QT-中级应用与课堂练习

本节重点：

- 中级应用
- 作业完成

1. 加载图片资源
2. 鼠标与键盘事件
3. 右键菜单
4. 菜单栏、工具栏与状态栏
5. 调用其他页面
6. 控件高级显示

1. 加载图片资源

在使用PyQt进行图形界面开发的时候不免要用到一些外部资源，比如图片，qss配置文件等。在前面代码中，遇到这类问题，我们使用绝对路径的方式来解决，这种方式，本身有其不方便之处(比如，调整图片路径后，对应代码需要修改，代码的维护性变差), 还有在app进行打包发布的时候，通常图片资源却不方便处理，app安装的时候还必须确保app可以正确地找到这些资源，还有出于保护一些资源文件的出发，不便于直接发布图片，qss资源等等。

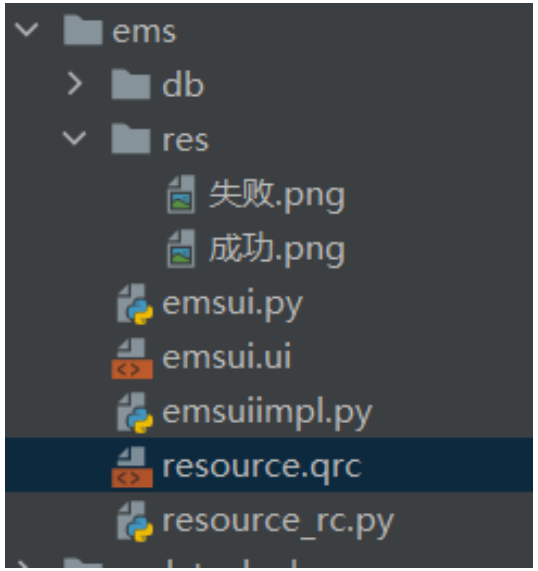
为解决这些问题，我们可以使用Qt官方提供的工具将这些资源文件编译成二进制文件，直接打包到程序中。为了编译这些资源文件，通常我们需要在代码目录下创建一个.qrc后缀的文件，为了叙述方便，我们假设这个文件的名字为resource.qrc，它和我们主py文件位于同一个目录。

一个 .qrc 资源集合文件是用来指定文件将被嵌入其资源的 XML 文件。根节点为 RCC ，里面可以包含若干 qresource 节点，每个 qresource 有自己的 prefix(路径前缀)属性，qresource 节点包含了若干 file 节点，描述了各个文件相对于 .qrc 的路径。在 C++ Qt 的 IDE 中，我们添加资源后他会自动编译出二进制文件。但在 PyQt 中，我们需要手动来完成这些操作。

1. 加载图片资源

配置过程

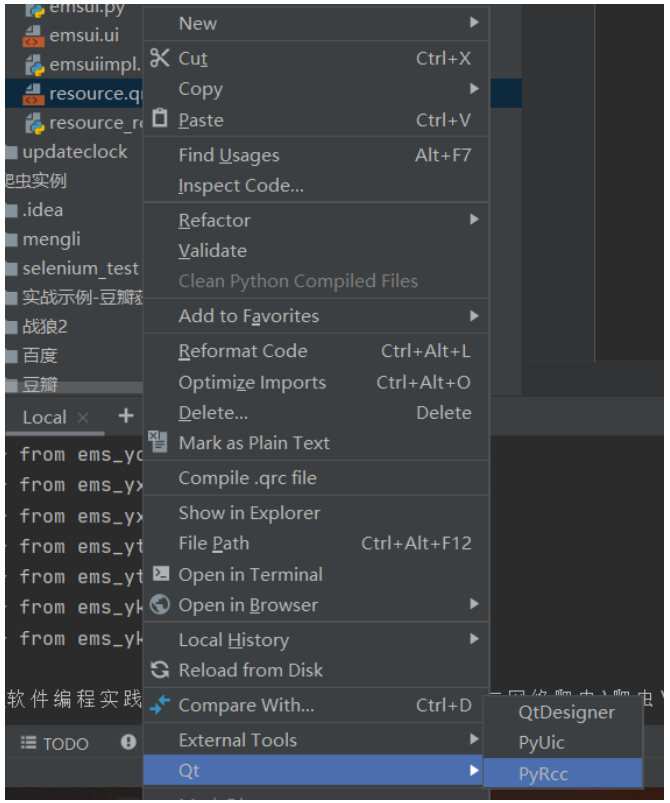
(1)
建立res文件夹，放置需要加载的图片资源



(2)
与python 同目录，编写resource.rc

```
<!DOCTYPE RCC>
<RCC version="1.0">
    <qresource>
        <file>res/成功.png</file>
        <file>res/失败.png</file>
    </qresource>
</RCC>
```

(3)
调用Qt/PyRcc，生成resource_rc.py



1. 加载图片资源

使用过程

(1)
import resource_rc.py

(2)
pix = QPixmap(":/res/失败.png")

```
# pix = QPixmap("./res/失败.png")
pix = QPixmap(":/res/失败.png")
height = self.ui.labelRtuStatus.height()
pix = pix.scaled(height, height)
self.ui.labelRtuStatus.setPixmap(pix)
```

提示：请注意这两种写法的区别(.与:)：

- ✓ 第1种是不依赖于rcc的资源加载， pix = QPixmap("./res/失败.png")
- ✓ 第2种是依赖于rcc的资源加载，pix = QPixmap(":/res/失败.png")

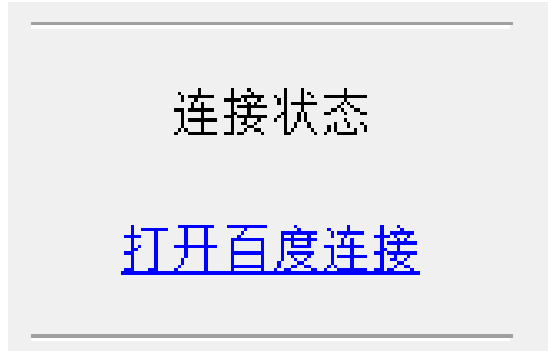
1. 控件中的高级显示功能

Qlabel中显示图片

```
# pix = QPixmap("./res/失败.png")
pix = QPixmap(":/res/失败.png")
height = self.ui.labelRtuStatus.height()
pix = pix.scaled(height, height)
self.ui.labelRtuStatus.setPixmap(pix)
```

Qlabel中显示网址

```
label.setText("<a href='http://www.baidu.com'>打开百度连接</a>")
```



2. 鼠标与键盘事件

- pyqt中，每个事件类型都被封装成相应的事件类，如鼠标事件为QMouseEvent，键盘事件为QKeyEvent等。而它们的基类是QEvent。

- 基类QEvent的几个重要方法：
  - ✓ accept() 表示事件已处理，不需要向父窗口传播
  - ✓ ignore()表示事件未处理，继续向父窗口传播
  - ✓ type()返回事件类型，如QtCore.QEvent.MouseButtonPress，一般由基事件调用。因为其它事件已经知道自己的事件类型了。
- 还有一个自定义事件的注册方法。

2. 鼠标与键盘事件

- 按下并释放键时，以下方法将被调用：
  - keyPressEvent(self,event) - 按下某一键时，该方法被调用直到键被释放为止；
  - keyReleaseEvent(self,event) - 释放之前按下的键时被调用。
- event参数为QKeyEvent对象，包括事件的相关信息，有以下方法可调用：
  - key():返回按下键的值；
  - text():返回按下键的Unicode字符编码信息，当按键为Shift, Control, Alt等时，则该函数返回的字符为空值；
  - modifiers():判断按下了哪些修饰键（Shift,Ctrl , Alt,等等）。返回值为QtCore.Qt 类以下枚举变量的组合：
- 控件必须可以设置为输入焦点。有些控件，如QLabel是不能接受输入焦点的。
- 捕获键盘事件要使用grabKeyboard( )方法，释放时，调用rekeaseKeyboard().
- 如果要让父控件继续收到键盘事件，要调用事件的ignore()方法；否则，调用accept()。

2. 鼠标与键盘事件

- 在某一时刻，只有一个控件(或根本没有)可以获得输入焦点。指定输入焦点可使用QWidget::setFocus([reason]) - 如果控件在活动窗口中，调用此方法后，该控件成为输入焦点。
  - ✓ clearFocus() - 去除输入焦点；
  - ✓ hasFocus() - 如果控件是输入焦点，返回True；否则，返回False;
  - ✓ focusWidget() - 返回最后调用setFocus()方法的控件对象；

```
"""重定义键盘事件"""
def keyPressEvent(self,e ):
    if e.key() == Qt.Key_Up:
        self.lab1.setText("↑")
    elif e.key() == Qt.Key_Down:
        self.lab1.setText("↓")
    elif e.key() == Qt.Key_Left:
        self.lab1.setText("←")
    else:
        self.lab1.setText("→")
```

```
@CatchExcept
def keyPressEvent(self, a0: QKeyEvent) -> None:
    self.ui.label.setText("keyPressEvent " + f"key:{a0.key()};" +
                          "text:" + str(a0.text()) +
                          ";modifiers:" + str(a0.modifiers()))

@CatchExcept
def keyReleaseEvent(self, a0: QKeyEvent) -> None:
    self.ui.label.setText("keyReleaseEvent " + f"key:{a0.key()};" +
                          "text:" + str(a0.text()) +
                          ";modifiers:" + str(a0.modifiers()))
```

2. 鼠标与键盘事件

- QMouseEvent鼠标事件：
  - buttons()返回哪个鼠标按键被按住了，如Qt.LeftButton
  - globalPos()返回鼠标相对屏幕的位置QPoint
  - pos()返回鼠标相对处理事件的窗口的位置
- 处理鼠标事件的响应函数（在QWidget及其继承类中）：
  - mousePressEvent(QMouseEvent)，鼠标键按下时调用;
  - mouseReleaseEvent(event)，鼠标键公开时调用;
  - mouseMoveEvent(event)，双击鼠标时调用。必须注意，在双击之前的其他事件
- 双击时的事件顺序如下：

(1)	(2)	(3)	(4)	(5)
MouseButton Press	MouseButton Release	MouseButton DblClick	MouseButton Press	MouseButton Release

2. 鼠标与键盘事件

- event参数是QMouseEvent对象，存储事件的其他信息。有以下方法：
  - ✓ x() 和 y() -返回相对于控件空间的鼠标坐标值;
  - ✓ pos() - 返回相对于控件空间的QPoint对象;
  - ✓ localPos()- 返回相对于控件空间的QPointF对象;
  - ✓ globalX() 和 globalY() - 返回相对于屏幕的x,y 坐标值；
  - ✓ globalPos() - 返回相对于屏幕的QPoint对象;
  - ✓ windowPos() - 返回相对于窗口的QPointF对象;
  - ✓ screenPos() - 返回相对于屏幕的QPointF对象;
  - ✓ button() - 返回以下枚举值

2. 鼠标与键盘事件

- 如果要让父控件继续收到鼠标事件，要调用事件的ignore()方法；
  - ✓ 否则，调用accept()。
- 如果一个控件的QtCore.Qt.WA_NoMousePropagation的属性设为True, 则不会将事件传递给父控件。
  - ✓ 调用setAttribute( )方法可修改此参数：
  - ✓ button.setAttribute (QtCore.Qt.WA_NoMousePropagation, True)
- 缺省情况下，鼠标事件只拦截控件区域上的鼠标操作。
  - ✓ 如果可拦截控件区域以下的鼠标事件，必须调用grabMouse( )方法；
- 释放时，调用releaseMouse( )。

2. 鼠标与键盘事件

- 鼠标移进和移出控件 鼠标移进和移出控件时，下列方法将被调用：
  - ✓ enterEvent (self, event) -鼠标进入控件;
  - ✓ leaveEvent (self, event) - 鼠标离开控件;
  - ✓ event是一个QEvent对象，并不包括附加信息。

更改鼠标指针形状要修改鼠标进入控件后的形状，可调用如下方法：

- QWidget::setCursor(QCursor qcr) 方法，参数qcr为QCursor对象或 Qtcore.Qt 类的枚举值，典型如：
  - ✓ ArrowCursor(标准箭头)
  - ✓ upArrowCursor(向上箭头)
  - ✓ CrossCursor(十字光标)
  - ✓ Waitcursor (沙漏) 。
- Qwidget:: unsetCursor() - 取消设置的鼠标形状。
- Qwidget:: cursor() - 返回当前鼠标形状的QCursor对象，

2. 鼠标与键盘事件

- 要处理鼠标指针的移动，需要重载mouseMoveEvent(self，event)方法。
  - ✓ 缺省情况下，只有按下鼠标键移动时，才会调用mouseMoveEvent( )。
  - ✓ 如果要处理包括普通的移动，需要以参数为True调用setMouseTracking() 方法。
  - ✓ 如果要处理窗口中鼠标移动的事件，需要调用grabMouse( )方法。

- event对象的pos( )返回值为相对控件的坐标，要坐标转换，需要调用QWidget类的以下方法：
  - mapToGlobal (QPoint) - 将窗口坐标转换成屏幕坐标;
  - mapFromGlobal(QPoint) - 将屏幕坐标转换成窗口坐标;
  - mapToParent(QPoint) - 将窗口坐标转换成父窗口坐标。若无父窗口，则相当于mapToGlobal (QPoint);
  - mapFromParent(QPoint) - 将父窗口坐标转换成窗口坐标。若无父窗口，则相当于mapFromGlobal(QPoint);
  - mapTo (QWidget, QPoint) - 将窗口坐标转换成 QWidget父窗口坐标;
  - mapFrom (QWidget, QPoint) - 将 QWidget父窗口坐标转换成窗口坐标;

2. 鼠标与键盘事件

- wheelEvent (self, event)方法可用来处理鼠标滚动事件。event是一个QWheelEvent对象，包含操作：
  - ✓ angleDelta( ) - 返回QPoint对象，为滚轮转过的数值，单位为1/8度，例如：
    - ✓angle=event.angleDelta( ) /8，angleX=angle.x()，angleY=angle.y()
  - ✓ pixelDeita () - 返回QPoint对象，为滚轮转过的像素值。
  - ✓ x() 和 y() - 返回相对于控件的当前鼠标的x,y位置;
  - ✓ pos() - 返回相对于控件的当前鼠标位置的QPoint对象;
  - ✓ posF() - 返回相对于控件的当前鼠标位置的QPoinFt对象;
  - ✓ globalX() 和globalY() - 返回相对于屏幕的当前鼠标的x,y位置;
  - ✓ globalPos() - 返回相对于屏幕的当前鼠标QPoint位置;
  - ✓ globalPosF() - 返回相对于屏幕的当前鼠标QPointF位置;
  - ✓ buttons(),modifiers()和timestamp()的用法参见上文。

如果要让父控件继续收到滚轮事件，要调用事件的ignore()方法；否则，调用accept()。

2. 鼠标与键盘事件

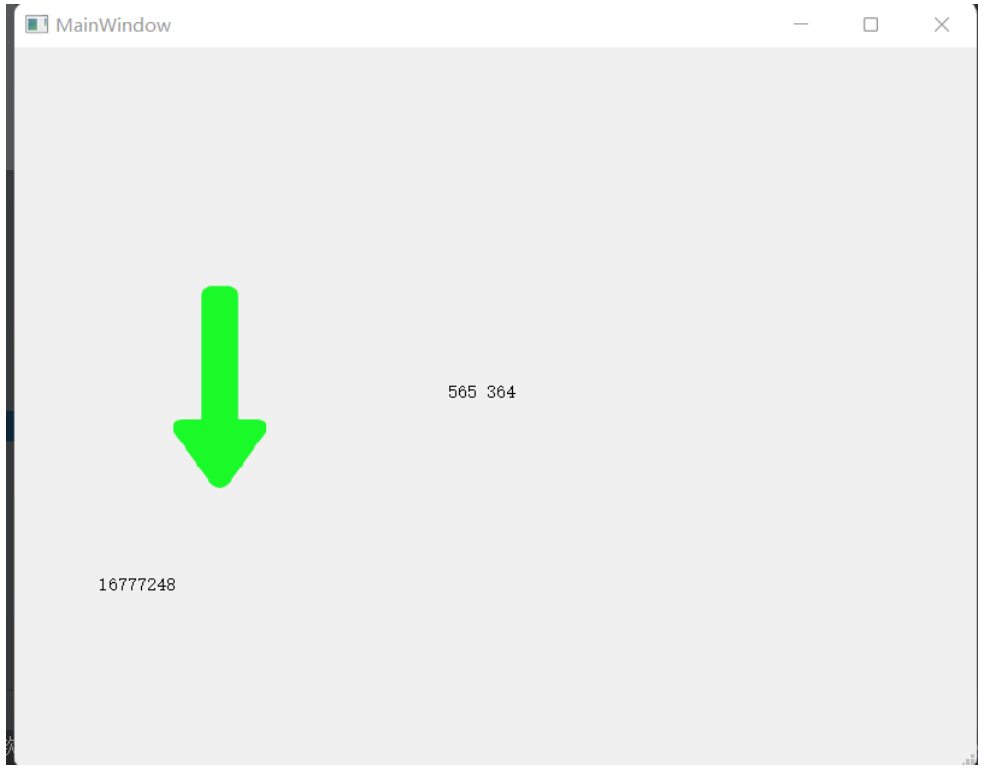
- 使用QApplication类中的以下静态方法来控制整个应用程序的鼠标形状：
  - ✓ setOverrideCursor(QCursor qcr) - 参数qcr为QCursor对象或 Qtcore.Qt 类的枚举值。
  - ✓ restoreOverrideCursor() - 取消全局鼠标形状设置；
  - ✓ changeOverrideCursor(QCursor qcr) - 将鼠标形状设置为qcr，只有先调用setOverrideCursor( )了，该函数才起作用，overrideCursor( ) - 返回当前鼠标形状的QCursor 对象；
- setOverrideCursor()和restoreOverrideCursor( )通常配合使用。

```python
@CatchExcept
def mouseMoveEvent(self, event) -> None:
    print("mouseMoveEvent", event.x(), event.y())

@CatchExcept
def mousePressEvent(self, event):
    print("mousePressEvent", event.x(), event.y())

@CatchExcept
def mouseReleaseEvent(self, event: QMouseEvent) -> None:
    print("mouseReleaseEvent", event.x(), event.y())
```

```python
@CatchExcept
def mouseDoubleClickEvent(self, event: QMouseEvent) -> None:
    print("mouseDoubleClickEvent", event.x(), event.y())

@CatchExcept
def leaveEvent(self, a0: QEvent) -> None:
    print("leaveEvent", a0)

@CatchExcept
def enterEvent(self, a0: QEvent) -> None:
    print("enterEvent", a0)
```

2.课堂练习（1）

3个控件，分别展示鼠标坐标、按键值、按键字符串（如果是上下左右按键，则展示图片）。

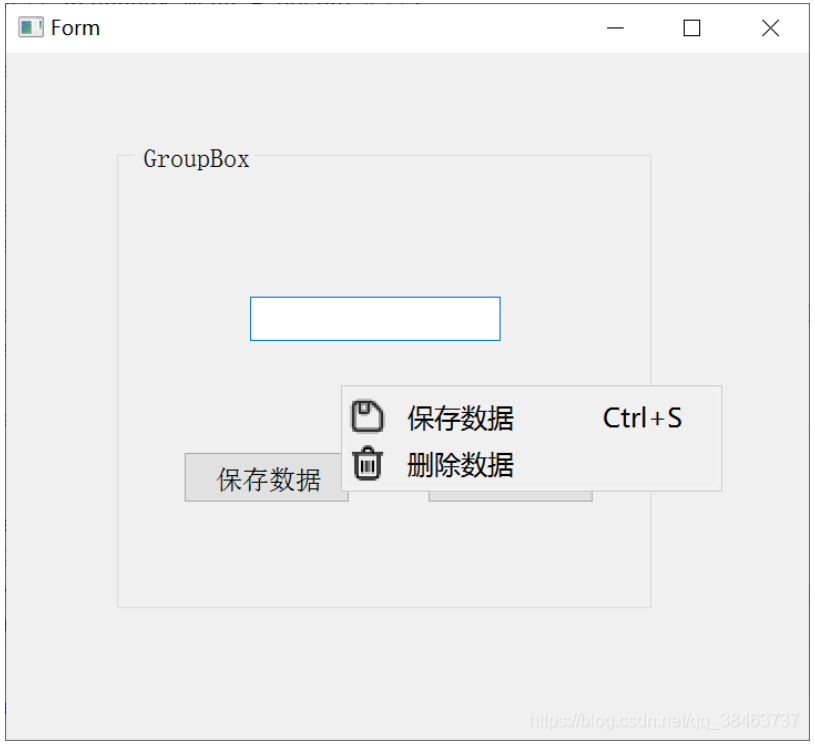


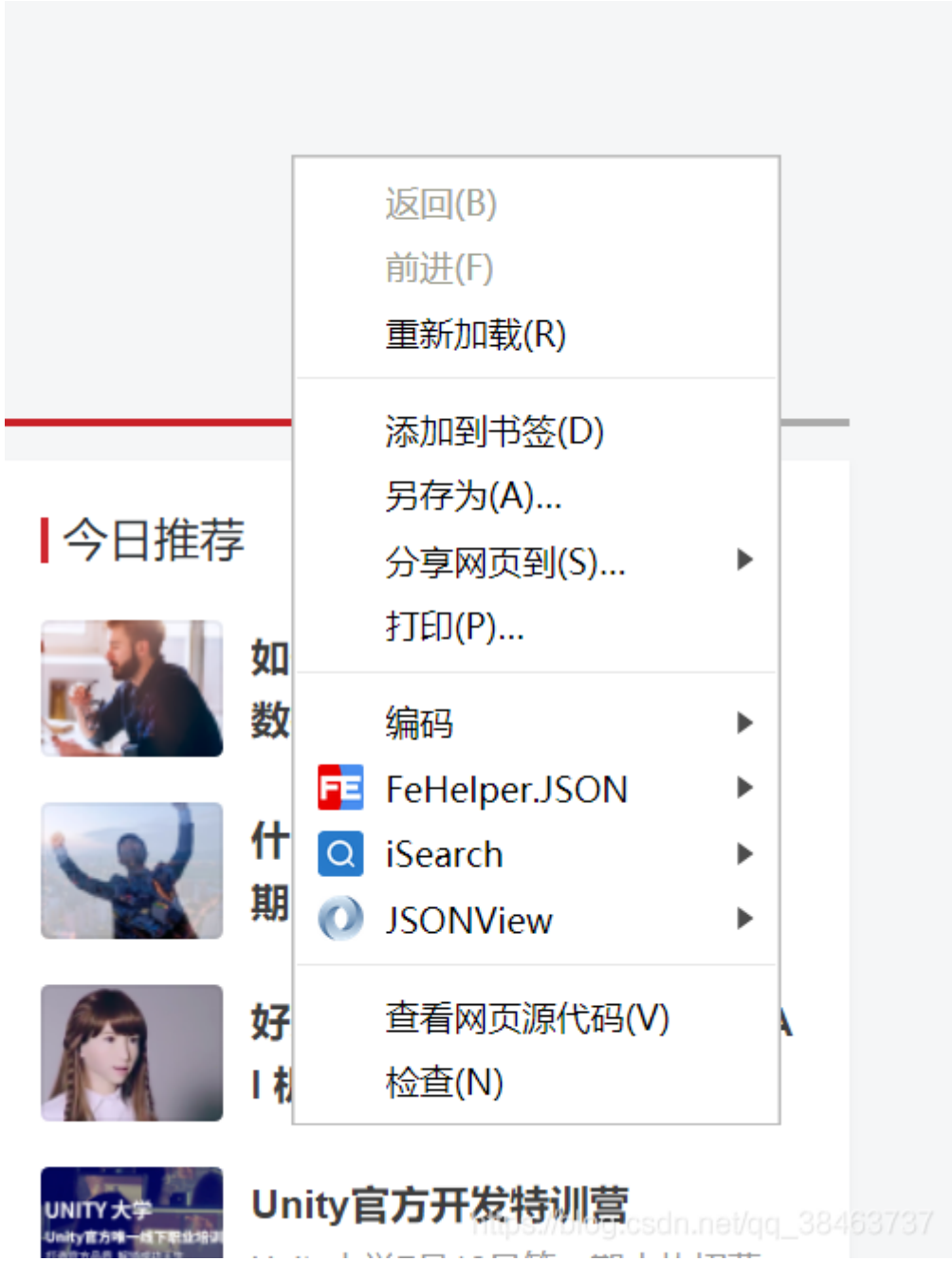

```python
def mouseMoveEvent(self, a0: QMouseEvent) -> None:
    self.ui.labelMousePos.setText(f"{a0.x()} {a0.y()}")

def keyPressEvent(self, a0: QKeyEvent) -> None:
    dic = {Qt.Key_Up: "up", Qt.Key_Down: "down",
           Qt.Key_Left: "left", Qt.Key_Up: "right"}
    png = dic.get(a0.key(), None)
    if png:
        width = self.ui.labelKeyPng.width()
        height = self.ui.labelKeyPng.height()
        pix = QPixmap(f":/png/{png}.png").scaled(width, height)
        self.ui.labelKeyPng.setPixmap(pix)
    else:
        self.ui.labelKeyPng.setText(a0.text())

    self.ui.labelKeyInfo.setText(f"{a0.key()}")
```

3. 鼠标右键菜单

一个完整的界面程序怎么少得了右键菜单呢？如在浏览器的右键菜单

我做的右键菜单如下图，其包括：图标，选项名称，快捷键





3.鼠标右键菜单

3.1 声明创建右键菜单

```
#声明在groupBox创建右键菜单
self.groupBox.setContextMenuPolicy(Qt.CustomContextMenu)
self.groupBox.customContextMenuRequested.connect(self.create_rightmenu)  # 连接到菜单显示函数
```

这两句声明语句是必须要有的。其中：

- self.grounpBox 是要创建右键菜单的控件，只有在groupBox控件上右击鼠标才弹出菜单。**如果是创建整个pyqt5 界面的右键菜单，用self 就可以。**
- self.create_rightmenu 是创建右键菜单的槽函数。虽然第一句声明了右键菜单，但其实它里面是没有内容的，槽函数就是创建右键菜单的具体内容。

其他关键字是PyQt5 创建右键菜单的关键字，如果没有把握，不能更改。

3.鼠标右键菜单

3.2 创建右键菜单槽函数

```
#创建右键菜单函数
def create_rightmenu(self):      #菜单对象
    self.groupBox_menu = QMenu(self)
    self.actionA = QAction(QIcon('image/保存.png'),u'保存数据',self)          #创建菜单选项对象
    self.actionA.setShortcut('Ctrl+S' )          #设置动作A的快捷键
    self.groupBox_menu.addAction(self.actionA)          #把动作A选项对象添加到菜单self.groupBox_menu上
    self.actionB = QAction(QIcon('image/删除.png'),u'删除数据',self)
    self.groupBox_menu.addAction(self.actionB)
    self.actionA.triggered.connect(self.button)          #将动作A触发时连接到槽函数 button
    self.actionB.triggered.connect(self.button_2)
    self.groupBox_menu.popup(QCursor.pos())
```

声明当鼠标在groupBox控件上右击时，在鼠标位置显示右键菜单 ,exec_,popup两个都可以，

3.鼠标右键菜单

3.3 代码示例：

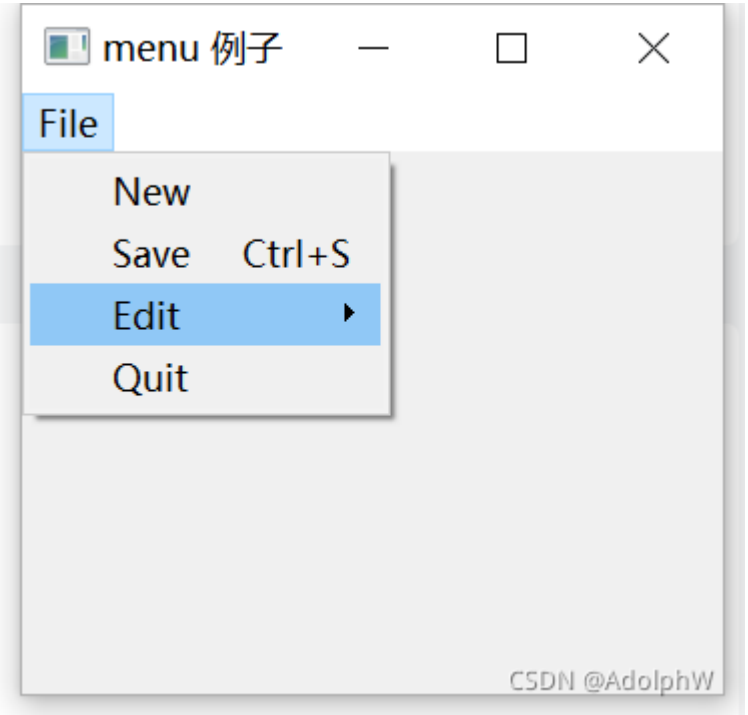
```
    self.tabYc.setContextMenuPolicy(Qt.CustomContextMenu)
    self.tabYc.customContextMenuRequested.connect(self.create_menu_tabyc)

def create_menu_tabyc(self):
    self.menu = QMenu(self)
    self.action1 = QAction(QIcon(":res/成功.png"), "打开1")
    self.action2 = QAction(QIcon(":res/失败.png"), "关闭2")
    self.action3 = QAction("其他3")
    self.menu.addAction(self.action1)
    self.menu.addAction(self.action2)
    self.menu.addSeparator()
    self.menu.addAction(self.action3)

    self.action1.triggered.connect(self.open)
    self.action2.triggered.connect(self.close)
    self.action3.triggered.connect(self.other)
    self.menu.exec(QCursor.pos())


def open(self):
    QMessageBox.information(self, "main", "open")
    pass
def close(self):
    QMessageBox.information(self, "main", "close")
    pass
```

```
    self.setContextMenuPolicy(Qt.CustomContextMenu)
    self.customContextMenuRequested.connect(self.create_menu)

def create_menu(self):
    self.menu = QMenu(self)
    self.action1 = QAction(QIcon(":res/成功.png"), "打开")
    self.action2 = QAction(QIcon(":res/失败.png"), "关闭")
    self.action3 = QAction("其他")
    self.menu.addAction(self.action1)
    self.menu.addAction(self.action2)
    self.menu.addSeparator()
    self.menu.addAction(self.action3)

    self.action1.triggered.connect(self.open)
    self.action2.triggered.connect(self.close)
    self.action3.triggered.connect(self.other)
    self.menu.exec(QCursor.pos())

def open(self):
    QMessageBox.information(self, "main", "open")
    pass
def close(self):
    QMessageBox.information(self, "main", "close")
    pass
```

4. 菜单栏、工具栏与状态栏

菜单栏menuBar

单击任何QAction按钮时，QMenu对象都会发射triggered信号。

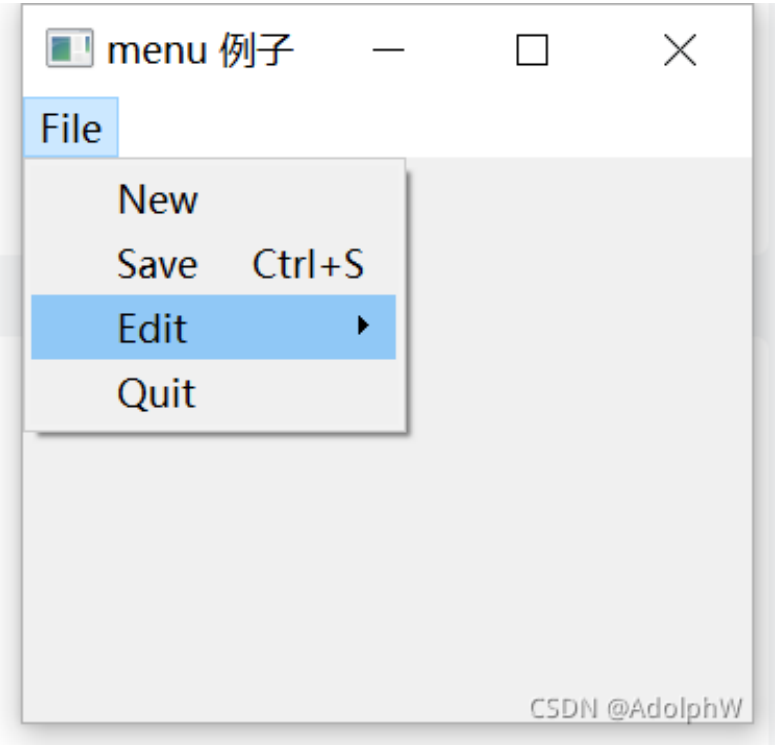


menuBar()	返回主窗口的QMenuBar对象
addMenu()	在菜单栏中添加一个新的QMenu对象
addAction()	向QMenu小控件中添加一个操作按钮，其中包含文本或图标
setEnabled()	将操作按钮状态设置为启用/禁用
addSeperator()	在菜单中添加一条分隔线
clear()	删除菜单/菜单栏的内容
setShortcut()	将快捷键关联到操作按钮
setText()	设置菜单项的文本
setTitle()	设置QMenu小控件的标题
text()	返回与QAction对象关联的文本
title()	返回QMenu小控件的标题

4. 菜单栏、工具栏与状态栏

菜单栏，menuBar

单击任何QAction按钮时，QMenu对象都会发射triggered信号。




```
class MenuDemo(QMainWindow):
    def __init__(self, parent=None):
        super(MenuDemo, self).__init__(parent)
        layout = QHBoxLayout()
        bar = self.menuBar()
        file = bar.addMenu("File")
        file.addAction("New")
        save = QAction("Save", self)
        save.setShortcut("Ctrl+S")
        file.addAction(save)
        edit = file.addMenu("Edit")
        edit.addAction("copy")
        edit.addAction("paste")
        quit = QAction("Quit", self)
        file.addAction(quit)
        # 菜单发射triggered信号，将该信号连接到槽函数proecesstrigger(),
        file.triggered[QAction].connect(self.processtrigger)
        self.setLayout(layout)
        self.setWindowTitle("menu 例子")
        self.resize(350, 300)

    def processtrigger(self, q):
        print(q.text() + " is triggered")
```

4. 菜单栏、工具栏与状态栏

菜单栏，menuBar，代码分析：

在这个例子中，顶层窗口必须是QMainWindow对象，才可以引用QMenuBar对象
通过addMenu（）方法将'File'菜单添加到菜单栏

```
        bar=self.menuBar()
        #向菜单栏中添加新的QMenu对象，父菜单
        file=bar.addMenu('File')123
```

菜单栏中的操作按钮可以是字符串或QAction对象

```
        file.addAction('New')

        #定义响应小控件按钮，并设置快捷键关联到操作按钮，添加到父菜单下
        save=QAction('Save',self)
        save.setShortcut('Ctrl+S')
        file.addAction(save)123456
```

将子菜单添加到顶级菜单中

```
        edit=file.addMenu('Edit')
        edit.addAction('Copy')
        edit.addAction('Paste')123
```

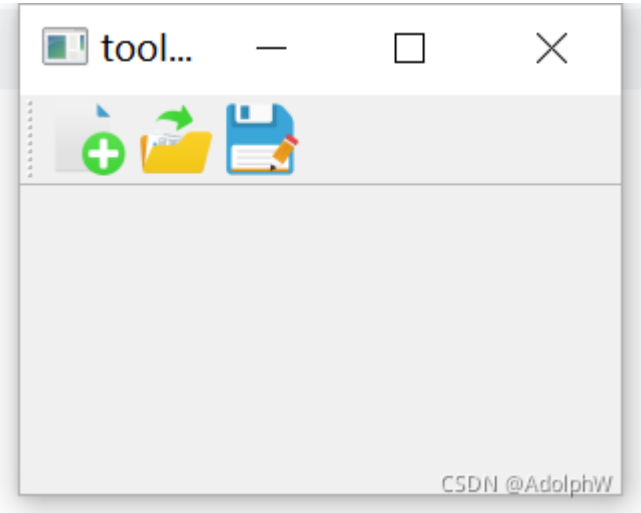
菜单发射triggered信号，将信号连接到槽函数processtrigger（）该函数接受信号的QAction对象

```
file.triggered[QAction].connect(self.processtrigger)
1
```

4. 菜单栏、工具栏与状态栏

工具栏，QToolBar

每当单击工具栏中的按钮时，都将发射**actionTriggered**信号。另外，这个信号将关联的**QAction**对象的引用发送到连接的槽函数上。




```
class ToolBarDemo(QMainWindow):

    def __init__(self, parent=None):
        super(ToolBarDemo, self).__init__(parent)
        self.setWindowTitle("toolbar 例子")
        self.resize(300, 200)

        layout = QVBoxLayout()
        tb = self.addToolBar("File")
        new = QAction(QIcon("./images/new.png"), "new", self)
        tb.addAction(new)
        open = QAction(QIcon("./images/open.png"), "open", self)
        tb.addAction(open)
        save = QAction(QIcon("./images/save.png"), "save", self)
        tb.addAction(save)
        tb.actionTriggered[QAction].connect(self.toolbtnpressed)
        self.setLayout(layout)

    def toolbtnpressed(self, a):
        print("pressed tool button is", a.text())
```

4. 菜单栏、工具栏与状态栏

工具栏，QToolBar，代码分析：

> 在这个例子中，首先调用addToolBar()方法在工具栏区域添加文件工具栏

```
  tb=self.addToolBar('File')
1
```

> 然后，添加具有文本标题的工具按钮，工具栏通常包含图形按钮，具有图标和名称的QAction对象将被添加到工具栏中

```
        new=QAction(QIcon('images\\new.png'),'new',self)
        tb.addAction(new)
        open=QAction(QIcon('images\open.png'),'open',self)
        tb.addAction(open)
        save=QAction(QIcon('images\save.png'),'save',self)
        tb.addAction(save)123456
```

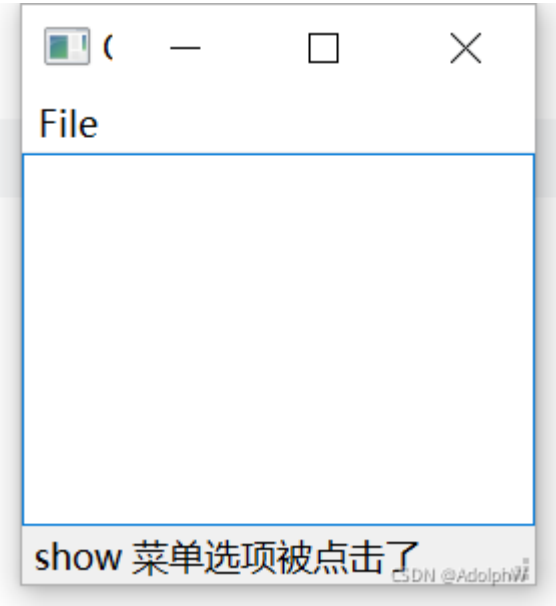
> 最后，将actionTriggered信号连接到槽函数toolbtnpressed（）

```
tb.actionTriggered[QAction].connect(self.toolbtnpressed)
```

4. 菜单栏、工具栏与状态栏

状态栏，QStatusBar

状态栏是位于主窗口的最下方,提供一个显示工具提示等信息的地方，QMainWindow类里面就有一个statusBar()函数，用于实现状态栏的调用。



addWidget()	在状态栏中添加给定的窗口小控件对象
addPermanentWidget()	在状态栏中永久添加给定的窗口小控件对象
showMessage()	在状态栏中显示一条临时信息指定时间间隔
clearMessage()	删除正在显示的临时信息
removeWidget()	从状态栏中删除指定的小控件

```
class StatusDemo(QMainWindow):
    def __init__(self, parent=None):
        super(StatusDemo, self).__init__(parent)
        bar = self.menuBar()
        file = bar.addMenu("File")
        file.addAction("show")
        file.triggered[QAction].connect(self.processTrigger)
        self.setCentralWidget(QTextEdit())
        self.statusBar = QStatusBar()
        self.setWindowTitle("QStatusBar 例子")
        self.setStatusBar(self.statusBar)

    def processTrigger(self, q):
        if (q.text() == "show"):
            self.statusBar.showMessage(q.text() + " 菜单选项被点击了", 5000)
```

4. 菜单栏、工具栏与状态栏

状态栏，QStatusBar，代码分析：

在这个例子中，顶层窗口MainWindow有一个菜单栏和一个QTextEdit对象，作为中心控件
当单击MenuBar的菜单时，将triggered信号与槽函数进行绑定

```
1        #当菜单对象被点击时，触发绑定的自定义的槽函数
2            file.triggered[QAction].connect(self.processTrigger)12
```

当单击show菜单选项时，会在状态栏提示信息，5秒后消失

```
1    #设置状态栏的显示文本以及显示时间
2                self.statusBar.showMessage(q.text()+'菜单选项被点击了',5000)12
```

通过主窗口的QMainWindow的setStatusBar（）函数设置状态栏，核心代码如下

```
1            #实例化状态栏
2            self.statusBar=QStatusBar()
3            #设置状态栏，类似布局设置
4            self.setStatusBar(self.statusBar)1234
```

6. 调用其他页面

调用QWidget/QMainWindow/Qframe，并置顶且非阻塞显示：

```
@CatchExcept
def LoadPage1(self, bChecked=False):
    print("LoadPage1...")
    if not self.inwnd:
        self.inwnd = InputWidget()
    self.inwnd.setWindowFlags(Qt.WindowStaysOnTopHint)  # 置顶
    self.inwnd.show()
    pass
```

6. 调用其他页面

调用QDialog，并阻塞显示：

此时，需要注意

show()

与exec()

的区别

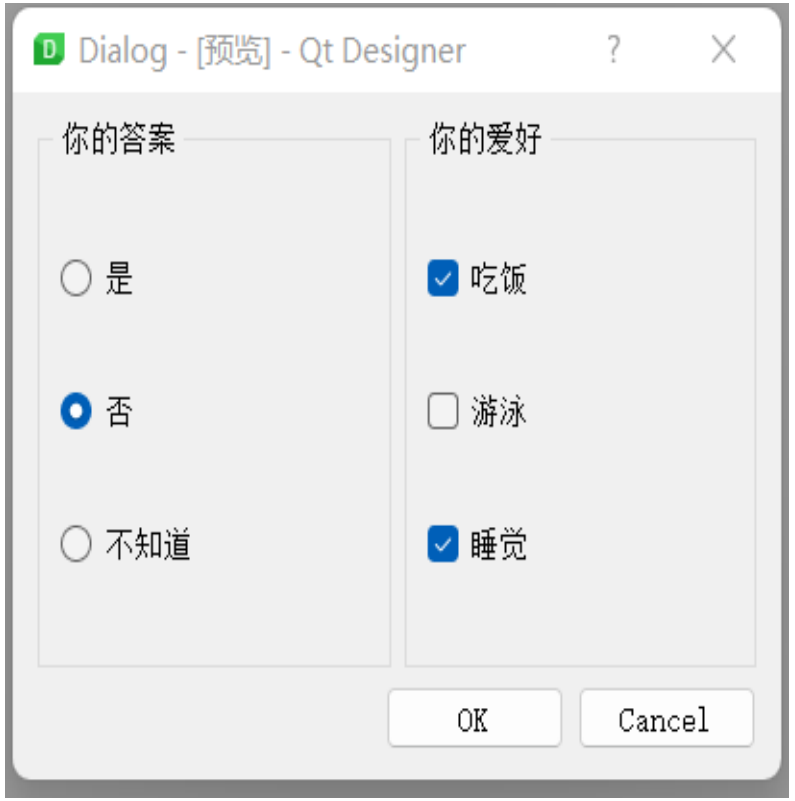
```python
@CatchExcept
def LoadPage2(self, bChecked=False):
    print("LoadPage2...")
    self.dialog = LcdNumDialog()
    # ret = self.dialog.show()
    ret = self.dialog.exec()
    print(ret)
    pass
```

Show：非阻塞运行。

Exec，阻塞运行，并且只有Qdialog才有这个方法

5. 控件高级显示

QButtonGroup，组合按钮，互斥选择与多重选择，如何获取被选中控件？




```
childs = self.ui.groupBoxSingleSelect.children()
selects = [btn for btn in childs if isinstance(btn, QRadioButton)
                and btn.isChecked()]
print(selects[0].text())

childs = self.ui.groupBoxMultiSelect.children()
texts = [btn.text() for btn in childs if isinstance(btn, QCheckBox)
                            and btn.isChecked()]
print(texts)
```

5. 控件高级显示

QTableWidget中显示控件，并居中显示（以Qlabel为例）：

1、生成Qlable对象

```
label = QLabel()
size = table.rowHeight(row)
pix = QPixmap(":/res/失败.png").scaled(size, size)
label.setPixmap(pix)
```

2、生成Qwidget对象，插入Qlabel，并设置为居中对其

```
widget = QWidget()
hLayout = QHBoxLayout()
hLayout.addWidget(label)
hLayout.setAlignment(Qt.AlignCenter)
widget.setLayout(hLayout)
```

2、将Qwidget对象插入到表格对应单元格中

```
table.setCellWidget(row, col, widget)
```

5. 控件高级显示

QTableWidget中的单元格事件（响应双击事件为例）

方式2：

itemDoubleClicked

事件

(只对文本框起作用)

```
        self.ui.tabYc.itemDoubleClicked.connect(self.OnItemDoubleClicked)

        ...

    @CatchExcept
    def OnItemDoubleClicked(self, item):
        if self.sender() == self.ui.tabYc:
            print(item)
            print(item.row())
            print(item.column())
            print(item.text())
```

5. 控件高级显示

QTableWidget中的单元格事件（响应双击事件为例）

方式1：

doubleClicked

事件

(只对文本框与Widget

都起作用)

```
        self.ui.tabYc.doubleClicked.connect(self.OnTableDoubleClick)

    # QTableWidget.cellWidget()
    @CatchExcept
    def OnTableDoubleClick(self, index):
        print("OnTableDoubleClick", self.sender(), index)
        if self.sender() == self.ui.tabYc:
            item = self.ui.tabYc.currentItem()
            if item is None:
                widget = self.ui.tabYc.cellWidget(index.row(), index.column())
                print(widget)
            else:
                print(item)
                print(item.row())
                print(item.column())
                print(item.text())
```

5. 控件高级显示

QTableWidget的高级操作：

```
#设置水平方向的表头标签与垂直方向上的表头标签，注意必须在初始化行列之后进行，否则，没有效果
TableWidget.setHorizontalHeaderLabels(['姓名','性别','体重（kg）'])
#Todo 优化1 设置垂直方向的表头标签
#TableWidget.setVerticalHeaderLabels(['行1', '行2', '行3', '行4'])

#TODO 优化 2 设置水平方向表格为自适应的伸缩模式
##TableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

#TODO 优化3 将表格变为禁止编辑
#TableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)

#TODO 优化 4 设置表格整行选中
#TableWidget.setSelectionBehavior(QAbstractItemView.SelectRows)

#TODO 优化 5 将行与列的高度设置为所显示的内容的宽度高度匹配
#QTableWidget.resizeColumnsToContents(TableWidget)
#QTableWidget.resizeRowsToContents(TableWidget)
```

5. 控件高级显示

QTableWidget的高级操作：

```
#TODO 优化 6 表格头的显示与隐藏
#TableWidget.verticalHeader().setVisible(False)
#TableWidget.horizontalHeader().setVisible(False)

#TOdo 优化7 在单元格内放置控件
# comBox=QComboBox()
# comBox.addItems(['男','女'])
# comBox.addItem('未知')
# comBox.setStyleSheet('QComboBox{margin:3px}')
# TableWidget.setCellWidget(0,1,comBox)
#
# searchBtn=QPushButton('修改')
# searchBtn.setDown(True)
# searchBtn.setStyleSheet('QPushButton{margin:3px}')
# TableWidget.setCellWidget(0,2,searchBtn)
```

```
#添加数据
newItem=QTableWidgetItem('张三')
TableWidget.setItem(0,0,newItem)

newItem=QTableWidgetItem('男')
TableWidget.setItem(0,1,newItem)

newItem=QTableWidgetItem('160')
TableWidget.setItem(0,2,newItem)
```

5. 控件高级显示

QTableWidget的高级操作：

```
#创建新条目，设置背景颜色，添加到表格指定行列中
newItem = QTableWidgetItem("张三")
#newItem.setForeground(QBrush(QColor(255, 0, 0)))
tableWidget.setItem(0, 0, newItem)

# 创建新条目，设置背景颜色，添加到表格指定行列中
newItem = QTableWidgetItem("男")
#newItem.setForeground(QBrush(QColor(255, 0, 0)))
tableWidget.setItem(0, 1, newItem)

# 创建新条目，设置背景颜色，添加到表格指定行列中
newItem = QTableWidgetItem("160")
#newItem.setForeground(QBrush(QColor(255, 0, 0)))
tableWidget.setItem(0, 2, newItem)

# newItem = QTableWidgetItem("李四")
# #将字体加粗，黑色字体
# newItem.setFont(QFont('Times',12,QFont.Black))
# tableWidget.setItem(1, 0, newItem)
```

```
# # 创建新条目，设置背景颜色，添加到表格指定行列中
# newItem = QTableWidgetItem("男")
# newItem.setFont(QFont('Times', 12, QFont.Black))
# tableWidget.setItem(1, 1, newItem)
#
# # 创建新条目，设置背景颜色，添加到表格指定行列中
# newItem = QTableWidgetItem("150")
# newItem.setFont(QFont('Times', 12, QFont.Black))
# tableWidget.setItem(1, 2, newItem)
#
# newItem = QTableWidgetItem("王五")
# #将字体加粗，黑色字体
# newItem.setFont(QFont('Times',12,QFont.Black))
# tableWidget.setItem(2, 0, newItem)
#
# # 创建新条目，设置背景颜色，添加到表格指定行列中
# newItem = QTableWidgetItem("女")
# newItem.setFont(QFont('Times', 12, QFont.Black))
# tableWidget.setItem(2, 1, newItem)
```

5. 控件高级显示

QTableWidget的高级操作：

```
# # 创建新条目，设置背景颜色，添加到表格指定行列中
# newItem = QTableWidgetItem("女")
# newItem.setFont(QFont('Times', 12, QFont.Black))
# tableWidget.setItem(2, 1, newItem)
#
# # 创建新条目，设置背景颜色，添加到表格指定行列中
# newItem = QTableWidgetItem("175")
# newItem.setFont(QFont('Times', 12, QFont.Black))

# 设置单元格文本的对齐方式
#newItem.setTextAlignment(Qt.AlignRight|Qt.AlignBottom)
#tableWidget.setItem(2, 2, newItem)


#按照体重排序
#Qt.DescendingOrder降序
#Qt.AscEndingOrder升序
#tableWidget.sortItems(2,Qt.DescendingOrder)
```

```
#合并单元格
#tableWidget.setSpan(2,0,4,1)

#设置单元格的大小
#将第一列的单元宽度设置为150
#tableWidget.setColumnWidth(0,150)
#将第一行的单元格高度的设置为120
#tableWidget.setRowHeight(0,120)


#表格中不显示分割线
#tableWidget.setShowGrid(False)

#隐藏垂直头标签
#tableWidget.verticalHeader().setVisible(False)
```

5. 控件高级显示

树形结构是通过QTreeWidget和QTreeWidgetItem类实现的，其中QTreeWidgetItem类实现了节点的添加

```
self.tree=QTreeWidget()
#设置列数
self.tree.setColumnCount(2)
#设置树形控件头部的标题
self.tree.setHeaderLabels(['Key','Value'])

#设置根节点
root=QTreeWidgetItem(self.tree)
root.setText(0,'Root')
root.setIcon(0,QIcon('./images/root.png'))
```

```
# todo 优化2 设置根节点的背景颜色
brush_red=QBrush(Qt.red)
root.setBackground(0,brush_red)
brush_blue=QBrush(Qt.blue)
root.setBackground(1,brush_blue)

#设置树形控件的列的宽度
self.tree.setColumnWidth(0,150)
```

5. 控件高级显示

树形结构是通过QTreeWidget和QTreeWidgetItem类实现的，其中QTreeWidgetItem类实现了节点的添加

```
#设置子节点1
child1=QTreeWidgetItem()
child1.setText(0,'child1')
child1.setText(1,'ios')
child1.setIcon(0,QIcon('./images/IOS.png'))

#todo 优化1 设置节点的状态
child1.setCheckState(0,Qt.Checked)

root.addChild(child1)
```

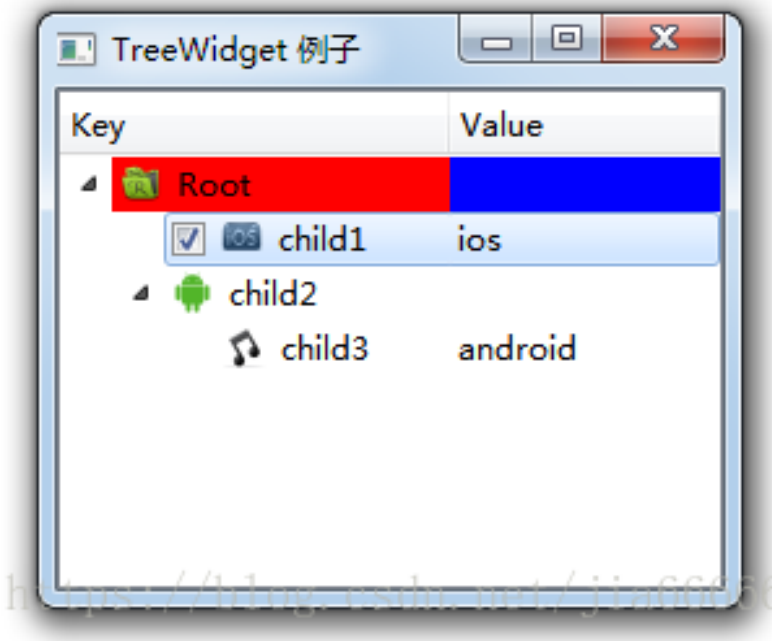
```
#设置子节点2
child2=QTreeWidgetItem(root)
child2.setText(0,'child2')
child2.setText(1,'')
child2.setIcon(0,QIcon('./images/android.png'))

#设置子节点3
child3=QTreeWidgetItem(child2)
child3.setText(0,'child3')
child3.setText(1,'android')
child3.setIcon(0,QIcon('./images/music.png'))
```

5. 控件高级显示

树形结构是通过QTreeWidget和QTreeWidgetItem类实现的，其中QTreeWidgetItem类实现了节点的添加..

```
    #加载根节点的所有属性与子控件
    self.tree.addTopLevelItem(root)

    #TODO 优化3 给节点添加响应事件
    self.tree.clicked.connect(self.onClicked)


    #节点全部展开
    self.tree.expandAll()
    self.setCentralWidget(self.tree)

def onClicked(self,qmodeLindex):
    item=self.tree.currentItem()
    print('Key=%s,value=%s'%(item.text(0),item.text(1)))
```

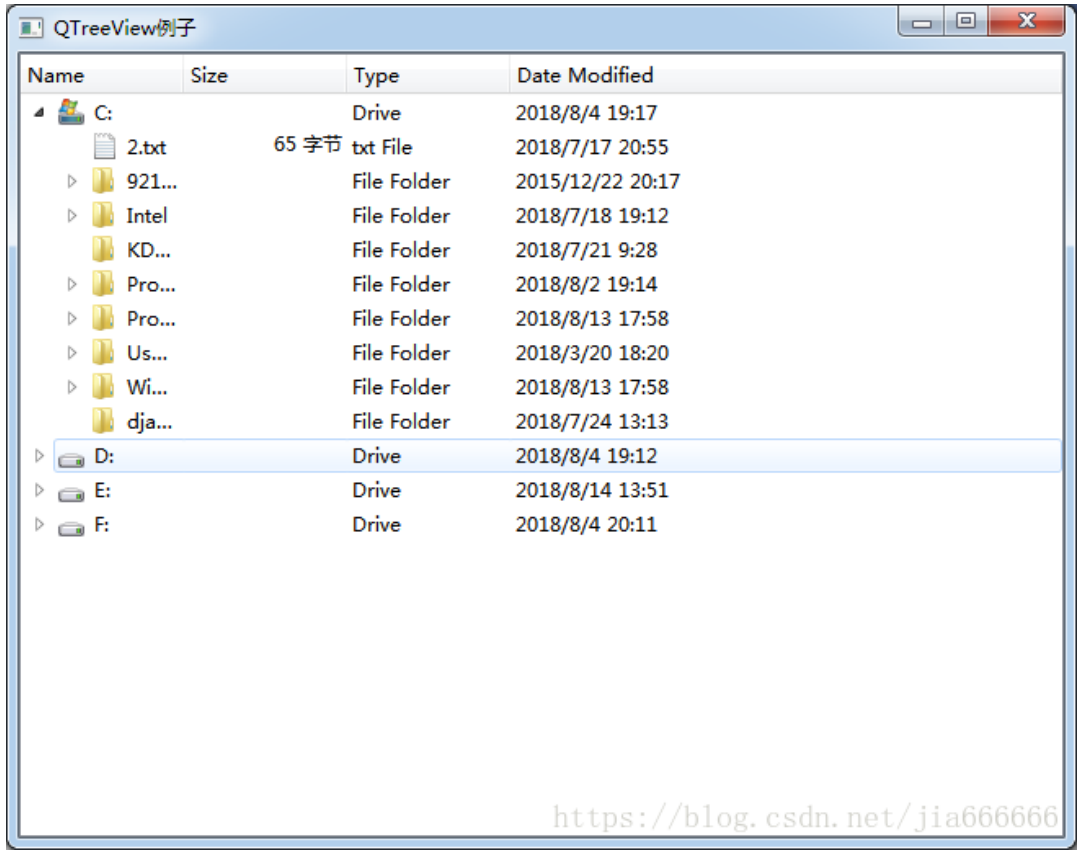
```
Key=child3,value=android

Key=child1,value=ios
```



5. 控件高级显示

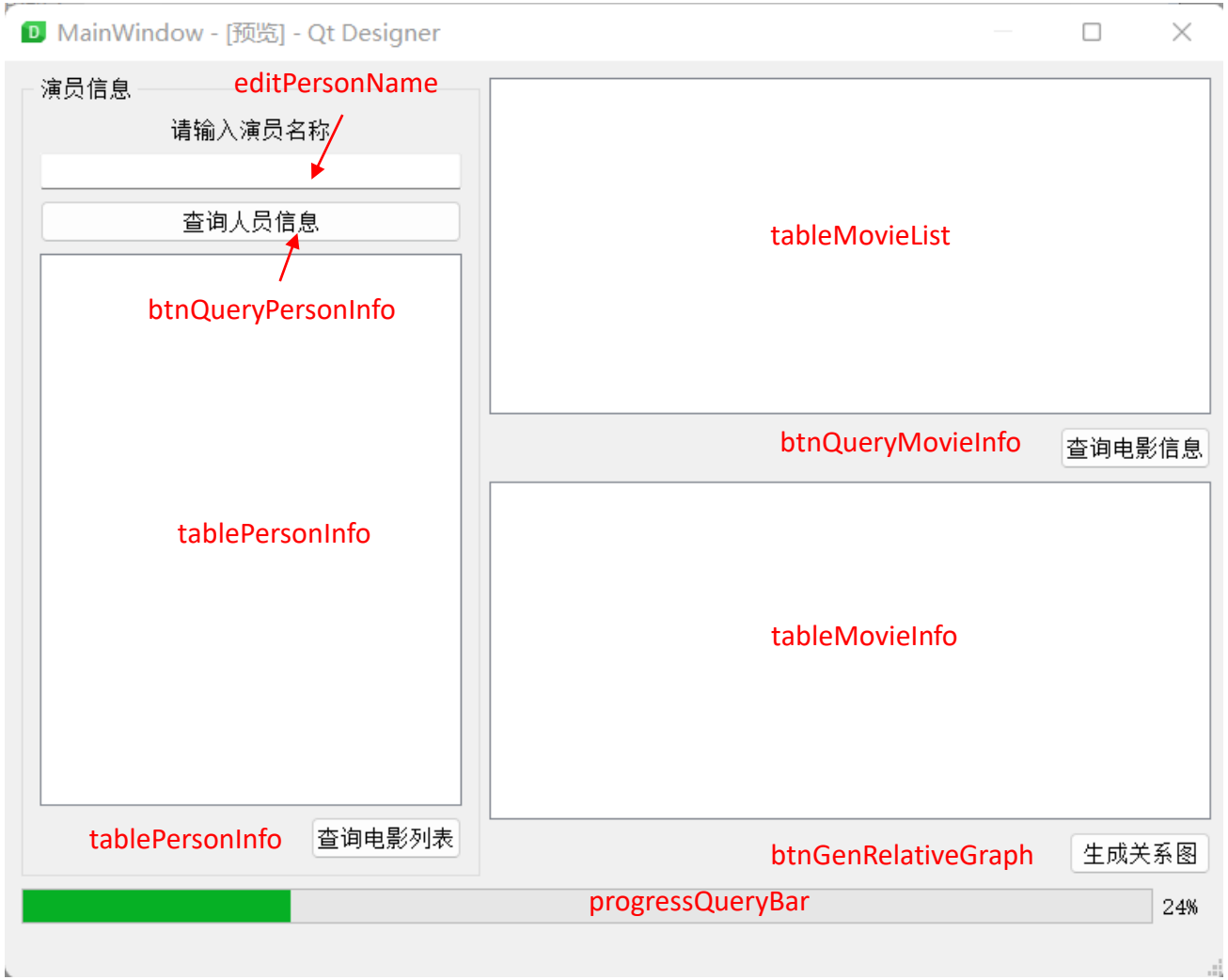
针对窗口产生比较复杂的树形结构时，一般都是通过QTreeView类来实现的，而不是QTreeWidget类，QTreeView和QTreeWidget类最大的区别就是，QTreeView类可以使用操作系统提供的定制模式，比如文件系统盘的树列表，如下所示：

```
if __name__ == '__main__':
    app=QApplication(sys.argv)

    #window系统提供的模式
    model=QDirModel()
    #创建一个QTreeView的控件
    tree=QTreeView()
    #为控件添加模式
    tree.setModel(model)

    tree.setWindowTitle('QTreeView例子')
    tree.resize(640,480)

    tree.show()
    sys.exit(app.exec_())123456789101112131
```



专题大作业（2选1）

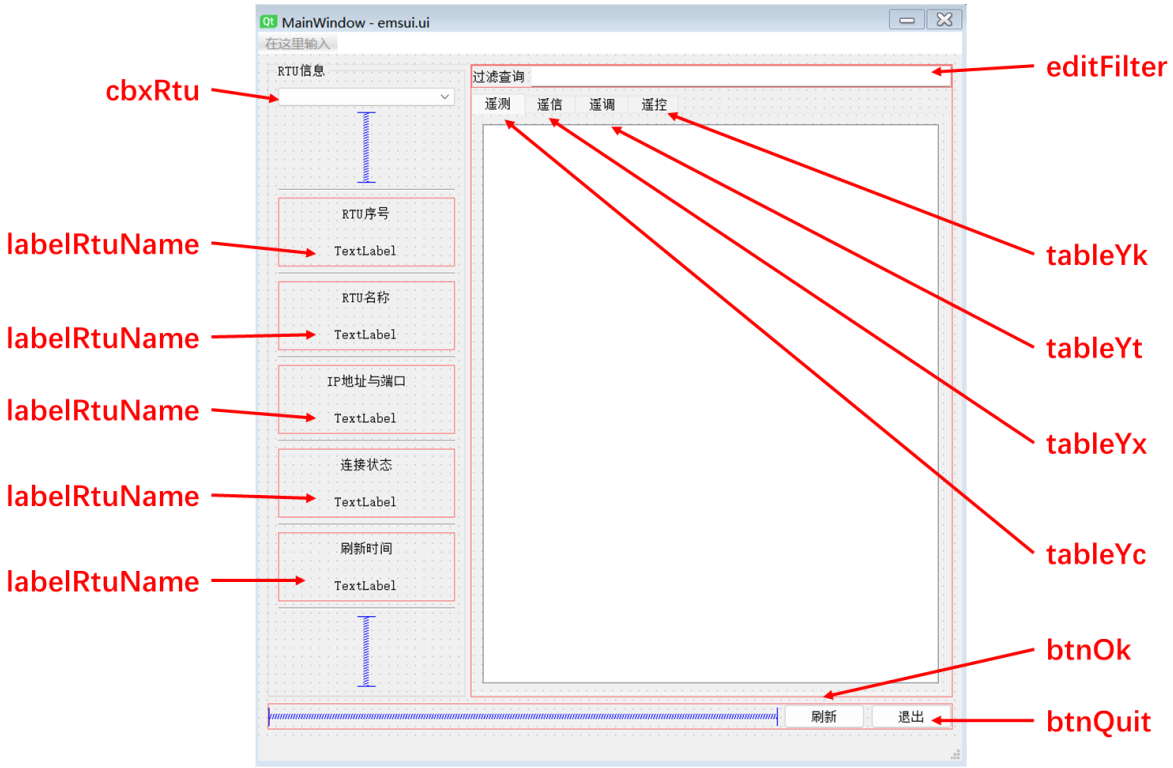
制作爬虫的人机页面：



- 点击**btnQueryPersonInfo**，根据**editPersonName**的输入查询该演员的个人信息，若演员信息不存在，则弹出告警窗口。
- 点击**tablePersonInfo**，根据已经查询的演员ID查询该演员所有的电影列表，结果写入到**tableMovieList**（考虑采用多线程，进度条）。
- 选择**tableMovieList**中某行，点击按钮**btnQueryMovieInfo**，则查询选中电影的相关信息，包括国家、时间、导演、演员等信息，结果写入**tableMovieInfo**。
- 点击**btnGenRelativeGraph**，根据已经查询到的所有电影列表，从网络获取到电影人员信息，并生成关系图（考虑采用多线程，进度条）。

专题大作业（2选1）

完成SCADA功能（前端人机+后台通信）：



- 完成EMS层的SCADA功能：
  - ✓ 点击按钮，重新获取数据库数据，并刷新到页面
  - ✓ 四遥与RTU的状态显示：图标显示。
  - ✓ 四遥与RTU的时间显示格式：”yyyy-MM-dd hh:mm:ss”
  - ✓ 通过表格，修改 遥控和遥调的值，保存到数据库时，同事更新该遥控/遥调的时间。
  - ✓ **修改ems.py，只向rtu 下发 数值或时间有变化的遥控和遥调。**
- 参考EMS层功能，完成RTU层的SCADA功能：
  - ✓ 点击按钮，重新获取数据库数据，并刷新到页面
  - ✓ 四遥与RTU的状态显示：图标显示。
  - ✓ 四遥与RTU的时间显示格式：”yyyy-MM-dd hh:mm:ss”
  - ✓ 通过表格，修改 遥测和遥信的值，保存到数据库时，同事更新该遥测/遥信的时间。
  - ✓ **修改rtu.py，只向ems上送 数值或时间有变化的遥测和遥信。**