西门子（中国）总代理S7-400PLC供应6ES7400-1JA01-0AA0

产品名称	西门子（中国）总代理S7-400PLC供应6ES7400-1JA01-0AA0
公司名称	湖南西控自动化设备有限公司
价格	.00/件
规格参数	西门子:授权代理商 模块:一级代理商 德国:售后保障服务
公司地址	中国（湖南）自由贸易试验区长沙片区开元东路1306号开阳智能制造产业园（一期）4#栋301
联系电话	17838383235 17838383235

产品详情

使用 Python 通过 ModbusTCP 连接 PLC（不限品牌 含示例程序）

引言

在现代工业自动化系统中，PLC（Programmable Logic Controller，可编程逻辑控制器）被广泛应用于监控和控制各种设备和过程。而与之配套的通信协议也是至关重要的。其中，Modbus TCP 协议作为一种常见的通信协议，被广泛应用于工业领域。

Modbus TCP 协议基于 TCP/IP 协议栈，并使用简单易懂的命令格式，使得各种设备和系统可以方便地进行数据交换。而 Python 作为一门灵活且功能强大的编程语言，在工控领域中也越来越受欢迎。

本文旨在介绍如何使用 Python 通过 Modbus TCP 连接 PLC，并实现对其读写操作的方式。我们将会介绍常用的 Python Modbus 库，并提供一个实际案例来展示其具体应用。通过本文的学习，读者将能够掌握使用 Python 与 PLC 进行数据交互的基本技巧，进一步开发出更加智能和灵活的工控系统。

值得注意的是，在实际应用中，安全性和异常处理也是非常重要的考虑因素。因此，我们还将提醒读者在使用 Python 连接 PLC 时，注意网络安全风险，并妥善处理异常情况，以确保系统的可靠性和稳定性。

通过本文的阅读，读者将进一步了解 Python 与 Modbus TCP 协议之间的结合，为工业自动化系统的开发和维护提供更加便利的解决方案。

希望本文能给读者带来启发和帮助，让我们一同深入了解如何使用 Python 通过 Modbus TCP 连接 PLC

吧！

2

Modbus TCP 简介

Modbus TCP 协议是 Modbus 协议的一种变种，基于 TCP/IP 协议栈进行通信。它是一种开放的通信协议，被广泛应用于工业自动化系统中，用于设备之间的数据交换和控制。

Modbus TCP 协议具有以下特点和优势：

简单易懂：Modbus TCP 采用简单的命令格式，使得不同设备和系统可以轻松地实现数据交互。它使用 16 位寄存器地址来表示设备内部的数据，通过读取和写入这些寄存器，可以实现对设备的控制和监控。

可靠性高：通过 TCP/IP 协议栈的传输机制，Modbus TCP 能够保证数据的可靠传输。TCP 提供了可靠的连接和错误检测机制，确保数据的完整性和准确性。

扩展性强：Modbus TCP 可以支持多个设备同时与一个主站进行通信，灵活应对各种复杂的工业场景。此外，Modbus TCP 还支持主从结构和广播通信，可以满足不同的通信需求。

平台独立性：由于 Modbus TCP 是基于 TCP/IP 协议的，因此它可以在不同的平台上实现，包括 Windows、Linux 等操作系统，以及各种硬件平台。

Python 作为一种流行的编程语言，提供了丰富的工具和库，使得使用 Modbus TCP 协议与 PLC 进行通信变得更加容易。通过几行简洁的 Python 代码，我们就可以实现对 PLC 的读写操作，从而实现设备的控制和数据采集。

在下面的章节中，我们将介绍常用的 Python Modbus 库，并提供示例代码来演示如何使用 Python 通过 Modbus TCP 连接 PLC 并进行数据交互。

3

Python 的 Modbus 库

在 Python 中，有一些常用的 Modbus 库可以帮助我们实现与 PLC 的通信。下面介绍一个常用的库：

pymodbus：pymodbus 是一个纯 Python 编写的 Modbus 库，提供了基于 TCP 和串口（RTU/ASCII）的 Modbus 通信功能。它支持 Modbus TCP、Modbus RTU 和 Modbus ASCII 三种传输模式，并提供了丰富的函数接口，使得读写操作变得简单方便。您可以使用 pip 安装 pymodbus 库：

```
pip install pymodbus
```

以下是一个使用 pymodbus 库读取保持寄存器数据的示例代码：

```
from pymodbus.client.sync import ModbusTcpClient
# 创建Modbus TCP客户端client = ModbusTcpClient('192.168.0.1') # 连接到PLCclient.connect()
# 读取保持寄存器数据result = client.read_holding_registers(address=0, count=10, unit=1) # 处理返
回结果if result.isError():    print("读取失败：{}".format(result))else:    data = result.registers
   print("读取成功：{}".format(data)) # 关闭连接client.close()
```

4

连接 PLC 的步骤

确定 PLC 的网络配置：首先，您需要获取 PLC 的 IP 地址和端口号。这些信息通常在 PLC 的文档或配置界面中可以找到，如图4.1和4.2所示。确保您的计算机和 PLC 在同一个局域网中，并且可以互相访问，可以通过PING指令进行测试。

图4.1

图4.2

安装必要的库和驱动程序：在使用 Python 与 PLC 进行通信之前，您可能需要安装一些必要的库和驱动程序。例如，在使用 Modbus TCP 通信时，您需要安装相应的 Modbus 库（如 pymodbus ）。按照库的文档说明安装和配置。

导入所需的库：在 Python 代码中，您需要导入相应的库以实现与 PLC 的通信。例如，如果您选择使用

pymodbus 库，则需要导入 pymodbus.client.sync 模块来创建 Modbus 客户端。

创建连接：使用所选的库，创建与 PLC 的连接。这通常涉及创建一个客户端对象并指定 PLC 的 IP 地址和端口号。例如，在使用 pymodbus 库时，您可以使用 ModbusTcpClient 类来创建 Modbus TCP 客户端。

连接到 PLC：使用创建的客户端对象，调用连接方法来与 PLC 建立连接。在 Modbus TCP 通信中，这将尝试连接到指定的 IP 地址和端口号。

进行读写操作：一旦连接建立成功，您可以使用相应的函数或方法读取或写入 PLC 的数据。具体的读写操作取决于您使用的 PLC。

关闭连接：在完成与 PLC 的通信后，记得关闭连接以释放资源。通过调用相应的方法（如 close()）来关闭连接。

请注意，上述步骤可能会因不同的 PLC 品牌、型号和通信协议而有所变化。确保仔细阅读 PLC 的文档和相关库的文档，以正确地进行连接和通信。

5

示例案例

当与 PLC 建立连接后，您可以使用 Python 代码进行读取和写入 PLC 的数据。以下是一个示例案例，演示如何使用 pymodbus 库读取和写入 Modbus TCP 通信协议下的保持寄存器数据：

```
from pymodbus.client.sync import ModbusTcpClient
# 创建Modbus TCP客户端client = ModbusTcpClient('192.168.0.22'，502)
# 连接到PLCclient.connect()
# 读取保持寄存器数据result = client.read_holding_registers(address=0, count=10, unit=1) # 处理返
回结果if result.isError():  print("读取失败：{}".format(result))else:  data = result.registers
  print("读取成功：{}".format(data)) # 写入保持寄存器数据write_data = [100, 200, 300, 400,
 500]result = client.write_registers(address=0, values=write_data, unit=1) # 处理返回结果if result.isErr
or():  print("写入失败：{}".format(result))else:  print("写入成功") # 关闭连接client.close()
```

在上述示例中，我们首先使用 ModbusTcpClient 类创建了一个 Modbus TCP 客户端对象，并指定 PLC 的 IP 地址为 192.168.0.1，端口为：502。然后，我们调用 connect()方法连接到 PLC。

接下来，我们使用 read_holding_registers()方法读取保持寄存器的数据，指定起始地址为 0，读取寄存器数为 10，设备地址为 1。读取的结果存储在 result 变量中，通过判断返回结果是否有错误，我们可以判断读取是否成功。如果成功，我们可以通过 registers 属性获取实际的寄存器数据。

然后，我们定义一个要写入保持寄存器的数据列表 write_data，其中包含了一些示例数值。使用 write_registers()方法将该数据写入到 PLC 的保持寄存器中，起始地址也是 0，设备地址为 1。同样地，我们检查返回结果以确定写入是否成功。

后，我们调用 close()方法关闭与 PLC 的连接，释放资源。

请注意，上述示例仅供参考，具体的读写操作和寄存器地址需要根据您的 PLC 和通信协议进行相应的调整。确保阅读相关库的文档和 PLC 的文档以正确地进行读写操作。

6

应用场景

PLC（可编程逻辑控制器）在工业自动化领域中扮演着至关重要的角色。使用 Python 与 PLC 建立连接和进行数据交互，可以实现各种应用场景。以下是几个常见的应用场景：

监控和数据采集：通过与 PLC 建立连接，您可以定期读取传感器数据、监测设备状态并记录生产数据。例如，您可以读取温度、压力、流量等传感器数据，并将其存储到数据库或进行实时监控。

远程控制和调整：借助 Python 与 PLC 的连接，您可以通过发送指令来实现对 PLC 控制的远程操作。例如，您可以编写 Python 代码来控制电机的启停、调整阀门的开闭、修改设备的运行参数等。

自动化生产线控制：使用 Python 与 PLC

通信，您可以实现对生产线的自动化控制。例如，在生产过程中，您可以通过与 PLC 交互来实现批次切换、产品跟踪、设备故障检测和处理等功能，从而提高生产线的效率和灵活性。

能耗管理与优化：通过读取 PLC 中的能耗数据，结合其他环境参数（如温度、湿度等），您可以编写 Python 代码对能源消耗进行监测和分析。这样可以帮助您找出能源浪费的原因，并采取相应措施进行能耗优化。

故障诊断和预测维护：通过实时监测和分析 PLC 中的数据，结合机器学习和数据挖掘技术，您可以构建故障诊断和预测维护系统。这可以帮助您及时发现设备故障、预测设备的寿命，并提前采取维护措施，以减少生产线的停机时间和维修成本。

总结

通过使用 Python 与 PLC 建立连接并进行数据交互，可以实现多种应用场景，包括监控和数据采集、远程控制和调整、自动化生产线控制、能耗管理与优化，以及故障诊断和预测维护等。

总之，Python与Modbus TCP连接PLC的步骤包括安装必要的库、建立连接、执行操作、处理响应数据以及关闭连接。通过这些步骤，你可以使用Python编写代码与PLC进行数据交互，实现对PLC的控制和监控。记得根据PLC的用户手册和Modbus协议规范进行操作，并根据自己的需求进行额外的扩展和处理。