# object数据

## object属性、值的设置和获取方法

**def** func1(self):

        obj1 **=** QObject()

        obj1.setProperty('level1', '第一')

        obj1.setProperty('level2', '第二')

        print(obj1.property('level1'))

        print(obj1.dynamicPropertyNames()) #dynamicPropertyNames()方法就是获取setProperty()所设置的属性名称

## **对象之间的父子关系继承和查找**

**def** func(self):

        obj1 **=**QObject()

        print('obj1',obj1)

        obj2 **=**QObject()

        print('obj2', obj2)

        obj3 **=**QObject()

        print('obj3', obj3)

        obj2.setParent(obj1)# 把obj1设置为obj2的父对象

        obj3.setParent(obj2)# 把obj2设置为obj3的父对象

        print(obj2.parent()) # 获取父对象

        print(obj2.children())# 获取obj2的子对象

        print(obj1.findChild(QObject))# 获取直接的子对象

        print(obj1.findChildren(QObject))# 获取所有的子对象

## **对象类型和继承关系的判定方法**

def func(self):

        obj = QObject()

        win = QWidget()

        btn = QPushButton()

        label = QLabel()

        obj\_list = [obj, win, btn, label]

        for o in obj\_list:

            # print(o.isWidgetType()) # 判定对象类型使用

            print(o.inherits('QPushButton')) # 判断'QPushButton'是否继承与另一个对象

## **Object中关于对象删除的方法**

|  |
| --- |
| **def** func(self):      obj1 **=** QObject()      self.obj1 **=** obj1 #相当于将obj1设置为全局变量，即使语句中有删除语句，只要窗口不关闭，那么obj1的值就不会被释放      obj2 **=** QObject()      obj3 **=** QObject()      obj2.setParent(obj1)      obj3.setParent(obj2)      print(obj1)      print(obj2)      print(obj3)        obj1.destroyed.connect(**lambda** :print('obj1被释放'))      obj2.destroyed.connect(**lambda** :print('obj2被释放'))      obj3.destroyed.connect(**lambda** :print('obj3被释放'))        # del obj2 #del无法删除对象，仅切断obj2与堆栈的联系      print(obj2.deleteLater()) #deletelater删除obj2的值，即使obj2是全局变量也会被删除值，而且deletelater是在所有代码执行完成后才会执行      print(obj1.children())        # 案例      label1 **=** QLabel(self)      label1.setText('label1')      label1.move(50,50)      label1.setStyleSheet('background-color:green')        label2 **=** QLabel(self)      label2.setText('label2')      label2.move(100,100)      label2.setStyleSheet('background-color:green')        label3 **=** QLabel(self)      label3.setText('label3')      label3.move(150,150)      label3.setStyleSheet('background-color:green')        # label2.deleteLater()  **del** label2 |

## **Object定时器**

**class** Obj(QObject):

**def** timerEvent(self, QTimerEvent): #添加定时器时，需要添加这个事件

        print(QTimerEvent,2)

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    app **=** QApplication(sys.argv)

    window **=** QWidget()

    obj **=**Obj()

    timer\_id **=** obj.startTimer(1000) #定时器开启，函数内的参数单位：毫秒

    # obj.killTimer(timer\_id) #定时器关闭

    window.show()

    sys.exit(app.exec\_())

# 事件处理

## Qlabel文本

常用的信号（事件）：

* 当鼠标划过Qlabel控件时触发：LinkHovered
* 当鼠标单击Qlabel控件时触发：linkActivated

label支持font的，所以我们可以通过font来设置文本样式

Label.setText(“<font color = red>XXXX</font>”)

# .ui文件转换为.py文件

方法一：

Python -m PyQt5.uic.pyuic xxx.ui -o xxx.py

方法二：

直接使用pyuic5 工具

安装目录\Python37\Lib\site-packages\PyQt5\uic\

Pyuit5 xxx.ui -o xxx.py

1. 错误汇总
2. 浏览器插件报“cannot import name ‘QtwebKitWidgets’ from ‘PyQt5’”

主要原因是新版pyqt5显示浏览器控件与老版不一致导致的

将之前的from PyQt5 import QtwebKitWidgets 修改为：from PyQt5.QtWebEngineWidgets import \*

将布局中的QtwebKitWidgets.QWebView(self.centralwidget) 修改为：QWebEngineView(self.centralwidget)

1. QMainWindow框架下，所有的布局不会有效果，需要添加一下4行才可以正常使用布局

widget = QWidget()

        vbox = QVBoxLayout(self)

        widget.setLayout(vbox)

        self.setCentralWidget(widget)