ESTUN ROBOTICS

ER 系列工业机器人 Python 脚本应用参考手册

RCS2 V2.0.2

南京埃斯顿自动化股份有限公司
南京埃斯顿机器人工程有限公司

-本文档符合 ISO 体系要求，并会根据 IPD 体系变化更新-

文档修订记录

序号	版本号	修订日期	修订概述	修订人
1	V2.0.1	2021.11.17	新建文档	应用软件部
2	V2.0.2	2022.04.15	修改 2.1 节的描述	应用软件部

目录

前言

本手册适用于控制系统 RCS2 系列，介绍埃斯顿 ER 系列机器人相关通信使用和调试方法。

读者对象

本手册仅供埃斯顿机器人相关技术支持人员使用。

注意事项

- 在安装和调试这些组件时，操作人员必须严格遵循本文档的说明和解释。
- 相关负责人员必须确保所述产品的应用或使用满足所有安全要求，包括相关法律、法规、准则和标准。
- 尽管本文档经过精心编制，但由于其中所描述的产品仍处于不断更新换代中，我们可能不会在每次更新后都检查文档中所描述的产品性能数据、标准或其它特性总是与实际产品相一致。
- 本文档中难免会出现一些技术或者编辑错误，我们保留随时对文档信息做出修改之权力，恕不另行通知。对于已经变更的产品，如果本文档中的数据、图表以及文字描述没有修改，我们将不再特别加以声明。
- 任何人不得对软、硬件配置进行文本档中规定之外的修改，ESTUN 公司对因此而造成的一切后果不承担任何责任。
- 本文档中出现图示单位在没有特别标注说明时，默认单位为毫米 mm。

安全说明

⚠ 警告	受伤的危险 不遵守本标志相关的安全说明将危及个人生命和健康安全。
⚠ 注意	对环境和设备有危险 不遵守本标志相关安全说明可能明显危害环境和设备安全。
ℹ 说明	说明或提示 该标志表示这些信息能够帮助您更好的理解安全说明。

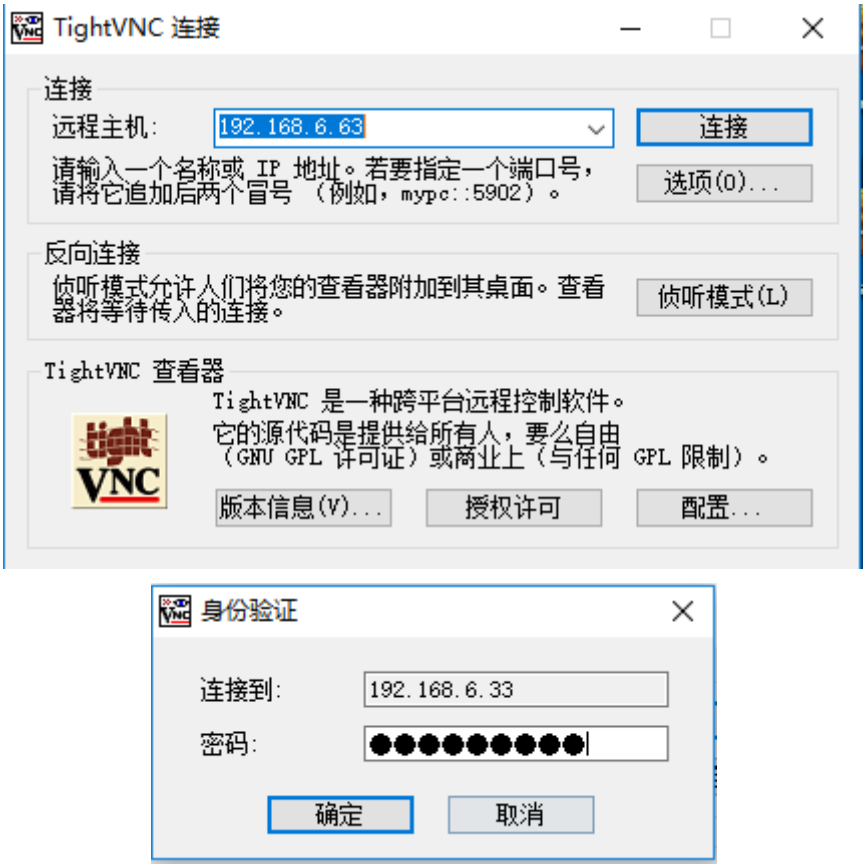
第 1 章 Python 脚本配置篇

1.1 本节概要

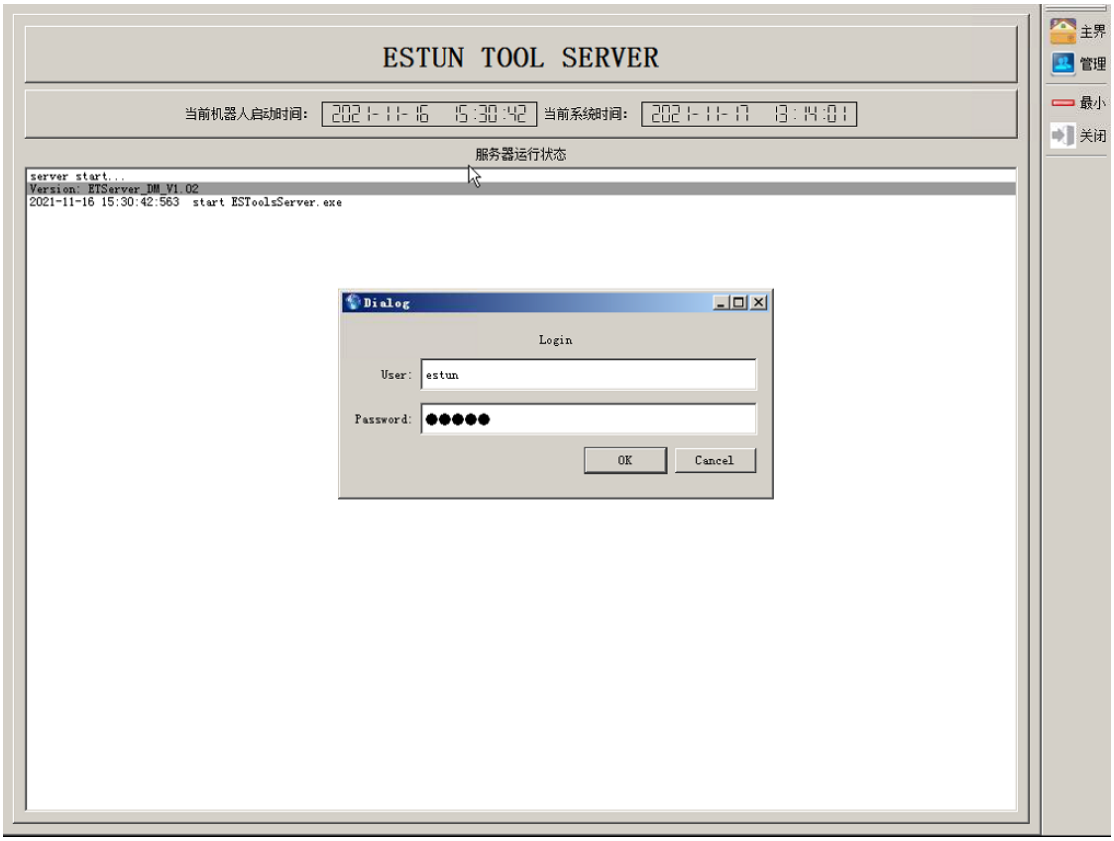
脚本的最新配置说明，在老版本的 runtime 下如何配置脚本的环境。在发布的 runtime 下如何升级脚本文件。

1.2 操作步骤

1. 用vnc连接进入控制器，在如下界面中输入控制器IP地址(默认IP地址是192.168.6.63)和密码：ERC654321



VNC连接控制器

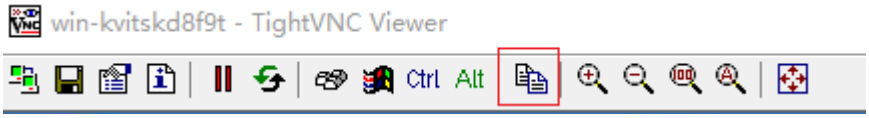


输入EstunTool密码,点击最小化进入系统

2. 检查控制器C:\runtime\python目录下是否存在PyServer文件夹：

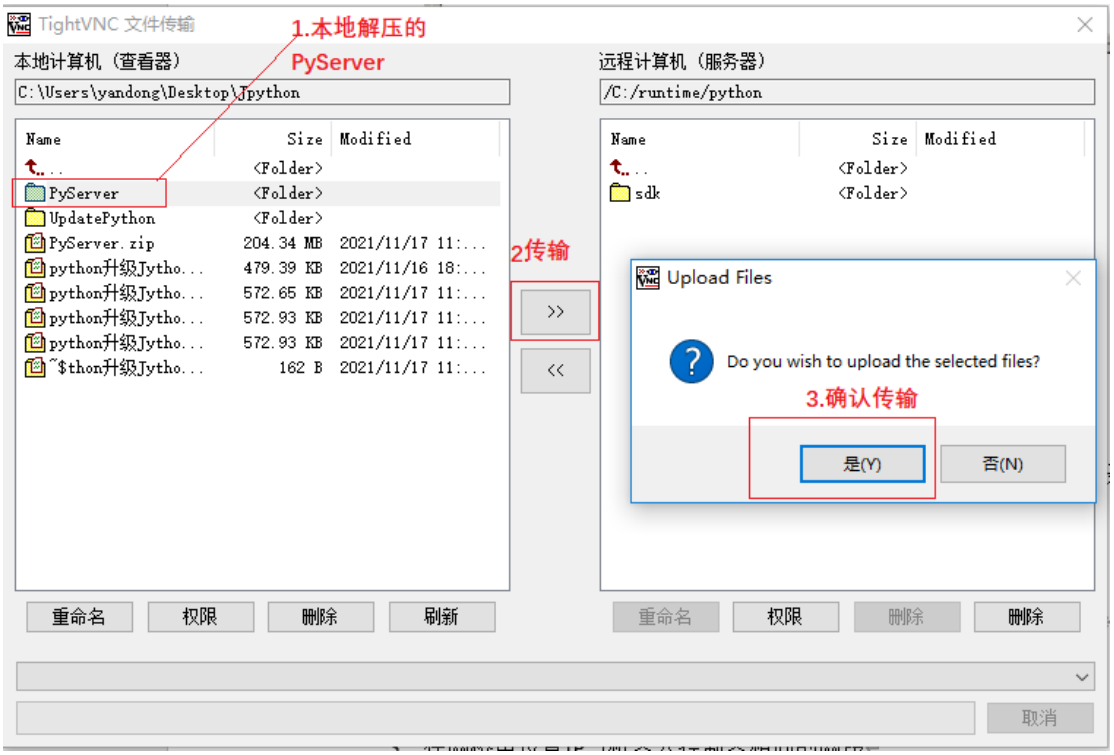
PyServer文件夹目录

如果不存在PyServer文件夹，把 PyServer.zip 文件夹解压到本地目录，点击vnc中的传输按钮，如下图

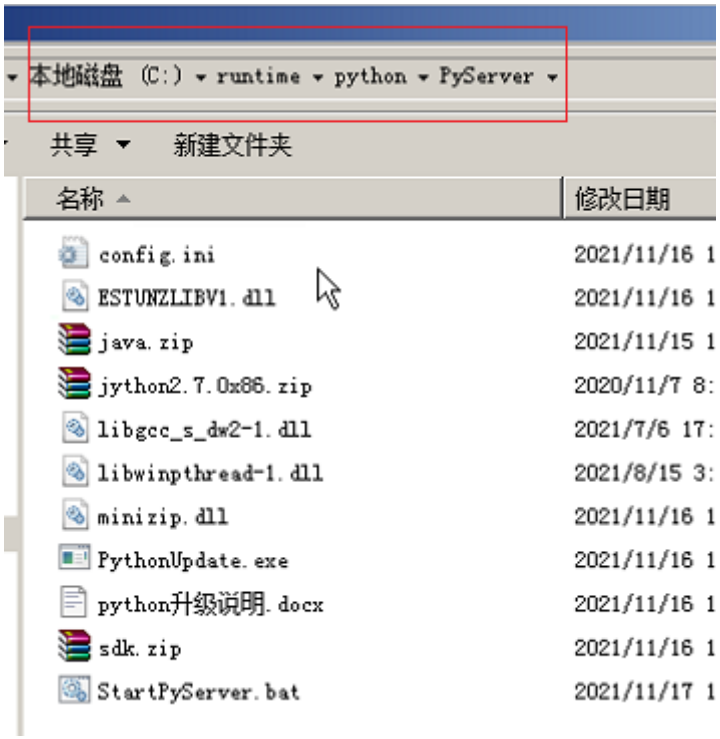


Vnc传输菜单

选中本地解压的PyServer文件夹，传输到控制器中，如下图所示



PyServer传输

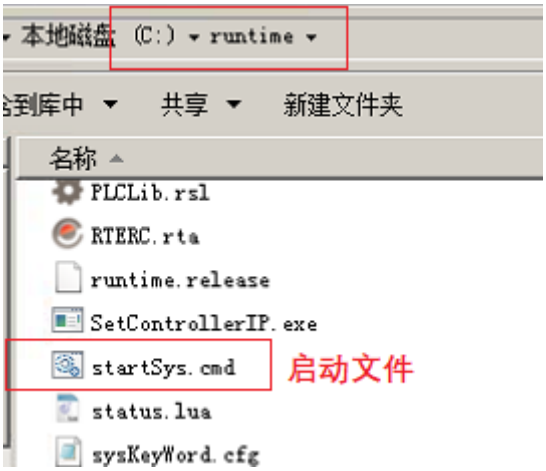


传输后控制器的PyServer目录

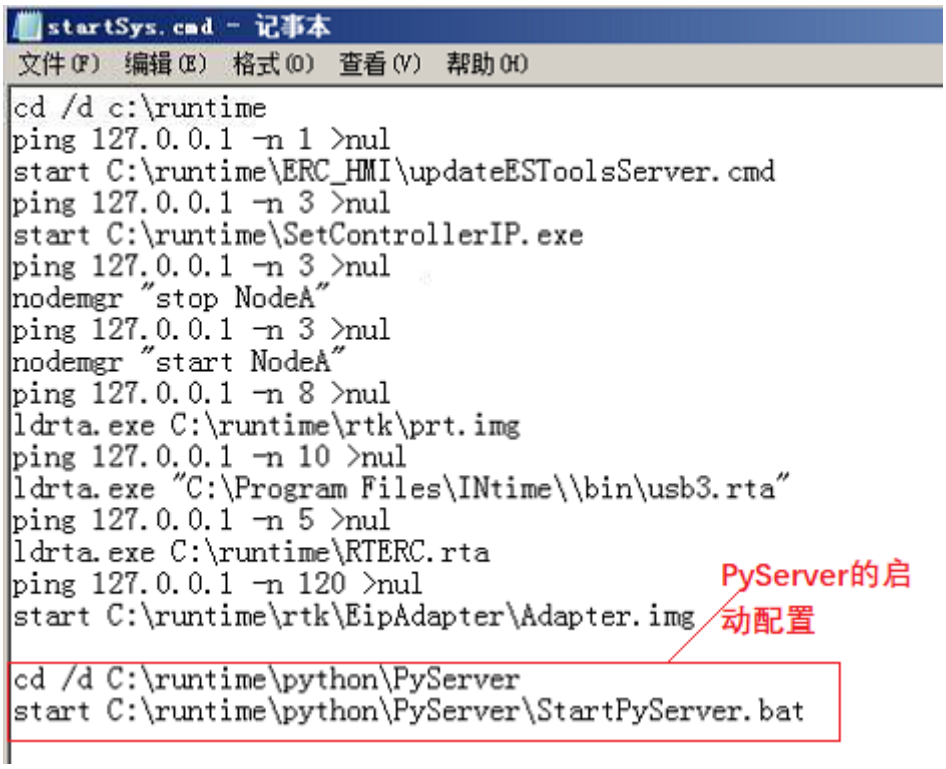
3. 在runtime包里的启动文件中配置PyServer的启动项，添加

   cd /d C:\runtime\python\PyServer

   start C:\runtime\python\PyServer\StartPyServer.bat



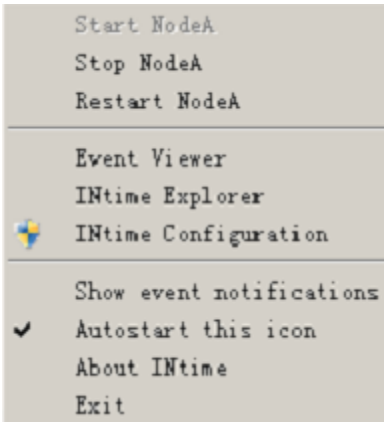
Runtime包里面的启动文件



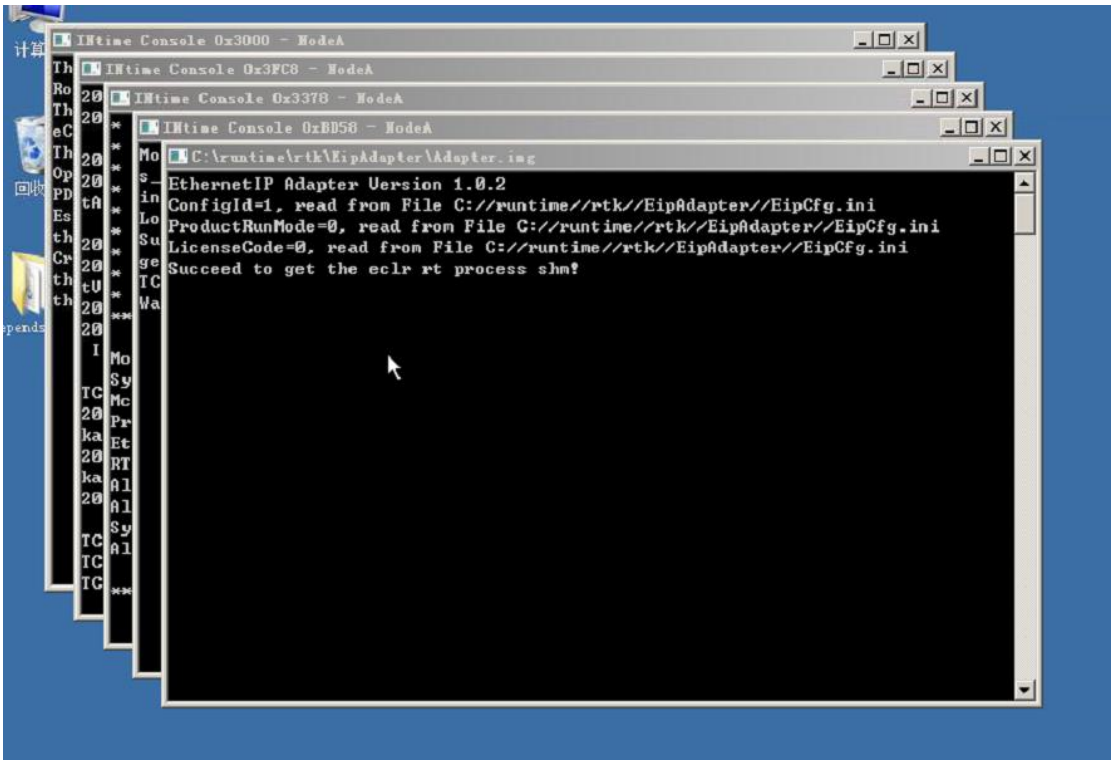
启动文件中添加的内容

4. 保存，关闭intime，以及现有窗口，再启动项中启动statSys.bat文件
5. 关闭intime操作：首先，右击电脑右下角intime图标，在左击出现的Stop NoteA，随后进行所有控制台窗口的关闭。

红色图标为intime图标



选择Stop NoteA



控制台窗口

6. 配置完成，立即生效

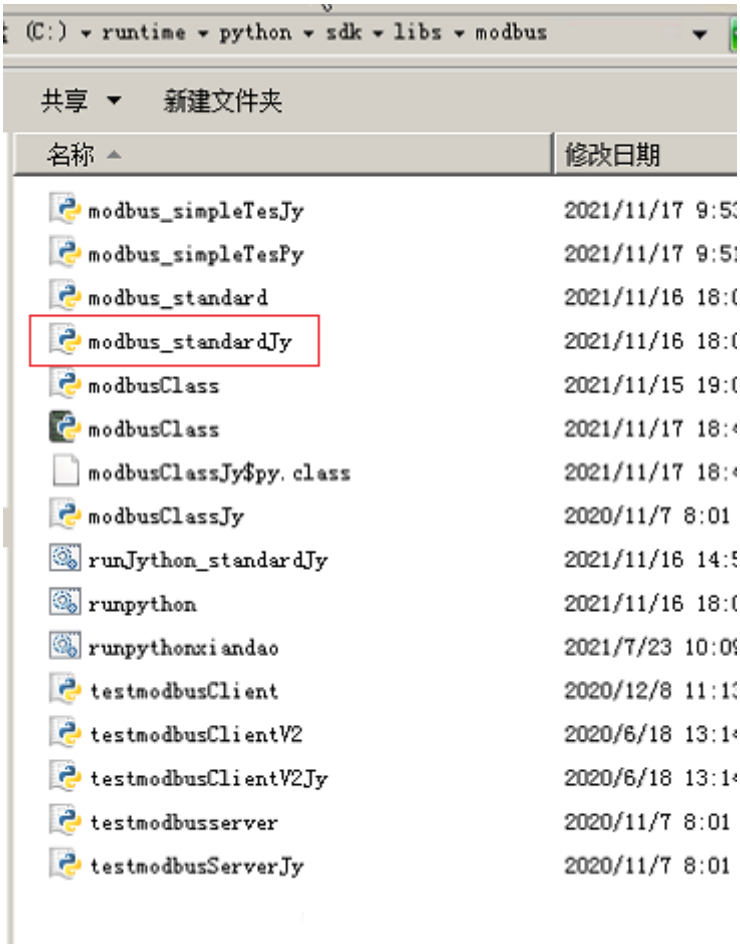
第 2 章 ModbusTcp 篇

2.1 本节概要

介绍如何配置modbus服务端的端口以及启动Modbus服务端。默认的modbusTCP的点表同<<ER系列工业机器人ModBusTCP接口调试手册_RCS2 V1.5>>,除地址40005~40014地址对应的显示当前加载工程名功能、40054~40063地址对应的设置加载工程名功能、40100地址对应的读写标志位请求功能的功能不再支持，其余的和文档一致。

2.2 配置启动脚本的 ModbusTCP

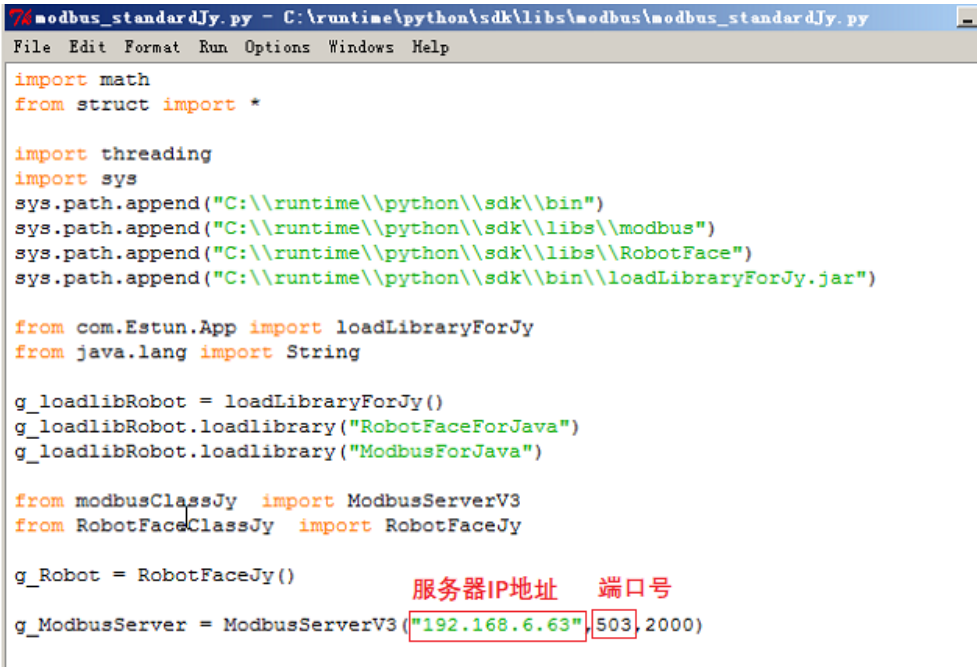
1. 同样使用vnc进入控制器，打开C:\runtime\python\sdk\libs\modbus目录，



Mosbus库目录

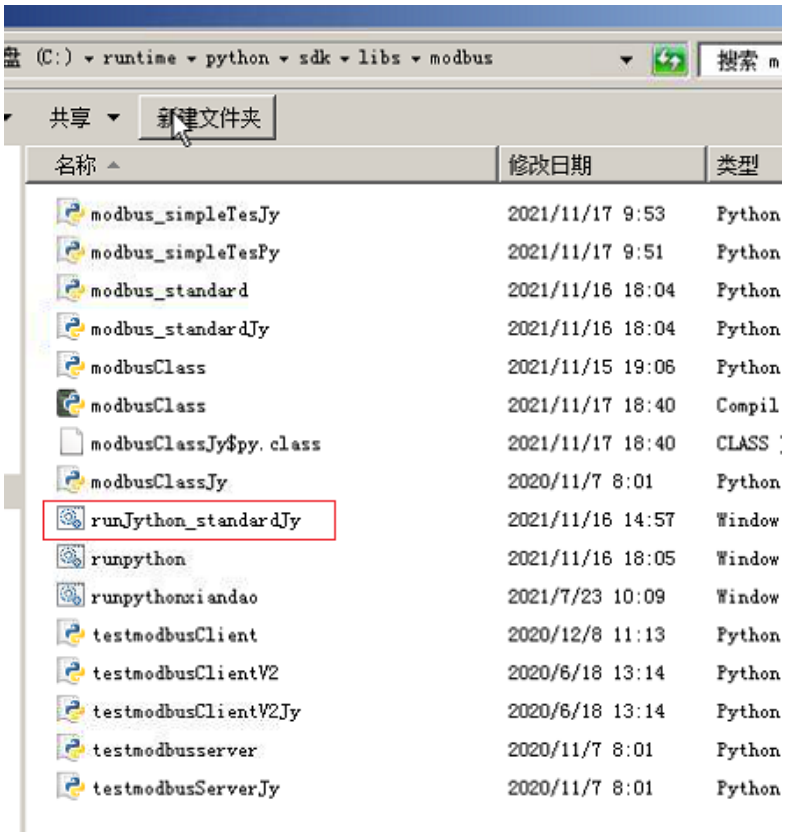
启动modbusTCP需要先配置一下modbusTCP的端口号，打开该目录下的

modbus_standardJy.py文件，如下图修改IP地址为控制器IP地址，端口号为modbusTCP的服务器的端口号

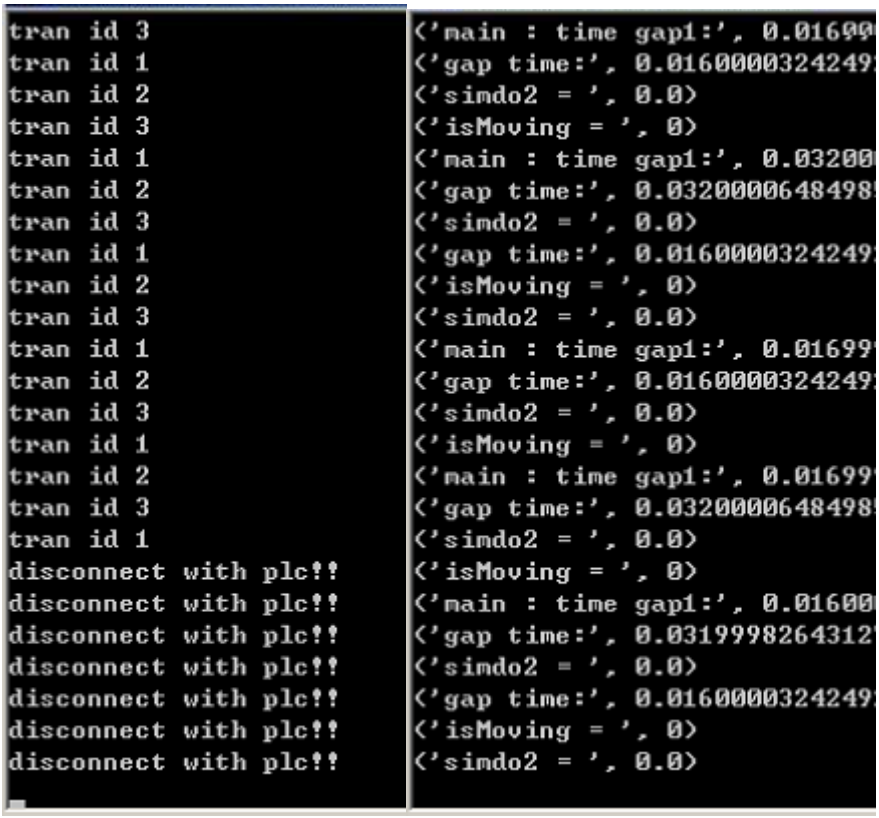


ModbusTCP服务器中的IP地址和端口号

2. 双击运行runJython_standarJy.bat启动文件，启动ModbusTCP



modbusTCP启动文件



Modbus服务器脚本正常运行的打印信息

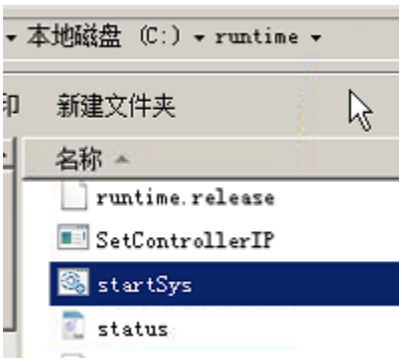
3. 脚本正常运行后，需要把runJython_standarJy.bat加到

   C:\runtime\startSys.bat启动文件中

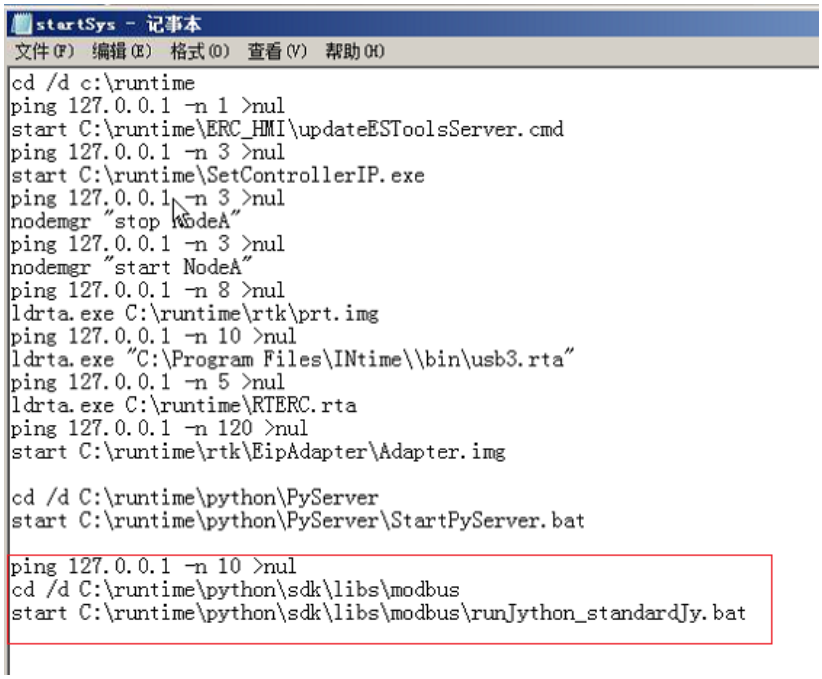
   ping 127.0.0.1 -n 10 >nul

   cd /d C:\runtime\python\sdk\libs\modbus

   start C:\runtime\python\sdk\libs\modbus\runJython_standardJy.bat



Runtime包启动文件



ModbusTCP启动项添加

4. 手动关闭系统，重启双击运行一下startSys.bat.一切正常，即可关机重启。

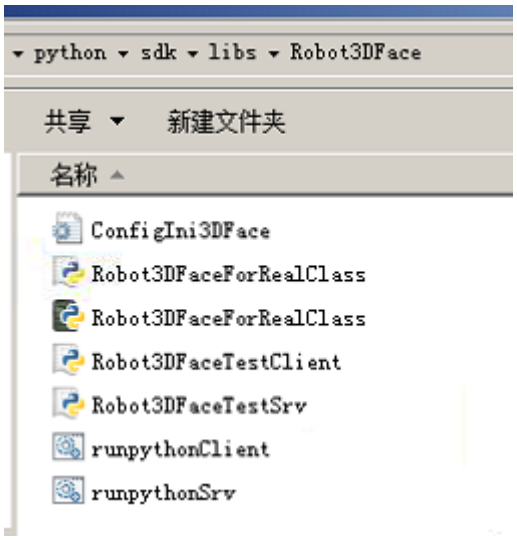
第 3 章 脚本机器人 3D 接口篇

3.1 本节概要

如何配置和启动3d接口的服务端和客户端。3d接口启动后，协议符合文档《ER系列工业机器人三维视觉调试手册(python下Robot3Dface库)_RCS2 V1.5.1》

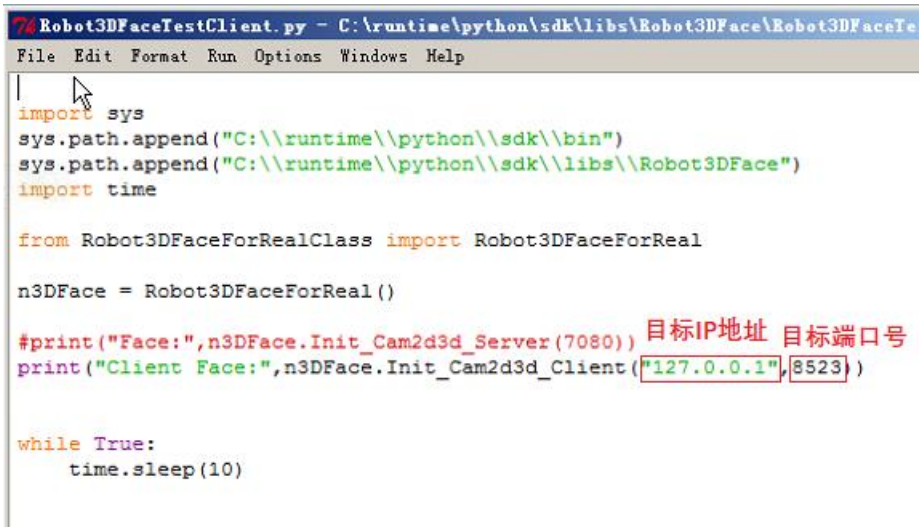
3.2 脚本 3d 接口配置

1. 同样使用 vnc 进入控制器，打开 C:\runtime\python\sdk\libs\Robot3DFace



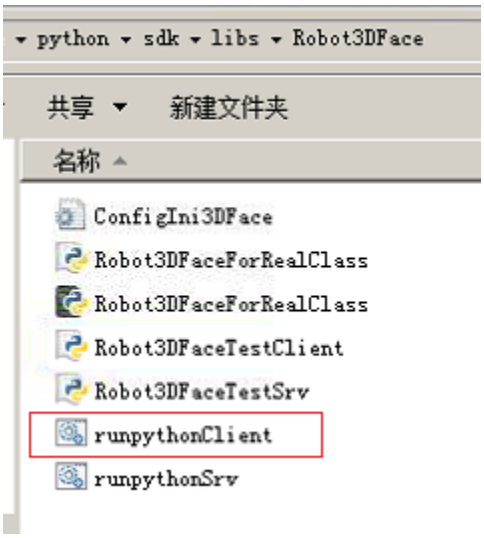
3d 接口目录

打开 Robot3DFaceTestClient.py 文件打开脚本，这是 3d 接口的客户端，需要设置其中的连接的目标的 IP 地址和端口号

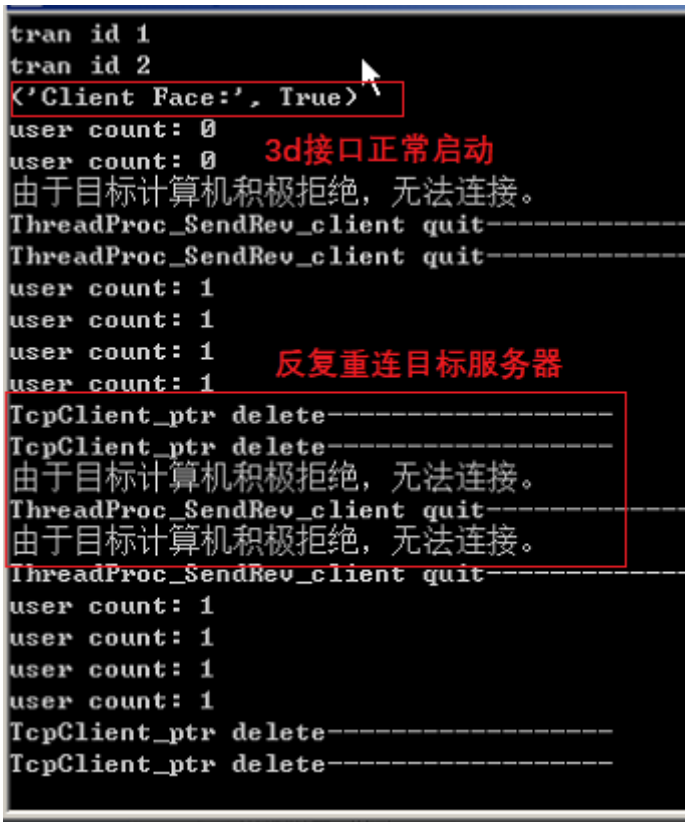


3d 接口客户端 IP 地址和端口号设置

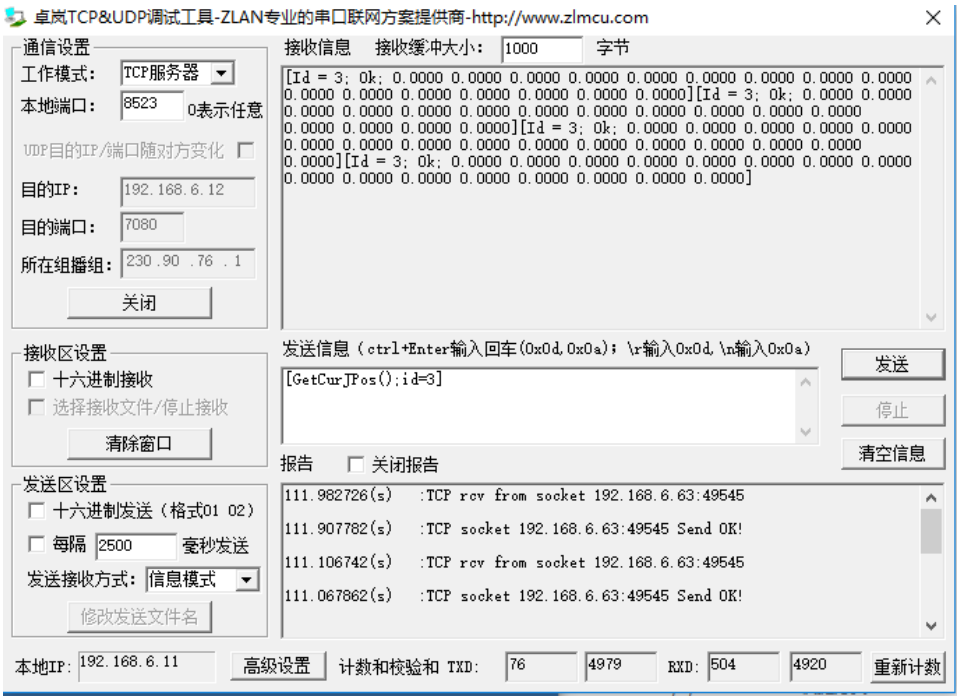
双击运行 runpythonClient.bat，启动 3d 接口客户端的脚本，客户端有自动重连功能



3d 接口启动文件



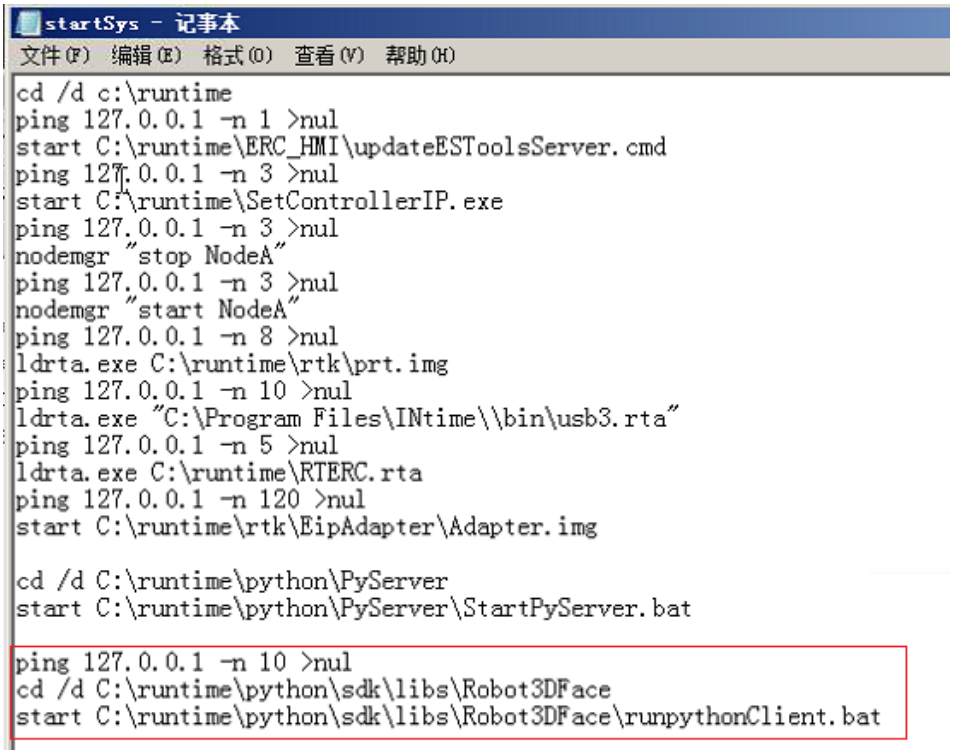
运行打印窗口



测试接口

2. 脚本正常运行后，需要把 runpythonClient.bat 加到 C:\runtime\startSys.bat 文件中

ping 127.0.0.1 -n 10 >nul
cd /d C:\runtime\python\sdk\libs\Robot3DFace
start C:\runtime\python\sdk\libs\ Robot3DFace \runpythonClient.bat

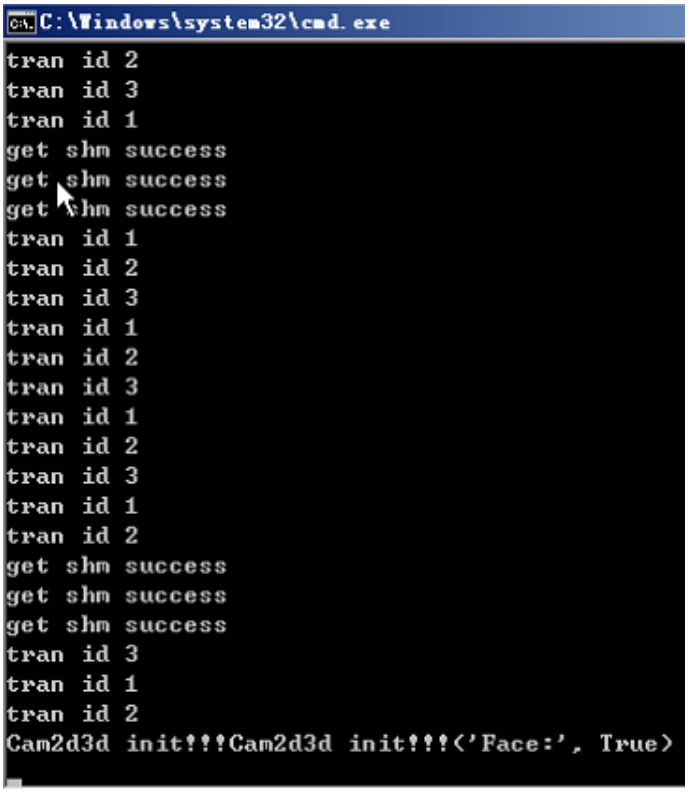


启动配置文件修改

3. 服务端的配置，需要打开 C:\runtime\python\sdk\libs\Robot3DFace 目录下的 Robot3DFaceTestSrv.py

3d 接口服务器脚本

修改脚本的端口号。启动 runpythonsrv.bat 文件。运行脚本

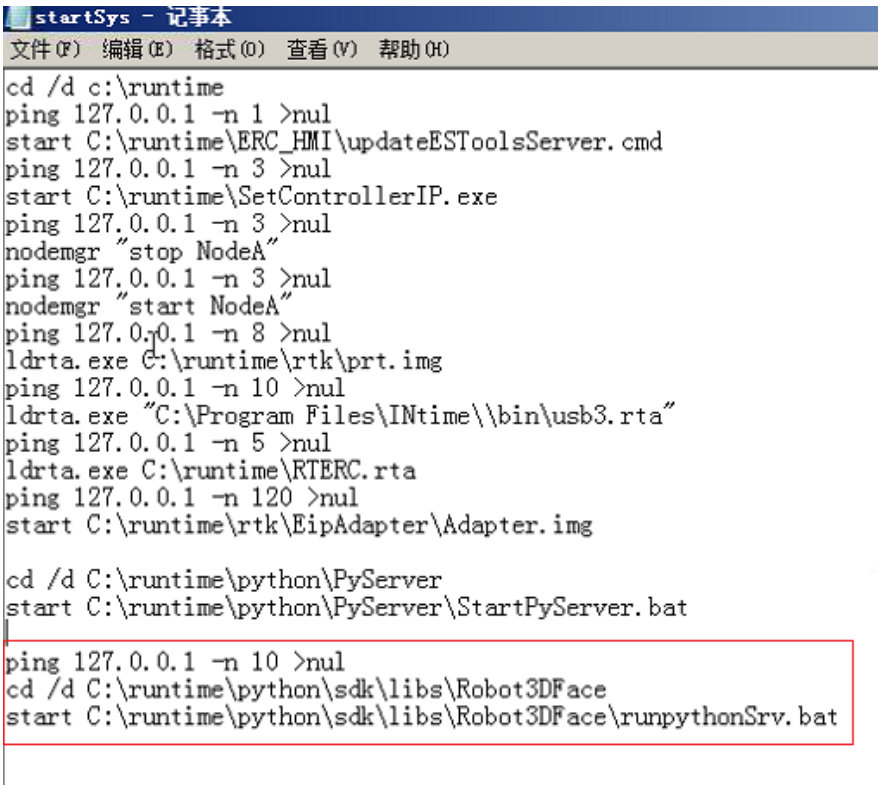


3d 接口服务端脚本

测试 3d 接口服务端

5. 脚本正常运行后，需要把 runpythonSrv.bat 加到启动项中

```
ping 127.0.0.1 -n 10 >nul
cd /d C:\runtime\python\sdk\libs\Robot3DFace
start C:\runtime\python\sdk\libs\ Robot3DFace \runpythonSrv.bat
```



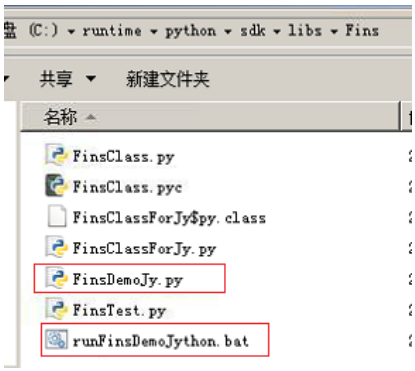
第 4 章 欧姆龙 Fins 协议篇

4.1 本节概要

介绍 Fins 通信模板程序；标准通信程序的配置、启动，标准通信程序是用了 modbusTCP 的点表，只不过地址是 Fins 的地址不是 modbus 地址，这样直接启动，安装点表可以与 PLC 程序通信。

4.2 Fins 通信模板程序

1. 同样进入 vnc，打开 C:\runtime\python\sdk\libs\Fins 目录



脚本 Fins 目录

打开 FinsDemoJy.py 文件，在自定义逻辑区就用 Fins 接口类写自己的逻辑了。

FinsDemo 模板

在通信对象创建的时候需要指定 PLC 的 IP 地址和端口号，以及本机 IP 地址的最后一位（例如 192.168.6.63 是控制器 IP 地址，这里就是本机 63）

```
g_finsclient = FinsTcpV2("192.168.6.100",9600,63)
```

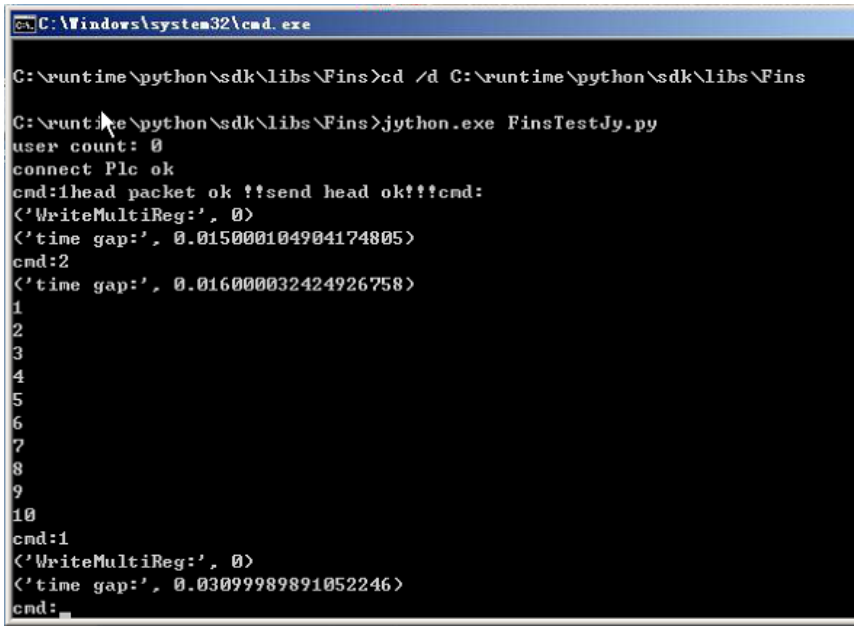
Fins 的 IP 地址和端口号以及本机 IP

```
FinsClassForJy.py - C:\runtime\python\sdk\libs\Fins\FinsClassForJy.py
File  Edit  Format  Run  Options  Windows  Help

import sys
sys.path.append("C:\\runtime\\python\\sdk\\bin\\FinsLibForJava.j


from com.Estun.App.FinslibV2 import FinsTcpForJyV2
from com.Estun.App.FinslibV2 import FinsUdpForJyV2
from com.Estun.App.FinslibV2 import vectors_v3_int

class FinsTcpV2:
    m_TcpFunction = FinsTcpForJyV2()

    def __init__(self,IpAddr,port,localIp):
        self.m_TcpFunction.initConfig(IpAddr,port,localIp)
        return

    def ReadSingleReg(self, addr):
        return self.m_TcpFunction.ReadSingleReg(addr)

    def WriteSingleReg(self, addr , valuedata):
        return self.m_TcpFunction.WriteSingleReg(addr,valuedata)

    #
    # return value : [unsigned short , ..... ]
    #
    def ReadMultiReg(self, addr, regNums):
        tmpdata = self.m_TcpFunction.ReadMultiReg(addr,regNums)
        res = []
        for n in tmpdata:
            res.append(n)
        return res

    #
    # dataInValues : [unsigned short , ..... ]
    #
    def WriteMultiReg(self, addr, regNums, dataInValues):
```

Fins 通信的接口类

接口类在文件 FinsClassForJy.py 中。

2. 运行 runFinsDemoJython.bat,就可以启动 demo 程序的脚本

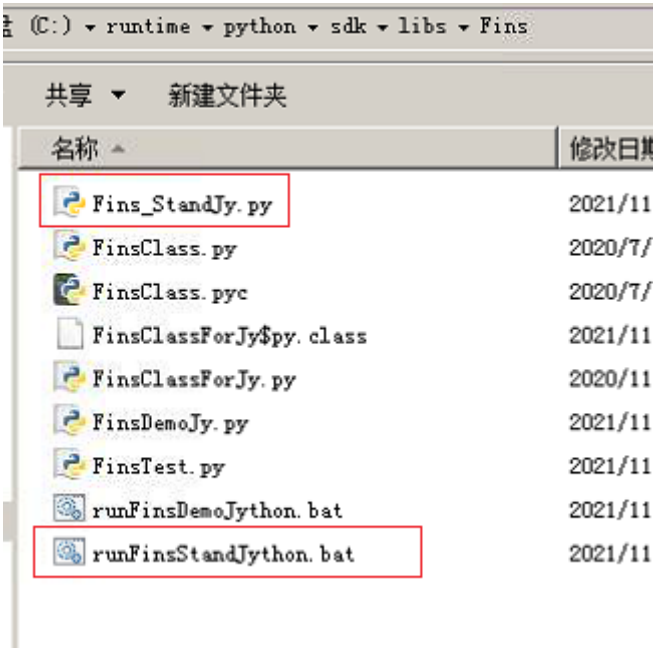


FinsDemo 测试程序运行窗口

3. 脚本正常运行后，需要把 runFinsDemoJython.bat 加到 c:\runtime\startSys.bat 中和上一章节都一样。这里不再重复

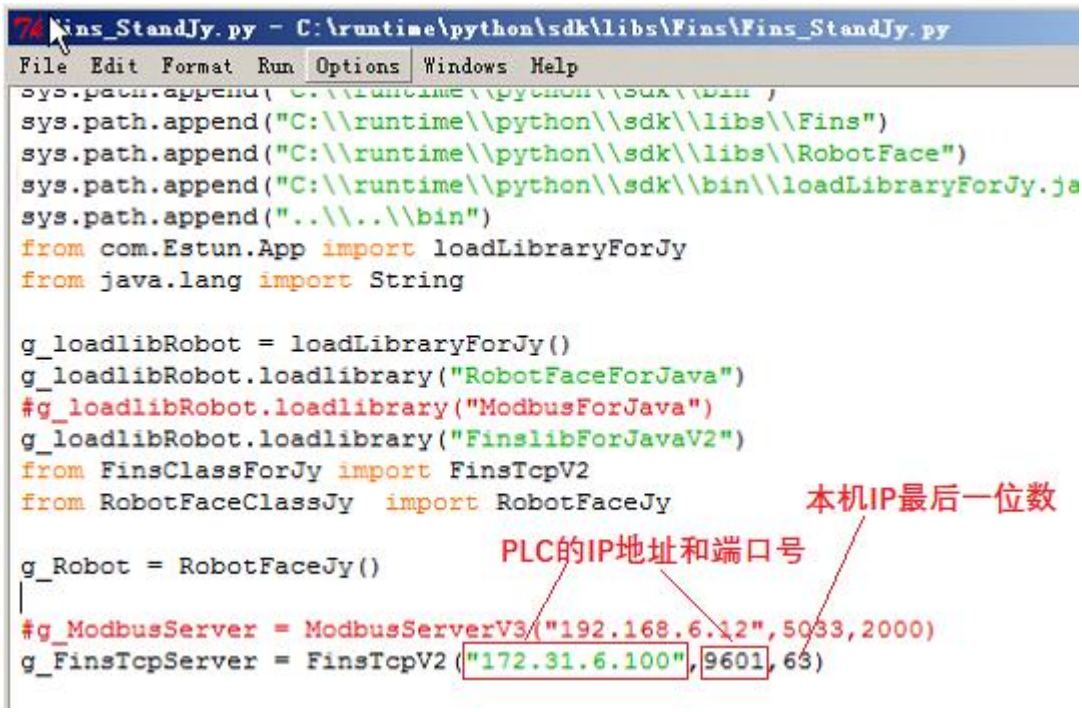
4.3 标准通信程序

6. 1. 同样使用vnc进入控制器，打开C:\runtime\python\sdk\libs\Fins目录，



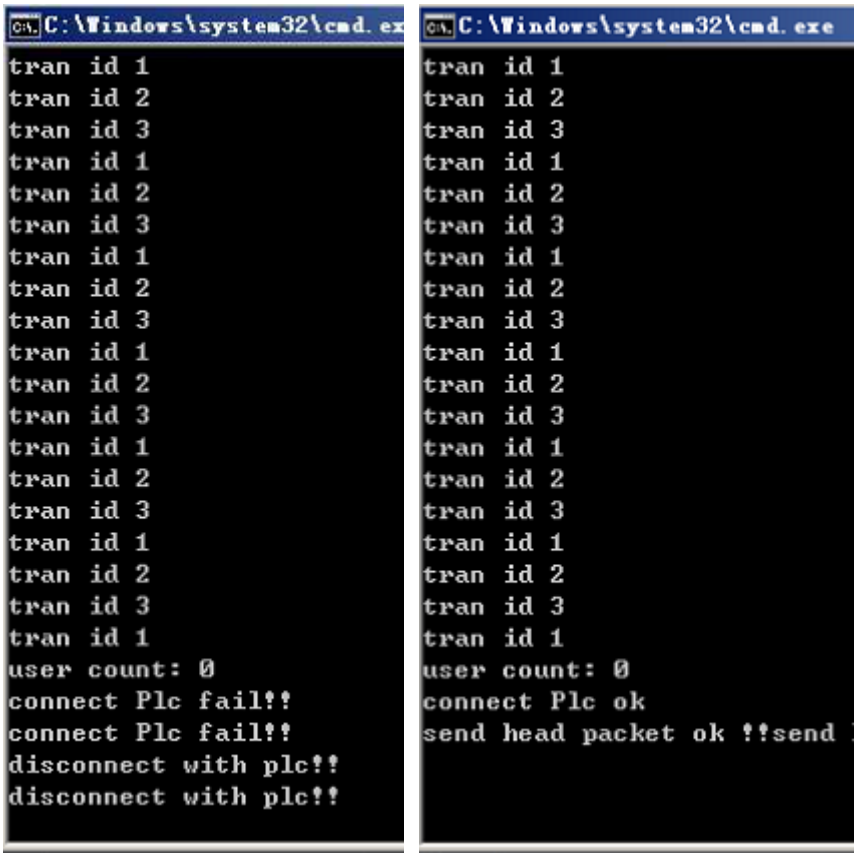
Fins库目录

启动Fins标准通信需要先配置一下PLC的IP地址和端口号，打开该目录下的Fins_StandJy.py文件，如下图修改IP地址为PLC的IP地址，端口号为PLC的端口号



PLC的IP地址和端口号以及本机IP

7. 双击运行runFinsStandJy.bat启动文件，启动Fins标准通信脚本



Fins标准通信脚本正常运行的打印信息

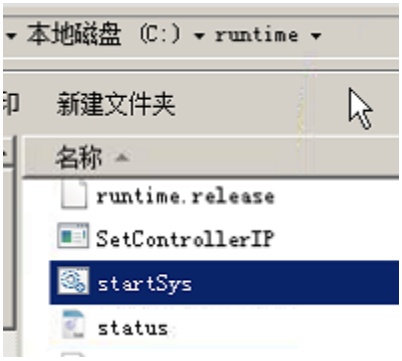
8. 脚本正常运行后，需要把runFinsStandJy.bat加到C:\runtime\startSys.bat启

动文件中

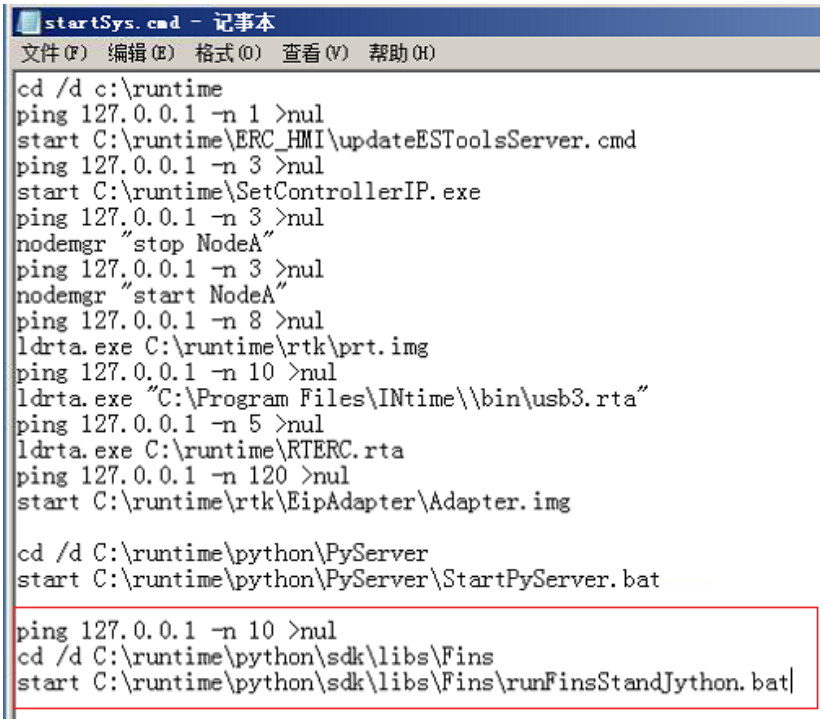
ping 127.0.0.1 -n 8 >nul

cd /d C:\runtime\python\sdk\libs\Fins

start C:\runtime\python\sdk\libs\Fins\runFinsStandJython.bat



Runtime包启动文件



Fins标准通信脚本启动项添加

手动关闭系统，重启双机运行一下 startSys.bat.一切正常，即可关机重启。

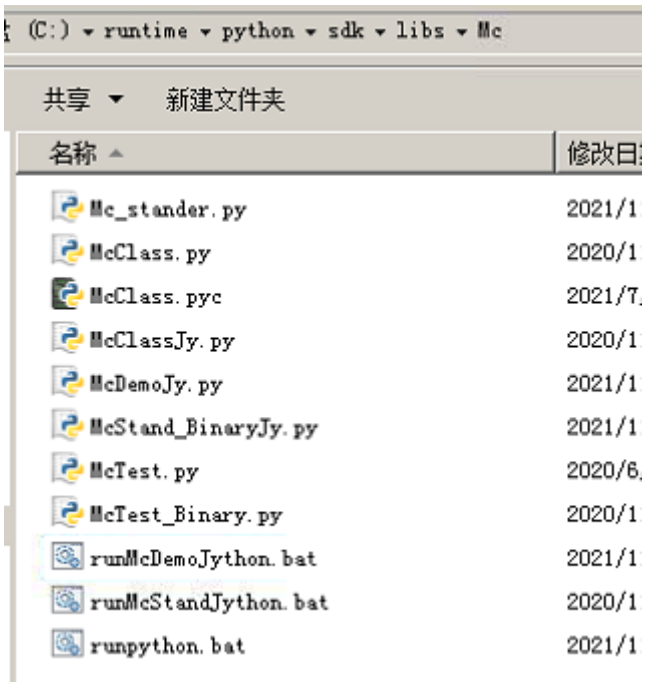
第 5 章 三菱 Mc 协议篇

5.1 本节概要

介绍 Mc 通信模板程序；标准通信程序的配置、启动，标准通信程序是用了 modbusTCP 的点表，只不过地址是 Mc 的地址不是 modbus 地址，这样直接启动，安装点表可以与 PLC 程序通信。

5.2 Mc 通信模板程序

1. 同样进入 vnc，打开 C:\runtime\python\sdk\libs\Mc 目录



脚本 Mc 目录

打开 McDemoJy.py 文件，在自定义逻辑区用 Mc 接口类写自己的逻辑了

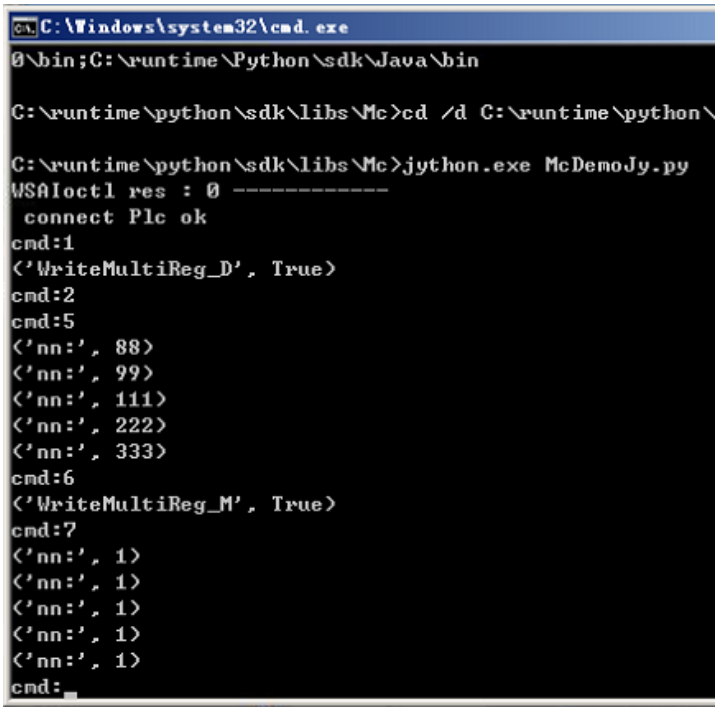
三菱McDemo程序

在创建通信对象的时候，需要指定PLC的IP地址和端口号。

```
g_ff = McClientTcpJy("192.168.6.200",5300)
```

通信IP地址和端口号

2. 运行runMcDemoJython.bat,启动Mcdemo的脚本，运行效果如下

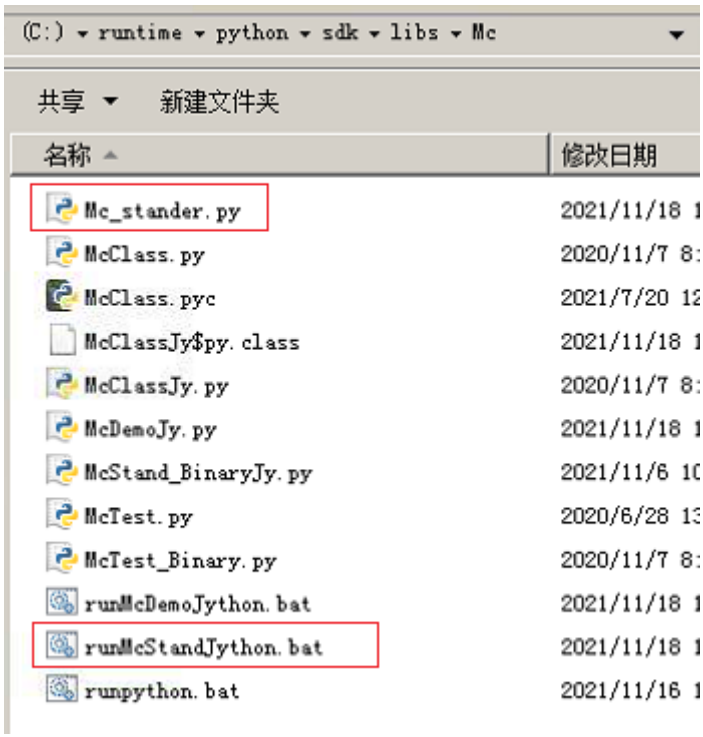


McDemo测试程序脚本运行

3. 脚本正常运行后，需要把runMcDemoJython.bat加到c:\runtime\startSys.bat中，同上面章节一致，这里不再重复

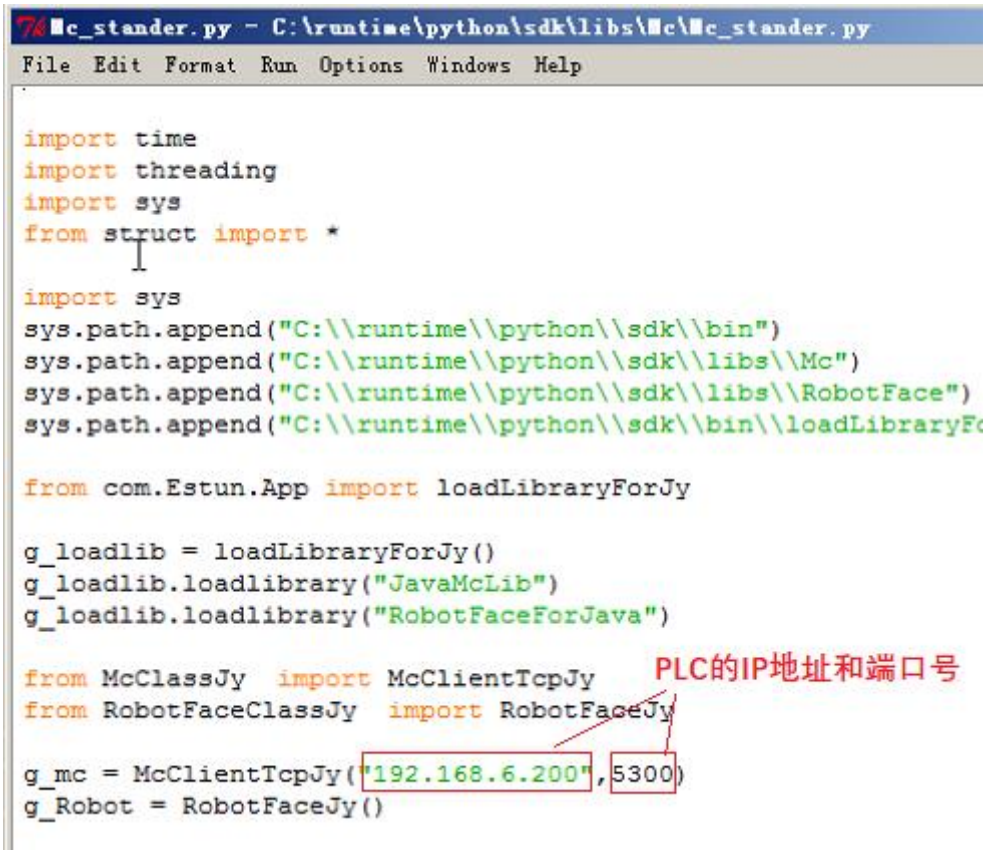
5.3 Mc 标准通信程序

1. 同样使用vnc进入控制器，打开C:\runtime\python\sdk\libs\Mc目录，



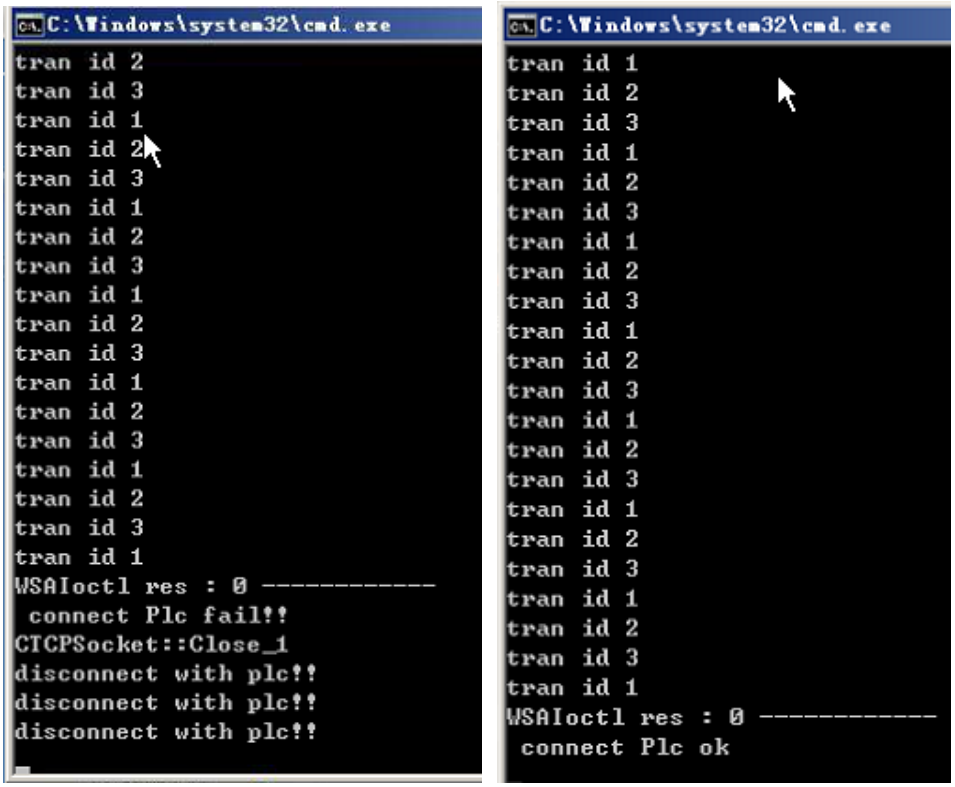
Mc库目录

启动Mc标准通信程序需要先配置一下PLC的Ip地址和端口号，打开该目录下的Mc_stander.py文件，如下图修改IP地址为PLC的IP地址，端口号为PLC的服务器的端口号



Mc的PLC的IP地址和端口号

9. 双击运行runMcStandJython.bat启动文件，启动Mc标准通信脚本程序



Mc标准通信脚本正常运行的打印信息

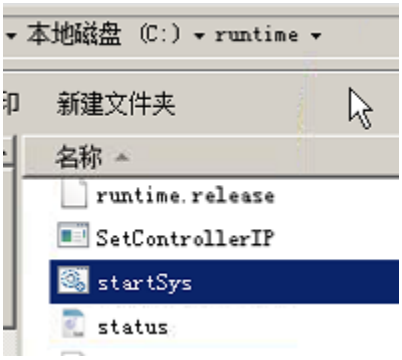
10. 脚本正常运行后，需要把runMcStandJython.bat加到

C:\runtime\startSys.bat启动文件中

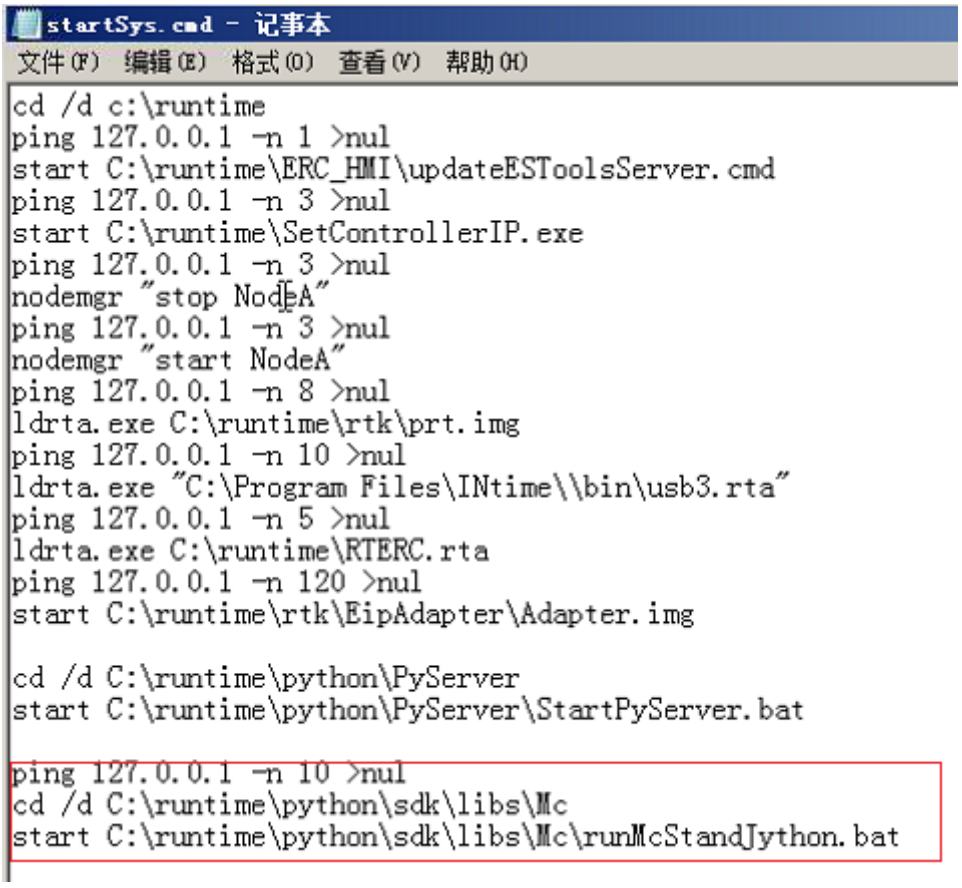
ping 127.0.0.1 -n 10 >nul

cd /d C:\runtime\python\sdk\libs\Mc

start C:\runtime\python\sdk\libs\Mc\runMcStandJython.bat



Runtime包启动文件



Mc标准启动项添加

手动关闭系统，重启双机运行一下 startSys.bat.一切正常，即可关机重启。。

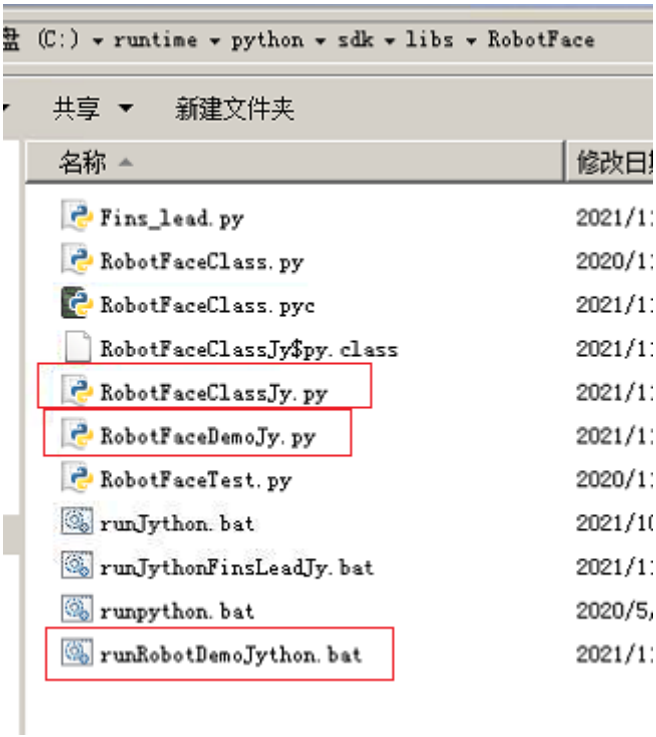
第 6 章 机器人接口篇

6.1 本节概要

介绍机器人接口的使用，和程序模板。

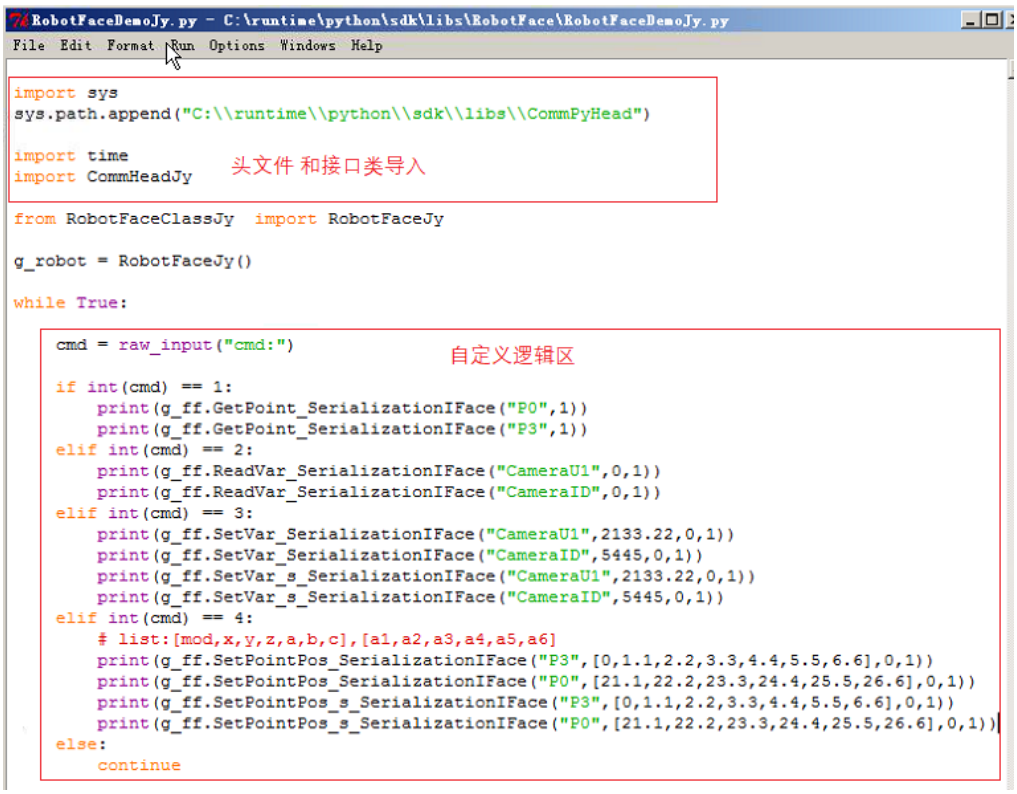
6.2 机器人接口 demo 例程

1. 同样进入vnc，打开C:\runtime\python\sdk\libs\RobotFace目录



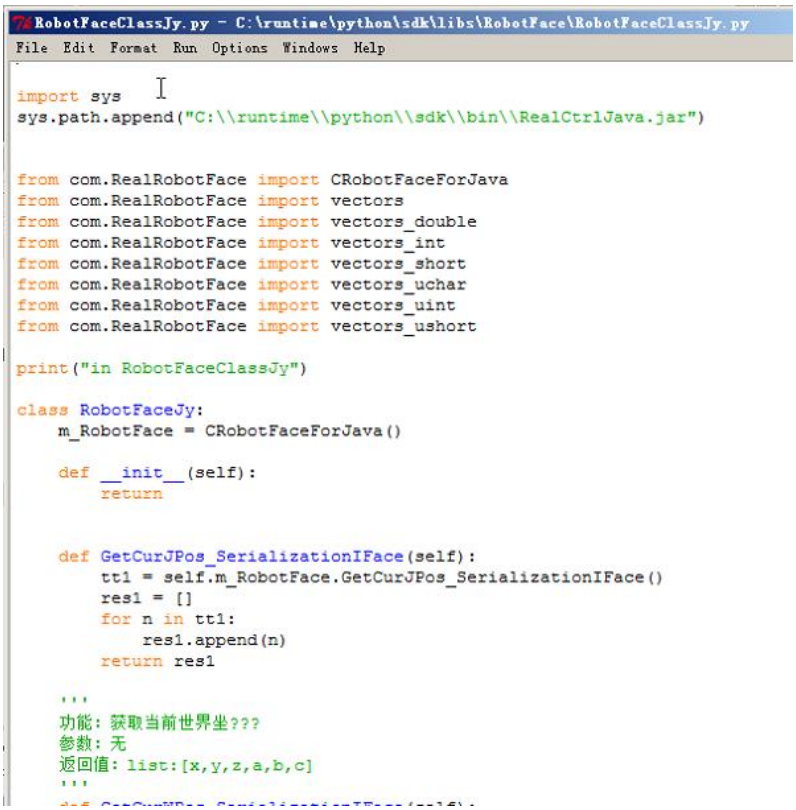
机器人接口目录

打开RobotFaceDemoJy.Py文件，在自定义逻辑区用Robot接口写程序



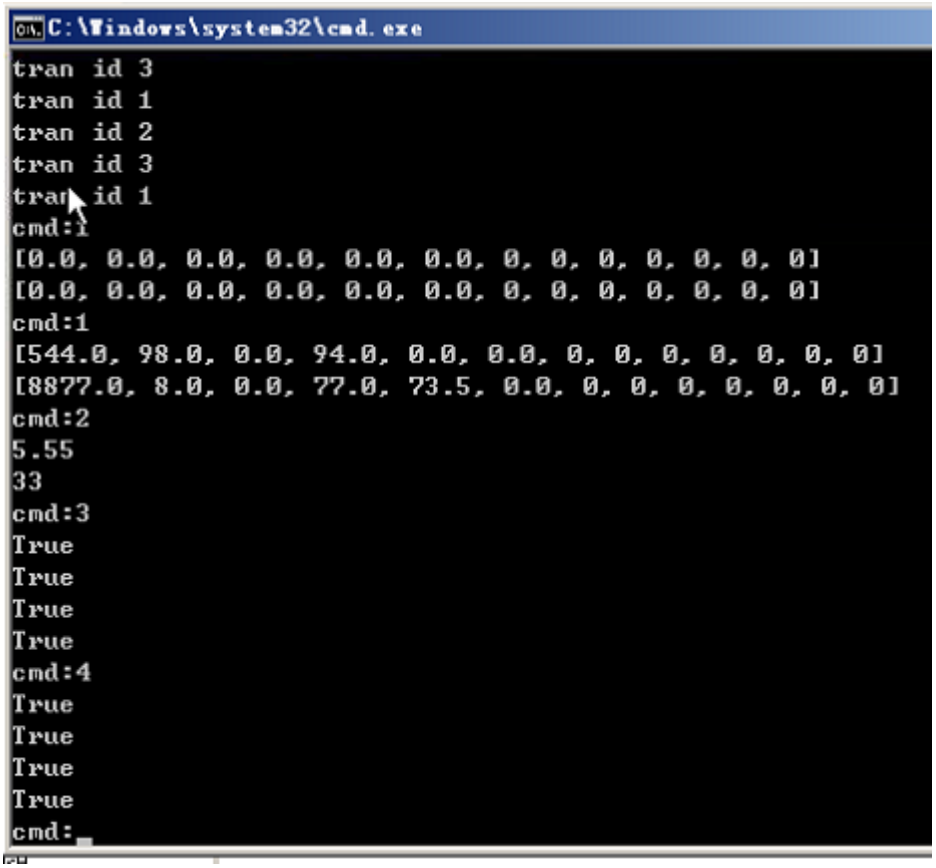
RobotFace的demo程序

2. 机器人的接口类比较多在文件RobotFaceClassJy.py文件中，跟以前的接口是一致的



机器人接口类

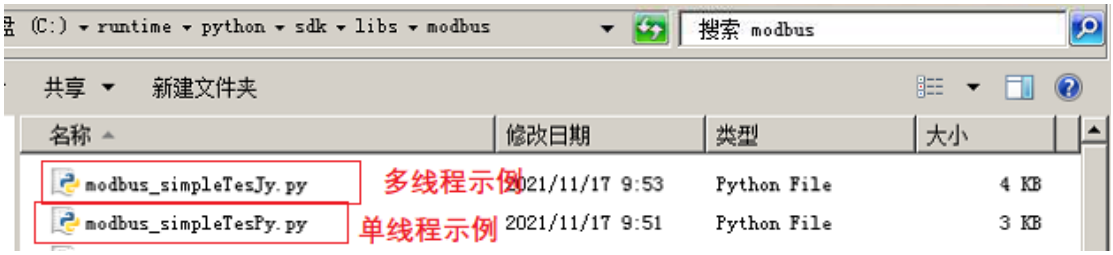
3. 运行runRobotDemoJython.bat，启动测试程序。



4. 脚本正常运行后，需要把runRobotDemoJython.bat加到c:\runtime\startSys.bat中，操作同上面章节一致，这样每次开机脚本都会启动

6.3 Java 下 Python 多线程编程示例

示例代码是在 c:\runtime\python\sdk\lib\modbus 目录下



Jython 多线程 demo

下面的代码是有两个操作

第一操作，读取机器人的速度、伺服状态、运行状态，把读取的值设置到 PLC 寄存器上。

第二操作，读取 PLC 寄存器的值，取上升沿，调用机器人的启动、停止、复位命令。

由于操作是顺序执行的，执行机器人接口的时候，PLC 的通信接口就必须等待；反之，执行 PLC 通信接口时，机器人接口就无法调用。

```
while True:
    t1 = time.time()

    #global readWritePlcError
    if g_ModbusServer.isConnected() == True:

        t1 = time.time()
        # ---------------过程1-----------------------
        # 设置PLC寄存器10号、11号、12号地址为速度、伺服状态、运行状态
        # 获得速度
        CurSpeed = g_Robot.GetGlobalSpeed_SerializationIFace()
        # 获得伺服状态
        servoStr = g_Robot.GetServoSts_SerializationIFace()
        # 获得运行状态
        sysState = g_Robot.CurSysMode_SerializationIFace()
        # 设置寄存器值
        g_ModbusServer.setMultiRegs(10,3,3,[int(CurSpeed), \
                        int(servoStr),int(sysState)])

        # ---------------过程2-----------------------
        # 读PLC寄存器20号、21号、22号地址值取上升沿，分别启动、停止、复位错误
        # 读PLC寄存器
        Reg20_23 = g_ModbusServer.getMultiRegs(20, 3, 3)
        startProgramRegister = Reg20_23[0]
        stopProgramRegister = Reg20_23[1]
        resetRobotStateRegister = Reg20_23[2]

        # 取上升沿，发送启动命令
        if startProgram == 0 and startProgramRegister == 1:
            g_Robot.StartRun_SerializationIFace()
        startProgram = startProgramRegister

        # 取上升沿，发送停止命令
        if stopProgram == 0 and stopProgramRegister == 1:
            g_Robot.StopRun_SerializationIFace()
        stopProgram = stopProgramRegister

        # # 取上升沿，发送复位命令
        if resetRobotState == 0 and resetRobotStateRegister == 1:
            g_Robot.rstError_SerializationIFace()
        resetRobotState = resetRobotStateRegister

        print("gap time:",time.time()-t1,"---------------")

    time.sleep(0.002)
```

使用 Jython 多线程修改该代码示例

第一步：把 2 个过程分解中的通信部分单独写一个线程，交互的数据使用全局变量保存，当其它线程需要 PLC 数据时从全局变量里取数据。当其它线程需要向 PLC 中设置数据时把数据放到全局变量里面，通信线程会取全局数据设置到 PLC 数据的。

```
setPLCData = [0] * 100
getPLCData = [0] * 100

# --------------过程0----------------------
# PLC的modbus通信,全局变量setPLCData和getPLCData是交互的值
def ReadWritePLC():
    global setPLCData
    global getPLCData

    while True:
        if g_ModbusServer.isConnected() == True:
            # 一次读10个寄存器值
            g_ModbusServer.setMultiRegs(10,10,3,setPLCData)
            # 一次写10个寄存器值
            getPLCData = g_ModbusServer.getMultiRegs(20, 10, 3)
        time.sleep(0.002)
```

```
thread0 = threading.Thread(target=ReadWritePLC)
thread0.setDaemon(True)
thread0.start()
```

第二步，把过程 1 写成一个线程，机器人的状态会在该线程中刷新，刷新后数据放到 PLC 的写全局变量里面，由通信线程把数据设置到 PLC 上

```
# --------------过程1----------------------
# 设置PLC寄存器10号、11号、12号地址为速度、伺服状态、运行状态
# 赋值到写PLC上去
def ReadRobotStatus():
    global CurSpeed
    global servoStr
    global sysState
    global setPLCData

    while True:
        # 获得速度
        CurSpeed = g_Robot.GetGlobalSpeed_SerializationIFace()
        # 获得伺服状态
        servoStr = g_Robot.GetServoSts_SerializationIFace()
        # 获得运行状态
        sysState = g_Robot.CurSysMode_SerializationIFace()

        setPLCData[10] = CurSpeed
        setPLCData[11] = servoStr
        setPLCData[12] = sysState
        time.sleep(0.002)
```

```
thread1 = threading.Thread(target=ReadRobotStatus)
thread1.setDaemon(True)
thread1.start()
```

第三步，把过程 2 写成一个线程，不断检测 PLC 数据的全局变量，全局变量的数据值，由通信线程刷新。该线程检测到全局变量有上述沿，会执行机器人控制命令。

```
# ---------------过程2-----------------------
# 读PLC寄存器20号、21号、22号地址值取上升沿，分别启动、停止、复位错误
# 读PLC寄存器
def SetRobotCmd():
    global getPLCData
    startProgram = 0;
    stopProgram = 0;
    resetRobotState = 0;

    while True:
        startProgramRegister = getPLCData[0]
        stopProgramRegister = getPLCData[1]
        resetRobotStateRegister = getPLCData[2]

        # 取上升沿，发送启动命令
        if startProgram == 0 and startProgramRegister == 1:
            g_Robot.StartRun_SerializationIFace()
        startProgram = startProgramRegister

        # 取上升沿，发送停止命令
        if stopProgram == 0 and stopProgramRegister == 1:
            g_Robot.StopRun_SerializationIFace()
        stopProgram = stopProgramRegister

        # # 取上升沿，发送复位命令
        if resetRobotState == 0 and resetRobotStateRegister == 1:
            g_Robot.rstError_SerializationIFace()
        resetRobotState = resetRobotStateRegister
        time.sleep(0.002)
```

```
thread2 = threading.Thread(target=SetRobotCmd)
thread2.setDaemon(True)
thread2.start()
```

说明：原来的1个线程变成3个线程，因为Jython是正真的多线程并行运行，所以效率会有很大提升。综合现场反馈，效率比原来提升1倍。