





# 《PySide6/PyQt6 快速开发与实战》附赠电子版

由于本书内容太多，初稿有快 800 页内容。这样的话由于成本因素，定价不会太低，会提高读者的学习成本。考虑经济形势不好，为减轻读者负担，必须节约篇幅。本章因此剥离出部分章节内容，以附赠电子版的形式呈现出来。由于是附赠内容，本章没有编辑校对，若有错误还请海涵。

本章电子版与源代码放在一起，路径为“appendix/《PySide6-PyQt6 快速开发与实战》附赠电子版.pdf”。本书源代码在 gitee 或 github 中都可以查到（打开 gitee 或 github 官网，搜索关键字：sunshe35/PyQt6-codes 或 sunshe35/PySide6-codes）。

## 目录

《PySide6/PyQt6 快速开发与实战》附赠电子版 .....	1
X.1 不规则窗口 .....	2
 案例 X-1 不规则窗口+拖拽 .....	2
 案例 X-2 不规则窗口+动画 .....	4
X.2 窗口数据传递 .....	6
X.2.1 多窗口数据传递：调用属性 .....	6
 案例 X-3 多窗口数据传递：使用属性 .....	7
X.2.2 多窗口数据传递：信号与槽 .....	8
 案例 X-4 多窗口数据传递：信号与槽 .....	8
X.3 UI 层的自动化测试 .....	9
X.3.1 手工测试与自动化测试 .....	10

X.3.2	模拟鸡尾酒的调酒器窗口 .....	11
X.3.2	启动测试程序 .....	14
X.3.3	单元测试程序 .....	16
X.3.4	运行测试用例 .....	26
X.3.5	生成测试报告 .....	27
X.4	金融数据显示 .....	29
X.4.1	控件级别的布局管理 .....	29
X.4.2	窗口级别的布局管理 .....	32
X.4.3	显示产品基本信息 .....	33
X.4.4	展示产品组合信息 .....	39

## X.1 不规则窗口

实现不规则窗口的最简单方式是使用一张遮罩层图片来建立不规则窗口，然后再利用 `paintEvent()` 函数在这个不规则窗口上重绘背景图。`QWidget.setMask()` 方法可以给控件增加一个遮罩，遮住所选区域以外的部分。它的参数可以为 `QBitmap` 或 `QRegion` 对象，使用方法如下：

```
"""
setMask(self, arg__1: Union[PySide6.QtGui.QBitmap, str]) -> None
setMask(self, arg__1: Union[PySide6.QtGui.QRegion,
PySide6.QtGui.QBitmap, PySide6.QtGui.QPolygon, PySide6.QtCore.QRect]) ->
None
"""
```

如果参数为 `QBitmap`，那么窗口像素中只有 `QBitmap` 为 1bit 的部分可见。如果参数为 `QRegion`，那么小部件中与 `QRegion` 重叠部分可见。



### 案例 X-1 不规则窗口+拖拽

本例文件路径为 `appendix\X\qt_setMask1.py`，演示了不规则窗口+拖拽的使用方法，代码如下：

```
class ShapeWidget(QWidget):
    def __init__(self, parent=None):
        super(ShapeWidget, self).__init__(parent)
        self.setWindowTitle("不规则的，可以拖动的窗体实现例子")
        self.updatePix()

    # 显示不规则 pic
    def updatePix(self):
        self.pix = QPixmap("./images/mask.png")
        self.resize(self.pix.size())
        self.setMask(self.pix)
        print(self.pix.size())
        self.dragPosition = None

    # 重定义鼠标按下响应函数 mousePressEvent (QMouseEvent) 和鼠标移动响应函数
    mouseMoveEvent (QMouseEvent)，使不规则窗体能响应鼠标事件，随意拖动。
    def mousePressEvent(self, event:QMouseEvent):
        if event.button() == Qt.LeftButton:
            self.m_drag = True
            self.m_DragPosition = event.globalPosition().toPoint() -
self.pos()
            event.accept()
            self.setCursor(QCursor(Qt.OpenHandCursor))
        if event.button() == Qt.RightButton:
            self.close()

    def mouseMoveEvent(self, event:QMouseEvent):
        if Qt.LeftButton and self.m_drag:
            # 当左键移动窗体修改偏移值
            self.move(event.globalPosition().toPoint() -
self.m_DragPosition)
            event.accept()

    def mouseReleaseEvent(self, event:QMouseEvent):
        self.m_drag = False
        self.setCursor(QCursor(Qt.ArrowCursor))

    # 一般 paintEvent 在窗体首次绘制加载，要重新加载 paintEvent 需要重新加载窗口使
    用 self.update() or self.repaint()
```

```
def paintEvent(self, event):
    painter = QPainter(self)
    painter.drawPixmap(0, 0, self.width(), self.height(),
QPixmap("./images/boy.png"))
```

运行结果如下图，可以看到，我们绘制了一张不规则的图：



updatePix 和 paintEvent 定义了遮罩和绘图方法，mousePressEvent，mouseReleaseEvent，mouseMoveEvent 定义了鼠标拖拽的事件，在 mousePressEvent 中，点击左键才能拖拽，点击右键会使窗口关闭。



## 案例 X-2 不规则窗口+动画

本例文件路径为 appendix\X\qt\_setMask2.py，演示了不规则窗口实现动画效果方法，代码如下：

```
class ShapeWidget(QWidget):
    def __init__(self, parent=None):
        super(ShapeWidget, self).__init__(parent)
        self.i = 1
        self.updatePix()
        self.timer = QTimer()
        self.timer.setInterval(1000) # 1000 毫秒
        self.timer.timeout.connect(self.timeChange)
        self.timer.start()

# 显示不规则 pic
def updatePix(self):
    if self.i == 5:
```

```
        self.i = 1
        self.mypic = {1: './images/left.png', 2: './images/up.png', 3:
            './images/right.png', 4: './images/down.png'}
        self.pix = QPixmap(self.mypic[self.i], "0", Qt.AvoidDither |
            Qt.ThresholdDither | Qt.ThresholdAlphaDither)
        self.resize(self.pix.size())
        self.setMask(self.pix.mask())
        self.dragPosition = None
        self.m_drag = False
        self.update()

    def mousePressEvent(self, event:QMouseEvent):
        if event.button() == Qt.LeftButton:
            self.m_drag = True
            self.m_DragPosition = event.globalPosition().toPoint() -
self.pos()
            event.accept()
            self.setCursor(QCursor(Qt.OpenHandCursor))
        if event.button() == Qt.RightButton:
            self.close()

    def mouseMoveEvent(self, event:QMouseEvent):
        if Qt.LeftButton and self.m_drag:
            self.move(event.globalPosition().toPoint() -
self.m_DragPosition)
            event.accept()

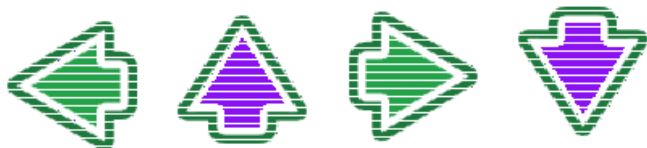
    def mouseReleaseEvent(self, event:QMouseEvent):
        self.m_drag = False
        self.setCursor(QCursor(Qt.ArrowCursor))

    def paintEvent(self, event:QPaintEvent):
        painter = QPainter(self)
        painter.drawPixmap(0, 0, self.pix.width(), self.pix.height(),
self.pix)

# 鼠标双击事件
    def mouseDoubleClickEvent(self, event:QMouseEvent):
        if event.button() == 1:
            self.i += 1
            self.updatePix()
```

```
# 每1000 毫秒修改 paint
def timeChange(self):
    self.i += 1
    self.updatePix()
```

运行效果，如下图所示，下面 4 幅图中每间隔 1 秒会循环切换一个：



这个案例和上一个案例差不多，每间隔 1000 秒会触发 `updatePix` 方法，在这个方法中，会切换遮罩图。`mousePressEvent`，`mouseReleaseEvent`，`mouseMoveEvent` 定义了鼠标拖拽的事件，功能和上一个案例一样。`mouseDoubleClickEvent` 触发鼠标双击事件，会触发 `updatePix` 从而切换遮罩图。

## X.2 窗口数据传递

在开发程序时，如果这个程序只有一个窗口，则应该关心这个窗口里面的各个控件之间是如何传递数据的；如果这个程序有多个窗口，那么还应该关心不同的窗口之间是如何传递数据的，这就是本节要解决的问题。本节之前的信号与槽的例子说明了单个窗口的数据传递情况，对于多窗口，一般有两种解决方法：一种是主窗口获取子窗口中控件的属性；另一种是通过信号与槽机制，一般是子窗口通过发射信号的形式传递数据，主窗口的槽函数获取这些数据。

### X.2.1 多窗口数据传递：调用属性

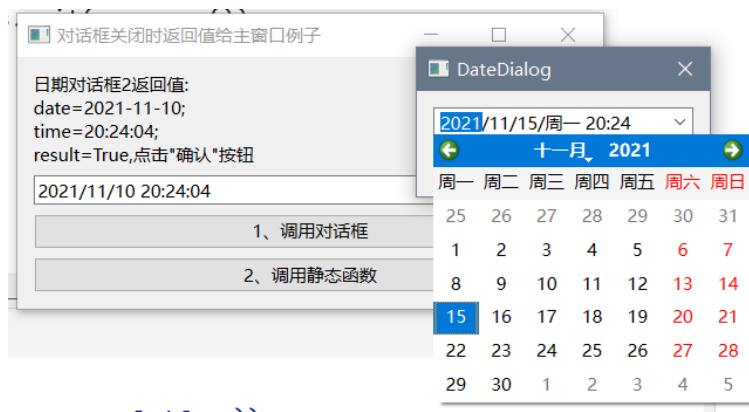
在 PySide/PyQt 编程过程中，经常会遇到输入或选择多个参数的问题。把多个参数写到一个窗口中，主窗口会显得很臃肿，所以一般是添加一个按钮，调用对话框，在对话框中进行参数的选择，关闭对话框时将参数值返回给主窗口。

Qt 提供了一些标准的对话框类，用于输入数据、修改数据、更改应用的设置等，常见的有 `QFileDialog`、`QInputDialog`、`QColorDialog`、`QFontDialog` 等。在不同的窗口之间传参有两种常用的方式：一种是在自定义对话框之间通过属性传参；另一种是在窗口之间使用信号与槽机制传参。本节主要介绍前一种方式。



## 案例 X-3 多窗口数据传递：使用属性

案例位于 `appendix\X\MultiWindow.py`, 代码比较简单, 节约篇幅不在此展示。运行脚本, 显示效果如下图所示。



本例提供了调用对话框的两种方法, 分别是实例化方法以及静态方法。

第一种方法: 直接在主窗口程序中实例化该对话框, 然后调用该对话框的函数来获取返回值, 根据对话框的返回值单击确认按钮还是取消按钮来进行下一步操作。对应按钮 1, 代码如下:

```
def onButton1Click(self):
    dialog = DateDialog(self)
    result = dialog.exec()
    datetime = dialog.dateTime()
    date = datetime.toString('yyyy-MM-dd')
    tim = datetime.time().toString()
    self.lineEdit.setText(datetime.toString('yyyy/MM/dd hh:mm:ss'))
    _str = f'日期对话框 1 返回值:\ndate={date};\ntime={tim};\nresult={result}'

    if result == QDialog.Accepted:
        _str += ', 点击"确认"按钮'
    else:
        _str += ', 点击"取消"按钮'

    self.label.setText(_str)
    dialog.destroy()
```

第二种方法: 在主窗口程序中调用子窗口的静态函数, 实际上这种方法与第一种方法是一样的, 只不过它是利用静态函数的特点, 在子窗口的静态函数中创建实例化对象。对应按钮 2, 代码如下:

```
def onButton2Click(self):
    datetime, result = DateDialog.getDateTime()
    date = datetime.toString('yyyy-MM-dd')
    tim = datetime.time().toString()
    self.lineEdit.setText(datetime.toString('yyyy/MM/dd hh:mm:ss'))
    _str = f'日期对话框 2 返回'
    值:\ndate={date};\ntime={tim};\nresult={result}'
    if result == QDialog.Accepted:
        _str += ',点击"确认"按钮'
    else:
        _str += ',点击"取消"按钮'
    self.label.setText(_str)
```

此处 getDateTime 是 DateDialog 类的静态函数，如下：

```
class DateDialog(QDialog):
    # 静态方法创建对话框并返回 (date, time, accepted)
    @staticmethod
    def getDateTime(parent=None):
        dialog = DateDialog(parent)
        result = dialog.exec()
        datetime = dialog.dateTime()
        return (datetime, result == QDialog.Accepted)
```

## X.2.2 多窗口数据传递：信号与槽

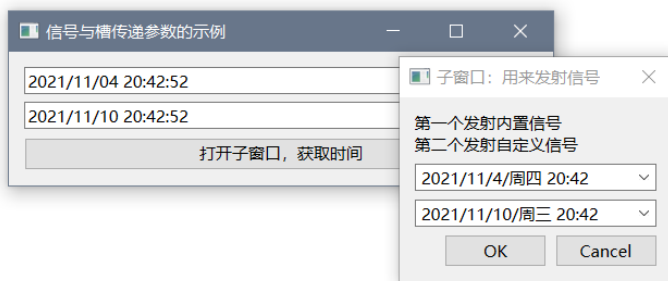
另一种多窗口传递数据方法是信号与槽，一般可以通过子窗口发射信号的，主窗口通过槽函数捕获这个信号，并执行相应操作。子窗口发射的信号有两种，其中一种是发射 Qt 内置的一些信号；另一种是发射自定义的信号。这两种方式的信号与槽机制在本例中都会介绍到。



### 案例 X-4 多窗口数据传递：信号与槽

案例位于 appendix\X\MultiWindow2.py，代码比较简单，节约篇幅不在此展示，运行脚本，显示效果如下图所示，当子窗口时间变化的时候会同步更新主窗口时间。





对于内置信号，直接通过 `dialog.datetime_inner.dateTimeChanged` 连接主窗口，如下：

```
dialog.datetime_inner.dateTimeChanged.connect(self.deal_inner_slot)
def deal_inner_slot(self, datetime):
    self.lineEdit_inner.setText(datetime.toString('yyyy/MM/dd hh:mm:ss'))
```

对于自定义信号，通过 `dialog.datetime_emit.dateTimeChanged` 连接 `dialog.datetimeSingnal`，然后连接主窗口，代码如下：

```
# from DateDialog
self.datetime_emit.dateTimeChanged.connect(self.emit_signal)
def emit_signal(self):
    datetime = self.datetime_emit.dateTime()
    self.datetimeSingnal.emit(datetime.toString('yyyy/MM/dd hh:mm:ss'))

# from WinForm
dialog.datetimeSingnal.connect(self.deal_emit_slot)
def deal_emit_slot(self, dateStr):
    self.lineEdit_emit.setText(dateStr)
```

## X.3 UI层的自动化测试

一般来说，UI 层的自动化测试是通过工具或编写脚本的方式来模拟手工测试的过程，通过运行脚本来执行测试用例，从而模拟人工对软件的功能进行验证。

PySide/PyQt 是 Qt 框架的 Python 语言实现，对于单元测试，Python 可以使用它内部自带的单元测试模块 `unittest`。对于模拟手工操作，PySide/PyQt 可以使用它内部的测试模块 `QTest`。本节将结合 `unittest` 和 `QTest` 模块对 PySide6/PyQt6 应用的 UI 窗

口进行自动化测试。



注意

虽然 Qt C++ API 包含了完整的单元测试框架，但是 PyQt/PySide 的 QTest 模块仅包含 QTest 类，它使用静态方法来模拟按键、鼠标单击和鼠标移动。

### X.3.1 手工测试与自动化测试

手工测试是传统、常规的软件测试方法，由测试人员依据设计文档手工编写测试用例，然后执行并记录测试结果。对于手工测试，大部分测试人员再熟悉不过了，例如某个测试用例，是在页面中输入不同的值反复提交一个表单，对查询结果进行测试，然后判断查询数据是否符合业务逻辑。这种测试方法适用于测试用例中输入项比较少的情況，如果需要不断地重复某个测试用例，例如不断地重复验证用户登录系统 10000 次后，是否还能登录系统成功，那么手工测试就会很累，这个时候就需要使用自动化测试来模拟手工登录系统的操作，从而避免重复的劳动。

自动化测试是指利用软件测试工具自动实现全部或者部分测试工作（测试管理、测试用例设计、测试执行和生成测试报告）。自动化测试可节省大量的测试资源，并能够完成一些手工测试无法实现的测试，比如单元测试、统计测试覆盖率等。随着技术的进步，又发展出 UI 层的自动化测试。

UI 层的自动化测试，是自动化测试的一类，是指编写代码、脚本，通过测试框架的驱动，让脚本自动运行，通过 UI 层面的键盘输入和鼠标操作，发现被测系统的缺陷，来代替部分手工测试。它的核心思想是，通过测试框架抓取被测元素对象，保存至对象库，通过脚本的编写以及配置必要的测试数据，在被测系统上进行回放，驱动被测系统完成我们期望的操作，获得最终结果，并将最终结果与预设的期望值进行比对，将比对结果进行报告输出。

综上所述，手工测试和自动化测试的特点总结如下：

（1）手工测试由人手工去执行测试用例；自动化测试由程序代替人去执行测试用例。

（2）手工测试非常消耗时间，持续进行手工测试会使测试人员感到疲惫；自动化测试可以代替一部分机械重复的手工测试。

（3）手工测试永远无法被自动化测试取代。在整个软件开发周期中，手工测试发现 Bug 所占的比例大，大约为 80%；而自动化测试只能发现大约 20% 的 Bug。

（4）手工测试适合测试业务逻辑；自动化测试适合进行回归测试。回归测试用于测试已有功能，而不是新增功能。自动化测试有利于测试项目底层的细节，比如可以测试出软件的崩溃、API 的错误返回值、业务逻辑异常和软件的内存使用等。

由于 PySide/PyQt 的 UI 自动化测试用例比较少, 因此这里参考了国外网友 John McGehee 编写的测试用例。原文地址是 <http://johnnado.com/pyqt-qtest-example>, 但目前网站已经关闭。John McGehee 使用 QtDesigner 设计了玛格丽特鸡尾酒的调酒器窗口界面, 采用的 PyQt 开发版本是 PyQt4, 笔者对他的代码进行了优化, 添加了新功能, 并使用最新版的 PySide6/PyQt6 重写了测试用例。自动化测试部分的编写要感谢 HP 的自动化测试专家楚建欣, 他是中国国内自动化测试方面的专家, 作为自动化项目的技术负责人, 有着多年的自动化测试经验。在此向 John McGehee 和楚建欣专家表示感谢。

接下来会使用 Qt Designer 制作一个模拟调制鸡尾酒的调酒器窗口, 然后使用 Python 的 unittest 模块和 PyQt/PySide 的 QTest 模块对调酒器窗口进行 UI 层的自动化测试。

### X.3.2 模拟鸡尾酒的调酒器窗口

现实生活中的调制玛格丽特鸡尾酒的机器如图 9-47 所示。



图 9-47

玛格丽特鸡尾酒 (Margarita Midori)

原料: 冰块 8 粒、Ei Charro Anejo 龙舌兰酒 20 毫升、Midori 10 毫升、新鲜柠檬汁 20 毫升、Triple Sec 20 毫升、细盐少许。

做法: 将龙舌兰酒、新鲜柠檬汁、Midori 和 Triple Sec 倒入摇酒壶中, 加入一些冰块, 摇匀后滤入挂有盐霜的酒杯中, 并在泡沫上撒少许细盐。

按照调酒方法将龙舌兰酒、新鲜柠檬汁和冰块装入量酒器 (jigger) 中, 就可以根据量酒器的体积 (升) 推算出可以调制出来的鸡尾酒的体积 (升) 了。因为不同液体的密度是不一样的, 在本例中, 一个量酒器可以容纳 0.0444 升的鸡尾酒。

一个量酒器只能调配出一杯鸡尾酒, 而调酒器一次可以调制出多杯鸡尾酒。

如图 9-48 和图 9-49 所示分别是在生活中使用的量酒器和玛格丽特鸡尾酒的成品, 读者可以很快明白量酒器和鸡尾酒的体积关系。



图 9-48

图 9-49

PyQt/PySide 自动化测试例子中各个文件说明如下。

- MatrixWinUi.ui, Qt Designer 的可视化编辑文件, 用于描述 GUI 对话框设计。
- MatrixWinUi.py, 用于描述 GUI 对话框设计的 Python 源代码文件, 可以使用如下命令来创建窗口的.py 文件。

```
pyside6-uic.exe -o MatrixWinUi.py MatrixWinUi.ui
```

- MatrixWinUiRun.py, 包含了实例化 GUI 对话框并处理结果的类, 是 MatrixWinUi.py 的启动文件。
- MatrixWinTest.py, 自动化单元测试。
- RunTestCase.py, 进行自动化测试, 生成测试报告。

以上文件都保存在 Chapter08/testCase 目录下。读者可以按照自己的想法进行修改, 测试代码的覆盖率越高, 软件的质量就越好。在本例中编写了自动化测试代码, 使用测试代码来模拟人工执行测试用例, 提高了代码测试的覆盖率, 并且执行完测试用例后会自动生成测试报告。

对于这个例子, 使用 Qt Designer 来设计鸡尾酒调酒器窗口, 如图 9-50 和图 9-51 所示。该窗口文件名为 MatrixWinUi.ui, 保存在 Chapter08/testCase 目录下。

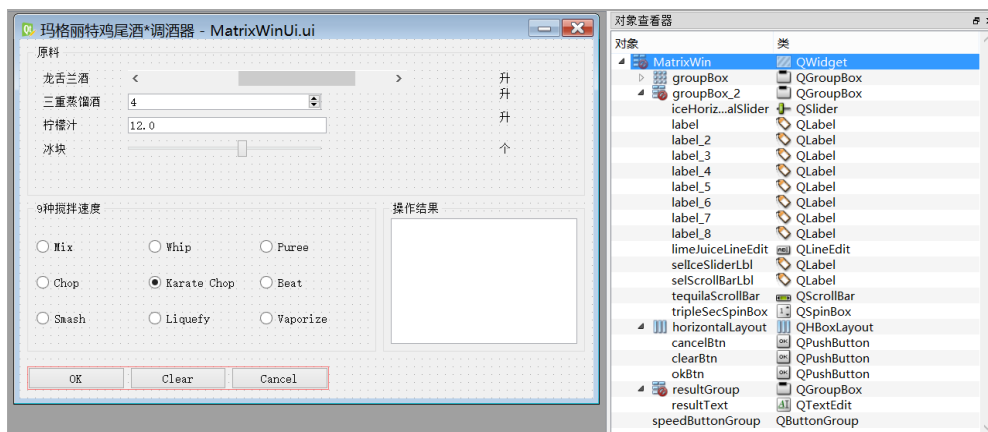


图 9-50

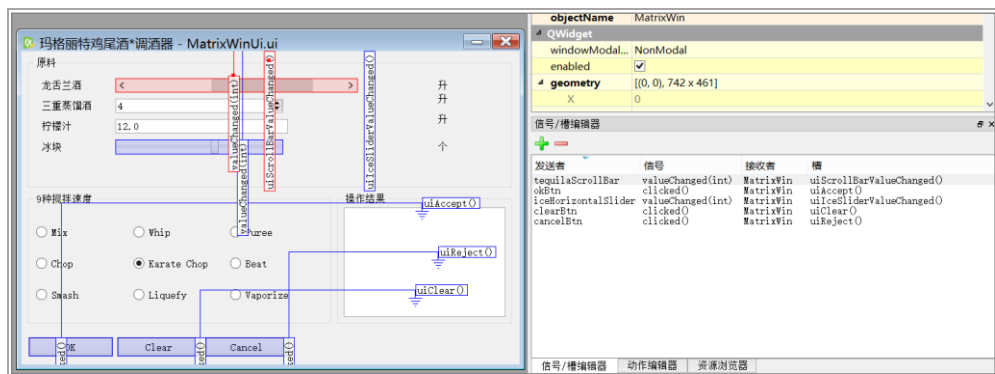


图 9-51

在窗口上部，指定各原料的量酒器容量（一个量酒器为 0.0444 升）。在“9 种搅拌速度”部分，选择搅拌速度。选定原料量和搅拌速度后，单击“Ok”按钮，就会模拟真实机器调制鸡尾酒了，并把结果显示在窗口的“操作结果”部分。单击“Clear”按钮，会清空“操作结果”部分的提示。单击“Cancel”按钮，会关闭调酒器窗口。

这里对窗口中的控件进行简要说明，如表 9-6 所示。

表 9-6

控件类型	控件名称	作 用
QScrollBar	tequilaScrollBar	显示龙舌兰酒的滑动条，连接 uiScrollBarValueChanged 函数的绑定。触发控件信号 valueChanged 的发射
QLabel	selScrollBarLbl	显示选择导入的龙舌兰酒的当前值
QSpinBox	tripleSecSpinBox	显示三重蒸馏酒的文本框
QLineEdit	limeJuiceLineEdit	显示柠檬汁的文本框
QSlider	iceHorizontalSlider	显示冰块的滑动条，连接 uiIceSliderValueChanged 函数的绑定。触发控件信号 valueChanged 的发射
QRadioButton	SpeedButton1	显示搅拌速度的单选钮，速度为 Mix
QRadioButton	SpeedButton2	显示搅拌速度的单选钮，速度为 Whip
QRadioButton	SpeedButton3	显示搅拌速度的单选钮，速度为 Puree
QRadioButton	SpeedButton4	显示搅拌速度的单选钮，速度为 Chop
QRadioButton	SpeedButton5	显示搅拌速度的单选钮，速度为 Karate Chop
QRadioButton	SpeedButton6	显示搅拌速度的单选钮，速度为 Beat
QRadioButton	SpeedButton7	显示搅拌速度的单选钮，速度为 Smash
QRadioButton	SpeedButton8	显示搅拌速度的单选，速度为 Liquefy
QRadioButton	SpeedButton9	显示搅拌速度的单选钮，速度为 Vaporize
QPushButton	okBtn	单击“OK”按钮，连接 uiAccept 函数的绑定。触发控件信号 clicked 的发射
QPushButton	clearBtn	单击“Clear”按钮，连接 uiClear 函数的绑定。触发控件信号 clicked 的发射
QPushButton	cancelBtn	单击“Cancel”按钮，连接 uiReject 函数的绑定。触发控件信号 clicked 的发射
QTextEdit	resultText	显示所调制的鸡尾酒的配置结果和搅拌速度

这里对在信号/槽编辑器中定义的信号和槽进行简要说明，如表 9-7 所示。

表 9-7

发 射 者	信 号	槽	作 用
tequilaScrollBar	valueChanged(int)	uiScrollBarValueChanged()	当改变导入龙舌兰酒的滑动条时，发射这个信号
okBtn	clicked()	uiAccept()	当单击“OK”按钮时，发射这个信号
iceHorizontalSlider	valueChanged(int)	uiIceSliderValueChanged()	当用户改变冰块滑块时，发射这个信号
clearBtn	clicked()	uiClear()	当用户单击“Clear”按钮时，发射这个信号
cancelBtn	clicked()	uiReject()	当用户单击“Cancel”按钮时，发射这个信号

### X.3.2 启动测试程序

需要使用如下命令把 MatrixWinUi.ui 文件转换为 MatrixWinUi.py 文件。如果命令执行成功，在 MatrixWinUi.ui 的同级目录下就会生成一个同名的.py 文件。

```
pyside6-uic.exe -o MatrixWinUi.py MatrixWinUi.ui
```

为了实现调用代码与界面的分离，需要新建一个文件 MatrixWinUiRun.py，直接继承界面类和主窗口类，该文件保存在 Chapter08/testCase 目录下，其完整代码如下：

```
import sys
from PySide6.QtWidgets import *
from MatrixWinUi import *

class CallMatrixWinUi(QWidget):
    def __init__(self, parent=None):
        super(CallMatrixWinUi, self).__init__(parent)
        self.ui = Ui_MatrixWin()
        self.ui.setupUi(self)
        self.initUi()

    # 初始化窗口
    def initUi(self):
        scrollVal = self.ui.tequilaScrollBar.value()
        self.ui.selScrollBarLbl.setText(str(scrollVal))
        sliderVal = self.ui.iceHorizontalSlider.value()
        self.ui.selIceSliderLbl.setText(str(sliderVal))

    # 获得一量杯酒的重量，单位：克
    def getJiggers(self):
```

```

# 返回玛格丽特就得总容量, 以 jigger 量酒器为单位。
# 一个量酒器可以容纳 0.0444 升的酒。
jiggersTequila = self.ui.tequilaScrollBar.value()
jiggersTripleSec = self.ui.tripleSecSpinBox.value()
jiggersLimeJuice = float(self.ui.limeJuiceLineEdit.text())
jiggersIce = self.ui.iceHorizontalSlider.value()
return jiggersTequila + jiggersTripleSec + jiggersLimeJuice +
jiggersIce

# 获得一量杯酒的体积, 单位: 升
def getLiters(self):
    '''返回鸡尾酒的总容量(升)'''
    return 0.0444 * self.getJiggers()

# 获得搅拌速度
def getSpeedName(self):
    speedButton = self.ui.speedButtonGroup.checkedButton()
    if speedButton is None:
        return None
    return speedButton.text()

# 点击 ok 按钮后, 把响应的结果显示在 resultText 文本框里
def uiAccept(self):
    print('* CallMatrixWinUi accept ')
    print('The volume of drinks is {0} liters ({1}
jiggers).'.format(self.getLiters() , self.getJiggers() ))
    print('The blender is running at speed
"{0}"'.format(self.getSpeedName() ))
    msg1 = '饮料量为: {0} 升 ({1} 个量酒器)'.format(self.getLiters() ,
self.getJiggers() )
    msg2 = '调酒器的搅拌速度是: "{0}"'.format(self.getSpeedName() )
    self.ui.resultText.clear()
    self.ui.resultText.append(msg1)
    self.ui.resultText.append(msg2)

# 点击 cancel 按钮, 关闭窗口
def uiReject(self):
    print('* CallMatrixWinUi reject ')
    '''Cancel.'''
    self.close()

# 点击 clear 按钮, 清空操作结果

```

```

def uiClear(self):
    print('* CallMatrixWinUi uiClear ')
    self.ui.resultText.clear()

def uiScrollBarValueChanged(self):
    print('* uiScrollBarValueChanged -----')
    pos = self.ui.tequilaScrollBar.value()
    self.ui.selScrollBarLbl.setText( str(pos) )

def uiIceSliderValueChanged( self):
    print('* uiIceSliderValueChanged -----')
    pos = self.ui.iceHorizontalSlider.value()
    self.ui.selIceSliderLbl.setText( str(pos) )

if __name__=="__main__":
    app = QApplication(sys.argv)
    demo = CallMatrixWinUi()
    demo.show()
    sys.exit(app.exec())

```

运行脚本，显示效果如图 9-52 所示。

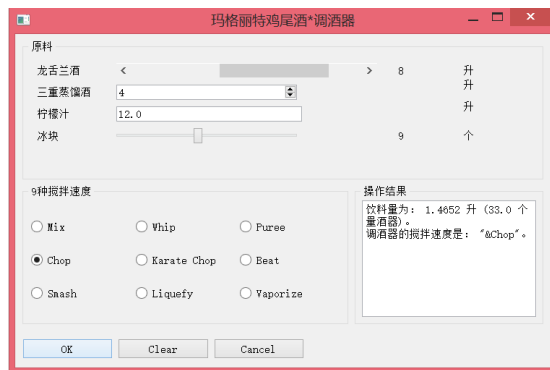


图 9-52

### X.3.3 单元测试程序

接下来要对窗口类 `MatrixWinUiRun` 编写单元测试程序。针对这个窗口编写单元测试类 `MatrixWinTest.py`。本例文件名为 `appendix\X\testCase\MatrixWinTest.py`，由于代码比较多，下面一步步进行介绍。



## 1. 编写单元测试类

编写单元测试类 `MatrixWinTest` 使用的是标准 Python 中的 `unittest` 模块，它是 Python 内部自带的一个单元测试模块。

(1) 首先导入 `unittest` 模块。

```
import unittest
```

(2) 定义一个继承自 `unittest.TestCase` 的测试用例类 `MatrixWinTest`。

(3) 定义 `setUp()` 和 `tearDown()`，在每个测试用例的前后做一些辅助工作。使用 `setUp()` 方法调用测试类之前的初始化工作；使用 `tearDown()` 方法调用测试类之后的清理工作。

(4) 定义测试用例，名字以 `test` 开头，比如 `test_moveScrollBar()` 和 `test_tripleSecSpinBox()`。

(5) 一个测试用例应该只测试一个方面，测试目的和测试内容很明确，主要是调用 `assertEqual`、`assertRaises` 等断言方法判断程序执行结果和预期值是否相符。如果测试未通过，则会输出相应的错误提示。

(6) 调用 `unittest.main()` 启动测试。

常用的断言方法如表 9-8 所示。

表 9-8

断言方法	说 明
<code>assertEqual(a, b)</code>	检测 <code>a==b</code>
<code>assertNotEqual(a,b )</code>	检测 <code>a!=b</code>
<code>assertTrue(x)</code>	检测 <code>bool(x) is True</code>
<code>assertFalse(x )</code>	检测 <code>bool(y) is False</code>
<code>assertIsNot(a, b)</code>	检测 <code>a is not b</code>

所编写的单元测试类 `MatrixWinTest` 需要继承自 `unittest.TestCase` 类，通过 `unittest` 模块编写测试业务，其完整代码如下：

```
class MatrixWinTest(unittest.TestCase):
    # 初始化工作
    def setUp(self):
        print('*** setUp ***')
        self.app = QApplication(sys.argv)
        self.form = CallMatrixWinUi.CallMatrixWinUi()
        self.form.show()

    # 退出清理工作
    def tearDown(self):
```

```
print('*** tearDown ***')
self.app.exec_()
```

## 2. 定时切换窗口

正常情况下测试运行速度会非常快，一闪而过，如果我们想在窗口上看下测试的效果，需要让窗口暂停。我们使用 `time.sleep()` 方法让窗口暂停一下，但是这样会造成窗口的阻塞，结果会卡死，什么也看不到。解决方案是使用 `QApplication.processEvents()` 强制刷新窗口，这点我们在多线程相关章节有更详细的介绍。代码如下，我们在测试窗口运行之前（`setUp`）以及之后（`tearDown`）都暂停 1 秒钟，并强制窗口刷新，方便我们查看测试效果。

```
class MatrixWinTest(unittest.TestCase):
    # 初始化工作
    def setUp(self):
        print('*** setUp ' + '=' * 20 + self._testMethodName)
        self.form = MatrixWinUiRun.CallMatrixWinUi()
        self.form.show()

        QApplication.processEvents()
        time.sleep(1)

    # 退出清理工作
    def tearDown(self):
        print('*** tearDown ' + '=' * 20 + self._testMethodName)
        time.sleep(1)
```

这样产生的效果是每 2 秒完成一个测试，我们有足够的时间查看测试的变化情况。

## 3. 测试调酒器窗口的默认值

测试设置“原料”部分每一个控件的默认值，以及“9 种搅拌速度”部分的单选钮（`QRadioButton`），当全部控件默认设置完毕后单击“OK”按钮，把结果显示在“操作结果”部分。

```
# 测试用例——在默认状态下测试 GUI
def test_defaults(self):
    '''测试 GUI 处于默认状态'''
    print('*** testCase test_defaults begin ***')
    self.form.setWindowTitle('开始测试用例 test_defaults ')

    self.assertEqual(self.form.ui.tequilaScrollBar.value(), 8)
```

```

self.assertEqual(self.form.ui.tripleSecSpinBox.value(), 4)
self.assertEqual(self.form.ui.limeJuiceLineEdit.text(), "12.0")
self.assertEqual(self.form.ui.iceHorizontalSlider.value(), 12)
self.assertEqual(self.form.ui.speedButtonGroup.
checkedButton().text(), "&Karate Chop")
print('*** speedName='+ self.form.getSpeedName() )

# 用鼠标左键单击“OK”按钮
okWidget = self.form.ui.okBtn
QTest.mouseClick(okWidget, Qt.LeftButton)

#测试窗口在默认状态下，各控件的默认值是否与预期值一样
self.assertEqual(self.form.getJiggers() , 36.0)
self.assertEqual(self.form.getSpeedName(), "&Karate Chop")
print('*** testCase test_defaults end ***')

```



注意

在测试例子中，QTest.mouseClick()用于实际单击“OK”按钮。

运行测试用例，显示的窗口如图 9-53 所示。

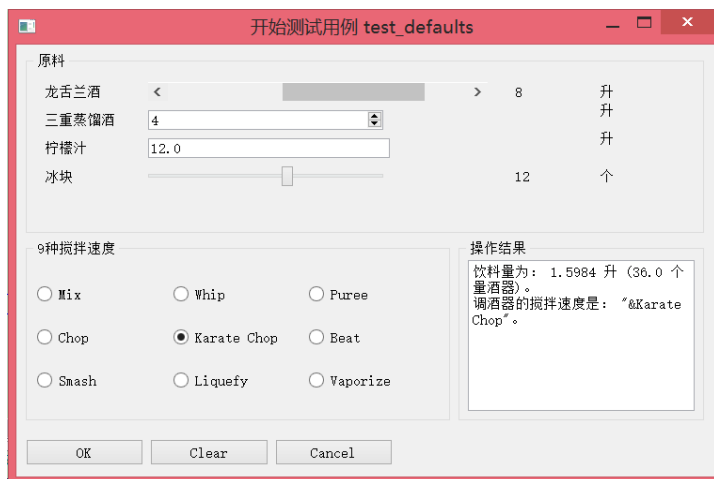


图 9-53

#### 4. 测试 QScrollBar

在运行测试用例前，将窗口中原料对应的所有控件中的所有成分数值设置为0，将待测试控件的成分设置为非零值（12 或-1）。比如将龙舌兰酒对应的滑动条控件的值超过它的合法范围，设置为 12，而在 UI 文件中实际它的最大值为 11、最小值为 0，查看窗口是否能正常运行，如图 9-54 所示。

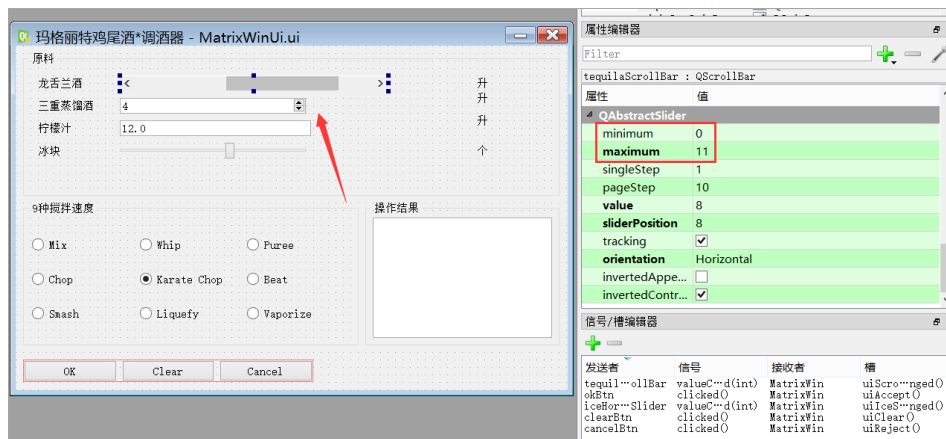


图 9-54

为方便起见，setFormToZero()将所有控件的值都设置为 0。

# 设置窗口中所有控件的值为 0，状态设置为初始状态。

```
def setFormToZero(self):
    print('* setFormToZero *')
    self.form.ui.tequilaScrollBar.setValue(0)
    self.form.ui.tripleSecSpinBox.setValue(0)
    self.form.ui.limeJuiceLineEdit.setText("0.0")
    self.form.ui.iceHorizontalSlider.setValue(0)

    self.form.ui.selScrollBarLbl.setText("0")
    self.form.ui.selIceSliderLbl.setText("0")
```

接下来测试使用滑动条控件来模拟往调酒器里倒入多少升龙舌兰酒。测试设置滑动条的最小值和最大值，然后尝试合法值，观察滑动条能否正常使用。

# 测试用例——测试滑动条

```
def test_moveScrollBar(self):
    '''测试用例 test_moveScrollBar'''
    print('*** testCase test_moveScrollBar begin ***')
    self.form.setWindowTitle('开始测试用例 test_moveScrollBar ')
    self.setFormToZero()

    # 测试将龙舌兰酒的滑动条的值设置为 12，在 UI 中实际它的最大值为 11
    self.form.ui.tequilaScrollBar.setValue( 12 )
    print('* 当执行 self.form.ui.tequilaScrollBar.setValue(12) 后,
    ui.tequilaScrollBar.value() => ' +
    str( self.form.ui.tequilaScrollBar.value() ) )
    self.assertEqual(self.form.ui.tequilaScrollBar.value(), 11 )
```

```

# 测试将龙舌兰酒的滑动条的值设置为 -1, 在 UI 中实际它的最小值为 0
self.form.ui.tequilaScrollBar.setValue(-1)
print('* 当执行 self.form.ui.tequilaScrollBar.setValue(-1) 后,
ui.tequilaScrollBar.value() => ' +
str( self.form.ui.tequilaScrollBar.value() ) )
self.assertEqual(self.form.ui.tequilaScrollBar.value(), 0)

# 重新将龙舌兰酒的滑动条的值设置 5
self.form.ui.tequilaScrollBar.setValue(5)

# 用鼠标左键单击“OK”按钮
okWidget = self.form.ui.okBtn
QTest.mouseClick(okWidget, Qt.LeftButton)
self.assertEqual(self.form.getJiggers() , 5)
print('*** testCase test_moveScrollBar end ***')

```

运行测试用例，显示的窗口如图 9-55 所示。



图 9-55

## 5. 测试 QSpinBox

将三重蒸馏酒对应的计数器控件（QSpinBox）的数值设置为非零值（12 或-1），在 UI 文件中计数器控件的最大值为 11、最小值为 0，通过测试用例设置控件的最小值和最大值，然后尝试设置合法值，最后验证结果，如图 9-56 所示。

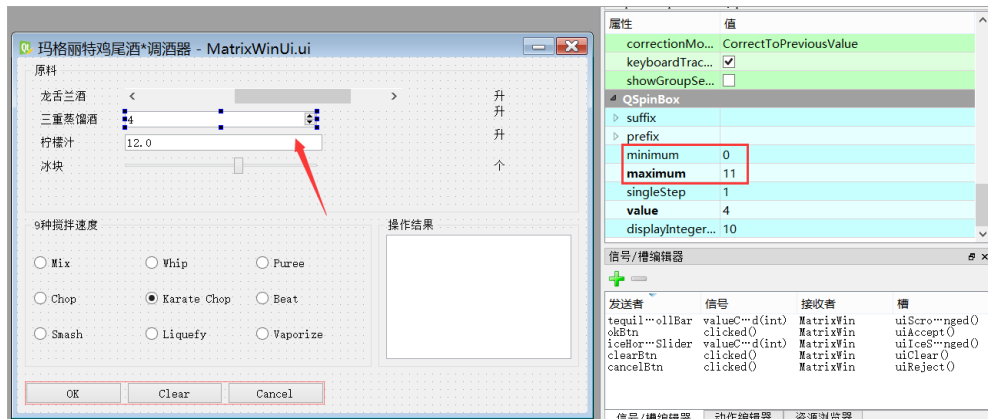


图 9-56

```
# 测试用例——测试计数器控件 (QSpinBox)
def test_tripleSecSpinBox(self):
    '''测试用例 test_tripleSecSpinBox '''
    print('*** testCase test_tripleSecSpinBox begin ***')
    self.form.setWindowTitle('开始测试用例 test_tripleSecSpinBox ')
    '''测试修改计数器控件 (QSpinBox) 的最大值、最小值
        测试它的最小值和最大值作为读者的练习
    '''
    self.setFormToZero()
    # tripleSecSpinBox 在界面中的取值范围为 0~11, 将它的最大值设置为 12, 看是否显示正常
    self.form.ui.tripleSecSpinBox.setValue(12)
    print('* 当执行 self.form.ui.tripleSecSpinBox.setValue(12) 后,
    ui.tripleSecSpinBox.value() => ' +
    str( self.form.ui.tripleSecSpinBox.value() ) )
    self.assertEqual(self.form.ui.tripleSecSpinBox.value(), 11 )

    # tripleSecSpinBox 在界面中的取值范围为 0~11, 将它的最小值设置为 -1, 看是否显示正常
    self.form.ui.tripleSecSpinBox.setValue(-1)
    print('* 当执行 self.form.ui.tripleSecSpinBox.setValue(-1) 后,
    ui.tripleSecSpinBox.value() => ' +
    str( self.form.ui.tripleSecSpinBox.value() ) )
    self.assertEqual(self.form.ui.tripleSecSpinBox.value(), 0 )

    self.form.ui.tripleSecSpinBox.setValue(2)

    # 用鼠标左键单击“OK”按钮
    okWidget = self.form.ui.okBtn
    QTest.mouseClick(okWidget, Qt.LeftButton)
```

```
self.assertEqual(self.form.getJiggers(), 2)
print('*** testCase test_tripleSecSpinBox end ***')
```

运行测试用例，显示的窗口如图 9-57 所示。

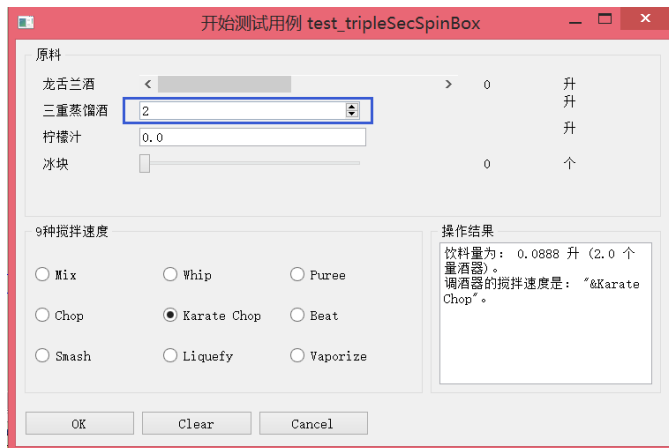


图 9-57

## 6. 测试 QLineEdit

使用 `QTest.keyClicks()` 实际输入一个字符串到 `limeJuiceLineEdit` 文本框控件中，在本例中是将字符串 "3.5" 输入到 `limeJuiceLineEdit` 文本框控件中。这里使用 `QTest.keyClicks()`，是因为本文强调 `QtTest`。如果使用 `QLineEdit.setText()` 直接设置控件文本，测试结果也是一样的。

### # 测试用例——测试柠檬汁单行文本框

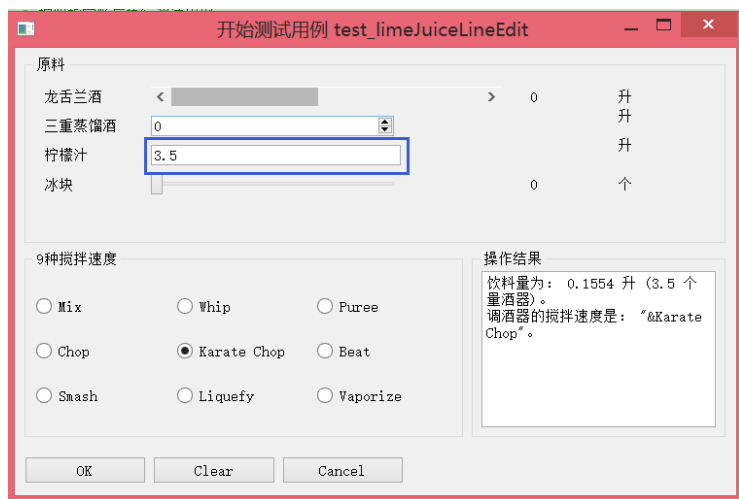
```
def test_limeJuiceLineEdit(self):
    '''测试用例 test_limeJuiceLineEdit '''
    print('*** testCase test_limeJuiceLineEdit begin ***')
    self.form.setWindowTitle('开始测试用例 test_limeJuiceLineEdit ')

    '''测试修改 lineEdit 文本框控件的最大值、最小值
    测试它的最小值和最大值作为读者的练习
    '''
    self.setFormToZero()
    # 清除 lineEdit 文本框控件值，然后在 lineEdit 文本框控件中输入"3.5"
    self.form.ui.limeJuiceLineEdit.clear()
    QTest.keyClicks(self.form.ui.limeJuiceLineEdit, "3.5")

    # 用鼠标左键单击"OK"按钮
    okWidget = self.form.ui.okBtn
    QTest.mouseClick(okWidget, Qt.LeftButton)
    self.assertEqual(self.form.getJiggers(), 3.5)
```

```
print('*** testCase test_limeJuiceLineEdit end ***')
```

运行测试用例，显示的窗口如图 9-58 所示。





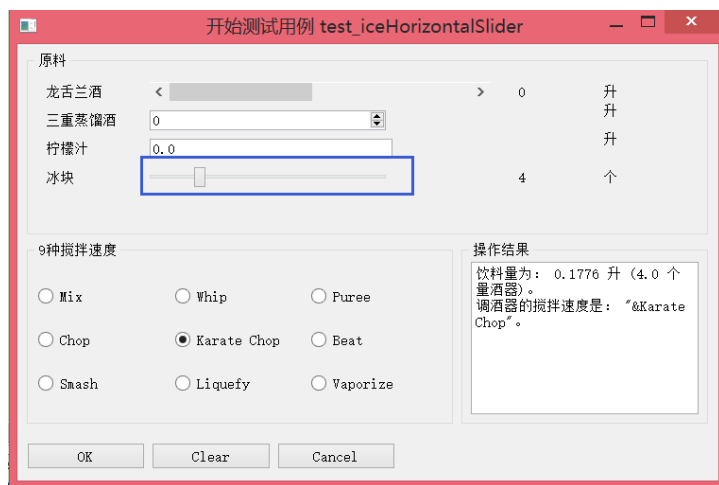


图 9-59

## 8. 测试 QRadioButton

在“9 种搅拌速度”部分包含了 9 个单选按钮，测试选中每个单选按钮，取得的值是否和单选按钮的文本一致。比如搅拌速度的文本显示的是'&Mix'，则意味着按住“Alt 键+M”快捷键，可以快速定位到“Mix”单选按钮。

### # 测试用例——测试搅拌速度单选按钮

```
def test_blenderSpeedButtons(self):
    print('*** testCase test_blenderSpeedButtons begin ***')
    self.form.ui.speedButton1.click()
    self.assertEqual(self.form.getSpeedName(), "&Mix")
    self.form.ui.speedButton2.click()
    self.assertEqual(self.form.getSpeedName(), "&Whip")
    self.form.ui.speedButton3.click()
    self.assertEqual(self.form.getSpeedName(), "&Puree")
    self.form.ui.speedButton4.click()
    self.assertEqual(self.form.getSpeedName(), "&Chop")
    self.form.ui.speedButton5.click()
    self.assertEqual(self.form.getSpeedName(), "&Karate Chop")
    self.form.ui.speedButton6.click()
    self.assertEqual(self.form.getSpeedName(), "&Beat")
    self.form.ui.speedButton7.click()
    self.assertEqual(self.form.getSpeedName(), "&Smash")
    self.form.ui.speedButton8.click()
    self.assertEqual(self.form.getSpeedName(), "&Liquefy")
    self.form.ui.speedButton9.click()
    self.assertEqual(self.form.getSpeedName(), "&Vaporize")
    print('*** testCase test_blenderSpeedButtons end ***')
```

运行测试用例，显示的窗口如图 9-60 所示。

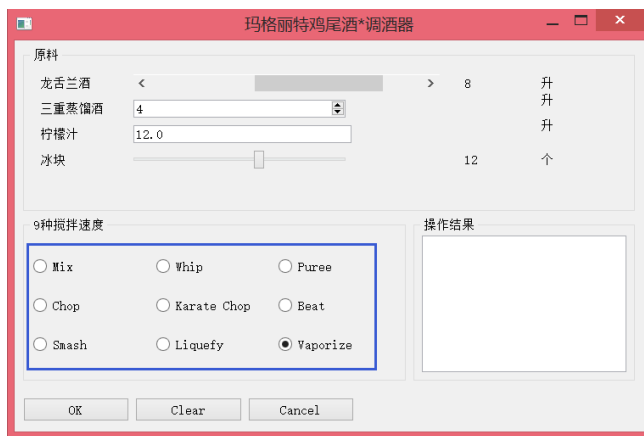


图 9-60

### X.3.4 运行测试用例

这一节讲解运行单元测试类 `MatrixWinTest`。运行单元测试类有两种方法：一种是默认执行所有的测试用例；另一种是按照指定顺序执行测试用例。

首先需要导入 `unittest` 模块。

```
import unittest
```

#### 1. 默认执行所有的测试用例

默认执行所有的测试用例，其核心代码如下：

```
if __name__ == "__main__":
    unittest.main()
```

使用以下命令，将测试结果报告保存到 `reportLog.txt` 日志文件中。

```
python MatrixWinTest.py >> ./reportLog.txt 2>&1
```

#### 2. 按照指定顺序执行测试用例

为了更加灵活地执行测试用例，可以按照指定顺序进行。其核心代码如下：

```
if __name__ == "__main__":
    suite = unittest.TestSuite()
    suite.addTest(MatrixWinTest("test_defaults"))
    suite.addTest(MatrixWinTest("test_moveScrollBar"))
    suite.addTest(MatrixWinTest("test_tripleSecSpinBox"))
    suite.addTest(MatrixWinTest("test_limeJuiceLineEdit"))
```

```
suite.addTest(MatrixWinTest("test_iceHorizontalSlider"))
suite.addTest(MatrixWinTest("test_liters"))
suite.addTest(MatrixWinTest("test_blenderSpeedButtons"))
runner = unittest.TextTestRunner()
runner.run(suite)
```

### X.3.5 生成测试报告

虽然可以通过 `unittest` 生成测试日志，但是把测试结果汇总到测试报告里，需要手工完成，这样又浪费了人力，有没有更简单的方法呢？

有的，可以使用 `HTMLTestRunner` 生成测试报告。

`HTMLTestRunner` 是 Python 标准库 `unittest` 模块的一个扩展库，使用它可以生成易于使用的 HTML 测试报告。

(1) 下载 `HTMLTestRunner.py` 文件，地址是：<http://tungwaiyip.info/software/HTMLTestRunner.html>，如图 9-61 所示（目前官网已经关闭）。

`HTMLTestRunner` 现在只支持 Python 2 环境，需要将 `HTMLTestRunner` 修改成支持 Python 3 版本，笔者参考了热心网友的博文 <http://www.cnblogs.com/sgtb/p/4169732.html>，修改 `HTMLTestRunner.py` 的源码让它支持 Python 3 环境。



图 9-61

支持 Python 3 环境的 `HTMLTestRunner.py` 文件位于 `Chapter08/testCase` 目录下，请读者自行下载。

(2) 生成测试报告

本例文件名为 `appendix\X\testCase\RunTestCase.py`，演示测试用例代码 `HTMLTestRunner` 整合测试报告。其完整代码如下：

```
import unittest
```

```

import HTMLTestRunner
import time
from MatrixWinTest import MatrixWinTest
from PySide6.QtWidgets import QApplication
import sys

if __name__ == "__main__":
    app = QApplication(sys.argv)
    now = time.strftime("%Y-%m-%d-%H_%M_%S", time.localtime(time.time()))
    print(now)
    testunit = unittest.TestSuite()
    testunit.addTest(unittest.makeSuite(MatrixWinTest))

    htmlFile = ".\\" + now + "HTMLtemplate.html"
    print('htmlFile=' + htmlFile)
    fp = open(htmlFile, 'wb')
    runner = HTMLTestRunner.HTMLTestRunner(
        stream=fp,
        title=u"Qt for Python 测试报告",
        description=u"用例测试情况")
    runner.run(testunit)
    app.exec()
    fp.close()

```

运行脚本后，会生成测试报告，测试报告的格式为：{当前日期-当前时间}HTMLtemplate.html，例如 2022-03-17-21\_38\_00HTMLtemplate.html，显示效果如图 9-62 所示。可以看到我们的 7 个测试结果都为 pass，测试全部通过。

#### Qt for Python测试报告

Start Time: 2022-03-17 21:38:00

Duration: 0:00:14.416826

Status: Pass 7

用例测试情况

Show [Summary](#) [Failed](#) [All](#)

Test Group/Test case	Count	Pass	Fail	Error	View
MatrixWinTest.MatrixWinTest	7	7	0	0	<a href="#">Detail</a>
test_blenderSpeedButtons			pass		
test_defaults: 测试GUI处于默认状态			pass		
test_iceHorizontalSlider: 测试用例 test_iceHorizontalSlider			pass		
test_limeJuiceLineEdit: 测试用例 test_limeJuiceLineEdit			pass		
test_liters: 测试用例 test_liters			pass		
test_moveScrollBar: 测试用例test_moveScrollBar			pass		
test_tripleSecSpinBox: 测试用例 test_tripleSecSpinBox			pass		
<b>Total</b>	<b>7</b>	<b>7</b>	<b>0</b>	<b>0</b>	

图 9-62

## X.4 金融数据显示

我们使用 Qt Designer 布局的时候需要涉及到局部布局以及全局布局，这里把它们称之为控件级别的布局管理和窗口级别的布局管理。

X.4.1-X.4.3 节案例对应 ui 文件为 appendix\X\fundDemo\fundFOF.ui，主要讲述如何使用 Qt Designer 设计一个窗口以及展示一些数据。

### X.4.1 控件级别的布局管理

首先打开 Qt Designer，新建一个窗口，并先后向里面添加一个 Scroll Area、两个 Widget 和一个 Tab 控件，然后把其他控件拖入 Scroll Area 中。

对于其中的一个 Widget，设置提升的窗口类，如图 11-1 所示；对于另一个 QWidget，设置对象名为“widget\_parameter\_tree”。

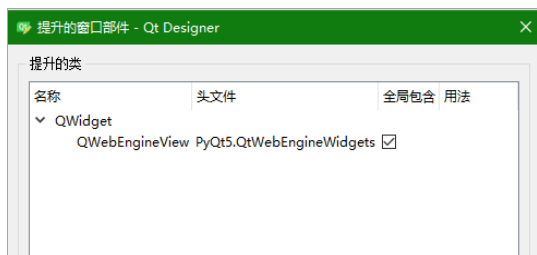


图 11-1

对于 Tab 控件，设置三个标签并分别命名为：月度收益，区间收益和回撤情况。然后在这三个标签中添加 QWidget，并设置提升的窗口类为 QWebEngineView。

最后对所有的 QWebEngineView 类修改样式，选中它并单击鼠标右键，从弹出的快捷菜单中选择“修改样式表”→“添加颜色”，background-color 选择灰色。这样做只是为了更好地区分 QWidget 和 QWebEngineView，如果你觉得比较麻烦或者不想这样做，也可以忽略这一步操作。初步效果如图 11-2 所示。

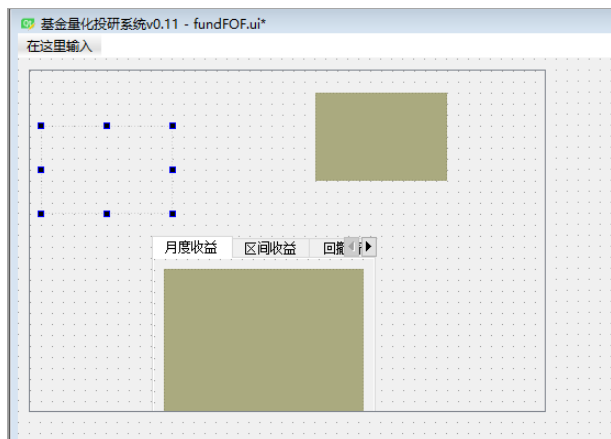


图 11-2



在窗口左边用鼠标选中的控件是 `widget_parameter_tree`, 其他控件都很容易识别出来。

接下来进行布局管理。

- (1) 设置 `tabWidget` 的 `minimumsize` 属性, 高度与宽度都为 200。
- (2) 选中 `tabWidget` 的每一个页, 然后单击鼠标右键, 从弹出的快捷菜单中选择“布局”→“水平布局”(也可以选择“垂直布局”)。
- (3) 选中 `tabWidget` 和上面的 `QWebEngineView`, 然后单击鼠标右键, 从弹出的快捷菜单中选择“布局”→“垂直布局”。

这样设置之后, 你会发现 `QWebEngineView` 不见了。设置 `QWebEngineView` 的 `minimumsize` 属性, 最小高度为 100, `QWebEngineView` 就可以看到了。

- (4) 选中上面的垂直布局管理器和 `widget_parameter_tree`, 然后单击鼠标右键, 从弹出的快捷菜单中选择“布局”→“水平布局”。



如果你不知道如何选择这个布局管理器, 则可以参考图 11-3。

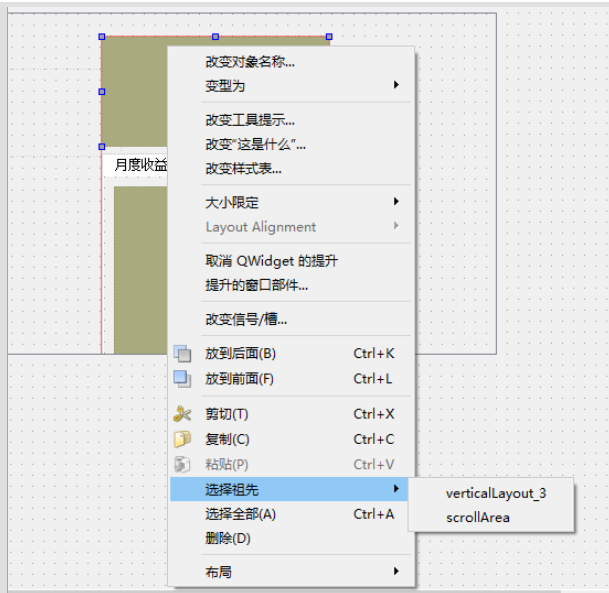


图 11-3

或者单击右上角的对象查看器进行选择操作，如图 11-4 所示。

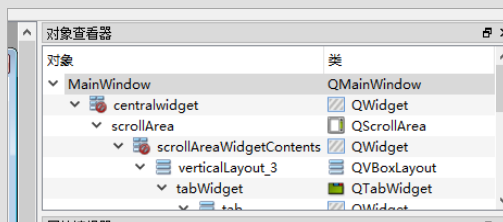


图 11-4

这时发现 widget\_parameter\_tree 又不见了，需要设置其最小宽度为 100。

(5) 单击 scrollArea，然后单击鼠标右键，从弹出的快捷菜单中选择“布局”→“水平布局”（也可以选择“垂直布局”），效果如图 11-5 所示。

至此，我们就完成了控件级别的布局管理。

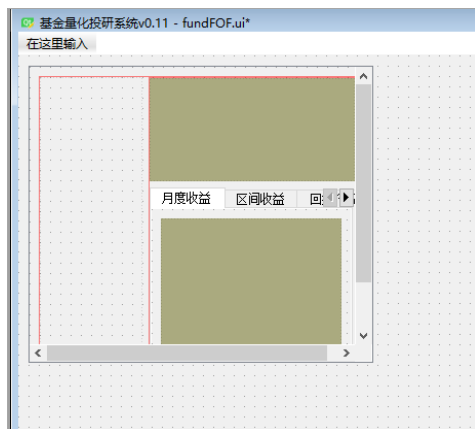


图 11-5

## X.4.2 窗口级别的布局管理

窗口级别的布局管理，就是指整个窗口的布局由布局管理器接管，随着窗口的缩放里面的控件也会跟着缩放。

窗口级别的布局管理非常简单，相对于控件级别的布局管理仅仅多了一两个步骤而已。

单击窗口的空白处，也就是选中 **MainWindow**，然后单击鼠标右键，从弹出的快捷菜单中选择“布局”→“水平布局”，效果如图 11-6 所示。

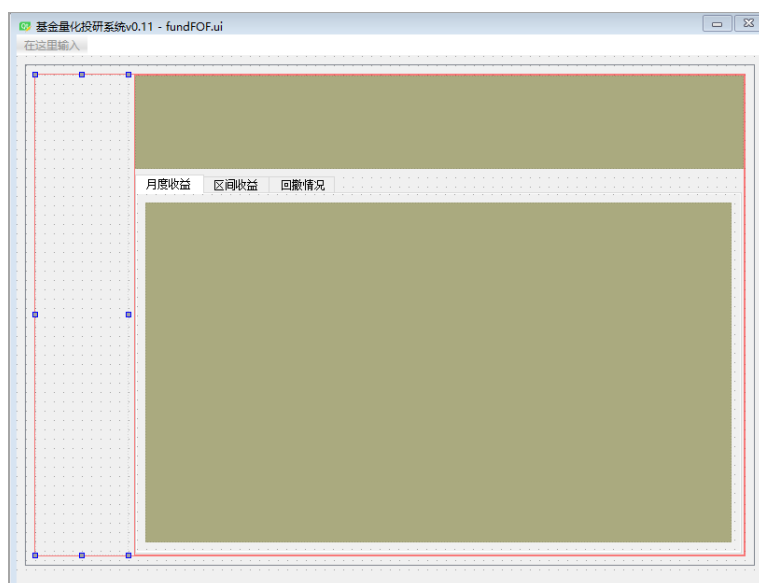




图 11-6

通过预览，发现整个布局会随着窗口大小的变化而变化。至此，我们的目的就达到了。

**注意**

实际上，对于布局管理，并没有窗口级别的布局管理和控件级别的布局管理这两个概念。之所以提出这两个概念，是因为有些人不知道如何用布局管理器接管窗口，导致设计出来的控件不能随着窗口大小的调整而自动缩放。

在这里 `scrollArea` 没有显示出滚动条，是因为控件的高度还不够高，设置 `widget_parameter_tree` 的最小高度为 1000，就可以看到滚动条了，如图 11-7 所示。

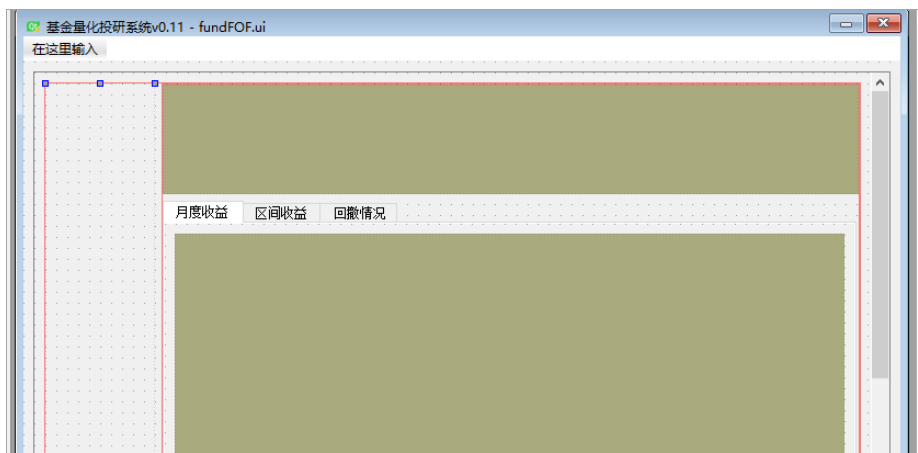


图 11-7

### X.4.3 显示产品基本信息

对于私募基金公司来说，不管公司大小，都需要拥有自己的投资研究系统，对于 FOF 型基金来说更是如此。所谓 FOF 型基金，就是指私募基金公司把钱投向表现比较好的其他私募公司的基金产品，把这些基金产品进行打包，就组成了自己的基金产品。因此，对于 FoF 私募公司来说，其投资研究系统比较重视市场上存在的基金产品信息。本节的主要目的就是教会大家如何用 PySide/PyQt 去呈现某种基金产品的信息。这里以从公开网络上获取一种基金产品的数据，如私募排排网的基金产品：千石资本 - 和聚光明 1 号资产管理计划为例，网址为：<http://dc.simuwang.com/product/HF00000XEG.html>。需要注意的是，打开这个网址需要注册，如果你不想注册的话，我们已经另存了一份后缀为 .mhtml 的离线文件，只

需使用浏览器打开即可。

首先,根据上一章的“扩展应用”内容,我们这里添加一个新成员: **ParameterTree** 类。这个类实际上是使用 **pyqtgraph** 改写的 **QTreeWidget** 的子类。但是相对于 **QTreeWidget**, **ParameterTree** 的表现更美观而且更具有实用价值。下面的示例中会用到 **ParameterTree** 类。

这一节的示例使用上一节设计的结果,我们先看一下示例结果,如图 11-8 所示。

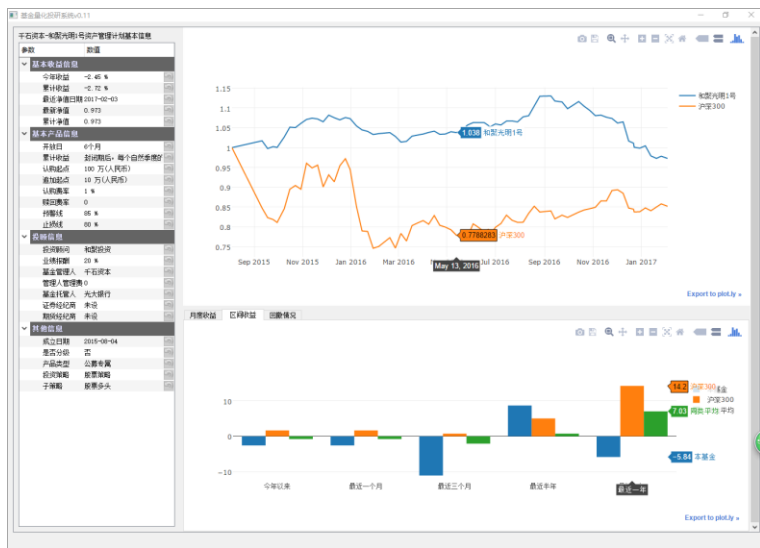


图 11-8

从图 11-8 可以看出,左侧使用 **ParameterTree** (**QTreeWidget** 的子类) 来显示产品基本信息,右侧从上到下使用四个 **QWebEngineView** 类来显示产品净值与沪深 300 对比图、月度收益图、区间收益图、回撤情况图。本节将重点介绍 **ParameterTree** 的使用方法。

我们发现图 11-8 所示的图片的大小与上一节设计的大小不一样,这是因为在窗口初始化时手动调整了窗口控件的大小。代码如下(本例文件名为 **appendix\X\fundDemo\fundFOF.py**):

```
def __init__(self, parent=None):
    """
    Constructor

    @param parent reference to the parent widget
    @type QWidget
```

```

"""

super(MainWindow, self).__init__(parent)

self.setupUi(self)

self.plotly_Qt = Plotly_Qt()

'''手动调整窗口控件的大小，使之看起来更美观'''

self.widget_parameter_tree.setMaximumWidth(300)
self.widget_parameter_tree.setMinimumWidth(200)
self.QWebEngineView_ProductVsHs300.setMinimumHeight(500)
self.tabWidget.setMinimumHeight(400)

```

上面的代码非常直观，这里就不解释了。下面看 **ParameterTree** 的使用方法。

'''显示 parametertree，这里通过布局管理器把 ParameterTree 间接地嵌套到 Widget 窗口中'''

```

from mypyqtgraph import p
from pyqtgraph.parametertree import ParameterTree

t = ParameterTree()
t.setParameters(p, showTop=False)
t.setHeaderLabels(["参数", "数值"])
# t.setWindowTitle('pyqtgraph example: Parameter Tree')
layout =QGridLayout()
self.widget_parameter_tree.setLayout(layout)
layout.addWidget(
    QLabel("千石资本-和聚光明 1 号资产管理计划基本信息"), 0, 0, 1, 1)
layout.addWidget(t)

```

我们看到，这里的代码没有说明如何使用数据创建成树。关键代码如下：

```

from mypyqtgraph import p
t = ParameterTree()
t.setParameters(p, showTop=False)

```

**mypyqtgraph** 是一个文件，为了使代码简洁、可读性好，我们把它放在另一个文件中了。下面对这个文件进行解读（代码见 `appendix\X\fundDemo\mypyqtgraph.py` 文件）。

首先看传入的部分数据。

```
# 创建参数树的数据

params = [
    {'name': '基本收益信息', 'type': 'group', 'children': [
        {'name': '今年收益', 'type': 'float', 'value': -2.45,
        'siPrefix': True, 'suffix': '%'},
        {'name': '累计收益', 'type': 'float', 'value': -2.72, 'step':
        0.1, 'siPrefix': True, 'suffix': '%'},
        {'name': '最近净值日期', 'type': 'str', 'value': "2017-02-03"},
        {'name': '最新净值', 'type': 'float', 'value': 0.9728, 'step':
        0.01},
        {'name': '累计净值', 'type': 'float', 'value': 0.9728, 'step':
        0.01},
    ]},
    .....
## 创建参数树

p = Parameter.create(name='params', type='group', children=params)
```

结果如图 11-9 所示。



图 11-9

我们看到,参数'type': 'group'对应的是树的分支,参数'type': 'str'对应的是 lineEdit,参数'type': 'float'对应的是 doubleSpinBox。

下面代码的作用是“数值”那一列的任何控件的 value 发生变化时,都会触发 change 函数,并输出变化。

```
## 若树里面的任何内容发生变化,则输出这些变化

def change(param, changes):
    print("tree changes:")
    for param, change, data in changes:
        path = p.childPath(param)
```

```

        if path is not None:
            childName = '.'.join(path)
        else:
            childName = param.name()
        print(' parameter: %s' % childName)
        print(' change:      %s' % change)
        print(' data:        %s' % str(data))
        print(' -----')

p.sigTreeStateChanged.connect(change)

```

下面代码的作用是如果对某一个控件的 **value** 撤销修改, 就会触发 **valueChanging** 函数。

```

def valueChanging(param, value):
    print("Value changing (not finalized):", param, value)

# Too lazy for recursion:
for child in p.children():
    child.sigValueChanging.connect(valueChanging)
    for ch2 in child.children():
        ch2.sigValueChanging.connect(valueChanging)

```

欲了解更多的 **ParameterTree** 使用方法, 可以参考 **pyqtgraph** 的官方示例。运行如下代码, 从中找到 **ParameterTree** 的示例。

```

import pyqtgraph.examples
pyqtgraph.examples.run()

```

接下来介绍几个绘图函数。

```

'''显示绘图函数'''

self.QWebEngineView_ProductVsHs300.load(

QtCore.QUrl.fromLocalFile(self.plotly_Qt.get_plotly_path_product_vs_hs300()))
self.QWebEngineView_LagestBack.load(QtCore.QUrl.fromLocalFile(self.plotly_Qt
.get_plotly_path_lagest_back()))
self.QWebEngineView_PeriodReturn.load(QtCore.QUrl.fromLocalFile(self.plotly_Qt
.get_plotly_path_period_return()))
self.QWebEngineview_MonthReturn.load(QtCore.QUrl.fromLocalFile(self.plotly_Q

```

```
t.get_plotly_path_month_return()))
```

这几个绘图函数的使用方法一样，意思是用 `QWebEngineView` 类实例载入本地 HTML 文件。注意 `plotly_Qt.get_plotly_path_product_vs_hs300()` 返回的是使用 `plotly` 绘图所保存的 HTML 文件路径。

下面对 `plotly_Qt.get_plotly_path_product_vs_hs300()` 代码进行解读（代码见 `appendix\X\fundDemo\Plotly_Qt.py` 文件）。

```
def get_plotly_path_product_vs_hs300(self,
file_name='product_vs_hs300.html'):
    path_plotly = self.path_dir_plotly_html + os.sep + file_name

    data = pd.read_excel(r'data\和聚光明 1 号_hs300_merge.xlsx',
index_col=0)
    data.rename_axis(lambda x: pd.to_datetime(x), inplace=True)
    data.dropna(inplace=True)

    data = [
        go.Scatter(
            x=data.index, # assign x as the dataframe column 'x'
            y=data.cumulative_nav,
            name='和聚光明 1 号'
        ),
        go.Scatter(
            x=data.index, # assign x as the dataframe column 'x'
            y=data.close,
            name='沪深 300'
        )
    ]
    pyof.plot(data, filename=path_plotly, auto_open=False)
    return path_plotly
```

首先获取存储 HTML 文件的路径。

```
path_plotly = self.path_dir_plotly_html + os.sep + file_name
data = pd.read_excel(r'data\和聚光明 1 号_hs300_merge.xlsx',
index_col=0)
```

然后对数据进行处理。下面代码的作用是把 `data` 数据索引设置成日期格式以及去掉 `data` 的缺失值。设置成日期格式的好处是绘图库会自动调整刻度大小，而不是

在坐标轴上呈现出每一天的日期。比如，如果 `data` 的数据跨度是 10 年，那么会以月为最小的时间单位进行绘图，而不是以日为最小的时间单位。

```
data.rename_axis(lambda x: pd.to_datetime(x), inplace=True)
data.dropna(inplace=True)
```

接下来是绘图函数，其中 `pyof.plot` 的参数 `auto_open=False` 表示绘图完成之后不用浏览器显示绘图结果；`filename=path_plotly` 表示将 HTML 文件保存到路径 `path_plotly` 中。

```
data = [
    go.Scatter(
        x=data.index, # assign x as the dataframe column 'x'
        y=data.cumulative_nav,
        name='和聚光明 1 号'
    ),
    go.Scatter(
        x=data.index, # assign x as the dataframe column 'x'
        y=data.close,
        name='沪深 300'
    )
]
pyof.plot(data, filename=path_plotly, auto_open=False)
return path_plotly
```

如果想要呈现更多的产品，则可以稍微修改一下这个示例，把数据源换成自己公司的数据库（前提是数据库中有这些产品信息），这样就具有了一个初步的基金调研框架。如果爬虫能力不错的话，也可以在私募排排网上抓取所需要的产品信息，然后把抓取到的数据与这个示例进行对接，这样你就拥有一个 mini 版本的私募排排网的客户端了。

#### X.4.4 展示产品组合信息

对于机构来说，它们可以从自己的产品信息数据库中筛选出一些优质标的（也就是基金产品）。但是对于投资者来说，他们不像机构那么专业，并不能很准确地识别某一种基金产品的好坏，也没有那么多的时间和精力去市场上寻找适合自己投资的基金产品。一边是投资者想要找出适合自己投资的基金产品，另一边是投资者没有能力找出适合自己投资的基金产品，于是就产生了信息堵塞，其表现就是市场上

的一些信息无法有效传递到投资者手中。而机构正好掌握了大量这种信息，具有信息优势，于是机构可以利用这种信息优势给投资者提供服务，并从中收取服务费用。

服务的方式很简单，对于投资者来说，他们其实并不知道适合自己的产品是什么类型的，但是可以预期自己的风险和收益。于是就产生了另一个问题：给定投资者的预期收益与风险的范围，为他们提供最优的投资方案。考虑到风险分散的原则，这个最优的投资计划最好是几种产品的组合。

我们先看一下示例结果，如图 11-10 所示（涉及文件名为 `combination.py` 等相关文件）。

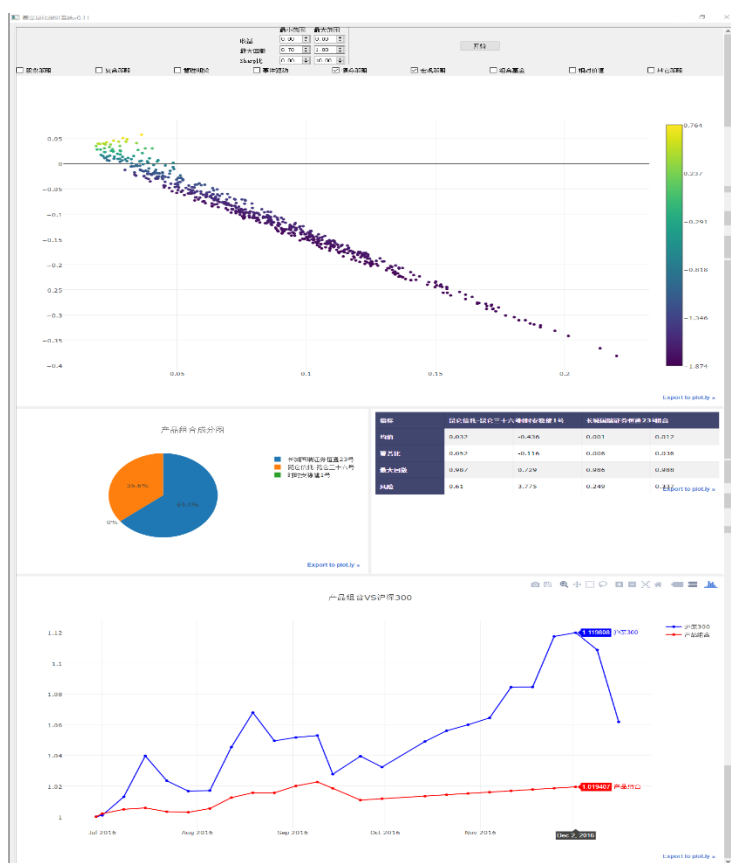


图 11-10

结果分为两个部分，上面部分是让用户选择预期收益、预期风险以及偏好的策略类型，截图如图 11-11 所示。



图 11-11

下面部分是从数据库中选取数据，并通过算法找出最优的产品组合以及各自的权重，然后用图表展现出来。



出于安全性考虑，这里只提供 GUI 的实现逻辑，而不会提供基金产品的数据库，也不会提供找出最优产品组合的算法。

对于使用 Qt Designer 进行界面设计这部分内容，这里不再介绍，读者可以参考 appendix\X\fundDemo\combination.ui 文件自行研究。

本例涉及的代码逻辑并不复杂，当用户输入预期收益和预期风险范围之后，单击“开始”按钮，就会触发唯一的信号槽机制。

```
@Slot()
def on_pushButton_start_combination_clicked(self):
    """
    产品组合分析
    """

    strategy_list = self.check_check_box()
```

`self.check_check_box()` 函数用来检测用户选中的策略是否符合要求，其内容如下，意思是在 `strategy_list` 中放入已经选中的 `checkBox` 的名字。

```
def check_check_box(self):
    strategy_list = []

    for each_checkBox in [self.checkBox_bond, self.checkBox_
        combination_fund, self.checkBox_compound, self.checkBox_event, self.checkBox_
        x_future_manage, self.checkBox_macro, self.checkBox_relative_fund, self.checkBox_
        others, self.checkBox_others]:
        if each_checkBox.isChecked():
            strategy_list.append(each_checkBox.text())

    return strategy_list
```

下面的内容对于有金融背景的人来说，理解起来应该没有问题；若没有金融背景，不能理解，则可以略去这部分内容。

当所选择的产品分类数大于 3 和小于 1 时都不合适，会弹出警告，并视为无效。原因是多个产品分类（大于 3）的组合由于产品之间的多重共线性，会导致分类的模型在数学上没有最优解，即使用计算机暴力算法找出最优解，其权重非零的个数一般也不超过 3 个。所以经过综合考虑，限制产品分类数为 1~3 是最合适的。

```
strategy_list = self.check_check_box()
if len(strategy_list) > 3:
    print('最多选择 3 个策略')
    QMessageBox.information(self, "注意", "最多选择 3 个策略")
    return None

if len(strategy_list) == 0:
    print('最少选择 1 个策略')
    QMessageBox.information(self, "注意", "最少选择 1 个策略")
    return None
```

接下来设置控件的大小和获取参数信息。由于这里并不准备把参数信息导入数据库，所以只是简单地输出参数信息。

```
self.QWebEngineview_Combination_monte_markovitz.setMinimumHeight(800)
self.QWebEngineview_Combination_Pie.setMinimumHeight(400)
self.QWebEngineview_Combination_Table.setMinimumHeight(400)
self.QWebEngineview_Combination_Versus.setMinimumHeight(700)

print('收益_min:', self.doubleSpinBox_returns_min.text())
print('收益_max:', self.doubleSpinBox_returns_max.text())
print('最大回撤_min:', self.doubleSpinBox_maxdrawdown_min.text())
print('最大回撤_max:', self.doubleSpinBox_maxdrawdown_max.text())
print('sharp比_min:', self.doubleSpinBox_sharp_min.text())
print('sharp比_max:', self.doubleSpinBox_sharp_max.text())
```

现在假设我们已经从数据库中找出 3 种组合最优的产品，分别为：昆仑三十六号、时时安稳健 1 号和长城国瑞证券恒通 23 号，它们的组合权重分别为 0.4、0.2 和 0.4。

'''假设已经获取产品组合和权重'''

```
df = pd.read_excel(r'data\组合.xlsx', index_col=[0])

w = [0.4, 0.2, 0.4]

df['组合'] = (df * w).sum(axis=1)
```

最后就是绘图了，代码如下：

```
self.QWebEngineview_Combination_monte_markovitz.load(
    QUrl.fromLocalFile(self.plotly_Qt.get_plotly_path_monte_markovitz(monte_count=600)))

self.QWebEngineview_Combination_Pie.load(
    QUrl.fromLocalFile(self.plotly_Qt.get_plotly_path_combination_pie(df=df, w=w)))

self.QWebEngineview_Combination_Versus.load(
    QUrl.fromLocalFile(self.plotly_Qt.get_plotly_path_combination_versus(df=df, w=w)))

self.QWebEngineview_Combination_Table.load(
    QUrl.fromLocalFile(self.plotly_Qt.get_plotly_path_combination_table(df=df, w=w)))
```

上面的绘图函数 `self.plotly_Qt.get_plotly_path_monte_markovitz` 对应的图相对有些难度，下面简单介绍一下。

通过建立随机的权重来进行蒙特卡洛模拟收益率和方差，然后依次计算出 sharp 比。

```
def get_plotly_path_monte_markovitz(self,
    file_name='monte_markovitz.html', monte_count=400, risk_free = 0.03):
    """
    """

    path_plotly = self.path_dir_plotly_html + os.sep + file_name
    df = pd.read_excel(r'data\组合.xlsx', index_col=[0])

    returns = df.pct_change()
    returns.dropna(inplace=True)

    noa = 3

    # 蒙特卡洛随机模拟结果

    port_returns = []
    port_variance = []
```

```

for p in range(monte_count):
    weights = np.random.random(noa)
    weights /= np.sum(weights)
    port_returns.append(np.sum(returns.mean() * 50 * weights)) #
加入模拟的均值

    port_variance.append(np.sqrt(np.dot(weights.T,
np.dot(returns.cov() * 50, weights)))) # 加入模拟的标准差

port_returns = np.array(port_returns)
port_variance = np.array(port_variance)

color_array = (port_returns - risk_free) / port_variance # sharp
比, 不同的 sharp 比对应的颜色是不同的

```

接下来对收益率和风险进行绘图，并且不同的 sharp 比对应不同的颜色，即颜色是可变的，并添加 colorbar。

```

# 此处位置为 get_plotly_path_monte_markovitz 函数内部
trace1 = go.Scatter(
    x=port_variance,
    y=port_returns,
    mode='markers',
    marker=dict(
        size='6',
        color=color_array, # 通过一个可变的变量表示颜色，结果是绘图颜色可变
        colorscale='Viridis',
        # 设置 colorbar
        colorbar=dict(
            tickmode='linear',
            tick0=color_array.min(),
            dtick=(color_array.max() - color_array.min()) / 5,
        ),
        showscale=True,
    )
)

data = [trace1]

```

```
pyof.plot(data, filename=path_plotly, auto_open=False)
return path_plotly
```

我们发现这部分内容与“展示产品基本信息”一节所涉及的 GUI 呈现技巧差不多，只是其背后的逻辑更复杂一些，而这些稍微复杂的逻辑由于不是本书的重点，所以都没有介绍。打开这个程序，然后运行，结果会看到使用 QWebEngineView 所绘制的图表非常的炫酷，而且它的设计又是那么的简单。