链滴

PyQt5 快速开发与实战（三）

作者：Ricky2020

原文链接：https://ld246.com/article/1612401756946

来源网站：链滴

PyQt5 高级界面控件

5.1 表格与树

1. 表格与树解决的问题是

如何在一个控件中有规律地呈现更多的数据

5.1.1 QTableView 表格结构视图

1. 一个应用需要和一批数据(比如数组、列表)进行交互，然后以表格的形式输出这些信息，这时就需用到QTableView视图类来使用数据模型显示内容，通过setModel来绑定数据源

2. QTableWidget继承自QTableView，主要的区别在于QTableView可以使用　自定义的数据模型而QTableWidget只能使用标准的数据模型，一般来说标准的模型已经可以满足我们的需求

3. 使用QTableWidget时，其单元格数据是通过QTableWidgetItem对象来设置实现的---不需要自再去建立数据模型

4. **常用的标准数据模型**

- QStringListModel　　存储一组字符串
- QStandardItemModel　　存储任意层次结构的数据模型
- QDirModel　　文件目录模型(对文件系统进行封装)
- QSqlTableModel　　对SQL中的表格进行封装
- QSqlQueryModel　　对SQL中的查询结果进行封装
- QSqlRelationalTableModel　对带有foreign key的SQL表进行封装
- QSortFilterProxyModel　　对模型中的数据进行排序或者过滤

案例5-1 QTableView表格视图类的使用

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

class TableViewDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5-1 QTableView表格视图类的使用")
        self.resize(500, 300)
        # 建立要绑定的数据模型
        self.model = QStandardItemModel(4, 4)
        self.model.setHorizontalHeaderLabels(['标题1', '标题2', '标题3', '标题4'])


        for row in range(4):
            for column in range(4):
                item = QStandardItem("row {}, column {}".format(row, column))
                # 设置模型单元格中的条目数据
                self.model.setItem(row, column, item)
         self.model.appendRow([
        self.model.appendRow([
            QStandardItem("row {}, column {}".format(11, 11)),
            QStandardItem("row {}, column {}".format(11, 11)),
            QStandardItem("row {}, column {}".format(11, 11)),
            QStandardItem("row {}, column {}".format(11, 11))
      ])
        self.tableView = QTableView()
        # 绑定数据模型到表格视图
        self.tableView.setModel(self.model)
        self.tableView.horizontalHeader().setStretchLastSection(True)
        self.tableView.verticalHeader().setSectionResizeMode(QHeaderView.Stretch)

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

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = TableViewDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 此例中表格填满了窗口，水平方向上到了显示的末尾部分，垂直方向上　占满

```
# 设置tableView的水平头伸展到最后内容的最后一部分
self.tableView.horizontalHeader().setStretchLastSection(True)
# 设置tableView的垂直头部域重构大小模式为，头部视图尽可能伸展
self.tableView.verticalHeader().setSectionResizeMode(QHeaderView.Stretch)
```

2. 追加数据

```
self.model.appendRow([
    QStandardItem("row {}, column {}".format(11, 11)),
    QStandardItem("row {}, column {}".format(11, 11)),
    QStandardItem("row {}, column {}".format(11, 11)),
    QStandardItem("row {}, column {}".format(11, 11))
])
```

3. 删除当前选中的数据

```
# 第一种方法
indexs = self.tableView.selectionModel().selection().indexs()
if len(indexs) > 0:
    self.model.removeRows(indexs[0].row(), 1)
# 第二种方法
index = self.tableView.currentIndex()
# 索引对应的那一行数据
print(index.row())
self.model.removeRow(index.row())
```

注意

1. 如果在表格中什么也不选默认就删除第一行，也就是索引为0的那一行数据
2. 选中一行时就删除那一行
3. 选中多行时，就删除焦点所在的那一行数据

5.1.2 QListView 列表视图

1. QListView列表视图类用于展示数据，子类是QListWidget
2. QListView是基于模型的(Model)的，需要程序来建立模型，然后再保存数据
3. QListWidget是对QListView优化升级，本身已经存在一个封装好的数据存储模型(QListWidgetIte)，可以直接调用addItem函数添加条目

常用的方法

4. setModel()　　用来设置View所关联的Model，可以使用Python原生的list作为数据源模型mo el
5. selectedItem()　　获取已选中Model中的条目
6. isSelected()　　判断Model中的某个条目是否被选中

常用的信号

7. clicked　　当点击Model中的某个条目Item时，发射该信号
8. doubleClicked　　当双击Model中的某项时，发射该信号

案例5-2 QListView列表视图的使用

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

class ListViewDemo(QWidget):
    def __init__(self):
```

```
        super().__init__()
        self.setWindowTitle("案例5-2 QListView列表视图的使用")

        self.resize(300, 270)
        layout =QVBoxLayout()
        listView = QListView()
        # 建立数据模型  空对象
        stringListModel = QStringListModel()
        self.strList = ['Item1', 'Item2', 'Item3', 'Item4']
        # 填充空对象数据模型
        stringListModel.setStringList(self.strList)
        # 列表视图绑定数据源模型
        listView.setModel(stringListModel)
        listView.clicked.connect(self.listViewItemClicked)

        layout.addWidget(listView)
        self.setLayout(layout)

    def listViewItemClicked(self, ListModelIndex):
        print(ListModelIndex)
        print(ListModelIndex.row())
        QMessageBox.information(self, "ListWidget", self.tr("你选择了：" + self.strList[ListModelI
dex.row()]))


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = ListViewDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 当用户点击QListView控件的Model中的某一个条目时会弹出消息对话框，显示选择的是哪一个条目

```
def listViewItemClicked(self, ListModelIndex):
    print(ListModelIndex)
    print(ListModelIndex.row())
    QMessageBox.information(self, "ListWidget", self.tr("你选择了：" + self.strList[ListModelInd
x.row()]))
```

注意

1. ListModelIndex是PyQt5.QtCore.QModelIndex类的实例化对象
2. ListModelIndex.row()这个才是真正的列表中的条目对应的索引

5.1.3 QListWidget 列表视图窗口控件

1. QListWidget是一个集成好数据模型的接口，可以直接绑定数据源
2. QListWidget可以设置为多重选择
3. QListWidget中的条目Item都是QListWidgetItem对象

常用的方法

4. addItem()　在列表中添加QListWidgetItem对象或者字符串

5. addItems()　在列表中添加多个条目，参数是可迭代对象

6. insertItem()　在指定的索引处插入条目

7. clear()　删除列表中的所有内容

8. setCurrentItem()设置当前所选中的条目

9. sortItems()　按升序重新排列列表的所有条目

常用的信号

10. itemClicked　当点击列表中的某项条目时，发射该信号

11. currentItemCahnged 当当前选中条目发生变化时，发射该信号

案例5-3 QListWidget列表控件的使用

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

class ListWidgetDemo(QListWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5-3 QListWidget列表控件的使用")
        self.resize(300, 120)
        self.addItem("Item1")
        self.addItem("Item2")
        self.addItem("Item3")
        self.addItem("Item4")

        self.itemClicked.connect(self.clicked)
    def clicked(self, item):
        QMessageBox.information(self, "ListWidget", self.tr("你选择了：" + item.text()))

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = ListWidgetDemo()
    win.show()
    sys.exit(app.exec_())
```

5.1.4 QTableWidget 表格控件窗口

1. QTableWidget是QTableView的子类，它自带有标准的数据模型，不需要再去建立，直接使用QTaleWidgetItem对象来表示表格中的每一个单元格条目Item

常用的方法

2. setRowCount(int row)　设置QTableWidget表格控件的行数

3. setColumnCount(int col)　设置QTableWidget的列数

4. setHorizontalHeaderLabels()　设置QTableWidget的水平头部标题标签

5. setVerticalHeaderLabels()　设置QTableWidget的垂直标题标签

6. setItem(int,int,QTableWidgetItem) 设置每一个单元格的Item

7. horizontalHeader()　　获取QTableWidget的水平表头

8. rowCount()　　获取QTableWidget行数

9. columnCount()　　获取QTableWidget列数

10. setEditTriggers(EditTriggers triggers) 设置表格是否可编辑，并设置编辑规则，枚举值如下

- QAbstractItemView.NoEditTriggers0No　　0

不能对表格内容进行修改操作

- QAbstractItemView.CurrentChanged1Editing　1

任何时候都能对单元格进行编辑修改

- QAbstractItemView.DoubleClicked2Editing　2

可以双击单元格修改

- QAbstractItemView.SelectedClicked4Editing　4

单击已选中的内容

- QAbstractItemView.EditKeyPressed8Editing　8

当修改键按下时对单元格进行修改

- QAbstractItemView.AnyKeyPressed16Editing　16

任意键被按下时对单元格进行修改

- QAbstractItemView.AllEditTriggers31Editing　31

包括以上所有的条件

11. setSelectionBehavior()　设置表格的选择行为

- QAbstractItemView.SelectItem0Selecting　0 选中单项
- QAbstractItemView.SelectRows1Selecting　1 选中一行
- QAbstractItemView.SelectColumns2Selecting 2 选中一列

12. setTextAlignment()　　设置单元格内的文本对齐方式

- Qt.AlignLeft　　左对齐
- Qt.AlignRight　　右对齐
- Qt.AlignHCenter　水平居中对齐
- Qt.AlignJustify　从左到右对齐
- Qt.AlignTop　　顶部对齐
- Qt.AlignBottom　底部对齐
- Qt.AlignVCenter　垂直居中对齐
- Qt.AlignBaseline　与基线对齐

13. setSpan(int row,int column,int rowSpanCount, int columnSpanCount)　　设置单格的合并形式

14. setShowDrid()　　设置网格是否显示，默认是True显示

15. setColumnWidth()　　设置单元格的宽度

16. setRowHeight()　　设置单元格的高度

实操案例

17. 基本用法

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

class tableWidgetDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle("QTableWidget 表格控件窗口的基本用法")
        self.resize(400, 300)
        layout = QHBoxLayout()
        tableWidget = QTableWidget()
        tableWidget.setRowCount(4)
        tableWidget.setColumnCount(3)
        tableWidget.setHorizontalHeaderLabels(['姓名', '性别', '体重(Kg)'])

        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)
        layout.addWidget(tableWidget)
        self.setLayout(layout)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = tableWidgetDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. QTableWidget与QTableView的实现等价代码

```
# QTableWidget
tableWidget = QTableWidget(4,3)


# QTableView
self.model = QStandardItemModel(4, 3)
self.tableView = QTableView()
self.tableView.setModel(self.model)
```

2. QTableWidget本身就就有数据模型，条目对象就是QTableWidgetItem，而QTableView需要程本身创建数据源模型，条目对象是QStandardItem，并且需要使用setModel函数来绑定数据模型

3. 默认时，表格中的字符串是可以修改的

4. 表头标签的设置必须要在整个表格初始化完成后，不然是没有效果的，即就是要先初始化行号和列(row/column)

5. 通过使用QTableWidget对象的horizontalHeader设置表格为自适应的伸缩模式，即可以根据窗口小来改变网格的大小

```
tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
tableWidget.verticalHeader().setSectionResizeMode(QHeaderView.Stretch)
```

6. 设置单元格的文本对齐方式

```
newItem.setTextAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
```

7. 表格头的隐藏与显示

```
tableWidget.horizontalHeader().setVisible(False)
```

8. 在单元格中放置控件

```
# 在单元格中放置控件
# 放置一个下拉列表框
comBox = QComboBox()
comBox.addItem("男")
comBox.addItem("女")
# 设置样式表  距离表格单元格的外边距为3px
comBox.setStyleSheet("QComBox{margin:3px};")
tableWidget.setCellWidget(1, 1, comBox)
# 放置一个按钮
serchBtn = QPushButton("修改")
serchBtn.setDown(True)
serchBtn.setStyleSheet("QPushButton{margin:3px};")
# 在表格中添加小部件
tableWidget.setCellWidget(1, 2, serchBtn)
```

9. 在表格中快速定位到指定行

_应用场景_当tableWidget表格的行数很多时，可以通过输入行号进行直接定位并显示，比如输入10就直接显示第10行

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

class TableWidgetDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("在表格中快速定位到指定行")
        self.resize(600, 800)
        layout = QHBoxLayout()
        tableWidget = QTableWidget(30, 4)
        layout.addWidget(tableWidget)
        for i in range(30):
```

```
            for j in range(4):
                itemContent = '(%d,%d)' % (i,j)
                tableWidget.setItem(i, j, QTableWidgetItem(itemContent))
        self.setLayout(layout)

        # 遍历表格查找对应选项
        text = "(10,1)"
        items = tableWidget.findItems(text, Qt.MatchExactly)
        item = items[0]
        # 选中查找到的单元格  选中的颜色是颜色，所以下面设置的单元格的背景颜色就看不出是红色
了
        item.setSelected(True)
        # 设置单元格的背景颜色为红色
        item.setForeground(QBrush(QColor(255, 0, 0)))

        # 获取到选中的那一行的索引
        row = item.row()
        print(item)
        print("row=", row)
        tableWidget.verticalScrollBar().setSliderPosition(row)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = TableWidgetDemo()
    win.show()
    sys.exit(app.exec_())
```

10. 设置单元格的文本颜色

```
item.setForeground(QBrush(QColor(255, 0, 0)))
```

11. 将单元格文本字体加粗

```
item.setFont(QFont("Times", 12, QFont.Black))
```

12. 设置单元格的排序方式

```
# 升序
tableWidget.sortItems(2, Qt.AscendingOrder)
# 降序
tableWidget.sortItems(2, Qt.DescendingOrder)
```

13. 设置单元格的对齐方式

```
# 设置单元格文本的对齐方式  ---此处设置为居中
item.setTextAlignment(Qt.AlignVCenter | Qt.AlignHCenter)
```

14. 合并单元格效果的实现

```
# 合并单元格---设置第一行第一列占据三行一列
tableWidget.setSpan(0, 0, 3, 1)
```

15. 设置单元格的大小

```
# 设置单元格的大小---设置第一行高为120，列宽为150
tableWidget.setColumnWidth(0, 120)
tableWidget.setRowHeight(0, 150)
```

16. 在表格中不显示分割线

```
# 在表格中设置不显示分割线
tableWidget.setShowGrid(False)
```

17. 只显示水平表头，不显示垂直表头

```
# 在表格中只显示水平表头，不显示垂直表头
tableWidget.verticalHeader().setVisible(False)
```

18. 为单元格添加图片

```
# 为单元格添加图片
newitem = QTableWidgetItem(QIcon("./images/open.png"), "背包")
tableWidget.setItem(0, 3, newitem)
```

19. 改变单元格中显示图片的大小

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

class TableWidgetDemo3(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("改变单元格中显示图片的大小")
        self.resize(600, 800)
        layout = QHBoxLayout()
        tableWidget = QTableWidget()
        tableWidget.setColumnCount(3)
        tableWidget.setRowCount(5)

        tableWidget.setHorizontalHeaderLabels(['图片1', '图片2', '图片3'])
        # 设置表格不可编辑
        tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)
        # 设置图标的大小
        tableWidget.setIconSize(QSize(300, 200))
        # 设置单元格的大小与图片的大小相同
        for i in range(3):
            tableWidget.setColumnWidth(i, 300)
        for i in range(5):
            tableWidget.setRowHeight(i, 200)
        # 这里使用一个简单的算法来遍历表格，除此之外当然可以使用嵌套循环
        for k in range(15):  # 模拟产生15个数据
            i = k/3
            j = k%3
            item = QTableWidgetItem()
            # 用户点击表格时，图片被选中
            item.setFlags(Qt.ItemIsEnabled)
```

```
            icon = QIcon(r"./images/bao%d.png" % k)
            item.setIcon(icon)

            print('e/icons/%d.png i=%d j=%d' % (k, i, j))
            tableWidget.setItem(i, j, item)
        tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        tableWidget.verticalHeader().setSectionResizeMode(QHeaderView.Stretch)
        layout.addWidget(tableWidget)
        self.setLayout(layout)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = TableWidgetDemo3()
    win.show()
    sys.exit(app.exec_())
```

20. 支持右键菜单

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


class tableWidgetDemo4(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("支持右键菜单")
        self.resize(500, 300)
        layout = QHBoxLayout()
        self.tableWidget = QTableWidget(5, 3)
        layout.addWidget(self.tableWidget)

        self.tableWidget.setHorizontalHeaderLabels(['姓名', '性别', '体重'])
        self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

        newItem = QTableWidgetItem("张三")
        self.tableWidget.setItem(0, 0, newItem)

        newItem = QTableWidgetItem("男")
        self.tableWidget.setItem(0, 1, newItem)

        newItem = QTableWidgetItem("160")
        self.tableWidget.setItem(0, 2, newItem)

        newItem = QTableWidgetItem("李四")
        self.tableWidget.setItem(1, 0, newItem)

        newItem = QTableWidgetItem("女")
        self.tableWidget.setItem(1, 1, newItem)

        newItem = QTableWidgetItem("170")
        self.tableWidget.setItem(1, 2, newItem)
```

```
        # 设置允许右键产生对应的菜单
        self.tableWidget.setContextMenuPolicy(Qt.CustomContextMenu)
        self.tableWidget.customContextMenuRequested.connect(self.generateMenu)
        self.setLayout(layout)
    def generateMenu(self, position):
        print(position)
        print("我是生成的菜单")
        row_num = -1
        indexes = self.tableWidget.selectionModel().selection().indexes()
        print(indexes)
        # 这里表示下面的显示行索引为在tableWidget控件上所有选择的最后选中项
        for i in indexes:
            row_num = i.row()
            print(row_num)
        if row_num < 2:
            menu = QMenu(self)
            item1 = menu.addAction("选项一")
            item2 = menu.addAction("选项二")
            item3 = menu.addAction("选项三")
            # 这里是进入了生成菜单的消息循环中，action一直在等待接收用户的右键菜单选择条目
            # 使用坐标系统的映射算法，将菜单显示在右键的大致位置mapToGlobal()
            # 应用程序的右键点击坐标映射到全局屏幕坐标
            action = menu.exec_(self.tableWidget.mapToGlobal(position))
            print(action)

            if action == item1:
                print(action)
                print(item1)
                print(row_num)
                print("你选择了选项一，当前行的文字内容是：",
                self.tableWidget.item(row_num, 0).text(),
                self.tableWidget.item(row_num, 1).text(),
                self.tableWidget.item(row_num, 2).text())
            elif action == item2:
                print(row_num)
                print("你选择了选项二，当前行的文字内容是：",
                self.tableWidget.item(row_num, 0).text(),
                self.tableWidget.item(row_num, 1).text(),
                self.tableWidget.item(row_num, 2).text())
            elif action == item3:
                print(row_num)
                print("你选择了选项三，当前行的文字内容是：",
                self.tableWidget.item(row_num, 0).text(),
                self.tableWidget.item(row_num, 1).text(),
                self.tableWidget.item(row_num, 2).text())
            else:
                return


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = tableWidgetDemo4()
    win.show()
```

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

解析

1. 选中某个单元行，单击右键，弹出的菜单显示位置是右键点击的坐标的全局映射

```
# 应用程序的右键点击坐标映射到全局屏幕坐标
action = menu.exec_(self.tableWidget.mapToGlobal(position))
```

2. 当选择弹出菜单的条目时，会根据用户选择来显示相关行内容，行索引的选取是根据用户鼠标选择的最后一个选中项

```
for i in indexes:
    row_num = i.row()

if action == item1:
    print(action)
    print(item1)
    print(row_num)
    print("你选择了选项一，当前行的文字内容是：",
    self.tableWidget.item(row_num, 0).text(),
    self.tableWidget.item(row_num, 1).text(),
    self.tableWidget.item(row_num, 2).text())
```

5.1.5 QTreeView 树形视图

1. QTreeWideget类是原生QTreeView的派生类，这个类实现了树形显示，并自带有绑定的数据模型不必像原生类要自定义数据模型

2. **常用的方法**

- setColumnWidth(int column,int width) 设置指定列的宽度
- insertTopLevelItems() 在视图的顶层插入项目列表
- expandAll() 展开所有的树形节点
- invisibleRootItem() 返回树形控件中不可见的根选项
- selectedItems() 返回所有选中的非隐藏项目列表

3. QTreeWideget的数据模型QTreeWidegetItem中的常用方法

- addChild() 将子项追加到子列表中
- setText() 设置显示的节点文本
- Text() 返回显示的节点文本
- setCheckState(int column,int state)设置指定列的选中状态
  - Qt.Checked 节点选中
  - Qt.Unchecked 节点未选中
- setIcon(int column,QIcon icon) 在指定列中显示图标

4. 树形结构的实现

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

class  treeWidget(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("treeWidget树形结构的实现")
        layout = QHBoxLayout()
        # 实例化一个树形控件
        self.tree = QTreeWidget()
        # 设置列数
        self.tree.setColumnCount(2)
        # 设置树形控件的头部标题
        self.tree.setHeaderLabels(['Key', 'Value'])
        # 设置根节点
        root = QTreeWidgetItem()
        root.setText(0, 'root')
        root.setIcon(0, QIcon("./images/root.png"))
        # 设置树形控件的列宽度
        self.tree.setColumnWidth(0, 160)

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

        # 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()
        child3.setText(0, 'child3')
        child3.setText(1, 'music')
        child3.setIcon(0, QIcon("./images/music.png"))
        child2.addChild(child3)

        # 将创建的所有条目列表添加到树形控件中
        #  self.tree.addTopLevelItem(root)
        self.tree.insertTopLevelItem(0, root)

        # 节点全部展开
        self.tree.expandAll()
        layout.addWidget(self.tree)
        self.setLayout(layout)

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

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

解析

- 可以使用继承来明确层级关系

```
child2 = QTreeWidgetItem(root)
```

- 也可以使用QTreeWidgetItem类的方法

```
child3 = QTreeWidgetItem()
child2.addChild(child3)
```

- 可以使用insertTopLevelItems方法添加节点

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

class  treeWidget(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("treeWidget树形结构的实现")
        layout = QHBoxLayout()
        # 实例化一个树形控件
        self.tree = QTreeWidget()
        # 设置列数
        self.tree.setColumnCount(2)
        # 设置树形控件的头部标题
        self.tree.setHeaderLabels(['Key', 'Value'])
        # 设置根节点
        root = QTreeWidgetItem()
        root.setText(0, 'root')
        root.setIcon(0, QIcon("./images/root.png"))
        # 设置树形控件的列宽度
        self.tree.setColumnWidth(0, 160)

        rootList = []
        rootList.append(root)

        # 设置子节点1
        child1 = QTreeWidgetItem()
        child1.setText(0, 'child1')
        child1.setText(1, 'ios')
        child1.setIcon(0, QIcon("./images/IOS.png"))
        root.addChild(child1)
        self.tree.insertTopLevelItems(0, rootList)
        # 节点全部展开
        self.tree.expandAll()
        layout.addWidget(self.tree)
        self.setLayout(layout)
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = treeWidget()
    win.show()
    sys.exit(app.exec_())
```

解析

- 设置节点的状态

```
 child1.setCheckState(0, Qt.Checked)
```

- 设置节点指定列的背景颜色

```
brush_red = QBrush(Qt.red)
brush_green = QBrush(Qt.green)
root.setBackground(0, brush_red)
root.setBackground(1, brush_green)
```

5. 给节点添加响应事件

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

class  treeWidget(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("treeWidget树形结构的实现")

        # 实例化一个树形控件
        self.tree = QTreeWidget()
        # 设置列数
        self.tree.setColumnCount(2)
        # 设置树形控件的列宽度
        self.tree.setColumnWidth(0, 160)
        # 设置树形控件的头部标题
        self.tree.setHeaderLabels(['Key', 'Value'])
        # 设置根节点
        root = QTreeWidgetItem()
        root.setText(0, 'root')
        root.setText(1, '0')
        root.setIcon(0, QIcon("./images/root.png"))

        child1 = QTreeWidgetItem(root)
        child1.setText(0, "child1")
        child1.setText(1, "1")

        child2 = QTreeWidgetItem(root)
        child2.setText(0, "child2")
        child2.setText(1, "2")

        child3 = QTreeWidgetItem(root)
```

```
        child3.setText(0, "child3")
        child3.setText(1, "3")

        child4 = QTreeWidgetItem(child3)
        child4.setText(0, "child4")
        child4.setText(1, "4")

        child5 = QTreeWidgetItem(child3)
        child5.setText(0, "child5")
        child5.setText(1, "5")

        self.tree.addTopLevelItem(root)
        self.tree.clicked.connect(self.onTreeClicked)
        # 节点全部展开
        self.tree.expandAll()
        self.setCentralWidget(self.tree)

    def onTreeClicked(self, modelIndex):
        item = self.tree.currentItem()
        print("key=%s, value=%s" % (item.text(0), item.text(1)))
if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = treeWidget()
    win.show()
    sys.exit(app.exec_())
```

6. 系统定制模式

概念QTreeView原生树形控件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形列表(QTreeWidget不可以使用自定义的数据模型，而QTreeView可以绑定基于元数据的所有数据模型)

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

if __name__ == '__main__':
    app = QApplication(sys.argv)
    # Window系统的定制模式
    model = QDirModel()
    # 创建一个QTreeView视图控件
    tree = QTreeView()
    tree.setWindowTitle("QTreeView绑定系统的目录列表数据模型")
    tree.resize(640, 480)
    tree.setColumnWidth(0, 300)
    # 为控件绑定数据模型
    tree.setModel(model)

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

5.2 容器：装载更多的控件

应用场景

5.2.1 QTabWidget

概念QTabWidget控件提供了一个选项卡和一个页面区域，默认显示第一个选项卡的页面，通过切换项卡可以查看对应的页面

QTabWidget常用的方法

1. addTab()　　将一个控件添加到Tab控件的选项卡中
2. insertTab()　　插入选项卡到Tab控件的指定位置
3. removeTab()　　根据指定的索引删除Tab控件的选项卡
4. setCurrentIndex()　设置当前可见的选项卡的索引
5. setCurrentWidget()　设置当前可见的页面
6. setTabBar()　　设置选项卡栏的小控件
7. setTabPosition()　设置选项卡的位置
   - QTabWidget.North　显示在页面的上方
   - QTabWidget.South　显示在页面的下方
   - QTabWidget.West　显示在页面的左侧
   - QTabWidget.East　显示在页面的右侧
8. setTabText()　　设置Tab选项卡的显示文本

QTabWidget常用的信号

9. currentCahnged　　切换当前选项卡页面时发射该信号

案例5-4 QTabWidget选项卡的使用

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

class TabWidgetDemo(QTabWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5-4 QTabWidget选项卡的使用")
        self.tab1 = QWidget()
        self.tab2 = QWidget()
        self.tab3 = QWidget()
        self.addTab(self.tab1, "Tab 1")
        self.addTab(self.tab2, "Tab 2")
        self.addTab(self.tab3, "Tab 3")

        self.tab1UI()
        self.tab2UI()
        self.tab3UI()

    def tab1UI(self):
        layout = QFormLayout()
```

```
        layout.addRow("姓名", QLineEdit())
        layout.addRow("地址", QLineEdit())
        self.setTabText(0, "联系方式")
        self.tab1.setLayout(layout)

    def tab2UI(self):
        layout = QFormLayout()
        sex = QHBoxLayout()
        sex.addWidget(QRadioButton("男"))
        sex.addWidget(QRadioButton("女"))
        layout.addRow(QLabel("性别"), sex)
        layout.addRow("生日", QLineEdit())
        self.setTabText(1, "个人详细信息")
        self.tab2.setLayout(layout)

    def tab3UI(self):
        layout = QHBoxLayout()
        layout.addWidget(QLabel("科目"))
        layout.addWidget(QCheckBox("物理"))
        layout.addWidget(QCheckBox("高数"))
        self.setTabText(2, "教育程度")
        self.tab3.setLayout(layout)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = TabWidgetDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 在QTabWidget控件中添加了三个选项卡

```
self.tab1 = QWidget()
self.tab2 = QWidget()
self.tab3 = QWidget()
self.addTab(self.tab1, "Tab 1")
self.addTab(self.tab2, "Tab 2")
self.addTab(self.tab3, "Tab 3")
```

5.2.2 QStackedWidget堆栈控件的使用

概念QStackedWidget是一个堆栈窗口，可以填充一些小控件，使用的是QStackedLayout布局

案例5-5 QStackedWidget堆栈控件的使用

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

class stackedWidgetDemo(QWidget):
    def __init__(self):
        super().__init__()
```

```
        self.setWindowTitle("案例5-5 QStackedWidget堆栈控件的使用")
        self.setGeometry(300, 50, 10, 10)

        self.leftList = QListWidget()
        firstItem = QListWidgetItem("联系方式")
        self.leftList.insertItem(0, firstItem)
        self.leftList.insertItem(1, "个人信息")
        self.leftList.insertItem(2, "教育程度")
        self.leftList.setCurrentItem(firstItem)

        self.stack1 = QWidget()
        self.stack2 = QWidget()
        self.stack3 = QWidget()

        self.stack1UI()
        self.stack2UI()
        self.stack3UI()

        self.stackedWidget = QStackedWidget(self)
        self.stackedWidget.addWidget(self.stack1)
        self.stackedWidget.addWidget(self.stack2)
        self.stackedWidget.addWidget(self.stack3)
        layout = QHBoxLayout()
        layout.addWidget(self.leftList)
        layout.addWidget(self.stackedWidget)

        self.setLayout(layout)
        self.leftList.currentRowChanged.connect(self.dispaly)

    def stack1UI(self):
        layout = QFormLayout()
        layout.addRow("姓名", QLineEdit())
        layout.addRow("地址", QLineEdit())
        self.stack1.setLayout(layout)

    def stack2UI(self):
        layout = QFormLayout()
        sex = QHBoxLayout()
        sex.addWidget(QRadioButton("男"))
        sex.addWidget(QRadioButton("女"))
        layout.addRow(QLabel("性别"), sex)
        layout.addRow("生日", QLineEdit())
        self.stack2.setLayout(layout)

    def stack3UI(self):
        layout = QHBoxLayout()
        layout.addWidget(QLabel("科目"))
        layout.addWidget(QCheckBox("物理"))
        layout.addWidget(QCheckBox("高数"))
        self.stack3.setLayout(layout)

    def dispaly(self, i):
        self.stackedWidget.setCurrentIndex(i)
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = stackedWidgetDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 使用了QListWidget的currentRowChanged信号来绑定用户的点击行为和右侧的stackedWidget视图

```
self.leftList.currentRowChanged.connect(self.dispaly)
```

5.2.3 QDockWidget

概念QDockWidget是一个可以停靠在_QMainWindow_内的窗口控件，它可以保持浮动状态，也可在指定的位置上作为子窗口附加到主窗口中

1. QMainWindow类的主窗口对象保留一个用户停靠的区域，就在控件的中央区域
2. QDockWidget停靠控件在主窗口内可以移到新的区域

常用的方法

3. setWidget()　　在Dock窗口区域设置QWidget
4. setFloating()　　设置Dock窗口可以浮动，True为可以浮动
5. setAllowedAreas() 设置窗口可以停靠的区域
    - LeftDockWidgetArea　左边停靠区域
    - RightDockWidgetArea 右边停靠区域
    - TopDockWidgetArea　顶部停靠区域
    - BottomDockWidgetArea底部停靠区域
    - NoDockWidgetArea　没有可以停靠的区域(不显示Widget)
6. setFeatures()　　设置停靠窗口的功能属性
    - DockWidgetClosable　可关闭
    - DockWidgetMovable　可移动
    - DockWidgetFloatable 可漂浮
    - DockWidgetVerticalTitleBar　在左边显示垂直的标题栏
    - AllDockWidgetFeatures具有前三种属性的所有功能
    - NoDockWidgetFeatures 不具备前三者的功能

案例5-6 QDockWidget停靠控件的使用

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

```
class dockWidgetDemo(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5-6 QDockWidget停靠控件的使用")

        bar = self.menuBar()
        file = bar.addMenu("File")
        file.addAction("New")
        file.addAction("Save")
        file.addAction("quit")
        self.dockWidget =QDockWidget("Dockable")
        self.listWidget = QListWidget()
        self.listWidget.addItem("item1")
        self.listWidget.addItem("item2")
        self.listWidget.addItem("item3")
        self.dockWidget.setWidget(self.listWidget)
        self.dockWidget.setFloating(False)
        self.setCentralWidget(QTextEdit())
        self.addDockWidget(Qt.RightDockWidgetArea, self.dockWidget)



if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = dockWidgetDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 在顶层窗口中添加一个中央小控件

```
self.setCentralWidget(QTextEdit())
```

2. 在停靠窗口内添加QListWidget对象

```
self.listWidget = QListWidget()
self.listWidget.addItem("item1")
self.listWidget.addItem("item2")
self.listWidget.addItem("item3")
self.dockWidget.setWidget(self.listWidget)
```

3. 在主窗口中添加停靠窗口控件并指定默认的停靠区域

```
self.setCentralWidget(QTextEdit())
self.addDockWidget(Qt.RightDockWidgetArea, self.dockWidget)
```

5.2.4 多文档界面QMdiArea

案例5-7 多文档界面QMdiArea控件的使用

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

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

class mdiAreaDemo(QMainWindow):
    count = 0
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5-7 多文档界面QMdiArea控件的使用")
        self.mdi = QMdiArea()
        self.setCentralWidget(self.mdi)

        bar = self.menuBar()
        file = bar.addMenu("File")
        file.addAction("New")
        file.addAction("cascade")
        file.addAction("Tiled")
        file.triggered[QAction].connect(self.windowAction)

    def windowAction(self, action):
        print("Triggered")

        if action.text() == "New":
            mdiAreaDemo.count = mdiAreaDemo.count + 1
            sub = QMdiSubWindow()
            sub.setWidget(QTextEdit())
            sub.setWindowTitle("subWindow" + str(mdiAreaDemo.count))
            self.mdi.addSubWindow(sub)
            sub.show()
        # 使子窗口的排布为级联模式
        elif action.text() == "cascade":
            self.mdi.cascadeSubWindows()
        # 使子窗口的排布为平铺模式
        elif action.text() == "Tiled":
            self.mdi.tileSubWindows()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = mdiAreaDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 主窗口拥有一个菜单和一个QMdiArea控件

```
self.mdi = QMdiArea()
self.setCentralWidget(self.mdi)

bar = self.menuBar()
file = bar.addMenu("File")
file.addAction("New")
file.addAction("cascade")
file.addAction("Tiled")
```

2. 设置了菜单栏的action的触发事件

```
 file.triggered[QAction].connect(self.windowAction)
```

3. 在触发事件中设置了mdi多文档界面的子窗口，并设置了排布模式

```
if action.text() == "New":
    mdiAreaDemo.count = mdiAreaDemo.count + 1
    sub = QMdiSubWindow()
    sub.setWidget(QTextEdit())
    sub.setWindowTitle("subWindow" + str(mdiAreaDemo.count))
    self.mdi.addSubWindow(sub)
    sub.show()
# 使子窗口的排布为级联模式
elif action.text() == "cascade":
    self.mdi.cascadeSubWindows()
# 使子窗口的排布为平铺模式
elif action.text() == "Tiled":
    self.mdi.tileSubWindows()
```

5.2.5 QScrollBar

概念通过提供水平垂直滚动条来扩大当前窗口的有效装载面积

常用的信号

1. valueCahnged　　当滚动条的值发生改变时发射此信号
2. sliderMoved　　当滑块被用户拖动时发射此信号

案例5-8 QScrollBar滚动条的使用

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

class scrollBarDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5-8 QScrollBar滚动条的使用")
        layout = QHBoxLayout()
        self.l1 = QLabel("拖动滑块改变颜色")
        self.l1.setFont(QFont("Arial", 16))
        layout.addWidget(self.l1)
        self.s1 = QScrollBar()
        self.s1.setMaximum(255)
        self.s1.sliderMoved.connect(self.sliderval)
        self.s2 = QScrollBar()
        self.s2.setMaximum(255)
        self.s2.sliderMoved.connect(self.sliderval)
        self.s3 = QScrollBar()
        self.s3.setMaximum(255)
        self.s3.sliderMoved.connect(self.sliderval)
        layout.addWidget(self.s1)
```

```
        layout.addWidget(self.s2)
        layout.addWidget(self.s3)

        self.setGeometry(300, 300, 300, 300)
        self.setLayout(layout)

    def sliderval(self):
        print(self.s1.value(), self.s2.value(), self.s3.value())
        palette = QPalette()
        color = QColor(self.s1.value(), self.s2.value(), self.s3.value())
        palette.setColor(QPalette.Foreground, color)
        self.l1.setPalette(palette)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = scrollBarDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 设置了三个滚动条来控制标签字体的颜色，通过移动滑块来关联槽函数

```
self.s3.sliderMoved.connect(self.sliderval)
```

2. 在设置调色板时，注意前景色和背景色的设置

```
# 前景色指的是设置主体自身的颜色
palette.setColor(QPalette.Foreground, color)
# 背景色指的是设置主体的底色也就是背景颜色
palette.setColor(QPalette.Background, color)
```

5.3 多线程

应用背景执行一个很耗时的操作，会使得程序卡顿很久，导致系统认为程序运行出错自动关闭程序

1. 一般来说，多线程技术大致有三种形式

- 使用定时器模块QTimer
- 使用多线程模块QThread
- 使用事件处理的功能

5.3.1 QTimer 定时器模块

使用创建QTimer实例，将其timeout信号关联到槽函数然后周期性运行槽函数的start函数

常用的方法

1. start(ms)　启动或者重启定时器，时间间隔为毫秒，如果已经在运行就先停止再去重启，如果_singeShot_信号为真，定时器仅激活一次

2. stop()　　停止定时器

常用的信号

3. singleSlot　在给定的时间间隔后调用一个槽函数时发射该信号

4. timeout　　当定时器超时时发射该信号

案例5.3.1 QTimer 定时器模块的使用

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

class timerDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5.3.1 QTimer 定时器模块的使用")
        layout = QGridLayout()
        self.listFile = QListWidget()
        self.label = QLabel("显示当前时间")
        self.startBtn = QPushButton("开始")
        self.endBtn = QPushButton("结束")
        # 初始化一个定时器
        self.timer = QTimer(self)
        # showTime()方法
        self.timer.timeout.connect(self.showTime)
        layout.addWidget(self.label, 0, 0, 1, 2)
        layout.addWidget(self.startBtn, 1, 0)
        layout.addWidget(self.endBtn, 1, 1)

        self.startBtn.clicked.connect(self.startTimer)
        self.endBtn.clicked.connect(self.endTimer)

        self.setLayout(layout)

    def showTime(self):
        # 获取系统的当前时间
        time = QDateTime.currentDateTime()
        # 设置系统时间的显示格式
        timeDisplay = time.toString("yyyy-MM-dd hh:mm:ss dddd")
        # 在标签上显示时间
        self.label.setText(timeDisplay)

    def startTimer(self):
        # 设置时间间隔并启动定时器
        self.timer.start(1000)
        self.startBtn.setEnabled(False)
        self.endBtn.setEnabled(True)

    def endTimer(self):
        self.timer.stop()
        self.startBtn.setEnabled(True)
        self.endBtn.setEnabled(False)

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

```
    win = timerDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 将timer的timeout信号关联showTime槽函数显示当前时间并显示在标签上

```
self.timer.timeout.connect(self.showTime)
def showTime(self):
    # 获取系统的当前时间
    time = QDateTime.currentDateTime()
    # 设置系统时间的显示格式
    timeDisplay = time.toString("yyyy-MM-dd hh:mm:ss dddd")
    # 在标签上显示时间
    self.label.setText(timeDisplay)
```

2. 单击开始，启动定时器并使startBtn失效

```
def startTimer(self):
    # 设置时间间隔并启动定时器
    self.timer.start(1000)
    self.startBtn.setEnabled(False)
    self.endBtn.setEnabled(True)
```

案例5.3.1 QTimer 定时器模块的使用(二)

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

class timerDemo2(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5.3.1 QTimer 定时器模块的使用(二)")
        layout = QVBoxLayout()
        self.label = QLabel("<font color=red size=128><b>Hello PyQt,窗口将在10秒后消失</b>
/font>")
        self.timer = QTimer(self)
        self.timer.singleShot(10000, app.quit)
        layout.addWidget(self.label)
        # 设置无边框窗口
        self.setWindowFlags(Qt.SplashScreen | Qt.FramelessWindowHint)
        self.setLayout(layout)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = timerDemo2()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 弹出窗口在10秒后消失，模仿启动画面并设置窗口边框为无

```
self.label = QLabel("<font color=red size=128><b>Hello PyQt,窗口将在10秒后消失</b></fon
>")

# 设置无边框窗口
self.setWindowFlags(Qt.SplashScreen | Qt.FramelessWindowHint)
```

2. 使用定时器的单槽模式来完成程序的10秒退出

```
self.timer = QTimer(self)
self.timer.singleShot(10000, app.quit)
```

5.3.2 QThread 多线程

概念QThread是Qt线程类中最核心的底层类，由于PyQt的跨平台性，QThread隐藏了所有与平台相的代码

应用场景可以在自定义的QThread实例中自定义信号，并将信号连接到指定的槽函数，当满足一定业条件后发射该信号

QThread常用的方法

1. start()　　启动线程
2. wait()　　阻止线程，直到满足如下条件之一
   - 等待时间超时返回False，单位是毫秒
   - 不超时(线程必须从run()返回)或者线程尚未启动，返回True
3. sleep()　　强制当前线程睡眠几秒

QThread常用的信号

4. started　　在启动线程执行run()之前，一般用来资源的初始化
5. finished　　在线程完成业务逻辑之后，一般用来资源的释放

—

案例5.3.2 QThread多线程的基本使用

_使用技巧_在简单数据读取时可以直接在窗口初始化时完成；在复杂数据读取时(比如网络请求数据时比较长)则可以将这部分逻辑放在QThred线程中

概念MVC(模型-视图-控制器)实现界面的数据显示和数据读取的分离

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


class myThread(QThread):
    # 自定义一个信号
    sinOut = pyqtSignal(str)

    def __init__(self):
```

```
        super().__init__()
        self.working = True
        self.num = 0

    def __del__(self):
        self.working = False
        self.wait()  # 阻塞程序，直到线程执行完毕

    def run(self):
        while self.working == True:
            file_str = "File index {0}".format(self.num)
            self.num += 1
            # 发射该信号
            self.sinOut.emit(file_str)
            # 线程休眠2秒
            self.sleep(2)

class threadWidgetDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5.3.2 QThread多线程的基本使用")
        layout = QGridLayout()
        self.thread = myThread()
        self.listFile = QListWidget()
        self.startBtn = QPushButton("开始")
        layout.addWidget(self.listFile, 0, 0, 1, 2)
        layout.addWidget(self.startBtn, 1, 1)
        self.startBtn.clicked.connect(self.slotStart)
        self.thread.sinOut.connect(self.slotAdd)
        self.setLayout(layout)

    def slotAdd(self, file_index):
        self.listFile.addItem(file_index)

    def slotStart(self):
        self.startBtn.setEnabled(False)
        self.thread.start()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = threadWidgetDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 单击开始按钮，启动线程并使按钮失效

```
self.startBtn.clicked.connect(self.slotStart)
def slotStart(self):
    self.startBtn.setEnabled(False)
    self.thread.start()
```

2. 比较复杂的是线程中自定义的信号，sinOut信号连接槽函数slotAdd()会从线程发射信号中定时读数据，并把返回的数据动态添加到列表

```
# 发射该信号
self.sinOut.emit(file_str)
# 主线程接收信号的触发
self.thread.sinOut.connect(self.slotAdd)
# 槽函数响应，动态添加条目到列表中
def slotAdd(self, file_index):
    self.listFile.addItem(file_index)
```

案例5.3.2 主UI的卡死现象

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

global sec
sec = 0

def setTime():
    global sec
    sec += 1
    # LCD显示数字+1
    lcdNumber.display(sec)

def work():
    # 计时器每秒计数
    TestBtn.setEnabled(False)
    timer.start(1000)
    for i in range(200000000):
        pass
    timer.stop()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    top = QWidget()
    top.resize(300, 120)
    # 垂直布局
    layout = QVBoxLayout()
    # 添加一个显示面板
    lcdNumber = QLCDNumber()
    layout.addWidget(lcdNumber)
    TestBtn = QPushButton("测试")
    layout.addWidget(TestBtn)
    timer = QTimer()

    # 每次计时结束，触发setTime函数
    timer.timeout.connect(setTime)
    TestBtn.clicked.connect(work)
    top.setLayout(layout)
    top.show()
```

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

解析

1. 在PyQt中所有的窗口都在UI主线程中(就是执行app.exec_()的线程)，在这个线程中执行耗时的操会阻塞UI线程，从而让窗口停止响应严重的会导致程序崩溃

案例5-9 应用案例：分离主UI线程和工作线程

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

global sec
sec = 0

class myThread(QThread):
    trigger = pyqtSignal()
    def __init__(self):
        super().__init__()

    def run(self):
        for i in range(200000):
            pass
        # 循环完毕发射信号
        self.trigger.emit()


def setTime():
    global sec
    sec += 1
    # LCD显示数字+1
    lcdNumber.display(sec)


def work():
    # 计时器每秒计数

    timer.start(1000)
    thread.start()
    thread.trigger.connect(timeStop)


def timeStop():
    timer.stop()
    print("运行线程中的业务用时" + lcdNumber.value())
    global sec
    sec = 0


if __name__ == '__main__':
    app = QApplication(sys.argv)
    top = QWidget()
    top.resize(300, 120)
```

```
    # 垂直布局
    layout = QVBoxLayout()
    # 添加一个显示面板
    lcdNumber = QLCDNumber()
    layout.addWidget(lcdNumber)
    TestBtn = QPushButton("测试")
    layout.addWidget(TestBtn)
    timer = QTimer()
    thread = myThread()

    TestBtn.clicked.connect(work)
    # 每次计时结束，触发setTime函数
    timer.timeout.connect(setTime)

    top.setLayout(layout)
    top.show()
    sys.exit(app.exec_())
```

5.3.3 事件处理

概念PyQt为事件处理提供了两种机制：高级的信号与槽机制，以及低级的事件处理程序processEvent()---简单说就是_刷新页面_

案例5.3.3 事件处理的低级处理程序的使用

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

class processEventDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5.3.3 事件处理的低级处理程序的使用")
        self.listFile = QListWidget()
        self.startBtn = QPushButton("开始")
        layout = QGridLayout()
        layout.addWidget(self.listFile, 0, 0, 1, 2)
        layout.addWidget(self.startBtn, 1, 1)
        self.startBtn.clicked.connect(self.slotAdd)
        self.setLayout(layout)

    def slotAdd(self):
        for i in range(10):
            str_i = "File index {}".format(i)
            self.listFile.addItem(str_i)
            QApplication.processEvents()
            time.sleep(1)

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

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

5.4 网页交互

1. PyQt5使用QWebEngineView控件来展示HTML5页面，使用的是Chrom内核
2. 可以通过PyQt5.QtWebEngineWidgets.QWebEngineView类来使用网页控件
3. **QWebEngineView常用的方法**
   - load(QUrl url)　　加载指定的URL并显示(本质是使用HTTP GET方法加载Web页面)
   - setHtml(QString &html)　在网页视图中设置指定的HTML内容

案例5-10 加载并显示外部的WEB页面

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

class loadUrlDemo(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5-10 加载并显示外部的WEB页面")
        self.setGeometry(5, 30, 1355, 730)
        self.browser = QWebEngineView()
        # 加载外部的WEB页面
        self.browser.load(QUrl("http://www.cnblogs.com/wangshuo1"))
        self.setCentralWidget(self.browser)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = loadUrlDemo()
    win.show()
    sys.exit(app.exec_())
```

案例5-11 加载并显示本地的WEB页面

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

class loadUrlDemo(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5-10 加载并显示外部的WEB页面")
        self.setGeometry(5, 30, 1900, 1000)
        self.browser = QWebEngineView()
        # 加载本地的WEB页面
        URL = r"D:/大背景VIP解析站html源码/index.html"
        self.browser.load(QUrl(URL))
```

```
        self.setCentralWidget(self.browser)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = loadUrlDemo()
    win.show()
    sys.exit(app.exec_())
```

案例5-12 加载并显示嵌入的HTML代码

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

class loadUrlDemo(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("案例5-10 加载并显示外部的WEB页面")
        self.setGeometry(5, 30, 1900, 1000)
        self.browser = QWebEngineView()
        # 加载HTML代码
        self.browser.setHtml('''
        <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>起风了视频解析</title>
    <meta name="keywords" content="vip视频解析,爱奇艺vip解析,腾讯vip解析,芒果TVvip解析"/>
    <meta name="descriptipn" content="视频站"/>
    <link href="css/kz.css" rel="stylesheet" type="text/css">
    <button class="sy" id="bybf" type="button" onclick="dihejk3()"><div style="color:#ffffff"
友情链接</div></button>
    <script type="text/javascript">
        function dihejk(){
            var diz=document.getElementById("dz").value;
            var jkdz=document.getElementById("jieko");
            var jk=document.getElementById("jieko").selectedIndex;
            var jkv=jkdz.options[jk].value;
            var cljdz=document.getElementById("bfq");
            cljdz.src=jkv+diz;
        }
        function dihejk2(){
            var diz=document.getElementById("dz").value;
            var jkdz=document.getElementById("jieko");
            var jk=document.getElementById("jieko").selectedIndex;
            var jkv=jkdz.options[jk].value;
            //var cljdz=document.getElementById("bfq");
            window.open(jkv+diz);
        }
        function dihejk3(){
            window.open("index.php")
```

```
        }
        window.onload=function() {
            var canvas = document.getElementById("tianxian");
            var context = canvas.getContext("2d");
            //弧线
            context.beginPath();

            context.moveTo(55, 20);
            context.lineTo(155, 150);
            context.lineTo(255, 20);
            context.lineJoin = "bevel";
            context.lineCap="butt";
            context.lineWidth="5";
            context.strokeStyle="#fff"
            context.stroke();

        }
    </script>


</head>
<body>
<div id="zhenti">
    <canvas id="tianxian">你的浏览器太旧不支持</canvas>
    <div id="wbk">
        <div id="bt">
            <span class="font-display">QFL SPIP</span>
            <br/>
            <a href="http://www.ismallcolor.com" target="_blank"></a>
        </div>

        <div id="kj">
        <iframe src="Explain.html" id="bfq" width="100%" height="100%"
        frameborder="0" scrolling="no">
        </iframe>

        </div>
        <div id="an">
        <select class="jieko" id="jieko" name="接口">
            <option value="http://api.nepian.com/ckparse/?url=">通用解析1</option>
            <option value="http://q.z.vip.totv.72du.com/?url=">通用解析2</option>
            <option value="http://www.wmxz.wang/video.php?url=">通用解析3</option>
            <option value="http://api.wlzhan.com/sudu/?url=">通用解析4</option>
            <option value="http://aikan-tv.com/?url=">爱奇艺高清</option>
            <option value="http://apiv.ga/magnet/" >磁力链接</option>

        </select>
        <input class="dzsr" type="search" placeholder="输入视频地址：例如(http://www.iqiyi.co
/v_19rr9k5h1s.html)" size="80" id="dz">
        <button class="bfann" style="background-color:transparent" id="bf" type="button" onc
ick="dihejk()">
            <img src="1.png" >
        </button>
        <button class="byan" id="bybf" type="button" onclick="dihejk2()"><div style="color:#ff
```

```
fff">网页打开</div></button>
        </div>
    </div>
</div>
<style type="text/css">
select.jieko
{
position:absolute;
left:620px;
top:715px
}
input.dzsr
{
position:absolute;
left:720px;
top:715px
}
button.bfann
{
position:absolute;
left:1240px;
top:715px
}
button.byan
{
position:absolute;
left:1290px;
top:718px
}
button.sy
{
position:absolute;
left:1360px;
top:718px
}
</style>

</body>

</html>
        ''')
        self.setCentralWidget(self.browser)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = loadUrlDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 使用QWebEngineView对象setHtml()函数渲染html代码时，如果页面中使用的JS代码超过_2MB，程序渲染就会失败，页面渲染后将会显示大面积的空白

案例5-13 PyQt调用JavaScript代码

_使用_通过QWebEnginePage类的runJavaScript(str, Callable)函数可以很方便地实现PyQt和HTML/avaScript的双向通信，也实现了Python代码和HTML/JavaScript代码的解耦

- QWebEnginePage.runJavaScript(str, Callable)

学习小技巧

1. 明显QWebEngineView实例的.page()方法也是QWebEnginePage类对象，所以也可以使用view.page().runJavaScript(str, Callable)

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

class webViewDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("PyQt调用JavaScript代码")
        layout = QVBoxLayout()
        self.view = QWebEngineView()
        self.view.setHtml('''
        <html>
        <head>
        <title>A Demo Page</title>
        <script language="JavaScript">
              // 获取输入的姓名，然后在页面中显示提交按钮
              function completeAndReturnName() {
                 var fname = document.getElementById('fname').value;
                 var lname = document.getElementById('lname').value;
                 var full = fname + ' ' + lname;
                 document.getElementById('fullname').value = full;
                 document.getElementById('submit-btn').style.display = 'block';
              }
        </script>
        </head>
        <body>
        <form>
        <label for="fname">First Name:</label>
        <input type="text" name="fname" id="fname">
        <br>
        <label for="lname">Last Name:</label>
        <input type="text" name="lname" id="lname">
         <br>
        <label for="fullname">Full Name:</label>
        <input disabled type="text" name="fullname" id="fullname">
         <br>

        <input style="display: none;" type="submit" id="submit-btn">
        </form>
        </body>
        </html>
```

```
        ''')
        # 创建一个按钮用于调用JavaScript代码
        self.btn = QPushButton("设置全名")
        self.btn.clicked.connect(self.complete_name)
        layout.addWidget(self.view)
        layout.addWidget(self.btn)

        self.setLayout(layout)

    def js_callback(self, result):
        print(result)

    def complete_name(self):
        self.view.page().runJavaScript('completeAndReturnName();', self.js_callback)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = webViewDemo()
    win.show()
    sys.exit(app.exec_())
```

解析

1. 初始化一个QWebEngineView对象view，通过view.page()函数获得QWebEnginePage对象

```
def js_callback(self, result):
    print(result)

def complete_name(self):
    self.view.page().runJavaScript('completeAndReturnName();', self.js_callback)
```

案例5-14 JavaScript调用PyQt代码

_概念_JavaScript调用pyqt代码，指的是pyqt可以与加载的WEB页面进行双向的数据交互

1. 使用QWebEngineView加载WEB页面，就可以获取到页面中表单输入数据，在WEB页面通过js代收集用户提交的数据
2. 在WEB页面中，js通过桥接方式传递数据给pyqt
3. pyqt接收到数据，经过业务处理后，还可以把处理的数据返回给WEB页面

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtWebEngineWidgets import *
from PyQt5.QtWebChannel import *
# from mySharedObject import MySharedObject

class MySharedObject(QWidget):
    def __init__(self):
        super().__init__()

    def _getStrValue(self):
        # 设置参数
```

```
        return '100'

    def _setStrValue(self, str):
        # 获取参数
        print("获得页面的参数是: %s" % str )
        QMessageBox.information(self, "Infomation", "获得页面参数：%s" % str, QMessageBox.Ye
| QMessageBox.No, QMessageBox.Yes)

        # 需要定义对外发布的方法
    strValue = pyqtProperty(str, fget=_getStrValue, fset=_setStrValue)


# 主函数部分切记不要写成一个类，不然会出错，原因大概是数据管道的双向连接问题
if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = QWidget()
    win.setWindowTitle("案例5-14 JavaScript调用PyQt代码")
    layout = QVBoxLayout()
    view = QWebEngineView()
    htmlUrl = r"D:/Python_集合/PyQt5_Projects/PyQt5快速开发与实战/Pycharm_PyQt/Chapter01
test.html"

    view.load(QUrl(htmlUrl))

    # 创建一个QWebChannel对象，用来传递PyQt的参数到JavaScript---搭建数据管道
    channel = QWebChannel()
    myObj = MySharedObject()
    channel.registerObject("bridge", myObj)
    view.page().setWebChannel(channel)

    # 把QWebEngineView控件和button控件添加到布局中
    layout.addWidget(view)

    win.setLayout(layout)
    win.show()

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

----------配套的HTML代码如下

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>A Demo Page</title>
    <script type="text/javascript" src="./qwebchannel.js"></script>
    <script type="text/javascript">
        // 添加一个事件监听器，事件触发响应函数为参数中的function
        document.addEventListener("DOMContentLoaded", function () {
            // 在WEB页面也建立起一个数据管道桥接PyQt5中建立的那个管道进行数据的双向通信
            new QWebChannel(qt.webChannelTransport, function (channel) {
                window.bridge = channel.objects.bridge;
                alert("bridge=" + bridge + '\n从pyqt传来的参数=' + window.bridge.strValue);
            });
```

```
        });
        function onshowMsgBox() {
            console.log(window.bridge);
            if (window.bridge) {
                var fname = document.getElementById("fname").value;
                window.bridge.strValue = fname
            }
        }
    </script>
</head>
<body>
<form action="#">
    <label for="fname">User Name:</label>
    <input type="text" name="fname" id="fname">
    <br>
    <input type="button" value="传递参数到pyqt" onclick="onshowMsgBox();">
    <input type="reset" value="重置">
</form>

</body>

</html>
```

解析

_注意_自定义的MySharedObject类可以必须要继承QObject和QWidget其中之一，继承QObject类以使用的父类方法和属性就比较少，因为QObject类是大多数类的基类

1. 创建QWebChannel对象，注册一个桥接对象，以便PyQt与WEB页面进行交互

```
# 创建一个QWebChannel对象，用来传递PyQt的参数到JavaScript
self.channel = QWebChannel()
myObj = MySharedObject()
self.channel.registerObject("bridge", myObj)
self.view.page().setWebChannel(self.channel)
```

2. 创建共享数据的PyQt对象，便于交互

```
class MySharedObject(QWidget):
    def __init__(self):
        super().__init__()

    def _getStrValue(self):
        # 设置参数
        return '100'

    def _setStrValue(self, str):
        # 获取参数
        print("获得页面的参数是: %s" % str )
        QMessageBox.information(self, "Infomation", "获得页面参数：%s" % str, QMessageBox.Ye
| QMessageBox.No, QMessageBox.Yes)

        # 需要定义对外发布的方法
    strValue = pyqtProperty(str, fget=_getStrValue, fset=_setStrValue)
```

pyqtProperty对外提供的PyQt对象方法，需要使用pyqtProperty()函数让它暴露出来

小知识点在PyQt5中使用pyqtProperty函数定义PyQt对象中的属性与Python中的property函数相同

- **pyqtProperty的参数**
  - type　　必填　属性的类型
  - fget　　选填　用于获取属性的值
  - fset　　用于设置属性的值
  - freset　　用于将属性的值重置为默认值
  - fdel　　用于删除属性
  - Doc　　属性的文档字符串
  - .....　　还有一些基本不会用到

pyqtProperty函数的测试用例

```
from PyQt5.QtCore import QObject, pyqtProperty

class MyObject(QObject):
    def __init__(self, inVal=20):
        super().__init__()
        self.val = inVal

    def readVal(self):
        print('readVal={}'.format(self.val))
        return self.val

    def setVal(self, val):
        print('setVal={}'.format(val))
        self.val = val

    ppVal = pyqtProperty(int, readVal, setVal)

if __name__ == '__main__':
    obj = MyObject()
    print("\n#1")
    obj.ppVal = 10
    print("\n#2")
    print("obj.ppVal={}".format(obj.ppVal))
    print("obj.readVal={}".format(obj.readVal()))
```

3. 建立双向通信的数据管道---搭桥

```
// 在WEB页面也建立起一个数据管道桥接PyQt5中建立的那个管道进行数据的双向通信
document.addEventListener("DOMContentLoaded", function () {

        new QWebChannel(qt.webChannelTransport, function (channel) {
            window.bridge = channel.objects.bridge;
            alert("bridge=" + bridge + '\n从pyqt传来的参数=' + window.bridge.strValue);
        });
    });
# 在PyQt中搭建桥
```

```
channel = QWebChannel()
myObj = MySharedObject()
# 给共享数据对象取别名为bridge
channel.registerObject("bridge", myObj)
view.page().setWebChannel(channel)
```