预备知识：Python基础，面向对象的编程思想

PyQt 教案

PyQt 6 文档；

PyQt 5 文档；

简介

PyQt 是一个用于创建图形用户界面（GUI）的强大工具包，基于 Qt 框架。它允许开发者使用 Python 语言来构建跨平台的桌面应用程序。PyQt 提供了丰富的控件和功能，使得开发复杂的 GUI 应用变得更加简单和高效。

教学目标

1. 理解 PyQt 的基本概念和架构。
2. 掌握 PyQt 的安装和配置方法。
3. 能够完成一个小车的图形上位机编写

课前准备

1. 安装 Python 和 PyQt5 库。
2. 安装 Qt Designer

教学内容

一， PyQt 基本机制

1. 信号与槽（Signals & Slots）—— 组件交互的核心

这是PyQt最独特、最核心的机制，用于实现**组件之间的通信**（比如“按钮被点击后执行某个操作”）。

- **信号（Signal）**：组件在特定事件发生时发出的“通知”。
  例如：按钮被点击（`clicked` 信号）、输入框内容变化（`textChanged` 信号）、窗口关闭（`closed` 信号）等，这些信号由PyQt自动定义，无需手动编写。

- **槽（Slot）**：用于接收并处理信号的函数（可以是自定义函数）。
  例如：点击按钮后弹出提示框、输入框内容变化时更新显示等，这些功能都可以写成槽函数。
- **关联方式**：通过 `connect()` 方法将信号与槽绑定，形成“事件触发→响应”的关系。
  简单例子：

```
import sys
from PyQt5.QtWidgets import QApplication, QPushButton, QMessageBox

def show_message():  # 自定义槽函数
    QMessageBox.information(None, "提示", "按钮被点击了！")

app = QApplication(sys.argv)
button = QPushButton("点击我")  # 创建按钮组件
button.clicked.connect(show_message)  # 将按钮的clicked信号与槽函数关联
button.show()
app.exec_()  # 启动事件循环
```

  当按钮被点击时， `clicked` 信号被触发，自动调用 `show_message` 槽函数，弹出提示框。

2. 事件循环（Event Loop）—— 程序运行的“心脏”

PyQt程序的运行依赖于**事件循环**，它是一个持续运行的“循环”，负责监听和处理所有用户操作（如点击、输入、窗口大小变化等）。

- **工作流程**：
  i. 程序启动时，通过 `QApplication.exec_()` 启动事件循环；
  ii. 循环不断“等待”用户操作（事件），一旦有事件发生（如点击按钮），就将事件分发给对应的组件处理；
  iii. 处理完成后，继续等待下一个事件，直到程序被关闭（如点击窗口关闭按钮），循环才会结束。
- **为什么需要它**：
  没有事件循环，程序会执行完代码后直接退出，无法响应用户操作。比如上面的例子中， `app.exec_()` 是必不可少的，否则窗口会一闪而过。

3. 组件模型（Widget Hierarchy）—— UI元素的“家族树”

PyQt的所有可视化元素（按钮、窗口、输入框等）都基于**QWidget**类（基础组件类），形成一套清晰的继承关系（类似“家族树”）。

- **核心特点**：
  - 所有UI组件都是 `QWidget` 的子类（或间接子类），例如：
    - `QPushButton` （按钮）继承自 `QAbstractButton` ，而 `QAbstractButton` 又继承自 `QWidget` ；

        - `QMainWindow`（主窗口）直接继承自 `QWidget`。
    - 组件可以“嵌套”：一个组件可以作为另一个组件的“父组件”（如窗口是按钮的父组件），父组件被删除时，子组件会自动被删除，避免内存泄漏。
- **好处**：
  统一的继承体系让所有组件拥有共同的基础功能（如显示、隐藏、设置大小等），同时各自扩展独特功能（如按钮的点击、输入框的文本编辑），方便学习和使用。

4. 布局管理（Layout Management）—— 自动排列组件的“智能管家”

手动设置组件的位置和大小（如 `setGeometry(x, y, width, height)`）很繁琐，且窗口大小变化时组件会错位。PyQt的**布局管理器**能自动管理组件的位置和大小，让界面更灵活。

- **常用布局**：
    - `QVBoxLayout`：垂直布局（组件从上到下排列）；
    - `QHBoxLayout`：水平布局（组件从左到右排列）；
    - `QGridLayout`：网格布局（组件按行列排列，类似表格）。
- **例子**：用垂直布局排列两个按钮，窗口大小变化时按钮会自动适应：

```
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout

app = QApplication([])
window = QWidget()
layout = QVBoxLayout()  # 创建垂直布局

# 向布局中添加组件
layout.addWidget(QPushButton("按钮1"))
layout.addWidget(QPushButton("按钮2"))

window.setLayout(layout)  # 为窗口设置布局
window.show()
app.exec_()
```

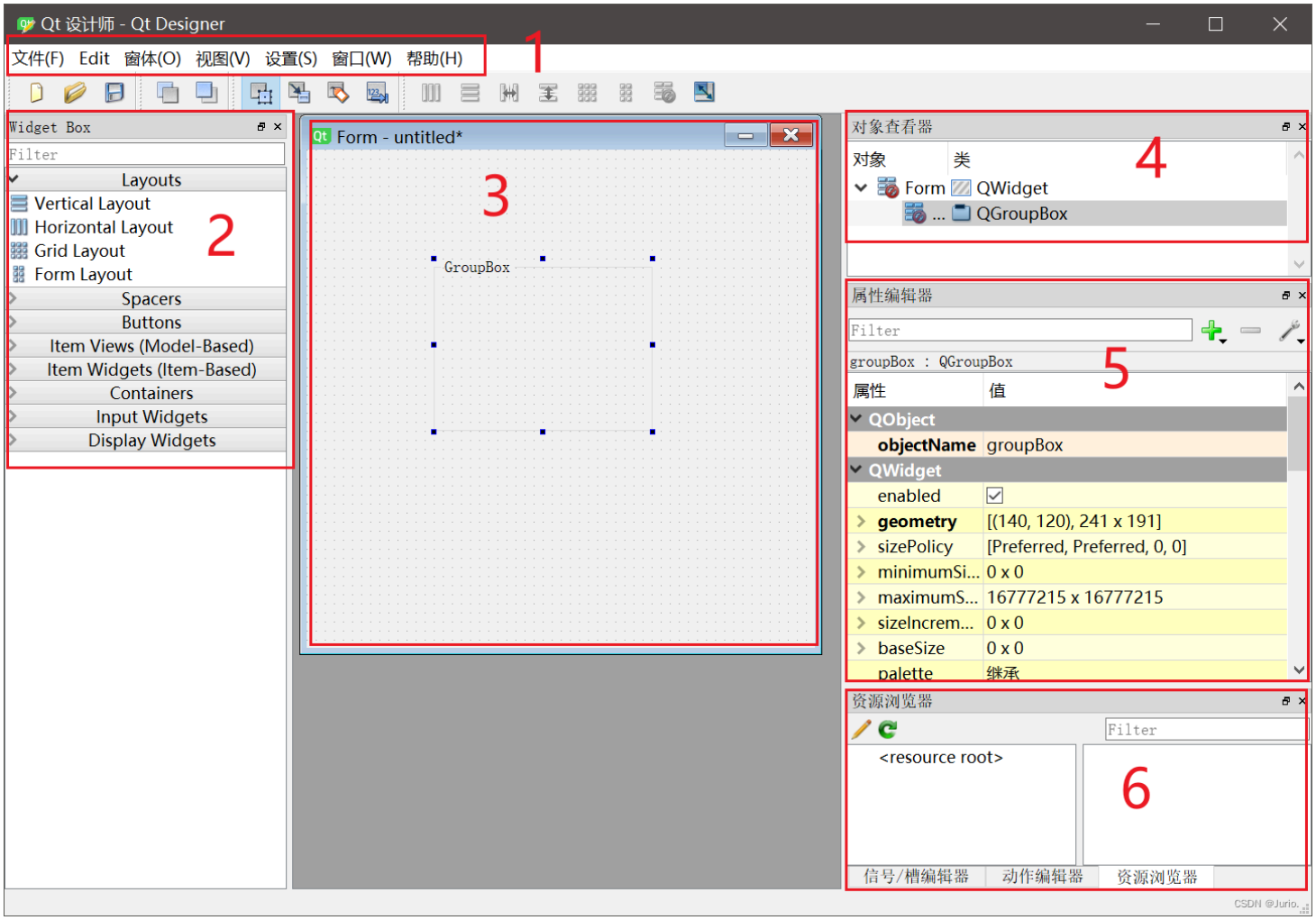
二， QtDesigner

PyQt5 界面设计工具 QtDesigner 详细使用

1. 简介

- QtDesigner是一个可视化的界面设计工具，用于快速创建PyQt的UI界面。它提供了拖放组件、布局管理、信号槽关联等功能，极大简化了界面设计的过程。

2. 界面简介



1. 顶部菜单栏：操作入口
   截图顶部显示了标准的菜单栏，包含常用操作入口，新生需重点关注3个核心菜单：

- **文件(F)**：负责UI文件的基础操作，如“新建窗体”(选择基于 `QWidget` / `QMainWindow` 等载体)、“保存”（将设计存为 `.ui` 文件)、“打开”(编辑已有UI)。
- **窗体(O)**：核心用于“布局管理”，比如选中控件后，通过“垂直布局”“水平布局”快速让控件自动排列，避免手动调整位置。
- **帮助(H)**：提供Qt Designer的基础教程，新手遇到控件用法问题时可查阅。

2. 存放所有可拖拽的UI组件，按功能分类：

- **Layouts**：布局管理器（垂直/水平/网格/表单布局)，先拖布局再放控件，自动对齐；
- **Buttons**：按钮（如 `QPushButton`)；
- **Containers**：容器（如截图中的 `QGroupBox` ，用于分组控件，比如把"用户名+密码输入框"放一组)；
- **Input Widgets**：输入控件（如文本框 `QLineEdit` 、下拉框 `QComboBox`)。

3. 中间设计区：可视化画布
   截图中间的“Qt Form - untitled*”是核心设计区，当前展示的是一个**基于 `QWidget` 的空白窗体**（标题“untitled”表示未保存），设计区中已添加：

- 1个 `QGroupBox` （分组框，默认名 `groupBox` ）：用于“组件分组”，比如后续可在里面拖入“性别选择按钮组”“兴趣复选框组”，让UI更规整；
- 隐含的布局管理器：截图左侧 `Widget Box` 中显示了“Vertical Layout（垂直布局）”“Horizontal Layout（水平布局）”，说明当前可能已为 `QGroupBox` 或窗体设置了布局，后续拖入控件会自动按布局排列。

4. 对象查看器

- 显示当前UI中所有组件的“层级关系”（类似“文件树”），比如截图中“Form（父载体）→ groupBox（子组件）”，可快速选中组件、修改组件名（如把 `groupBox` 改名为 `userGroup` ）。

5. 属性编辑器

- 修改选中组件的“属性”，无需写代码。
- 管理UI中用到的“资源”（如图片、图标）：比如给按钮加图标时，先在这里导入图片文件，再在“属性编辑器”中选择图片，避免代码中手动处理路径。

3. 导出UI文件

三， 如何使用.ui文件

3. 导出UI文件

1. 保存后会获得一个 `.ui` 文件，后续可以用 `pyuic5` 将其转换为Python代码。

```
pyuic5 -o ui_main.py main.ui
```

2. 在你的程序中导入这个 `ui_main.py` 中的 `Ui_MainWindow` 类

```
from ui_main import Ui_MainWindow
from PyQt5.QtWidgets import QApplication, QMainWindow
import sys
```

3. 创建主窗口类

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
```

4. 运行应用

```
if __name__ == "__main__":
    app = QApplication([])
    window = MainWindow()
    window.show()
    app.exec_()
```

5. 需要注意的点

- 信号与槽的连接
  - 信号：组件的状态变化，如按钮点击、文本框输入等。
  - 槽：响应信号的函数，当信号触发时，会自动调用槽函数。
  - 连接：在代码中使用 `connect` 方法将信号与槽连接起来，例如：

    ```
    self.ui.pushButton.clicked.connect(self.button_click)
    ```

    这表示当 `pushButton` 按钮被点击时，会调用 `button_click` 方法。
  - 槽函数的定义：在主窗口类中定义一个方法，用于响应信号。例如：
  - 同时也可以在designer中设置信号与槽的连接
    - 选中组件，在属性编辑器中找到 `clicked` 信号，点击 `+` 号添加槽函数。
    - 槽函数的定义：在主窗口类中定义一个方法，用于响应信号。
  - 推荐使用designer中设置信号与槽的连接，而不是在代码中设置。
- 为什么要使用designer生成ui文件，而不是直接在代码中编写ui，为什么ui界面要单独一个文件？
- i. 可视化设计：designer提供拖拽组件、布局管理等功能，极大简化了界面设计过程。
- ii. 代码与界面分离：ui文件与业务逻辑代码分离，方便团队合作开发。
- iii. 代码生成：使用 `pyuic5` 工具可以将ui文件转换为Python代码，避免手动编写ui代码。

四， 综合案例：小车上位机界面设计

1. 需求分析

- 设计一个简单的上位机界面，用于控制小车的运动和显示状态信息。
- 界面包括：

    - 小车运动控制按钮（前进、后退、左转、右转、停止）
    - 当前模式，机械臂位置

2. 界面设计

- 使用QtDesigner设计界面，布局合理，易于操作。
- 包含一个 `QGroupBox` ，用于分组显示运动控制按钮。
- 包含一个 `QLabel` ，用于显示当前模式。
- 包含一个 `QLabel` ，用于显示机械臂位置。
- ……

3. Designer实操

- 打开QtDesigner，创建一个新的窗体。
- 从左侧的组件库中拖拽组件到设计区，设置组件的属性。
- 使用布局管理器组织组件，比如使用垂直布局将按钮组垂直排列。
- 信号与槽的连接
    - 选中组件，在属性编辑器中找到 `clicked` 信号，点击 `+` 号添加槽函数。
    - 槽函数的定义：在主窗口类中定义一个方法，用于响应信号。
    - 槽函数的参数：槽函数的参数与信号的参数一致，比如按钮点击信号没有参数，所以槽函数也没有参数。
    - 槽函数的实现：在主窗口类中定义一个方法，用于响应信号。
- 保存.ui文件，后续可以使用 `pyuic5` 将其转换为Python代码。

4. 代码实现

- 导入生成的ui文件，创建主窗口类。
- 初始化ui组件。
- 实现按钮的槽函数，控制小车的运动。
- 实现其他组件的槽函数，更新状态信息。
- 运行应用。

```python
from PyQt5.QtWidgets import QApplication, QMainWindow
from ui_main import Ui_MainWindow
import sys

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

    def move_forward(self):
        print("小车前进")
        # 这里添加控制小车前进的代码

    def move_backward(self):
        print("小车后退")
        # 这里添加控制小车后退的代码

    def turn_left(self):
        print("小车左转")
        # 这里添加控制小车左转的代码

    def turn_right(self):
        print("小车右转")
        # 这里添加控制小车右转的代码

    def stop(self):
        print("小车停止")
        # 这里添加控制小车停止的代码

if __name__ == "__main__":
    app = QApplication([])
    window = MainWindow()
    window.show()
    app.exec_()
```