

借助于 Python 进行QT开发---主要组件详解

Author: qyan.li

Date: 2022.7.7

Topic: 学习整理python进行QT开发基础知识, 主要组件详解

一、写在前面

最近计划开始系统整理一下python进行QT开发的基础知识和技能, 一方面帮助老师整理相关的知识和实例 Demo, 另外一方面也完善自己QT开发的知识, 作为之前QT树形组件的补充和拓展。关于QT树形组件的基本内容, 可以参考我之前的博文: <https://blog.csdn.net/DALEONE/article/details/123676385?spm=1001.2014.3001.5501>

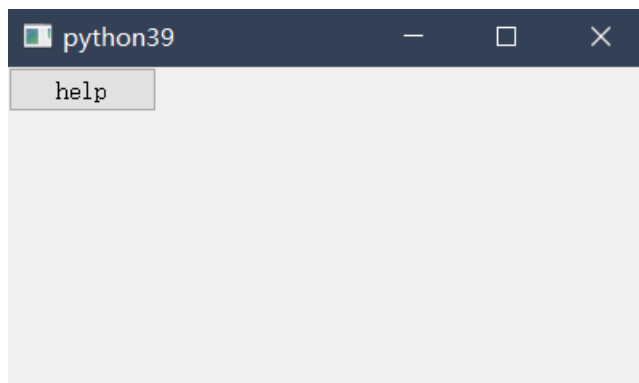
二、QT for Python 基础组件

2.1 QPushButton 组件

- 组件简介:

QPushButton组件, 顾名思义是定义QT界面中按钮的类, QT官网对其的界定为Perhaps the most commonly used widget in user's interface, 足见其在QT界面开发设计中的重要性, 但是从应用开发的层面讲, 相对而言较为简单。

- 界面展示:



- 组件使用:

- QPushButton的实例化及基础使用:

```
button1 = QPushButton("OK", self) ## QPushButton实例化
button1.clicked.connect(lambda: self.onClicked(button1))
def onClicked(self, button):
    print("Button {0} is clicked.".format(button.text()))
```

小Tips:

- QPushButton的构造函数中传入str作为Button的label, 显示在创建成功的按钮上
- Button的text方法用于获取Button的名称(上述实例化时传入的label)

此处的connect函数重点强调：

QT开发过程中，每个组件都不可能单独存在，都必须与一定的作用效果相关联（不是光秃秃摆个图标上去，否则图形化界面就没有意义）。

`Button.clicked.connect()` 即说明Button按钮背后关联的作用：`print("Button {0} is clicked.".format(button.text()))` 输出被按下按钮的名称。

提醒：

此处onClicked函数需要传入参数，因此借助于lambda表达式完成函数传入调用，后续在其他组件中还会有其他传入的方式

关于lambda表达式，也可以参考我的博文：[\(2条消息\).python中lambda表达式，列表生成式，map\(\)函数的基础用法详解 隔壁李学长的博客 CSDN 博客lambda 列表](#)

2. QPushButton的其他使用：

- `setText()`函数修改Button的标签信息：

按钮Button的两种设定方式：

1. constructor方法实例化时直接传入
2. button借助于`setText`方法进行修改

- QPushButton快捷键设置：

QPushButton官网提供一种快捷键设置的方法，在Button的label前添加“&”，注意此时的快捷键由Alt键和&后的第一个字母确定。

```
button1.setText("&help")
```

- 根据上述解释，此时Button1的快捷键为Alt+h
- 顺便提醒一下，使用&&表示真正的&

- 默认按键设置`setAutoDefault`

官网关于默认键设置给出的说明为 `A default btton is a push button that is activated when the user presses the Enter or Return key in the dialog`，简单讲：将现有的按钮设置为默认，在用户按下Enter或者Return时会被默认激活

```
button1.setAutoDefault(True)
```

OK，上述关于QPushButton的全部内容已经讲解完成，完整的代码：

```
import sys
from PyQt5.Qtwidgets import QApplication, QPushButton, QWidget

class MainWidget(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        button1 = QPushButton("OK", self) ## 创建名为ok的Button

        ## label的内容可以在构造函数创建，也可借助于setText函数实现后期修改
        button1.setText("help")
```

```

## 设置默认按键驱动Button(Enter or Return key in the dialog)
button1.setAutoDefault(False)

## 快捷键设置(简单的快捷键的设置方法: 字母前加&, 快捷键即为alt+字母)
# button1.setText("&help")

button1.clicked.connect(lambda: self.onClicked(button1))

def onClicked(self, button):
    print("Button {0} is clicked.".format(button.text()))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWidget()
    window.resize(400, 200)
    window.show()

    sys.exit(app.exec_())

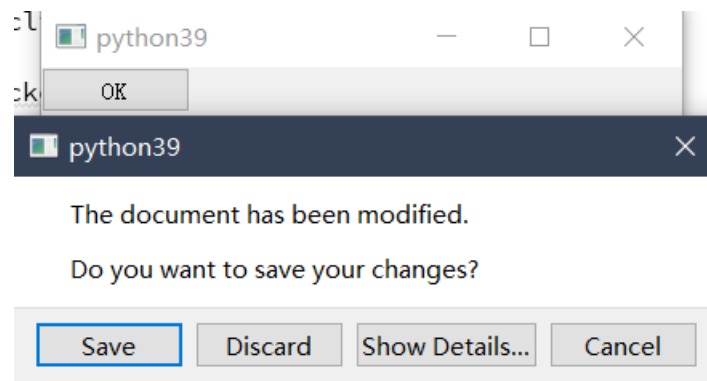
```

2.2 QMessageBox 组件

- 组件简介:

QMessageBox 主要用于弹出对话框显示信息提示和指导用户, 官方文档给出的定义为 The QMessageBox class provides a modal dialog for informing the user or for asking the user a question and receiving an answer, 翻译过来简单讲就是给予用户信息, 询问用户问题, 接收用户输入。

- 界面展示:



- 组件应用:

QMessageBox 在 QT 创建中也是非常常见的, 在警告框, 确认框等方面应用颇多, 下面也还是主要介绍 QMessageBox 的两种构造方法:

1. 借助于静态函数 information, critical, question, warning

```
ret = QMessageBox.question(self, "question Message",
                            "The document has been modified.\n Do you want to\n save your changes?",
                            QMessageBox.Save | QMessageBox.Discard
                            | QMessageBox.Cancel,
                            QMessageBox.Save)
```

上述代码展示如何借助于静态函数实现询问对话框的制作

小Tips:

1. 关于 `question` 函数的几点说明:

关于 `question` 函数, 官网给出两种构造方法, 简单总结下:

- `question(parent, title, text, button1, button2)`, 返回值为 `int`
- `question(parent, title, text, buttons, defaultButton)`, 返回值为 `StandardButton`

在本例中, 使用的为重载过后的 `question` 方法, `QMessageBox.Save | QMessageBox.Discard | QMessageBox.Cancel` 表示 `Buttons`, 而 `QMessageBox.Save` 表示 `defaultButton`

这一点可以在代码中借助于 `print('type(ret)')` 说明, 发现输出值为 `<class 'PyQt5.QtWidgets.QMessageBox.StandardButton'>` (虽然此处代码直接输出为数字)

2. 关于重载函数中 `buttons` 和 `defaultButton` 的说明:

`buttons` 使用 `|` 连接, 表示会直接在对话框中显示的按钮, 而 `defaultButton` 则表示默认情况下按钮情况

3. `question, warning, critical, information` 按钮创建和使用方法类似, 区别仅在于对话框左侧的图形显示不同, 具体内容详见官方文档

2. 借助于函数定义方法自己创建:

与 `QFileDialog` 相似, 借助于函数定义的方法创建比静态方法更自由, 限制小, 但是创建的难度和复杂度比静态方法要高。

```
msgBox = QMessageBox() # QMessageBox class 实例化
msgBox.setText("The document has been modified.")
msgBox.setInformativeText("Do you want to save your changes?")

msgBox.setStandardButtons(QMessageBox.Save | QMessageBox.Discard |
                          QMessageBox.Cancel)
msgBox.setDefaultButton(QMessageBox.Save)
msgBox.setDetailedText("这是一个测试用例\n用于测试setDetailedText的具体用法")

ret = msgBox.exec()
print(ret == QMessageBox.Save)
```

小Tips:

1. `setText` 和 `setInformativeText` 可以用于设置在对话框中显示的信息, 内部传入 `str` 类型字符串即可

2. `setStandardButtons` 和 `setDefaultButton` 分别用于设置标准按钮和默认按钮，与静态方法创建时相同
3. `setDetailedText` 用于添加详细信息，调用函数后对话框上会自动添加 `show Details` 按钮，点击后即可显示详细信息，内部传入 `str` 字符串即可
4. `ret = msgBox.exec()` 用于调用对话框，该函数必须被调用，否则对话框不显示，看不见任何效果
5. `ret` 用于接收函数返回值(用户点击的按钮)，因此，假设用户此时点击 `save`，`print` 的输出值为 `true`

OK，上述关于QMessageBox组件的全部内容已经介绍完毕，提供完整代码，方便大家参考借鉴：

```
import sys
from PyQt5.Qtwidgets import QApplication, QMessageBox, QWidget, QPushButton

class MainWidget(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        button = QPushButton("OK", self) ## 界面中设置一个Button

        self.resize(800, 600)
        button.clicked.connect(self.onOKClicked)

    def onOKClicked(self):
        msgBox = QMessageBox()
        ## 显示信息
        msgBox.setText("The document has been modified.")
        msgBox.setInformativeText("Do you want to save your changes?")
        ## 小界面中设置Button
        msgBox.setStandardButtons(QMessageBox.Save | QMessageBox.Discard |
QMessageBox.Cancel)
        msgBox.setDefaultButton(QMessageBox.Save)

        # 设置详细信息的显示，会自动添加Button按钮
        msgBox.setDetailedText("这是一个测试用例\n用于测试setDetailedText的具体用法")

        ## ret本质上会返回用户选择的按钮
        ret = msgBox.exec()
        print(ret == QMessageBox.Save)

'''借助于静态函数进行实例化'''
# ## warning的弹出框
# ret = QMessageBox.warning(self, "Warning Message",
#                             "The document has been modified.\n Do you want to
save your changes?",
#                             QMessageBox.Save | QMessageBox.Discard
#                             | QMessageBox.Cancel,
#                             QMessageBox.Save)

# ## information的弹出框
```

```

        # ret = QMessageBox.information(self, "Information Message",
        #                               "The document has been modified.\n Do you want to
save your changes?",
        #                               QMessageBox.Save | QMessageBox.Discard
        #                               | QMessageBox.Cancel,
        #                               QMessageBox.Save)

        # ## question的弹出框
        # ret = QMessageBox.question(self, "Question Message",
        #                             "The document has been modified.\n Do you want to
save your changes?",
        #                             QMessageBox.Save | QMessageBox.Discard
        #                             | QMessageBox.Cancel,
        #                             QMessageBox.Save)

        # ## critical的弹出框
        # ret = QMessageBox.critical(self, "critical Message",
        #                             "The document has been modified.\n Do you want to
save your changes?",
        #                             QMessageBox.Save | QMessageBox.Discard
        #                             | QMessageBox.Cancel,
        #                             QMessageBox.Save)
        # print(type(ret))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWidget()
    window.resize(400, 200)
    window.show()

    sys.exit(app.exec_())

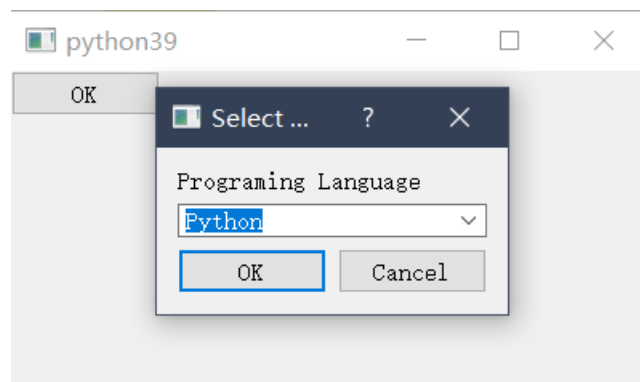
```

2.3 QInputDialog 组件

- 组件简介：

QInputDialog 组件是一个可接收用户输入的对话框，官网给出的解释为：The QInputDialog class provides a simple convenience dialog to get a single value from the user，翻译过来即为提供一个方便的对话框用于接收用户输入。

- 界面展示：



- 组件应用：

`QInputDialog`与上述组件相同，同样提供**静态函数**便捷的创建对话框实例，提供的静态函数主要有 `getDouble()` , `getInt()` , `getItem()` , `getText()` , `getMultiLineText()` , 下面我们主要介绍其中的两种用于说明此静态函数的使用方法：

1. `getInt()`静态函数：

```
num, ok = QInputDialog.getInt(self, "Input an int number", "num:")
if ok:
    print("input num: ", num)
```

`getMultiLineText()`静态函数：

```
text, ok = QInputDialog.getMultiLineText(self, "Input MultiLineText", "Text:")
if ok:
    print("input text: ", text)
```

关于上述函数的一些小 Tips：

- 大家会发现 `getInt()` , `getMultiLineText()` 调用方式基本相同，实际上，其余静态函数的调用方式都类似(下面提及的 `getItem()` 函数传入的参数与其他有些不同)
- 静态函数的输入参数主要包含两个：`Input an int number` 为弹窗的标题，`num:` 为在对话框中显示的具体内容
- 静态函数的返回值有两个：`value` 和 `ok` 标签，`value` 表示用户输入的内容（可以为数字，字符串等等），`ok` 返回 `bool` 值，表示用户是否进行选择
- 上述展示的均为基本用法，更多详细的用法和函数见官方文档，参考链接：<https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QInputDialog.html>

2. `getItem()`静态函数：

```
items = ["C++", "Python", "Java", "Go"]
item, ok = QInputDialog.getItem(self, "Select an Item", "Programing Language",
                                items, 0, False)
if ok and item:
    print("selected item: ", item)
```

小 Tips：

1. `getItem()` 与其余静态函数功能相同，同样是提供对话框供用户选择，但区别在于 `getItem` 给用户一个区间供其选择，而并非用户随意输入
2. `getItem()`接收输入的参数多些，前面的两个 `text` 相同，分别表示对话框标题和具体内容，后续需要接收三个参数，`items---list, current---int, editable---bool` 三个参数分别代表，**供用户选择的待选列表，列表默认显示字条index，是否可编辑**
3. 返回值与上述 `getInt()` 相同，更多详细信息参照官方文档

上述 `QInputDialog` 的介绍全部完成，完整代码，方便大家参考借鉴：

```
import sys
from PyQt5.Qtwidgets import QApplication, QInputDialog, QWidget, QPushButton
```

```

class MainWidget(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        button = QPushButton("OK", self)

        self.resize(800, 600)
        button.clicked.connect(self.onOKClicked)

    def onOKClicked(self):

        # QInputDialog提供多个用户选择getItem,getText,getInt,getDouble
        items = ["C++", "Python", "Java", "Go"]
        item, ok = QInputDialog.getItem(self, "Select an Item", "Programing Language",
items, 1, True)
        if ok and item:
            print("selected item: ", item)

        text, ok = QInputDialog.getText(self, "Input an text", "text:")
        if ok:
            print("input text: ", text)

        num, ok = QInputDialog.getInt(self, "Input an int number", "num:")
        if ok:
            print("input num: ", num)

        text, ok = QInputDialog.getMultiLineText(self, "Input MultiLineText", "Text:")
        if ok:
            print("input text: ", text)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWidget()
    window.resize(400, 200)
    window.show()

    sys.exit(app.exec_())

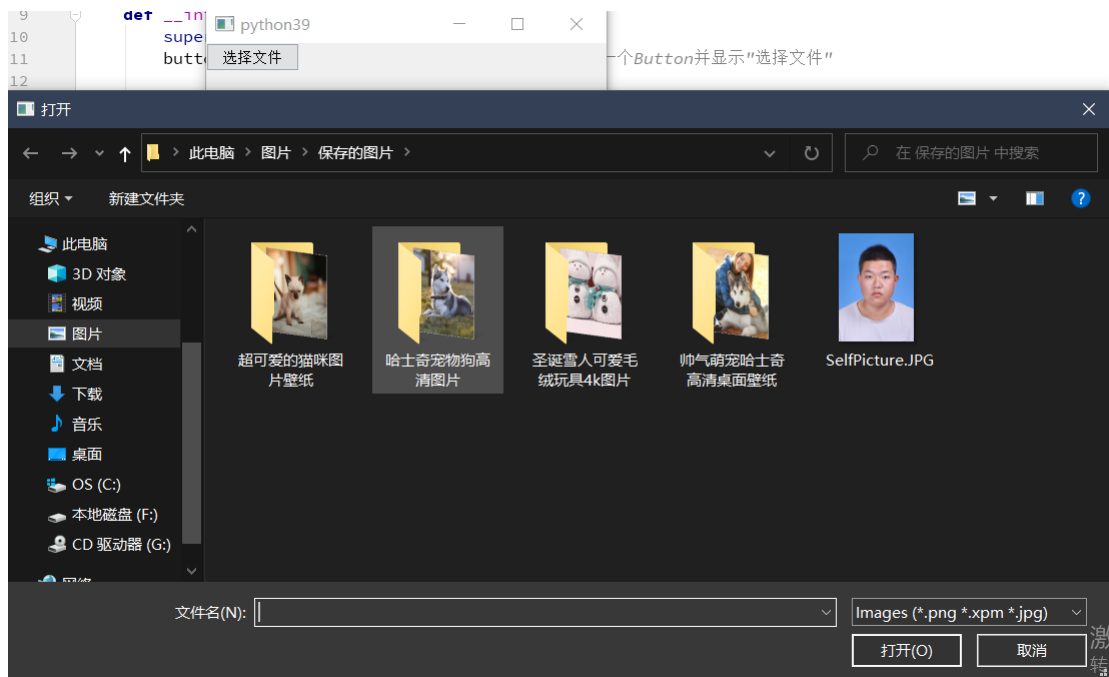
```

2.4 QFileDialog组件

- 组件简介:

印象中,应该是在[树形组件博文](#)中简单介绍过此组件,官网对其的界定为 `The QFileDialog class provides a dialog that allows users to select files or directories`,翻译过来就是提供一个对话框供用户选择文件或者目录。

- 界面展示:



- 组件应用:

在官网中, QFileDialog提供两种构造的方式: 1. 借助于静态函数- `static function` 2. 自己创建

1. 借助于静态函数 `getOpenFileName`

```
fname, _ = QFileDialog.getOpenFileName(self, "open file", 'c:/Users\臧味\Desktop\ClashForWindows', "Images (*.jpg *.gif);;Text files (*.txt)") # 选择文件并将选择的文件路径进行返回
print(fname)
```

小Tips:

1. 常用的几个参数:

`caption:{str}` 对应于 "open file" 用于定义打开窗口的名称

`dir:{str}` 用于定义打开窗口默认的文件路径

`selectedFilter:{str}` 用于定义待操作文件的可选类型

2. 函数返回值:

`fileName, selectedFilter` 原代码中的 `_` 即代表 `selectedFilter`, 用下划线表示我们不关心, 但是原函数确实返回, 故借其占位

3. 其他:

官方文档永远是最好的参考文档, `getOpenFileName` 更多可选参数见官方文档, 传送门:

[QFileDialog — Qt for Python](#)

2. 借助于函数自行创建

相比于使用官方提供的静态函数, 借助于函数自我创建的方法显然更加复杂, 但借助于此种方法的优点在于限制小, 可操作性大

```

fileNames = []
dialog = QFileDialog(self) # QFileDialog类实例化
dialog.setFileMode(QFileDialog.AnyFile)

## 指定打开的文件类型:源代码中添加tr,但查阅相关资料后发现tr可以不适用,且一般情况下不推荐使用
# 参考文献:https://blog.csdn.net/weixin_41567783/article/details/118416484
dialog.setNameFilter("Images (*.png *.xpm *.jpg)")

## 理论上, list仅显示文件名和文件夹列表, 而Details同时显示详细信息(文件大小, 修改时间等)
# dialog.setViewMode(QFileDialog.Detail)
# dialog.setViewMode(QFileDialog.List)

dialog.setDirectory('C:/Users\腻味\Pictures\Saved Pictures') ## 此函数可以设计打开时的文件夹

```

小Tips:

1. `setFileMode` 函数用于设置 `FileMode`, 此处有三种模式可供选择:
`AnyFile, ExistingFile, Directory`
 - `AnyFile` 可选任何类型, 甚至不存在的文件, 因此常用于 `save as` 类的对话框
 - `ExistingFile` 可选确实存在的文件
 - `Directory` 可选目录, 不可选具体文件
2. `setNameFilter` 用于指定文件的待选格式, 如果阅读源码会发现源码中字符串前加 `tr`, 但查阅资料未找到 `tr` 的相关定义, 最终在一篇博文中提及: `tr` 似乎是方便语言切换设计的模块, 此处可以不使用

Reference: https://blog.csdn.net/weixin_41567783/article/details/118416484

3. `setViewMode` 设置文件显示信息, 从官方文档看, 其包含有两种模式 `QFileDialog.Detail` 和 `QFileDialog.List`, 一种仅显示文件列表, 另外一种还可以显示文件详细信息(如文件大小, 修改时间等), 但从代码实际运行的效果观察, 并未起到作用, 无论设置任何 `Mode`, 最终显示均为详细信息
4. `setDirectory` 用于设定对话框打开默认的文件路径, 内部参数传入 `DirPath` 即可
5. 更多信息参照官方文档

上述有关于QFileDialog的介绍以全部完成, 附完整代码方便大家参考借鉴:

```

## QFileDialog模块学习
import sys
from PyQt5.Qtwidgets import QApplication, QFileDialog, QWidget, QPushButton

class MainWindow(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        button = QPushButton("选择文件", self)

        self.resize(800, 600)
        button.clicked.connect(self.onOKClicked)

```

```

def onOKClicked(self):

    '''1. Using a static function-getOpenFileName'''
    # fname, _ = QFileDialog.getOpenFileName(self, "Open file", 'C:/Users\臧味
\ Desktop\ClashForWindows', "Images (*.jpg *.gif);;Text files (*.txt)") # 选择文件并将选择的
文件路径进行返回
    # print(fname)

    '''2. creat our own QFileDialog'''
    fileNames = [] # 首先赋值为empty directory
    dialog = QFileDialog(self)
    dialog.setFileMode(QFileDialog.AnyFile)
    dialog.setNameFilter("Images (*.png *.xpm *.jpg)")

    # # dialog.setViewMode(QFileDialog.Detail)
    # # dialog.setViewMode(QFileDialog.List)
    dialog.setDirectory('C:/Users\臧味\Pictures\Saved Pictures')

    if dialog.exec_():
        fileNames = dialog.selectedFiles()
        print(fileNames)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWidget()
    window.resize(400, 200)
    window.show()

    sys.exit(app.exec_())

```

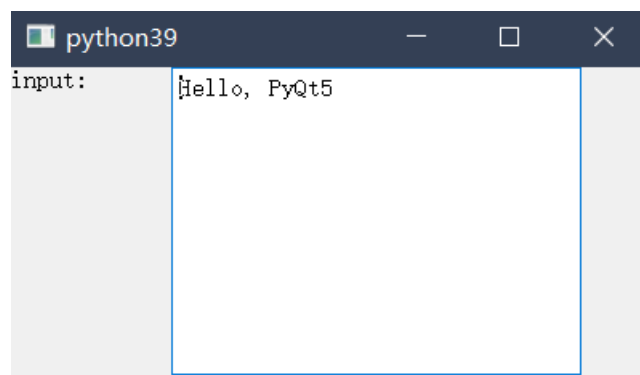
这里作为一个分界线，上面主要针对于QT开发中的核心模块进行介绍，下面会针对于一些小部件进行介绍：

2.5 QTextEdit组件

- 组件简介：

QTextEdit组件也是QT开发中重要且常用的组件，官方文档中给出的界定为 `QTextEdit class provides a widget which is used to edit and display plain and rich text`。翻译过来QTextEdit组件提供一个用于显示或编辑纯文本和富文本的组件。

- 界面展示：



- 组件应用:

```
import sys
from PyQt5.QtWidgets import QApplication, QTextEdit, QWidget, QLabel
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QIntValidator

class MainWidget(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        label = QLabel("input: ", self)
        label.move(0, 0)
        self.textEdit = QTextEdit(self)
        self.textEdit.move(100, 0)

        self.textEdit.setPlainText("Hello, PyQt5")
        self.textEdit.textChanged.connect(self.displayText)

    def displayText(self):
        print(self.textEdit.toPlainText())

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWidget()
    window.resize(400, 200)
    window.show()
```

小Tips:

上述代码展示QtextEdit的基础用法，主要涉及两个函数：

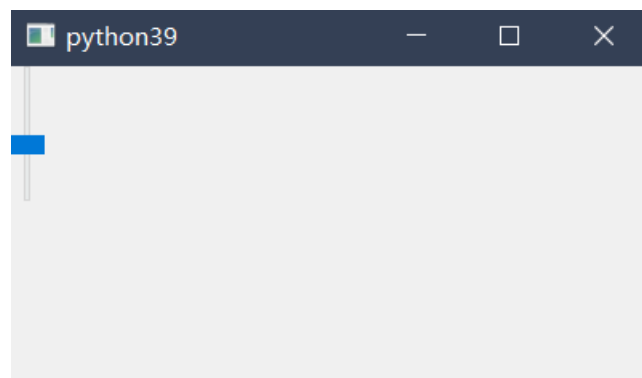
1. setPlainText()用于在组件中显示纯文本内容"hello PyQt5"
2. textChanged.connect()函数用于在组件内容改变时打印组件的文本内容，调用的形式和作用与QPushButton组件类似

2.6 QSlider组件

- 组件简介:

QSlider组件提供以上下或左右滑动的方式交互式选择数据，QT开发中在某些特定场景下应用较为广泛

- 界面展示:



- 组件应用:

```
import sys
from PyQt5.QtWidgets import QApplication, QSlider, QWidget
from PyQt5.QtCore import Qt

class MainWidget(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)

        ## 水平滑栏
        # slider = QSlider(Qt.Horizontal, self)
        ## 竖直滑栏
        slider = QSlider(Qt.Vertical, self)
        ## 设置slider的上下限
        slider.setMaximum(20)
        slider.setMinimum(10)

        slider.valueChanged.connect(self.onValueChanged)

    def onValueChanged(self, value):
        print("current value is {}".format(value))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWidget()
    window.resize(400, 200)
    window.show()

    sys.exit(app.exec_())
```

小Tips:

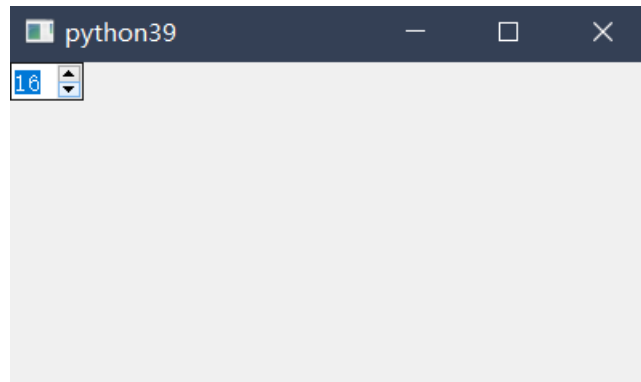
1. QSlider组件主要分为水平和竖直两种方式，分别在constructor构造函数时通过不同的参数传入实现水平或者竖直Slider的创建
Qt.Horizontal代表水平，Qt.Vertical代表竖直
2. setMaximum和setMinimum函数分别用于设置滑栏滑动过程中的最大和最小值
3. valueChanged.connect()用于在滑栏选值改变时，输出对应选择的数值

2.7 QSpinBox组件

- 组件简介:

QSpinBox组件提供一组离散的数值集合供用户选择, `handle integers or discrete sets of values`, 个人理解功能与上面QSlider差别不太, 不同的效果。

- 界面展示:



- 组件应用:

```
import sys
from PyQt5.Qtwidgets import QApplication, QSpinBox, QWidget

class MainWidget(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)

        spinBox = QSpinBox(self)
        ## SpinBox各种方法测试
        spinBox.setMinimum(10)
        spinBox.setMaximum(20)
        ## 设置调整的步长
        spinBox.setSingleStep(2)

        ## 循环操作
        spinBox.setWrapping(True)

        ## valueChanged信号函数
        spinBox.valueChanged.connect(self.onValueChanged)

        ## 将选择的数据进行返回
        def onValueChanged(self, value):
            print("current value is {}".format(value))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWidget()
    window.resize(400, 200)
    window.show()

    sys.exit(app.exec_())
```

小Tips:

1. spinBox同样有setMinimum()和setMaximum()函数用于设置数值选择的上限和下限
2. setSingleStep()函数内部传入数值, 用于设置调整一次的步长(点击一下, 数值变化多少)
3. setWrapping()函数用于设定数值变换的过程中是否会循环(即到最大值后再点击是否会回到最小值)

2.8 QComboBox组件

- 组件简介:

QComboBox组件用于构造图形化界面开发中的下拉框，以满足用户可以在开发者事先设定的若干词条中进行选择。

- 界面展示:



- 组件应用:

```
import sys
from PyQt5.Qtwidgets import QApplication, QComboBox, QWidget

class MainWidget(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)

        self.combo = QComboBox(self)
        ## 添加元素addItem
        self.combo.addItem("Apple")
        self.combo.addItem("HuaWei")
        self.combo.addItem("XiaoMi")
        self.combo.addItem("Oppo")

        ## 添加元素insertItem(index,str)
        self.combo.insertItem(1,"Meizu")
        ## 修改元素setItemText(index,str)
        self.combo.setItemText(1,"Chuizi")
        ## 删除元素removeItem(index)
        # self.combo.removeItem(1)
        ## 清除所有元素clear(all items can be removed)
        ## 获取指定index处的标签itemText(idnex)
        # string = self.combo.itemText(1)
        # print(string)

        ## comboBox每个词条内容可编辑
        # self.combo.setEditable(True)
        ## 设置内容是否可以重复
        # self.combo.setDuplicatesEnabled(True)

        self.combo.currentIndexChanged.connect(self.onCurrentIndex)
```

```
def onCurrentIndex(self, index):
    ## 获取当前词条的标签内容
    print("current item is {}".format(self.combo.currentText()))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.resize(400, 200)
    window.show()

    sys.exit(app.exec_())
```

小Tips:

1. QComboBox**添加词条**的两个函数: addItem和insertItem
addItem(item)按照由上而下的顺序添加词条
insertItem(index,item)可以将词条插入在指定的index位置上
2. QComboBox修改词条: setItemText(index,item)
3. QComboBox删除和清空函数: remove(index), clear()
4. itemText(index)用于获取指定词条处元素的标签label
5. setEditable()和setDuplicatesEnabled()分别用于设置QComboBox()中item词条是否可编辑, 是否可重复
6. currentText()结合currentIndexChanged实现在屏幕上打印用户选择的词条内容

三、总结

上面简单的介绍和总结下使用Python进行QT开发时常用组件的基本用法, 但是都只是皮毛, 远没有达到开发的水平, 作者最近在尝试利用上述组件开发一个整体的界面。也推荐读者自行尝试, 真正开发过一个小系统, 才会真正的掌握此项技术, 一起努力吧。

作者初次学习, 若有谬误, 批评指正!!!!