PortMonitor

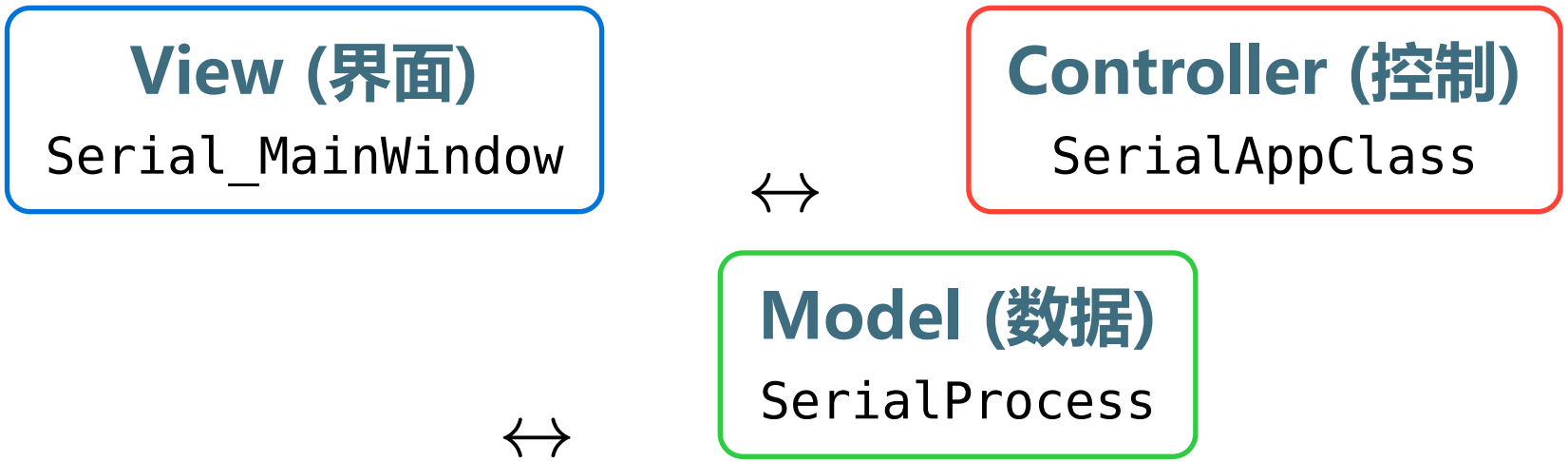
深度架构剖析与优势说明

MVC 架构 | PyQt5 | 工业级设计

2026-01-17

1. 架构概览：精密的 MVC 模式

本项目采用 **Model-View-Controller** 架构，结合 PyQt5 **信号与槽**机制。



此处插入图表: 详细类关系图 (Class Diagram)，展示 SerialProcess, SerialAppClass, Serial_MainWindow 的方法与引用关系

设计理念: 高内聚、低耦合，专业软件工程素养。

Model (模型层)

核心逻辑与数据

- **主要组件**:
  - `SerialProcess.py`: 串口引擎
  - `config_manager.py`: 配置管家
- **核心职责**:
  - 封装底层 `QSerialPort` 操作
  - 负责应用状态的持久化 (Load/Save)

- **架构亮点**:
  - **完全独立于 UI**
  - 纯逻辑封装，可轻松移植到自动化脚本或无头环境

View (视图层)

用户交互界面 - “Passive View”

- **主要组件**: `Serial_MainWindow.ui / .py`
- **核心职责**:
  - 定义布局、样式、控件属性
  - **不**包含复杂的业务逻辑代码

此处插入图表: 软件运行主界面截图，高亮标注出 '数据接收区', '配置区', '图表区' 等 Passive View 区域

- **架构亮点**:
  - 代码由 UI 设计器自动生成，与逻辑物理分离

Controller (控制器层)

业务逻辑的中枢神经

- **主要组件**: `app_SerialWindows.py`
- **核心职责**: 程序的“胶水”，初始化 View/Model，调度核心。

此处插入图表: 信号与槽时序图 (Sequential Diagram)：展示 Data Received -> Signal Emit -> Slot Triggered -> UI Update 的完整流程

- **工作流**: Model 发出 `data_received` → Controller 捕获 → View 刷新

2. 核心优势 - A. 极致解耦

🚫 传统初学者写法

- `serial.read()` 阻塞在 GUI 线程
- 逻辑散落在按钮点击事件中
- 代码面条化，难以维护

✅ PortMonitor 架构

- `SerialProcess` 独立对象
- 仅通过信号通讯
- **单元测试友好 (Unit Testing)**

此处插入图表: 对比图：'单体代码块' vs '模块化组件' 的结构差异示意

核心优势 - B. 响应式与异步 I/O

不再有 “界面假死”

- **机制**: 基于 PyQt5 强大的事件循环 (Event Loop)
- **表现**: 无论波特率多高 (115200+)，界面始终**丝滑流畅**

此处插入图表: 线程模型图：展示 'GUI 主线程' 负责渲染，'底层 IO' 通过事件驱动回调，两者互不阻塞的时间轴

- **原理**: 硬件操作分离 → 信号触发 → 异步更新

核心优势 - C. 像素级配置还原

JSONConfigManager 的细节体验

此处插入图表: 数据流图：User Action (Change BaudRate) -> Signal -> AutoSave -> JSON File 的自动化闭环

- **“修改即保存”**：改变波特率/窗口大小/Hex 模式 → Auto Save

- **用户价值**: 程序重启后**完全还原**，区分 **Demo** 与 **专业工具** 的关键细节

核心优势 - D. 架构扩展性

为未来而设计

- **类型安全**: 广泛使用的 Python Type Hints (`window_manager: 'WindowManagerClass'`)

此处插入图表: 系统架构图：展示 WindowManager 持有 QStackedWidget，下辖 SerialApp, PlotApp(未来), NetworkApp(未来) 等子模块

- **窗口管理**: `QStackedWidget` 架构，轻松扩展多页面应用

3. 总结

一份优秀的 PyQt5 架构教科书

- Model-View-Controller 完美实践
- 兼顾 高稳定性 与 高性能

- 清晰的代码结构赋予项目无限生命力