上位机

上位机开发需求列表－工作日志：

- 工控机－SQL数据库连接通信（**7.2已完成**）
  - [x] ~~本地化数据库创建~~
  - [x] ~~数据库管理员权限配置~~
  - [x] ~~Python连接SQL并访问数据~~
  - [x] ~~数据库基础表创建~~
- 工控机－单片机连接通信（**7.10已完成**）
- 工控机－PLC连接通信（**7.18已完成**）
  - [x] ~~PLC通信模块梯形图绘制~~
  - [x] ~~PLC组态软件软硬件联调~~
- 工控机Ｍｏｄｂｕｓ　ＲＴＵ协议数据帧发送（**7.28已完成**）
- 工控机界面开发－SQL数据直连（进行中）
  - [ ] Ｐｙｑｔ前端界面开发－数字孪生
  - [ ] 后端开发
  - [ ] 协议帧异常管理：错误、缺失、稳定性
- 数据库系统开发（进度２０％）

- [ ] 顶层用例设计、需求分析、数据流程开发（正在进行）
- [ ] 用户权限开发、数据库安全、触发器开发（待开始）

1.MODBUS－RTU通信协议开发

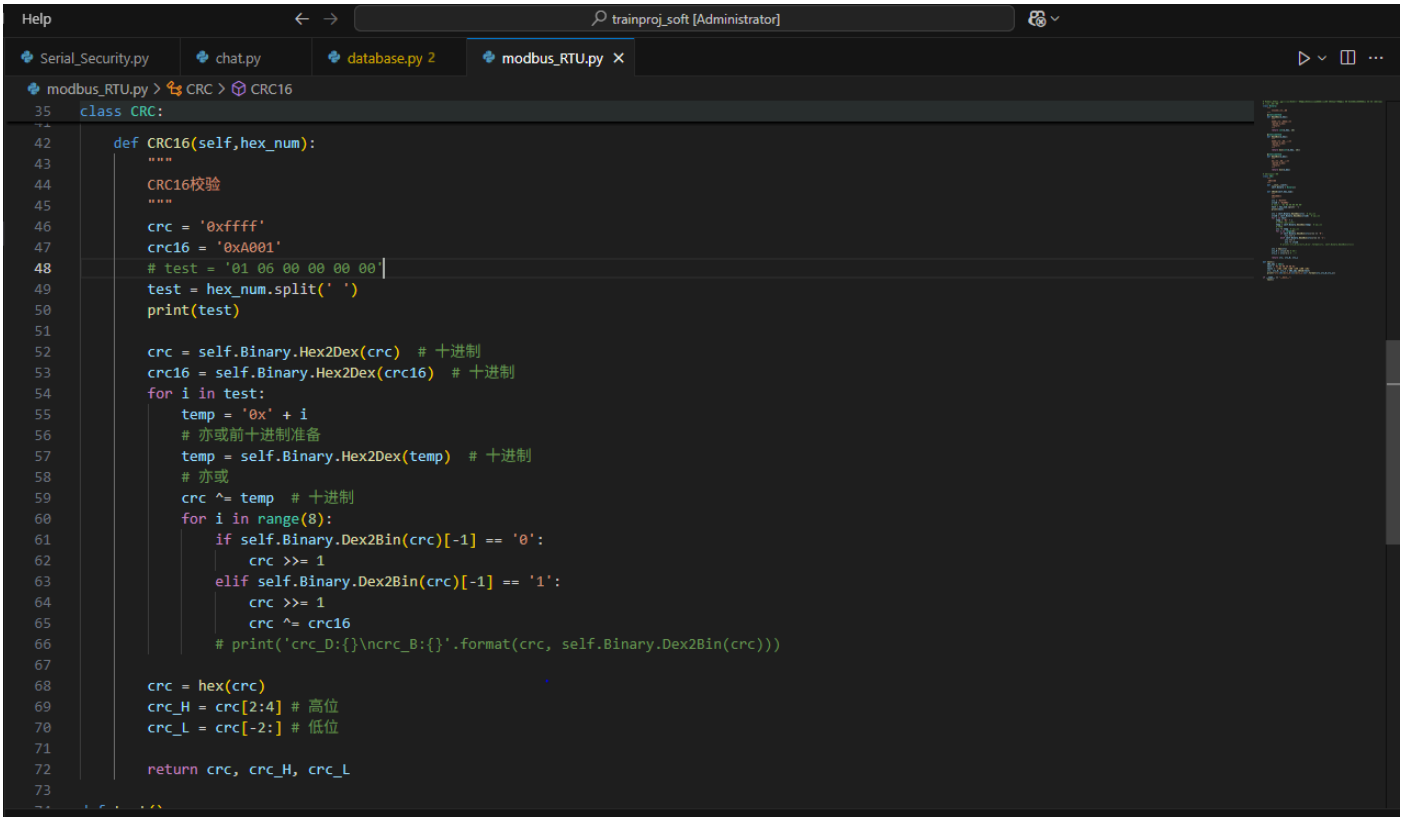
本部分详细描述了基于 PyQt5 的 MODBUS-RTU 通信协议实现方案，包括串口通信框架、CRC 校验算法以及 MODBUS-RTU 报文处理机制。

本系统采用分层设计架构，主要分为三层：

1. **硬件接口层**：通过QSerialPort实现底层串口通信
2. **协议处理层**：实现MODBUS协议帧的封装、解析和校验
3. **应用接口层**：提供简洁的API供上层应用调用

首先开发了MODBUS－RTU通信协议中CRC帧格式，随后开发了硬件接口，保证工控机实现与下层单片机的COM串口通信。

经过协议验证，代码可靠，硬件连接稳定



Serial_Security.py | chat.py | database.py 2 | modbus_RTU.py

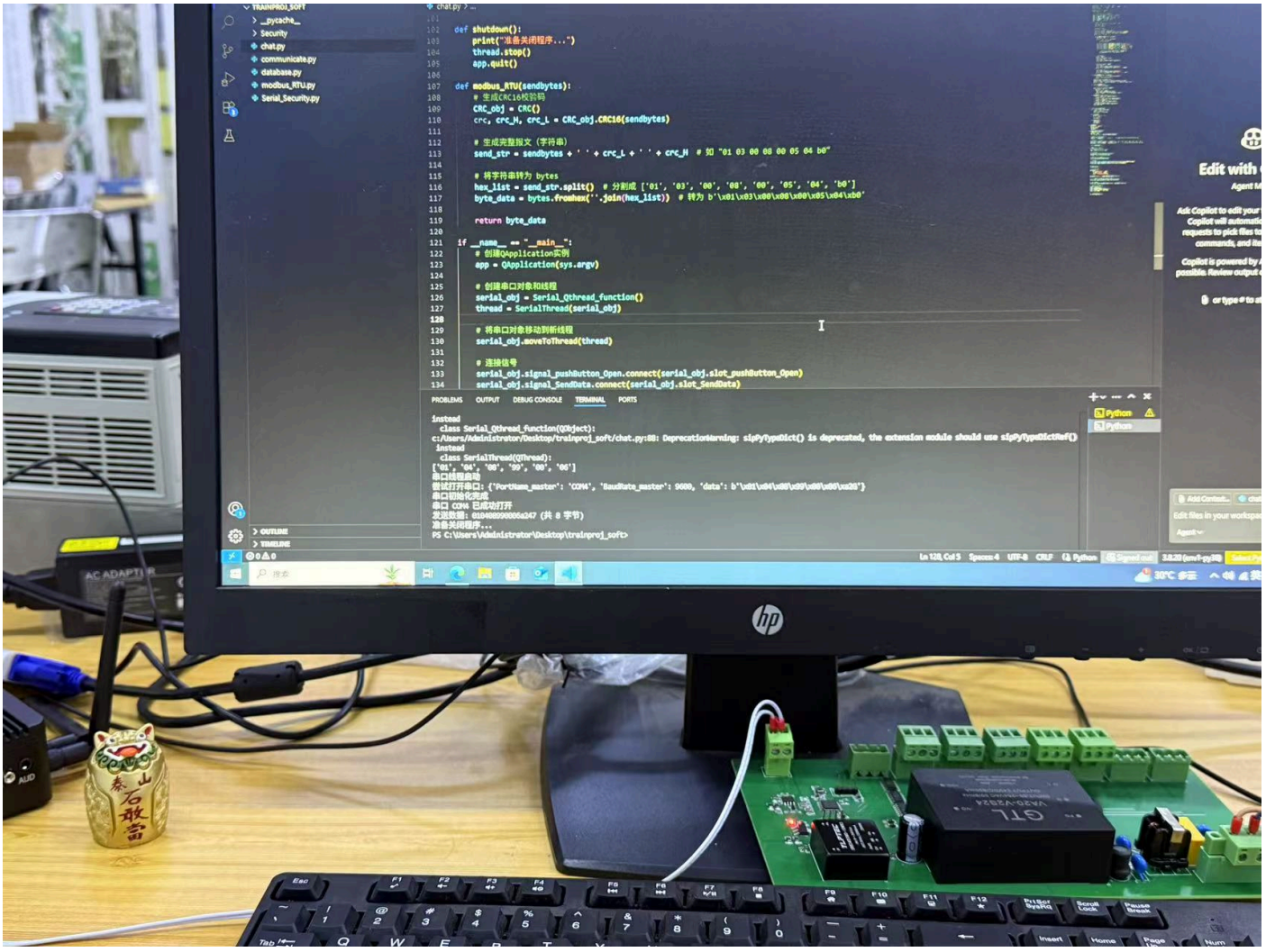
chat.py > ...

```python
class Serial_Qthread_function(QObject):
    signal_Serial_qthread_function_Init = pyqtSignal()
    signal_pushButton_Open = pyqtSignal(object)
    signal_pushButton_Open_flage = pyqtSignal(object)
    signal_readyRead = pyqtSignal(object)
    signal_SendData = pyqtSignal(object)

    def __init__(self):
        super(Serial_Qthread_function, self).__init__()
        self.state = 0  # 0未打开，1已打开
        self.serial_master = None

    def init_serial(self):
        """在线程中初始化串口"""
        self.serial_master = QSerialPort()
        self.serial_master.readyRead.connect(self.slot_readyRead)
        self.serial_master.errorOccurred.connect(self.handleError)
        print("串口初始化完成")

    def slot_pushButton_Open(self, paremeter):
        if self.state == 0:
            print("尝试打开串口:", paremeter)
            if not self.serial_master:
                self.init_serial()

            self.serial_master.setPortName(paremeter['PortName_master'])
            self.serial_master.setBaudRate(paremeter['BaudRate_master'])
            self.serial_master.setDataBits(QSerialPort.Data8)  # 设置数据位
            self.serial_master.setParity(QSerialPort.NoParity)  # 设置校验位
            self.serial_master.setStopBits(QSerialPort.OneStop)  # 设置停止位

            if not self.serial_master.open(QIODevice.ReadWrite):
                self.handleError(self.serial_master.error())
                return
```

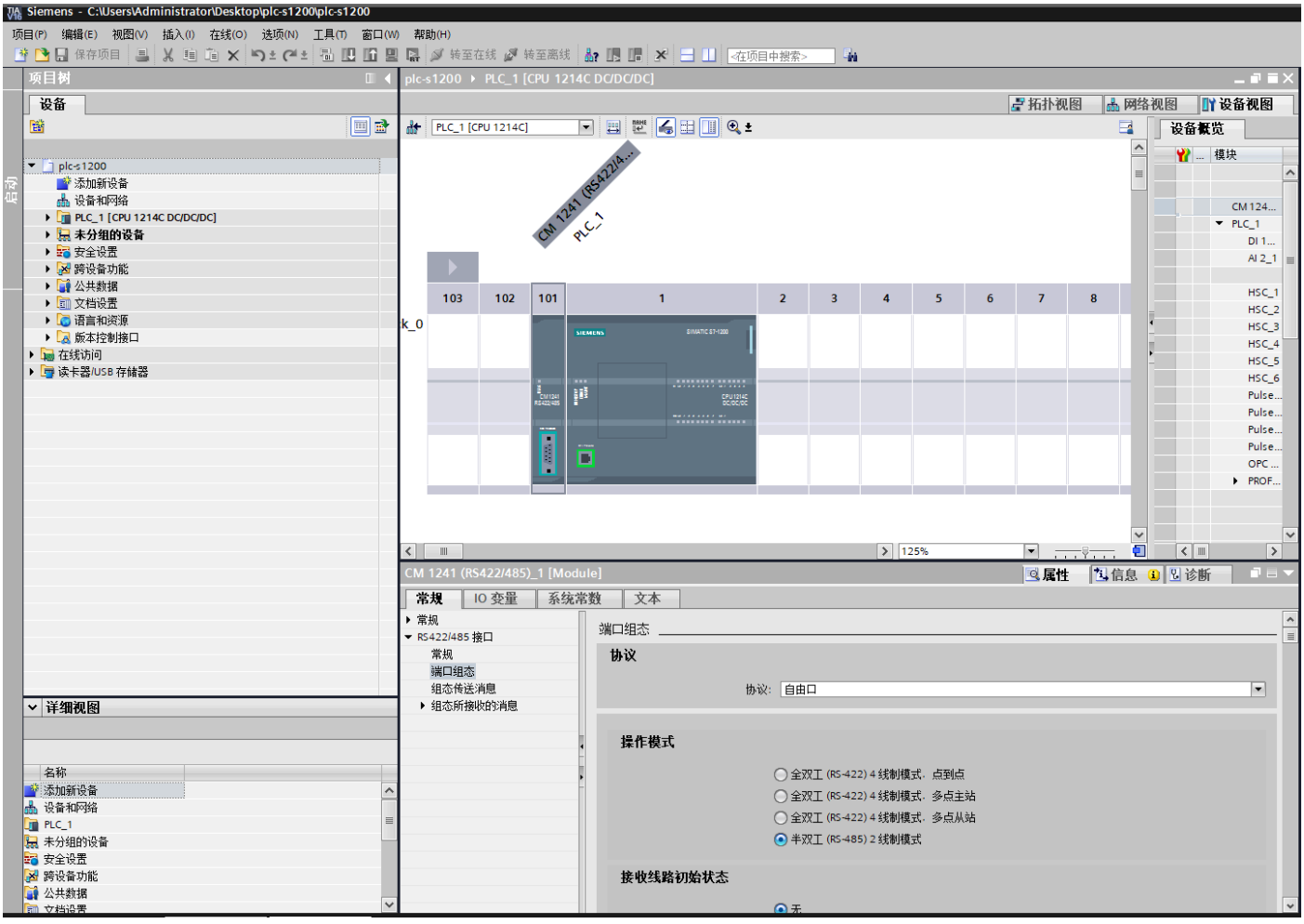
PROBLEMS 2 | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS

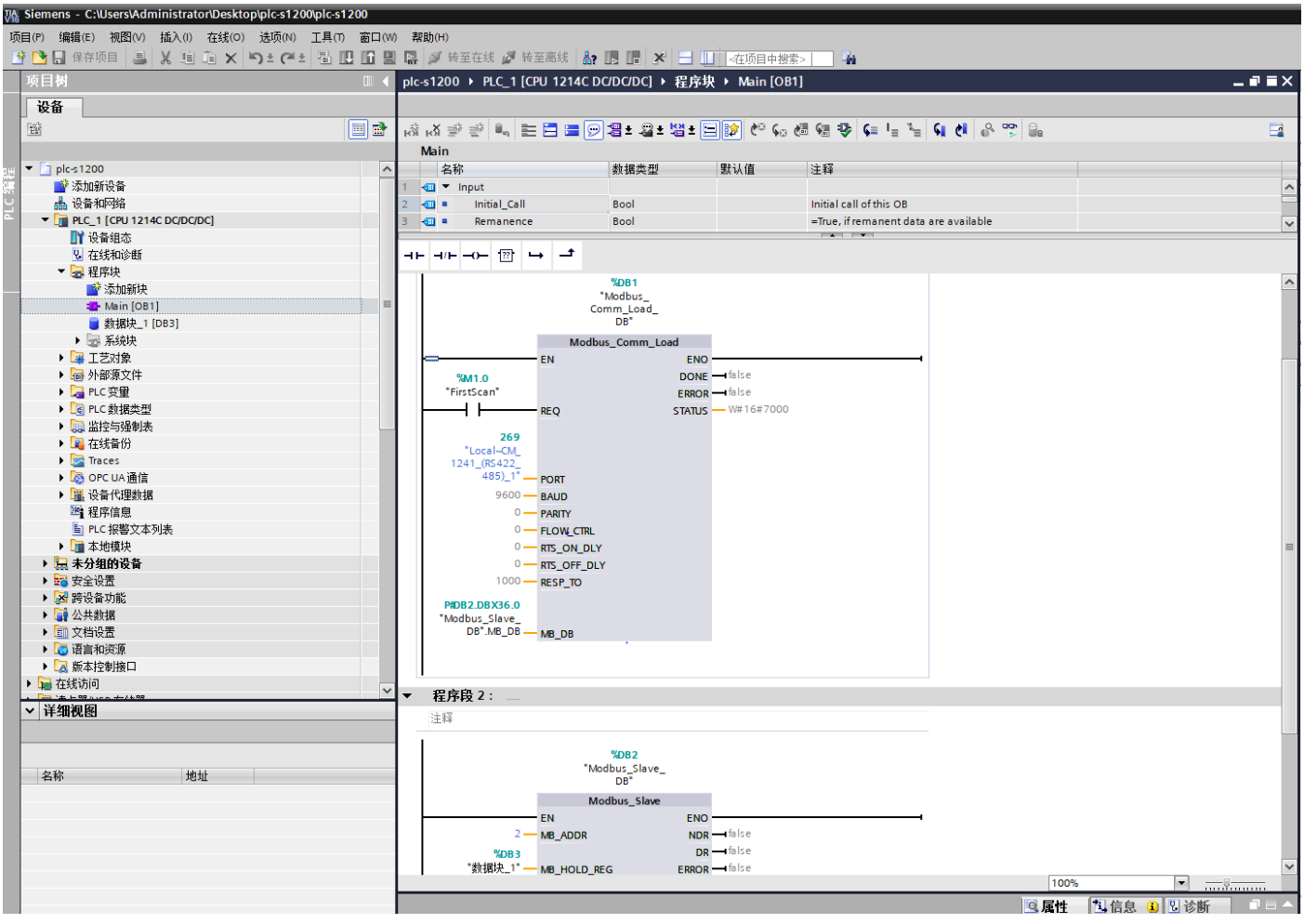
Python | Python

```
instead
  class Serial_Qthread_function(QObject):
c:/Users/Administrator/Desktop/trainproj_soft/chat.py:88: DeprecationWarning: sipPyTypeDict() is deprecated, the extension module should use sipPyTypeDictRef()
 instead
  class SerialThread(QThread):
['01', '04', '08', '99', '00', '06']
串口线程启动
尝试打开串口: {'PortName_master': 'COM4', 'BaudRate_master': 9600, 'data': b'\x01\x04\x08\x99\x00\x06\xa2G'}
串口初始化完成
串口 COM4 已成功打开
发送数据: 010408990006a247 (共 8 字节)
准备关闭程序...
PS C:\Users\Administrator\Desktop\trainproj_soft>
```



2． 工控机连接PLC并进行通信


Siemens - C:\Users\Administrator\Desktop\plc-s1200\plc-s1200
项目树
设备
plc-s1200
添加新设备
设备和网络
PLC_1 [CPU 1214C DC/DC/DC]
未分组的设备
安全设置
跨设备功能
公共数据
文档设置
语言和资源
版本控制接口
在线访问
读卡器/USB 存储器
详细视图
名称
添加新设备
设备和网络
PLC_1
未分组的设备
安全设置
跨设备功能
公共数据
文档设置
plc-s1200 ▸ PLC_1 [CPU 1214C DC/DC/DC]
拓扑视图
网络视图
设备视图
PLC_1 [CPU 1214C]
设备概览
模块
CM 1241 (RS422/4...
PLC_1
103
102
101
1
2
3
4
5
6
7
8
125%
CM 1241 (RS422/485)_1 [Module]
属性
信息
诊断
常规
IO 变量
系统常数
文本
常规
RS422/485 接口
常规
端口组态
组态传送消息
组态所接收的消息
端口组态
协议
协议: 自由口
操作模式
全双工 (RS-422) 4 线制模式，点到点
全双工 (RS-422) 4 线制模式，多点主站
全双工 (RS-422) 4 线制模式，多点从站
半双工 (RS-485) 2 线制模式
接收线路初始状态
无


Siemens - C:\Users\Administrator\Desktop\plc-s1200\plc-s1200
项目树
设备
plc-s1200
添加新设备
设备和网络
PLC_1 [CPU 1214C DC/DC/DC]
设备组态
在线和诊断
程序块
添加新块
Main [OB1]
数据块_1 [DB3]
系统块
工艺对象
外部源文件
PLC 变量
PLC 数据类型
监控与强制表
在线备份
Traces
OPC UA 通信
设备代理数据
程序信息
PLC 报警文本列表
本地模块
未分组的设备
安全设置
跨设备功能
公共数据
文档设置
语言和资源
版本控制接口
在线访问
详细视图
名称
地址
plc-s1200 ▸ PLC_1 [CPU 1214C DC/DC/DC] ▸ 程序块 ▸ Main [OB1]
Main
名称
数据类型
默认值
注释
Input
Initial_Call
Bool
Initial call of this OB
Remanence
Bool
=True, if remanent data are available
%DB1
"Modbus_Comm_Load_DB"
Modbus_Comm_Load
EN
ENO
%M1.0
"FirstScan"
REQ
DONE
false
ERROR
false
STATUS
W#16#7000
269
"Local~CM_1241_(RS422_485)_1"
PORT
9600
BAUD
0
PARITY
0
FLOW_CTRL
0
RTS_ON_DLY
0
RTS_OFF_DLY
1000
RESP_TO
P#DB2.DBX36.0
"Modbus_Slave_DB".MB_DB
MB_DB
程序段 2：
注释
%DB2
"Modbus_Slave_DB"
Modbus_Slave
EN
ENO
2
MB_ADDR
NDR
false
%DB3
"数据块_1"
MB_HOLD_REG
DR
false
ERROR
false
100%
属性
信息
诊断

3． SQL本地化数据库连接工控机

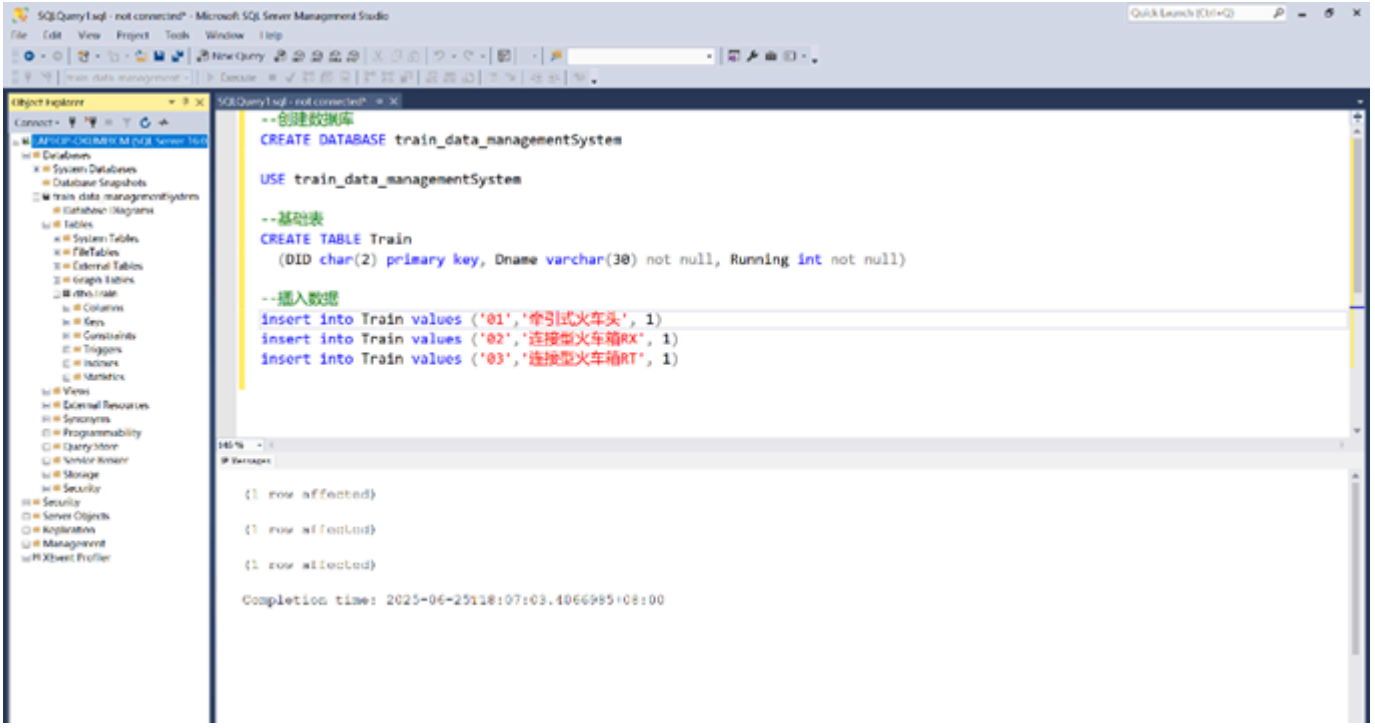
创建本地SQL Server数据库

• 工作：创建安装SQL Server，安装SQL Server Management Studio，设置工控机用户权限，创建数据库train_data_managementSystem，E-R图设计数据库存储结构，定义数据库表单以及依赖，插入虚拟数据。

> 具体数据来源，需要与下游联调后，才能确定，所以数据库表单设计只能根据构想进行理想化编程，后期需要根据业务更改表单内容

• 截图：

[图片]

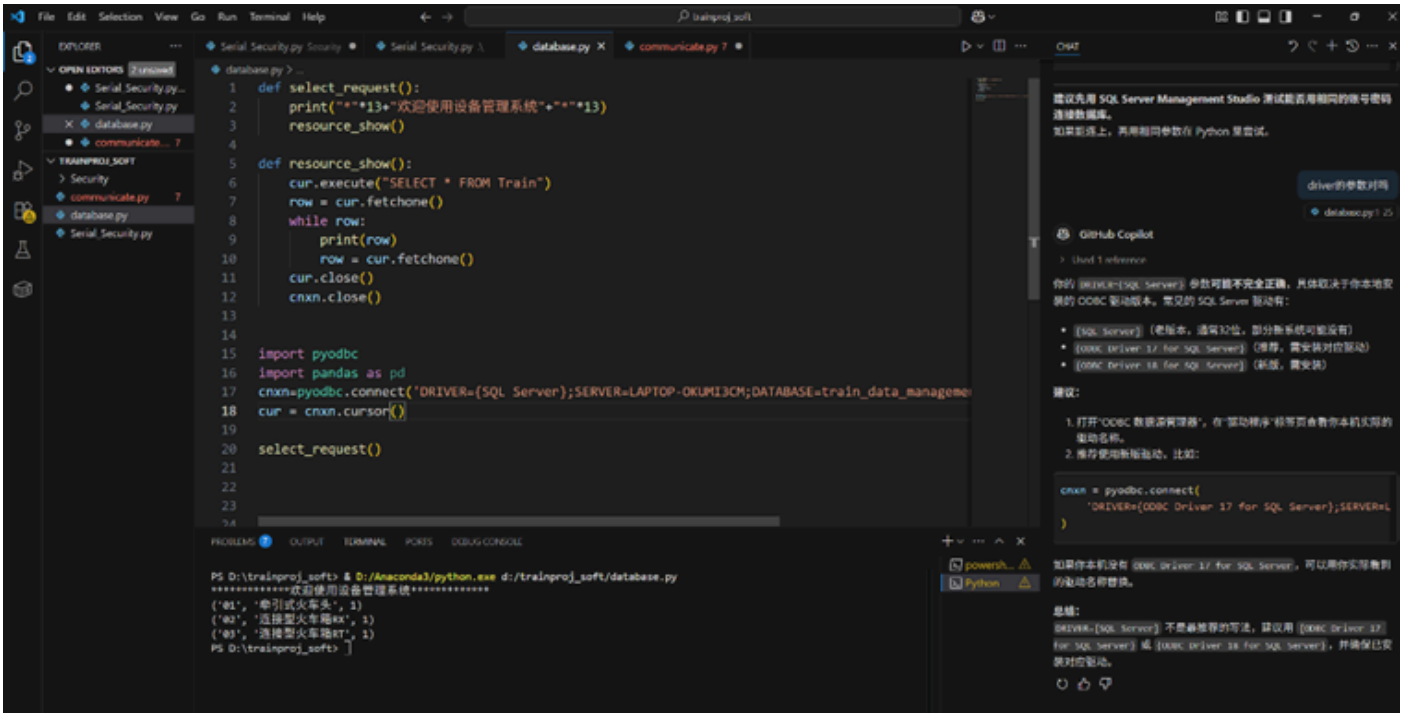


4． SQL本地化数据库连接工控机

（1） 工作：编写python代码，使得上位机程序可以直接访问数据库中的数据

（2） 方案：采用pyodbc中的游标cursor依次访问数据库中的内容

（3） 截图：



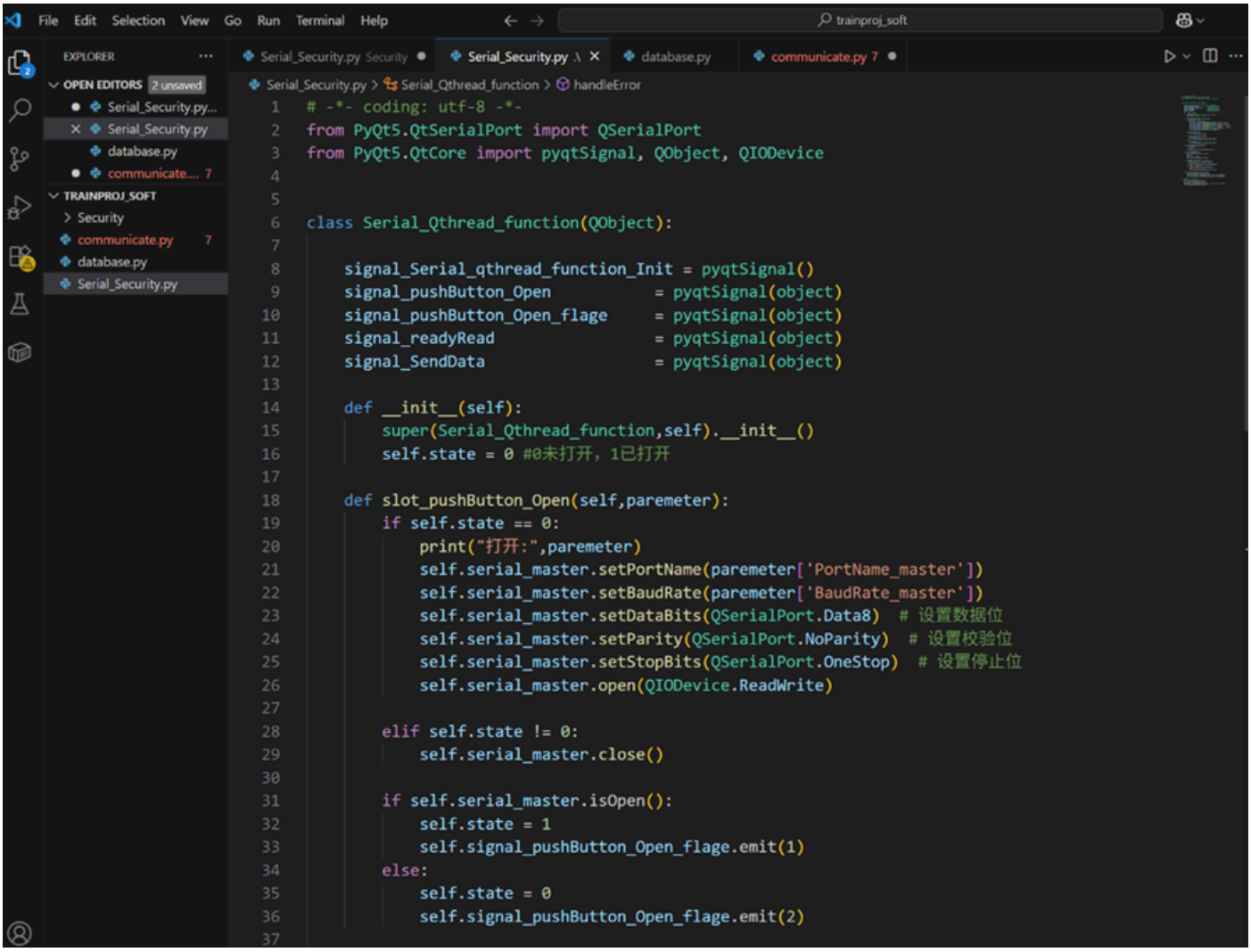
5. 实现工控机与plc的串口通信

5.1 创建本地SQL Server数据库

（1） 工作：实现PLC通过串口向工控机发送数据，再有工控机中的python脚本，将数据自动写入到SQL Server，实现发送数据的功能，通常采用中间件程序作为桥梁处理两者之间的交互。

（2） 方案：PySerial可用于控制RS-485接口设备，符合我们的方案选型，而pyodbc则负责与SQL Server的交互，编写脚本监听来自PLC的消息并通过API插入记录至目标表格

（3） 截图：


File Edit Selection View Go Run Terminal Help
trainproj_soft
EXPLORER
OPEN EDITORS 2 unsaved
Serial_Security.py...
Serial_Security.py
database.py
communicate.... 7
TRAINPROJ_SOFT
Security
communicate.py 7
database.py
Serial_Security.py
Serial_Security.py Security
Serial_Security.py .\
database.py
communicate.py 7
Serial_Security.py > Serial_Qthread_function > handleError
# -*- coding: utf-8 -*-
from PyQt5.QtSerialPort import QSerialPort
from PyQt5.QtCore import pyqtSignal, QObject, QIODevice
class Serial_Qthread_function(QObject):
signal_Serial_qthread_function_Init = pyqtSignal()
signal_pushButton_Open = pyqtSignal(object)
signal_pushButton_Open_flage = pyqtSignal(object)
signal_readyRead = pyqtSignal(object)
signal_SendData = pyqtSignal(object)
def __init__(self):
super(Serial_Qthread_function,self).__init__()
self.state = 0 #0未打开，1已打开
def slot_pushButton_Open(self,paremeter):
if self.state == 0:
print("打开:",paremeter)
self.serial_master.setPortName(paremeter['PortName_master'])
self.serial_master.setBaudRate(paremeter['BaudRate_master'])
self.serial_master.setDataBits(QSerialPort.Data8) # 设置数据位
self.serial_master.setParity(QSerialPort.NoParity) # 设置校验位
self.serial_master.setStopBits(QSerialPort.OneStop) # 设置停止位
self.serial_master.open(QIODevice.ReadWrite)
elif self.state != 0:
self.serial_master.close()
if self.serial_master.isOpen():
self.state = 1
self.signal_pushButton_Open_flage.emit(1)
else:
self.state = 0
self.signal_pushButton_Open_flage.emit(2)