Text edited...')
    print(text)
```

- 完整代码如下

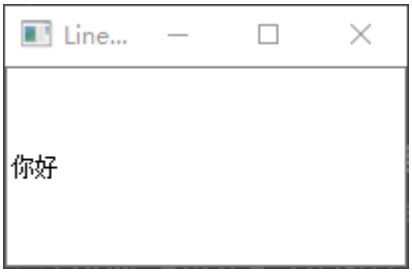
```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *


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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('Line Edit')

        line_edit = QLineEdit(self)
        # 设置行编辑的最大字符长度
        line_edit.setMaxLength(10)
        # 设置提示词
        line_edit.setPlaceholderText('Enter your Text')

        # 设置行编辑为只读
        # line_edit.setReadOnly(True)

        # 信号.returnPressed是接收回车信号
        line_edit.returnPressed.connect(self.return_pressed)
        # 信号.selectionChanged是接收鼠标选中的文本信号
        line_edit.selectionChanged.connect(self.selection_changed)
        # 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
        line_edit.textEdited.connect(self.text_edit)

        self.setCentralWidget(line_edit)
        # print(self.centralWidget())

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

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

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

    def text_edit(self, text):
```

```
        print('Text edited...')
        print(text)


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    main()
```

运行程序会出现以下窗口：



输入你好，则会出现以下情况：

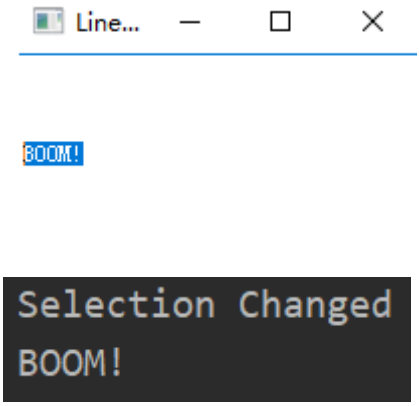



```
Text edited...
你好
Text changed...
你好
```

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：



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

```
list_widget = QListWidget(self)
```

- 添加列表选项

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

- 将列表部件的选项变化信号与槽函数相连

```
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

```
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

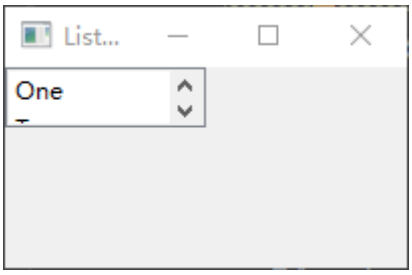
```
def text_changed(self, text):
    print(text)
```

- 完整代码如下

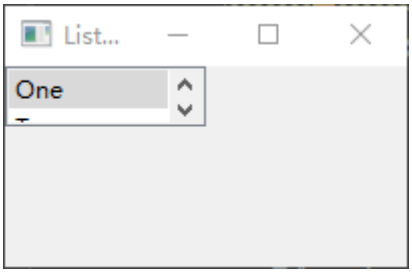
```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *


class MainWindow(QMainWindow):
```

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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('ListBox')

        list_widget = QListWidget(self)
        list_widget.addItems(['One', 'Two', 'Three'])

        list_widget.currentItemChanged.connect(self.index_changed)
        list_widget.currentTextChanged.connect(self.text_changed)

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

    def text_changed(self, text):
        print(text)


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    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):
    def __init__(self, color, *args, **kwargs):
        super(Color, self).__init__(*args, **kwargs)
        self.setAutoFillBackground(True)

        # 获取当前调色板
        palette = self.palette()
        # 为窗口调色板设置传入参数的颜色
        palette.setColor(QPalette.Window, QColor(color))
        # 应用调色板颜色
        self.setPalette(palette)
```

- 创建标签集实例

```
tabs = QTabWidget()
```

- 设置标签集部件位置

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

- 设置可方向键切换标签

```
tabs.setMovable(True)
```

- 在标签集中加入标签，并给每个标签设置一种颜色

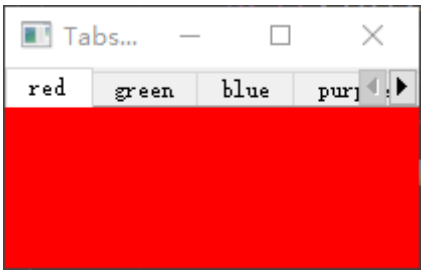
```
colors = ['red', 'green', 'blue', 'purple', 'pink']
for n, color in enumerate(colors):
    tabs.addTab(Color(color), color)
```

- 完整代码如下

```
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)
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.Window, QColor(color))
        self.setPalette(palette)


class MainWindow(QMainWindow):
```

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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('Tabs Widget')

        tabs = QTabWidget()
        # 设置标签集部件位置
        tabs.setTabPosition(QTabWidget.North)
        # 方向键可切换标签
        tabs.setMovable(True)

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

        self.setCentralWidget(tabs)


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    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

```
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

```
widget = QWidget()
```

- 在部件上设置布局

```
widget.setLayout(h_layout)
```

- 完整代码如下

```
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)
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.Window, QColor(color))
        self.setPalette(palette)


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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('H Layout')

        h_layout = QHBoxLayout()

```

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

        widget = QWidget()
        widget.setLayout(h_layout)

        self.setCentralWidget(widget)


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    main()
```

运行程序会出现以下窗口：

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

```
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

```
widget = QWidget()
```

- 在部件上设置布局

```
widget.setLayout(v_layout)
```

- 完整代码如下

```
import sys
```

```python
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)
        # 填充背景
        self.setAutoFillBackground(True)

        # 获取当前调色板
        palette = self.palette()
        # 对窗口调色板设置颜色
        palette.setColor(QPalette.Window, QColor(color))
        # 应用更新的调色板
        self.setPalette(palette)


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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('V Layout')

        v_layout = QVBoxLayout()

        color_widget = Color('red')
        v_layout.addWidget(color_widget)
        v_layout.addWidget(Color('green'))
        v_layout.addWidget(Color('blue'))

        widget = QWidget()
        widget.setLayout(v_layout)

        self.setCentralWidget(widget)


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    main()
```

运行程序会出现以下窗口：

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

```
v_layout2.addWidget(Color('yellow'))
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

```
widget = QWidget()
```

- 在部件上设置布局

```
widget.setLayout(h_layout)
```

- 完整代码如下

```
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)
```

```
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.Window, QColor(color))
        self.setPalette(palette)


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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('Nested Layout')

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

        v_layout1.addWidget(Color('red'))
        v_layout1.addWidget(Color('green'))
        v_layout1.addWidget(Color('blue'))

        v_layout2.addWidget(Color('yellow'))
        v_layout2.addWidget(Color('purple'))

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

        widget = QWidget()
        widget.setLayout(h_layout)

        self.setCentralWidget(widget)


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    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

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

- 给水平布局设置间隔大小

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

- 给第一个垂直布局设置间隔大小

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

- 给第二个垂直布局设置间隔大小

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

- 完整代码

```
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)
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.Window, QColor(color))
        self.setPalette(palette)


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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('Margins and Spacing')

        h_layout = QHBoxLayout()
        h_layout.setContentsMargins(0, 0, 0, 0)
        h_layout.setSpacing(20)

        v_layout1 = QVBoxLayout()
        v_layout1.setSpacing(10)

        v_layout2 = QVBoxLayout()
```

```
        v_layout2.setSpacing(5)

        v_layout1.addWidget(Color('red'))
        v_layout1.addWidget(Color('green'))
        v_layout1.addWidget(Color('blue'))

        v_layout2.addWidget(Color('yellow'))
        v_layout2.addWidget(Color('purple'))

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

        widget = QWidget()
        widget.setLayout(h_layout)

        self.setCentralWidget(widget)


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    main()
```

运行程序会出现以下窗口：

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

```
grid_layout = QGridLayout()
```

- 在网格特定位置添加特定颜色部件

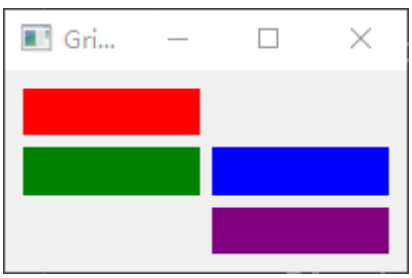
```
grid_layout.addWidget(Color('red'), 0, 0)
grid_layout.addWidget(Color('green'), 1, 0)
grid_layout.addWidget(Color('blue'), 1, 1)
grid_layout.addWidget(Color('purple'), 2, 1)
```

- 完整代码如下

```
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)
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.Window, QColor(color))
        self.setPalette(palette)


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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('Grid Layout')

        grid_layout = QGridLayout()

        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)

        widget = QWidget()
        widget.setLayout(grid_layout)

        self.setCentralWidget(widget)


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()


if __name__ == '__main__':
    main()
```

运行程序会出现以下窗口：



Stacked Layout

下面来看看堆叠布局。

- 创建堆叠布局实例

```
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

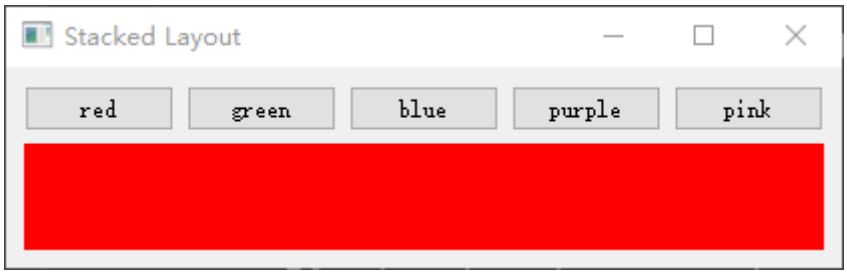
```
colors = ['red', 'green', 'blue', 'purple', 'pink']
for n, color in enumerate(colors):
    stacked_layout.addWidget(Color(color))

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

- 完整代码如下

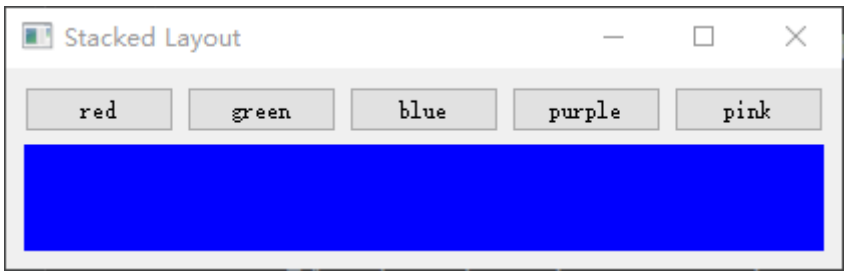
```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *


class Color(QWidget):
    def __init__(self, color, *args, **kwargs):
        super(Color, self).__init__(*args, **kwargs)
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.Window, QColor(color))
        self.setPalette(palette)


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

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('Stacked Layout')
```

```
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'))
32          # stacked_layout.addWidget(Color('blue'))
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
40          colors = ['red', 'green', 'blue', 'purple', 'pink']
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()
49          widget.setLayout(page_layout)
50
51          self.setCentralWidget(widget)
52
53
54  def main():
55      app = QApplication(sys.argv)
56      window = MainWindow()
57      window.show()
58      app.exec_()
59
60
61  if __name__ == '__main__':
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

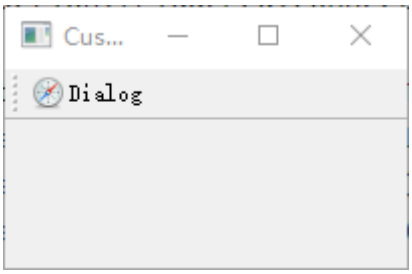
```
class CustomDialog(QDialog):
    def __init__(self, *args, **kwargs):
        super(CustomDialog, self).__init__(*args, **kwargs)

        self.setWindowTitle('Hello')

        # 创建按钮
        btn = QDialogButtonBox.Ok | QDialogButtonBox.Cancel

        # 将按钮放在按钮盒中
        self.buttonbox = QDialogButtonBox(btn)
        # 将信号与槽相连
        self.buttonbox.accepted.connect(self.accept)
        self.buttonbox.rejected.connect(self.reject)

        self.layout = QVBoxLayout()
        self.layout.addWidget(self.buttonbox)
        self.setLayout(self.layout)
```

- 完整代码如下

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


class CustomDialog(QDialog):
    def __init__(self, *args, **kwargs):
        super(CustomDialog, self).__init__(*args, **kwargs)

        self.setWindowTitle('Hello')

        # 创建按钮
        btn = QDialogButtonBox.Ok | QDialogButtonBox.Cancel

        # 将按钮放在按钮盒中
        self.buttonbox = QDialogButtonBox(btn)
        # 将信号与槽相连
        self.buttonbox.accepted.connect(self.accept)
        self.buttonbox.rejected.connect(self.reject)

        self.layout = QVBoxLayout()
        self.layout.addWidget(self.buttonbox)
        self.setLayout(self.layout)


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

```
30
31          self.init_ui()
32
33      def init_ui(self):
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))
42          self.setToolButtonStyle(Qt.ToolButtonTextBesideIcon)
43          self.addToolBar(toolbar)
44
45          toolbar.addAction(action)
46
47      def showDialog(self, s):
48          print('Click', s)
49
50          dialog = CustomDialog()
51          if dialog.exec_():
52              print('Success!')
53          else:
54              print('Cancel!')
55
56
57  def main():
58      app = QApplication(sys.argv)
59      window = MainWindow()
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!。