InGateway Documentation

Release 0.0.1

zhangning

Aug 25, 2020

快速入门

CHAPTER 1

InGateway 文档网站导航

Device Supervisor App 为用户提供了便捷且可靠的数据采集、数据二次处理和数据上云等功能，支持 ISO on TCP、ModbusRTU 等多种工业协议解析。如果您想快速实现多种工业协议的数据采集并在本地预处理设备数据，处理过滤后的数据能够经过简单配置即可上传至自建 MQTT 云平台，那么 Device Supervisor 将是您理想的选择。

1.1 Device Supervisor App 用户手册

Device Supervisor App（以下简称 Device Supervisor）为用户提供了便捷的数据采集、数据处理和数据上云功能，支持 ISO on TCP、ModbusRTU 等多种工业协议解析。本手册以采集 PLC 的数据并上传至 Thingsboard 云平台为例说明如何通过 Device Supervisor App 实现 PLC 数据采集和数据上云。以下将 InGateway501 简称为 “IG501”；InGateway902 简称为 “IG902”。

- 概览
- *1.* 准备硬件设备及其数据采集环境
  - *1.1* 硬件接线
    - *1.1.1* 以太网接线
    - *1.1.2* 串口接线
  - *1.2* 设置 *LAN* 网络参数：在局域网访问 *PLC*
  - *1.3* 设置 *WAN* 网络参数：传输数据至 *MQTT* 服务器
  - *1.4* 更新 *InGateway* 设备软件版本

1.1.1 概览

使用过程中，您需要准备以下项：

- 边缘计算网关 IG501/IG902
- PLC 设备
- 网线/串口线
- * 更新软件版本所需的固件、SDK 和 App
  - 固件版本：V2.0.0.r12622 及以上
  - SDK 版本：py3sdk-V1.3.7 及以上
  - App 版本：1.1.2 及以上
- *Thingsboad 演示账号

整体流程如下图所示：

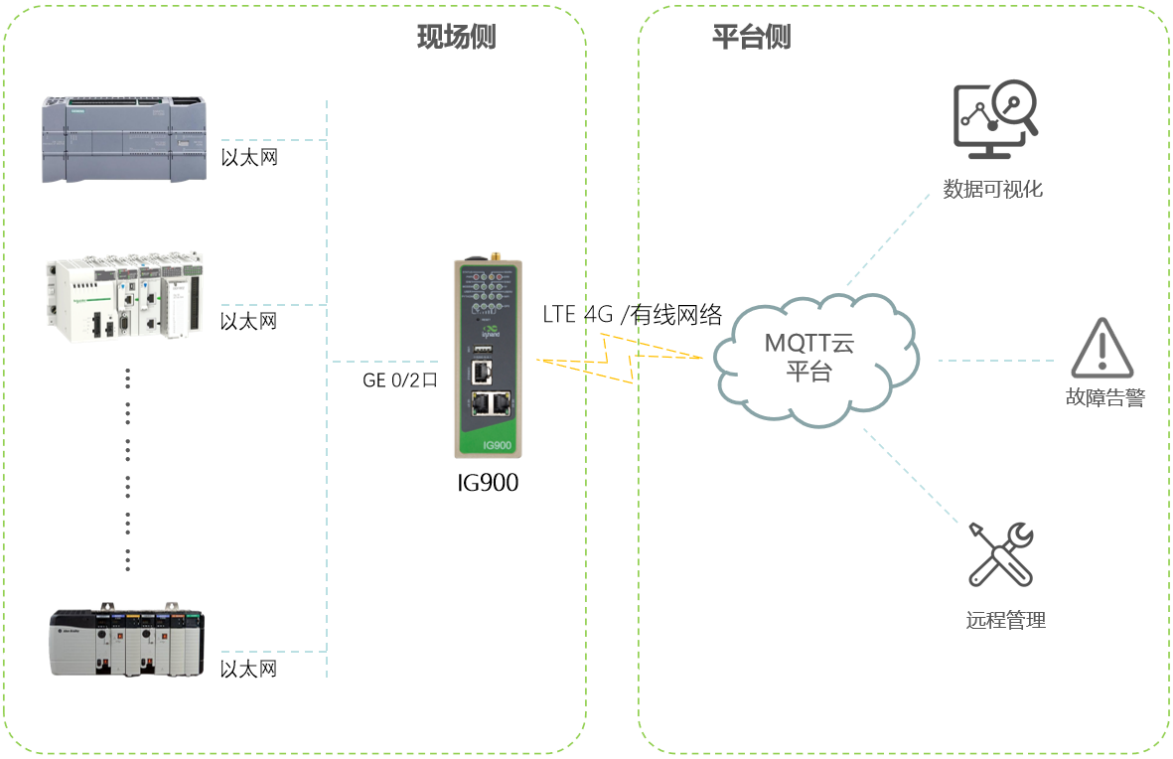
1.1.2 1. 准备硬件设备及其数据采集环境

- *1.1 硬件接线*
- *1.2 设置 LAN 网络参数：在局域网访问 PLC*
- *1.3 设置 WAN 网络参数：传输数据至 MQTT 服务器*
- *1.4 更新 InGateway 设备软件版本*

1.1 硬件接线

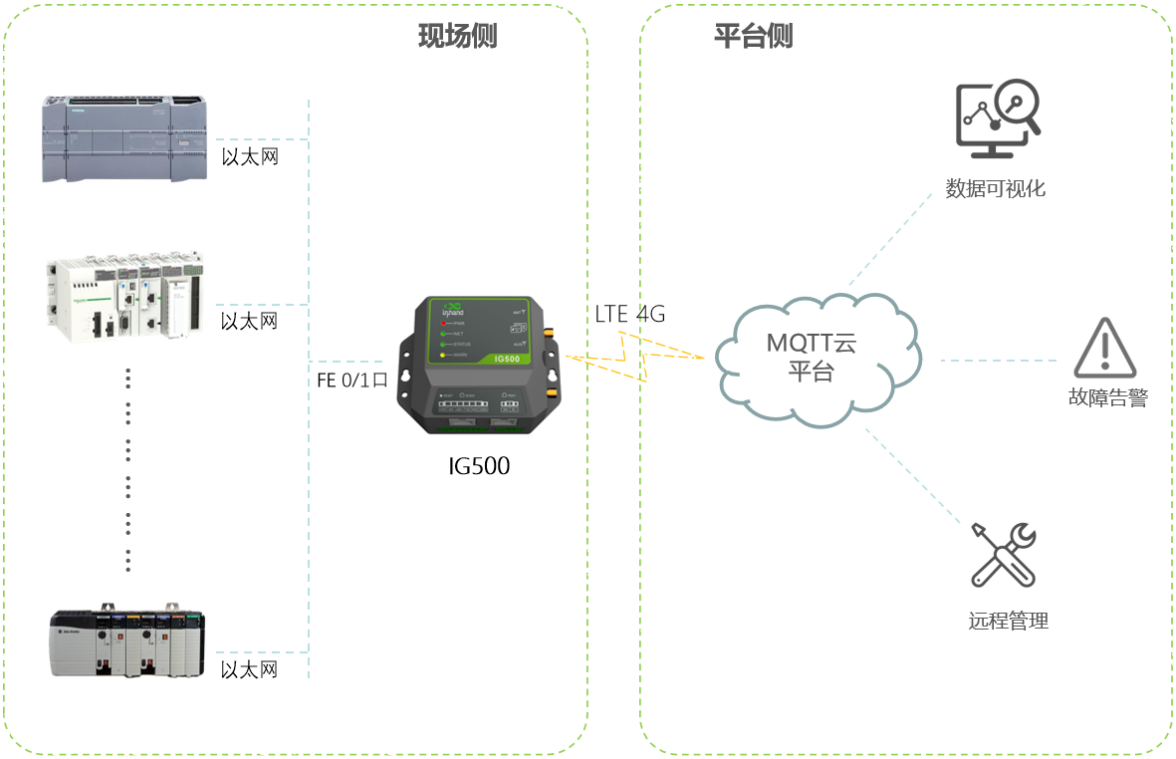
1.1.1 以太网接线

- IG902 以太网接线接通 IG902 的电源并按照拓扑使用以太网线连接 IG902 和 PLC。



- IG501 以太网接线

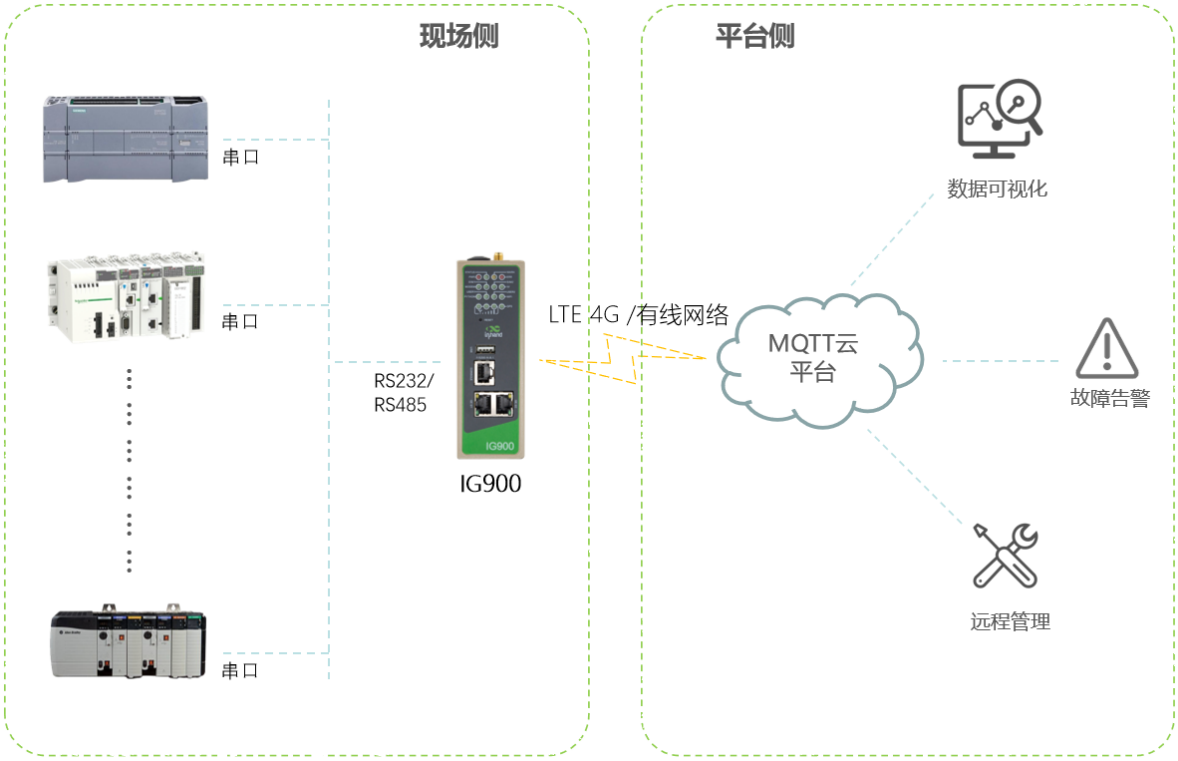
  接通 IG501 的电源并按照拓扑使用以太网线连接 IG501 和 PLC。



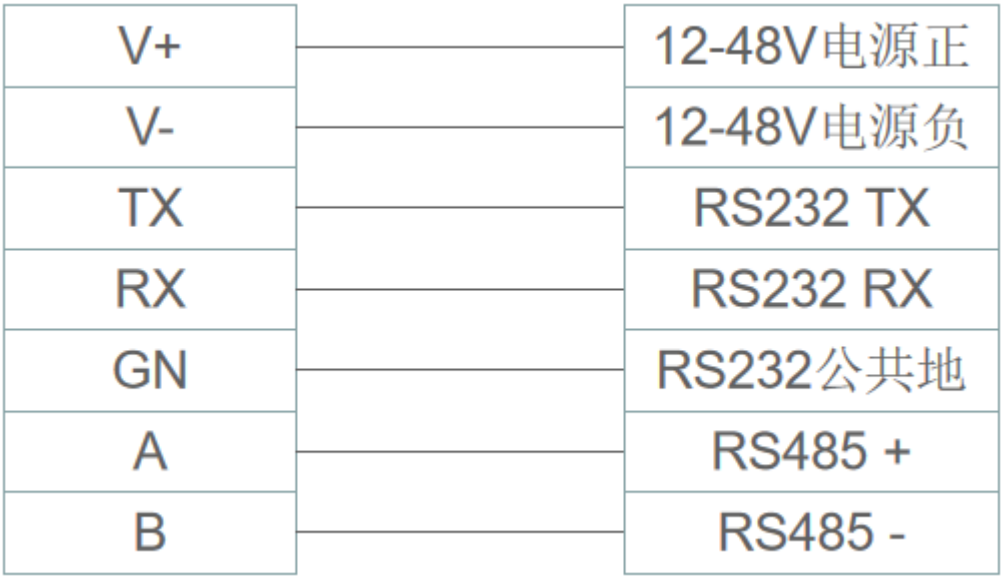
1.1.2 串口接线

- IG902 串口接线

  接通 IG902 的电源并按照拓扑连接 IG902 和 PLC。

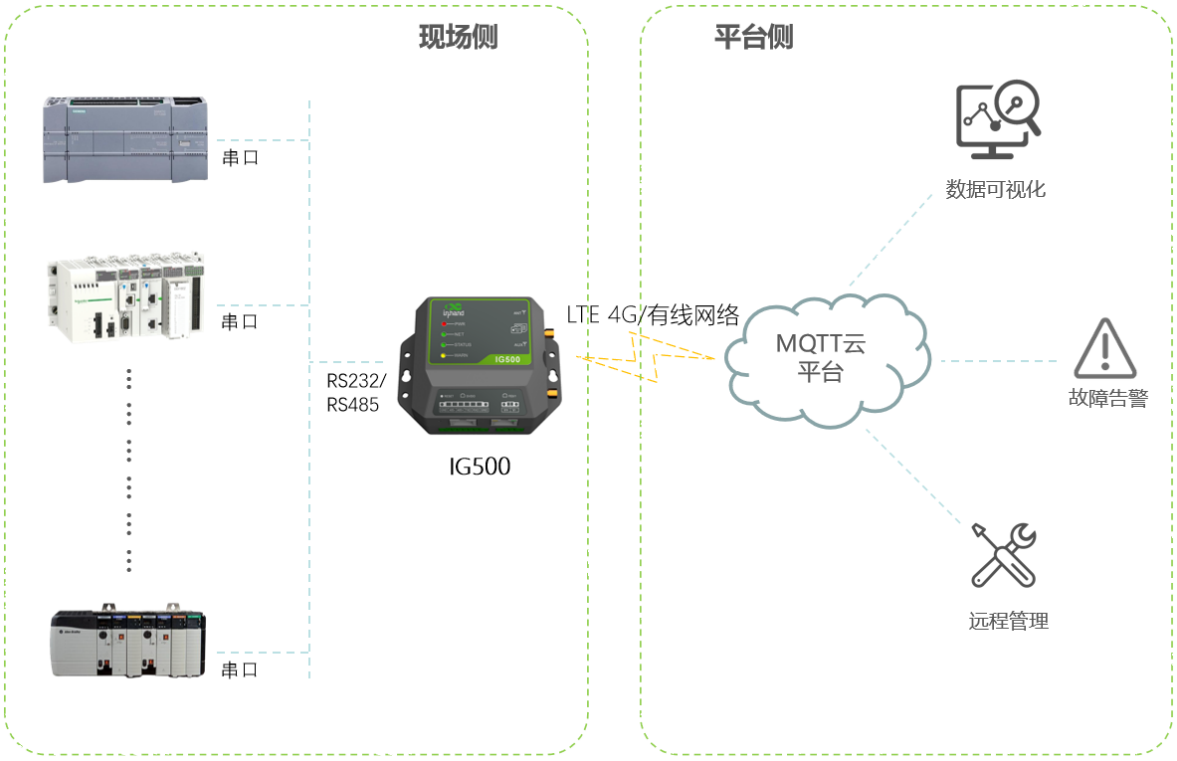


  IG902 串口端子接线说明如下图：

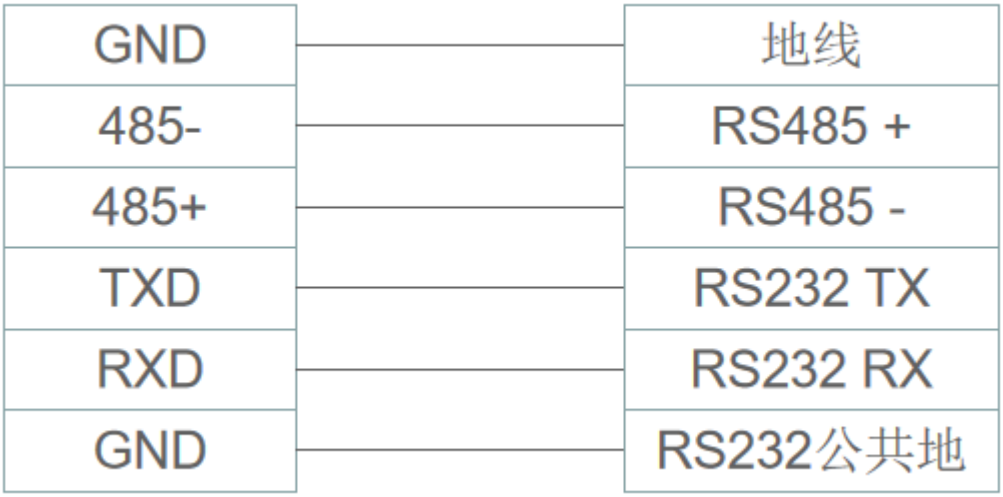


- IG501 串口接线

接通 IG501 的电源并按照拓扑连接 IG501 和 PLC。



IG501 串口端子接线说明如下图：



1.2 设置 LAN 网络参数：在局域网访问 PLC

- IG902 的 GE 0/2 口的默认 IP 地址为 `192.168.2.1`。为了使 IG902 能够通过 GE 0/2 口访问以太网 PLC，需要设置 GE 0/2 口与 PLC 处于同一网段，设置方法请参考在局域网访问 IG902。

- IG501 的 FE 0/1 口的默认 IP 地址为 `192.168.1.1`。为了使 IG501 能够通过 FE 0/1 口访问以太网 PLC，需要设置 FE 0/1 口与 PLC 处于同一网段，设置方法请参考在局域网访问 IG501。

1.3 设置 WAN 网络参数：传输数据至 MQTT 服务器

- 设置 IG902 WAN 网络参数，请参考IG902 连接 Internet。
- 设置 IG501 WAN 网络参数，请参考IG501 连接 Internet。

1.4 更新 InGateway 设备软件版本

如需获取 InGateway 产品最新软件版本及其功能特性信息，请联系客服。如需更新软件版本，请参考如下链接：

- 更新 IG902 软件版本 使用 Device Supervisor 时，IG902 的固件版本应为 `V2.0.0.r12537` 及以上；SDK 版本应为 `py3sdk-V1.3.5` 及以上。
- 更新 IG501 软件版本

1.1.3 2. 配置 Device Supervisor App

- *2.1 安装并运行 Device Supervisor*
- *2.2 Device Supervisor 数据采集配置*

2.1 安装并运行 Device Supervisor

- IG902 如何安装并运行 Python App 请参考IG902 安装和运行 Python App，Device Supervisor 正常运行后如下图所示：

- IG501 如何安装并运行 Python App 请参考IG501 安装和运行 Python App，Device Supervisor 正常运行后如下图所示：

2.2 Device Supervisor 数据采集配置

- *2.2.1 添加 PLC 设备*
- *2.2.2 添加变量*
- *2.2.3 配置告警策略*
- *2.2.4 配置分组*

2.2.1 添加 PLC 设备

- 添加 ISO on TCP 通讯的 PLC 设备

  进入“边缘计算 > 设备监控 > 设备列表”页面，点击“添加 PLC”按钮，在添加设备页面选择 PLC 协议为“ISO on TCP”并配置 PLC 的通讯参数。注意：设备名称不能重复。

  下图是添加 S7-1500、S7-1200、S7-400、S7-300、S7-200 Smart 系列 PLC 的示例（模式选择 `Rack/Slot`）。除 PLC 为 S7-200 Smart 时机架号和槽号需要配置为 0，1；其余类型的 S7 系列 PLC 默认使用 0，0 即可：

  下图是添加 S7-200 和西门子 LOGO 系列 PLC 的示例（模式选择 `TSAP`）：

  添加成功后如下图所示：

- 添加 ModbusTCP 通讯的 PLC 设备

  进入“边缘计算 > 设备监控 > 设备列表”页面，点击“添加 PLC”按钮，在添加设备页面选择 PLC 协议为“ModbusTCP”并配置 PLC 的通讯参数。（端口号和字节序默认为 502 和 abcd；使用时需根据实际情况调整）注意：设备名称不能重复。

  添加成功后如下图所示：

- 添加 ModbusRTU 通讯的 PLC 设备

  进入“边缘计算 > 设备监控 > 设备列表”页面，点击“添加 PLC”按钮，在添加设备页面选择 PLC 协议为“ModbusRTU”并配置 PLC 的通讯参数。注意：设备名称不能重复。

  添加成功后如下图所示：

  如需修改 RS232/RS485 串口的通讯参数，请在“边缘计算 > 设备监控 > 全局参数”页面修改。修改后所有串口设备的通讯参数将自动修改并按照修改后的通讯参数通讯。

2.2.2 添加变量

- 添加 ISO on TCP 变量

  在“设备列表”页面点击“添加变量”按钮，在弹出框中配置变量参数：

- 变量名：变量名称（同一设备下变量名称不能重复）
- 寄存器类型：变量寄存器类型，包括 I/Q/M/DB 四种类型
- DB 索引：寄存器类型为 DB 时变量的 DB 号
- 地址：变量的寄存器地址
- 数据类型：变量数据类型，包括：
  - BOOL：True 或 False
  - BIT：0 或 1
  - BYTE：8 位无符号数据
  - SINT：8 位有符号数据
  - WORD：16 位无符号数据
  - INT：16 位有符号数据
  - DWORD：32 位无符号数据
  - DINT：32 位有符号数据
  - FLOAT：32 位浮点数
  - STRING：8 位字符串
  - BCD：16 位 BCD 码
- 小数位：数据类型为 FLOAT 时变量小数点后的数据长度，最大 6 位
- 长度：数据类型为 STRING 时字符串长度，读取 1 个字符串的长度为 1
- 位：数据类型为 BOOL 或 BIT 时变量的位偏移，可输入 0~7 中任一数字
- 读写权限：
  - Read：只读，不可写
  - Write：只写，不可读
  - Read/Write：可读可写
- 采集模式：
  - Realtime：按照所属分组的采集间隔采集变量并按照上报间隔上报数据
  - Onchange：变量数值有变化时才采集变量并按照上报间隔上报数据
- 单位：变量单位
- 描述：变量描述
- 所属分组：变量所属的采集组

下图是添加一个地址为%I0.0 的开关变量的例子：

下图是添加一个地址为%IB1 的字节变量的例子：

下图是添加一个地址为%IW3 的字变量的例子：

下图是添加一个地址为%ID4 的双字变量的例子：

下图是添加一个地址为%DB6.DBD18 的浮点数变量的例子：

- 添加 Modbus 变量

  在“设备列表”页面点击“添加变量”按钮，在添加变量弹出框中配置 PLC 变量参数：

  - 变量名：变量名称（同一设备下变量名称不能重复）
  - 地址：变量的寄存器地址
  - 数据类型：变量数据类型，包括：
    - BOOL：True 或 False
    - BIT：0 或 1
    - WORD：16 位无符号数据
    - INT：16 位有符号数据
    - DWORD：32 位无符号数据
    - DINT：32 位有符号数据
    - FLOAT：32 位浮点数
    - STRING：8 位字符串
  - 小数位：数据类型为 FLOAT 时变量小数点后的数据长度，最大 6 位
  - 长度：数据类型为 STRING 时字符串长度
  - 位：地址为 `30001~40000`,`310001~365535`，`40001~50000`，`410001~465535` 且数据类型为 BOOL 或 BIT 时变量的位偏移，可输入 0~15 中任一数字
  - 读写权限：
    - Read：只读，不可写
    - Write：只写，不可读

    * Read/Write：可读可写
  - 采集模式：
    * Realtime：按照固定采集间隔采集变量并按照上报间隔上报数据
    * Onchange：变量数值变化后才采集并按照上报间隔上报数据
  - 单位：变量单位
  - 描述：变量描述
  - 所属分组：变量所属的采集组

下图是添加一个地址为 00001 的线圈变量的例子：

下图是添加一个地址为 10001 的开关变量的例子：

下图是添加一个地址为 30001 的整数变量的例子：

下图是添加一个地址为 40001 的浮点数变量的例子：

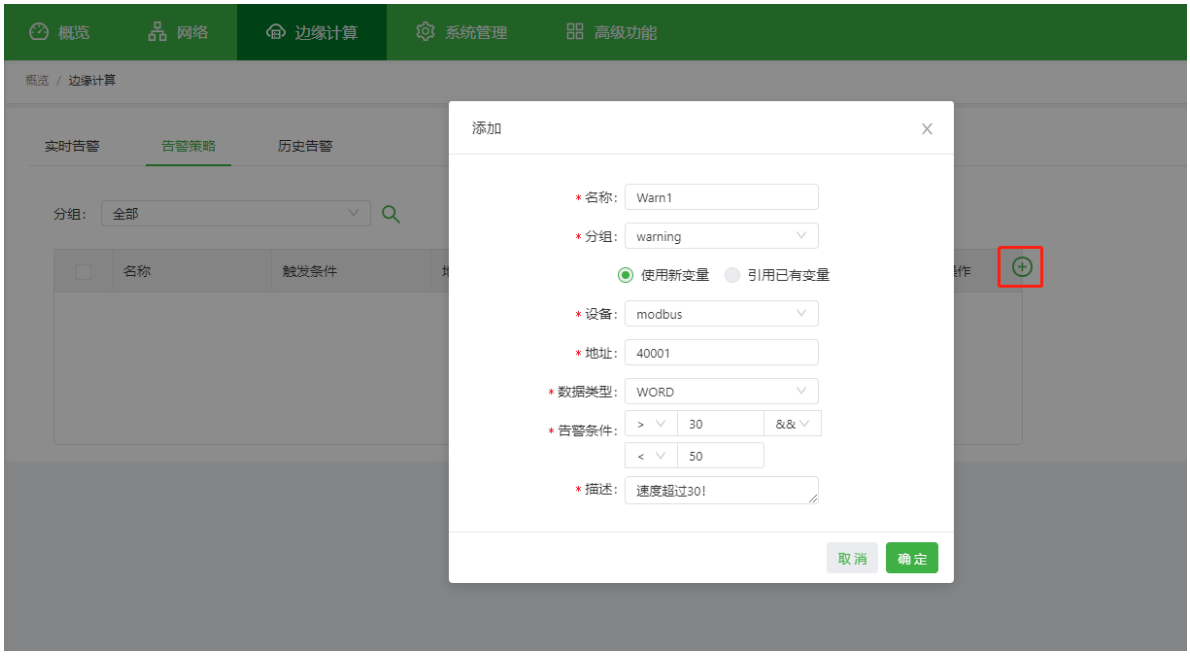
2.2.3 配置告警策略

你可以进入“边缘计算 > 设备监控 > 告警 > 告警策略”页面配置告警策略，点击“添加”按钮后，在弹出框中配置告警策略参数。告警策略支持两种配置方式“使用新变量”和“引用已有变量”，参数如下：

- 使用新变量
  - 名称：告警名称
  - 分组：告警所属分组
  - 变量来源：“使用新变量”即告警变量未在“设备列表”中配置，需要自行设置变量参数（该操作不会在“设备列表”中新增变量）
  - 设备：告警变量所属设备
  - 地址：告警变量地址
  - 数据类型：告警变量数据类型
  - 告警条件
    * 判断条件：支持“=”、“!=”、“>”、“ ”、“<”、“ ”
    * 逻辑条件
      · 无逻辑条件：仅通过单个判断条件判断告警

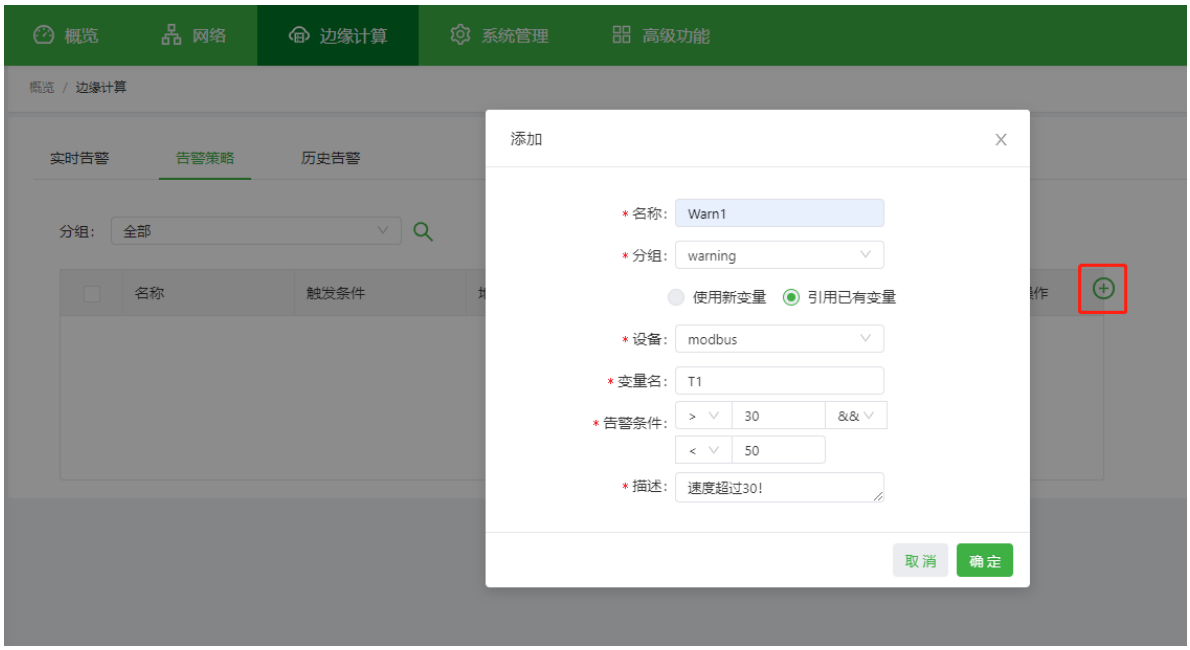
        - &&：通过两个判断条件相与判断告警
        - ||：通过两个判断条件相或判断告警
    - 描述：告警描述

下图是添加一个告警变量，该变量数值 >30 且 <50 时产生告警；不在此范围时不产生告警或告警消除。



- 引用已有变量
    - 名称：告警名称
    - 分组：告警所属分组
    - 变量来源：“引用已有变量”即告警变量已在“设备列表”中配置，可以输入已有变量名称直接使用
    - 设备：告警变量所属设备
    - 地址：告警变量的地址
    - 告警条件
        * 判断条件：支持“=”、“!=”、“>”、“ ”、“<”、“ ”
        * 逻辑条件
            - 无逻辑条件：仅通过单个判断条件判断告警
            - &&：通过两个判断条件相与判断告警
            - ||：通过两个判断条件相或判断告警
    - 描述：告警描述

下图是引用已有变量生成一条告警变量，该变量数值 >30 且 <50 时产生告警；不在此范围时不产生告警或告警消除。



2.2.4 配置分组

如需为变量或告警配置不同的采集间隔或需要按照不同的 MQTT 主题上报相应的变量数据时，可在“边缘计算 > 设备监控 > 分组”页面添加新分组。

- 添加采集分组

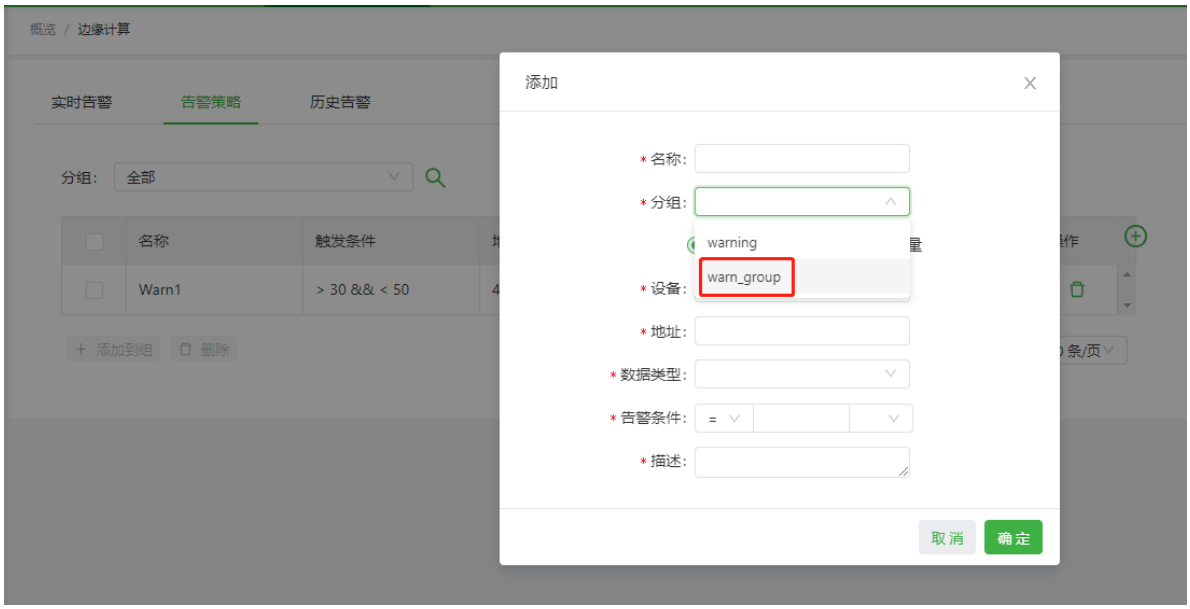
  下图添加了一个名为“group2”的采集分组，该采集分组每 5 秒采集一次分组中的变量：

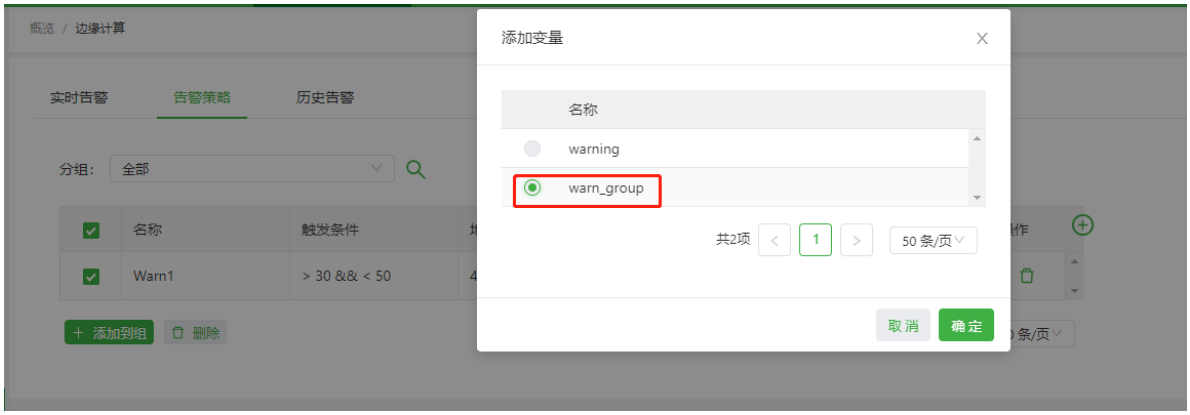
添加采集分组后，添加变量时可以选择将变量关联到该分组或者在变量列表中选择需要关联的变量添加到指定分组中，分组中的变量会按照分组的采集间隔采集数据。

- 添加告警分组

  下图添加了一个名为 warn_group2 的告警分组，该告警分组每 5 秒检测一次分组中的告警变量是否处于告警状态:

添加告警分组后，添加告警策略时可以选择将告警策略关联到该分组或者在告警列表中选择需要关联的告警策略添加到指定分组中，分组中的告警策略会按照分组的采集间隔检测变量告警状态。





1.1.4 3. 监控 PLC 数据

- *3.1 本地监控 PLC 数据*
- *3.2 在 Thingsboard 上监控 PLC 数据*

3.1 本地监控 PLC 数据

- *3.1.1 本地监控数据采集*
- *3.1.2 本地监控告警*

3.1.1 本地监控数据采集

数据采集配置完成后，可以在“边缘计算 > 设备监控 > 设备列表”页面查看数据采集情况。点击设备列表中的设备卡片可切换需要查看的 PLC 数据。

点击数值栏的按钮可进行写入操作。

修改成功如下图所示：

3.1.2 本地监控告警

告警策略配置完成后，可以在“边缘计算 > 设备监控 > 告警”页面查看变量告警情况。

- 实时告警：查看当前未消除的告警信息

- 历史告警：筛选查看任意告警信息

3.2 在 Thingsboard 上监控 PLC 数据

- *3.2.1 配置 Thingsboard*

- *3.2.2 配置云服务*

3.2.1 配置 Thingsboard

Thingsboard 的详细使用方法请查看Thingsboard 入门手册，您也可以按照 *Thingsboard 参考流程*进行测试。

3.2.2 配置云服务

进入“边缘计算 > 设备监控 > 云服务”页面，勾选启用云服务并配置相应的 MQTT 连接参数，配置完成后点击提交。

- 服务器地址：Thingsboard 的 demo 地址为 `demo.thingsboard.io`
- 端口号：默认为 `1883`
- `MQTT` 客户端 `ID`：任一唯一 ID
- `MQTT` 用户名：Thingsboard 设备的访问令牌，访问令牌获取方式见*传输 PLC 数据到 Thingsboard 设备*
- MQTT 密码：任意 6~32 位密码
- 其余项使用默认配置即可

配置完成后如下图所示：

提交后点击“高级设置”以配置发布和订阅主题。发布和订阅主题的配置方法请参考高级设置（*自定义 MQTT 发布/订阅*）。以下是示例配置：

- 发布主题：
  - 主题：`v1/devices/me/telemetry`
  - `Qos(MQTT)`：`1`
  - 分组类型：采集
  - 分组：需要上传数据至 thingsboard 的分组名称，本文档为 `default`
  - 主函数：入口函数名称，本文档为 `upload_test`
  - 脚本：

```
from common.Logger import logger # 导入打印日志模块 logger

def upload_test(data, wizard_api): # 定义主函数 upload_test
    logger.info(data) # 在日志中以 info 等级打印采集到的数据
    value_dict = {} # 定义上报的数据字典 value_dict
```

(continues on next page)

(continued from previous page)

```
    for device, val_dict in data['values'].items(): # 遍历 data 中的 values 字典，该字典中包含设备名称和设备下的变量数据
        for id, val in val_dict.items(): # 遍历变量数据，为 value_dict 字典赋值
            value_dict[id] = val["raw_data"]
        value_dict["timestamp"] = data["timestamp"]
    logger.info(value_dict) # 在日志中以 info 等级打印 value_dict
    return value_dict # 将 value_list 发送给 App，由 App 自行顺序上传至 MQTT 服务器。最终的 value_list 格式为{'bool': False, 'byte': 7, 'real': 0.0, 'timestamp': 1583990892.5429199}
```

配置完成后如下图所示：

- 订阅主题
  - 主题：v1/devices/me/rpc/request/+
  - Qos(MQTT)：1
  - 主函数：入口函数名称，本文档为 ctl_test
  - 脚本：

```
from common.Logger import logger # 导入打印日志模块 logger
import json # 导入 json 模块

def ctl_test(topic, payload, wizard_api): # 定义主函数 ctl_test
    logger.info(topic) # 打印订阅主题
    logger.info(payload) # 打印订阅数据，Thingsboard 的下发数据格式为{"method": "setValue","params":true}
    payload = json.loads(payload) # 反序列化订阅数据
    if payload["method"] == "setValue": # 检测是否为写入数据
        message = {"bool":payload["params"]} # 定义修改变量值消息，包括变量名称和变量值
        ack_tail = [topic.replace('request', 'response'), message] # 定义确认数据，包括响应的主题和消息
        wizard_api.write_plc_values(message, ack, ack_tail) # 调用 write_plc_values 方法，将 message 字典中的数据下发至指定变量；调用 ack 方法并发送 ack_tail 给 ack 方法


def ack(data, ack_tail, wizard_api): # 定义 ack 方法
    resp_topic = ack_tail[0] # 定义响应主题
```

(continues on next page)

(continued from previous page)

```
    resp_data = ack_tail[1] # 定义响应数据
    wizard_api.mqtt_publish(resp_topic, json.dumps(resp_data), 1) # 调用 mqtt_
↪publish 将响应数据发送给 MQTT 服务器，响应数据格式为{'bool': True}
```

配置完成后如下图所示：

1.1.5 附录

- 导入导出配置
- 高级设置（自定义 *MQTT* 发布/订阅）
- 全局参数
- 其他网关操作
- *Thingsboard* 参考流程

导入导出配置

Device Supervisor 的数据采集配置总共包含三个 CSV 格式的配置文件，您可以通过导入导出配置文件快速实现采集配置。各配置文件内容如下：

- device.csv：设备配置文件, 详细参数如下
  - `Device Name`：设备名称
  - `Protocol`：通讯协议名称，如 `ModbusTCP`
  - `Ip/Serial`：以太网设备填写 ip 地址；串口设备填写 `RS485` 或 `RS232`
  - `Port`：以太网设备的通讯端口号
  - `Rack`（仅 ISO on TCP 设备）：设备机架号
  - `Slot`（仅 ISO on TCP 设备）：设备槽号
  - `Mode`（仅 ISO on TCP 设备）：ISO on TCP 模式，包括 `TSAP` 和 `Rack/Slot`
  - `Slave`（仅 Modbus 设备）：从站地址
  - `Byte Order`（仅 Modbus 设备）：字节序，包括 `abcd`、`badc`、`cdab`、`dcba`

  导出方式为设备列表页面的设备列表导出。

  示例配置如下：

- var.csv: 变量配置文件，详细参数如下
  - `Var Name`：变量名称
  - `Device`：变量所属设备
  - `Protocol`：通讯协议名称
  - `Dbnumber`（仅 ISO on TCP 设备）：DB 号
  - `Register Type`（仅 ISO on TCP 设备）：寄存器类型，如 `DB`
  - `Register Addr`：寄存器地址
  - `Register Bit`：位偏移
  - `Data Type`：数据类型
  - `Read Write`：读写权限，包括 `Read/Write`、`Write`、`Read`
  - `Float Repr`：小数位，`1~6`
  - `Mode`：采集模式，包括 `realtime`、`onchange`
  - `Unit`：单位
  - `Size`：字符串长度
  - `Desc`：描述
  - `Group`：所属分组

  导出方式为设备列表页面的变量列表导出。

  示例配置如下：

- group.csv: 分组配置文件，详细参数如下
  - `Group Name`：分组名称
  - `Polling Interval`：采集间隔
  - `Upload Interval`：上传间隔。分组类型为 `alarm` 时此项为空即可
  - `Group Type`：分组类型，支持 `alarm` 和 `collect`

  导出方式为分组页面的分组导出。

  示例配置如下：

高级设置（自定义 MQTT 发布/订阅）

您可以在“边缘计算 > 设备监控 > 云服务”配置你的 MQTT 连接参数，并支持通过高级设置功能配置上报数据的 MQTT 主题、数据来源等参数并支持使用 Python 语言自定义 MQTT 发布和订阅主题的数据上报、处理等逻辑。无需二次开发即可实现与多种 MQTT 服务器进行数据上传和下发。以下将为您说明“高级设置”的使用方法。

- 发布
- 订阅
- *Device Supervisor* 的 *api* 接口说明
- 回调函数说明

发布

自定义发布主题中包含以下配置项：

- 名称：用户自定义发布名称
- 主题：发布主题，与 MQTT 服务器订阅的主题保持一致
- `Qos(MQTT)`：发布 Qos，建议与 MQTT 服务器的 Qos 保持一致
  - `0`：只发送一次消息，不进行重试
  - `1`：最少发送一次消息，确保消息到达 MQTT 服务器
  - `2`：确保消息到达 MQTT 服务器且只收到一次
- 分组类型：发布变量数据时请选择“采集”，随后在分组中仅能选择“采集组”；发布告警数据时请选择“告警”，随后在分组中仅能选择“告警组”
- 分组：选择相应的分组后，分组下所有变量通过该发布配置将数据上传至 MQTT 服务器（可选择多个分组）
- 主函数：主函数名称，即入口函数名称，与脚本中的入口函数名称保持一致
- 脚本：使用 Python 代码自定义组包和处理逻辑，发布中的主函数参数包括：
  - 参数 `1`：Device Supervisor 将采集后的变量数据发送给该参数，数据格式如下：
    * 变量数据格式：

```
{
    'timestamp': 1589434519.5458372,  # 数据产生时间戳
    'group_name': 'default'  # 采集组名称
    'values':  # 变量数据字典，包含 PLC 名称，变量名称和变量值
    {
        'S7-1200':  #PLC 名称
```

(continues on next page)

(continued from previous page)

```
        {
            'Test1':  # 变量名称
            {
                'raw_data': False,  # 变量值
                'status': 1  # 采集状态，非 1 即采集异常
            },
            'Test2':
            {
                'raw_data': 2,
                'status': 1
            }
        }
    },
}
```

* 告警数据格式：

```
{
    'timestamp': 1589434527.3628697,  # 告警产生时间戳
    'group_name': 'warning'  # 告警组名称
    'values':  # 告警数据字典，包含告警名称等告警信息
    {
        'Warn1':  # 告警名称
        {
            'timestamp': 1589434527,  # 告警产生时间戳
            'current': 'on',  # 告警状态。on: 已触发，off: 已消除
            'status': 0,  # 告警状态。0: 已触发，1: 已消除
            'value': 33,  # 告警触发时告警变量的数值
            'alarm_content': '速度超过 30! ',  # 告警描述
            'level': 1  # 预留字段
        }
    },
}
```

– 参数 2：该参数为 Device Supervisor 提供的 api 接口，参数说明见 *Device Supervisor 的 api 接口说明*

以下是常见的自定义发布方法示例（请勿将 `mqtt_publish` 或 `save_data` 方法与 `return` 命令同时使用）：

- 发布示例 1：使用 `return` 方式上传变量数据

  本示例实现了使用 `return` 方式上传变量数据时，将处理后的变量数据使用 `return` 命令发送给 Device Supervisor。Device Supervisor 自行根据发布中配置的主题和 Qos 将变量数据按照采集时间顺序上传至

MQTT 服务器。如果发送失败则缓存变量数据等待 MQTT 连接正常后按采集时间顺序上传至 MQTT 服务器。发布和代码配置示例如下：

```
import logging
"""
在网关中打印日志通常有两种办法。
1.import logging: 使用 logging.info(XXX) 打印日志，该方法的日志显示不受全局参数页面中的日志等级参数控制。
2.from common.Logger import logger: 使用 logger.info(XXX) 打印日志，该方法的日志显示受全局参数页面中的日志等级参数控制。
"""

def vars_upload_test(data_collect, wizard_api): # 定义发布主函数
    value_list = [] # 定义数据列表
    for device, val_dict in data_collect['values'].items(): # 遍历 values 字典，该字典中包含设备名称和设备下的变量数据
        value_dict = { # 自定义数据字典
                      "Device": device,
                      "timestamp": data_collect["timestamp"],
                      "Data": {}
                      }
        for id, val in val_dict.items(): # 遍历变量数据，为 Data 字典赋值
            value_dict["Data"][id] = val["raw_data"]
        value_list.append(value_dict) # 依次将 value_dict 添加到 value_list 中
    logging.info(value_list) # 在 App 日志中打印 value_list，数据格式为 [{'Device': 'S7-1200', 'timestamp': 1589538347.5604711, 'Data': {'Test1': False, 'Test2': 12}}]
    return value_list # 将 value_list 发送给 App，App 将自行按照采集时间顺序上传至 MQTT 服务器。如果发送失败则缓存数据等待连接恢复后按采集时间顺序上传至 MQTT 服务器
```

- 发布示例 2：使用 return 方式上传告警数据

  本示例实现了上传告警数据。发布和代码配置示例如下：

```
import logging
"""
在网关中打印日志通常有两种办法。
1.import logging: 使用 logging.info(XXX) 打印日志，该方法的日志显示不受全局参数页面中的日志等级参数控制。
```

(continues on next page)

(continued from previous page)

```
2.from common.Logger import logger: 使用 logger.info(XXX) 打印日志，该方法的日志显示受
全局参数页面中的日志等级参数控制。
"""

def alarms_upload_test(data_collect, wizard_api): # 定义发布主函数
    alarm_list = [] # 定义告警列表
    for alarm_name, alarm_info in data_collect['values'].items(): # 遍历 values 字典，
该字典中包含告警名称和告警时间等信息
        alarm_dict = { # 自定义数据字典
                    "Alarm_name": alarm_name,
                    "timestamp": data_collect["timestamp"],
                    "Alarm_status": alarm_info['current'],
                    "Alarm_value": alarm_info['value'],
                    "Alarm_content": alarm_info['alarm_content']
                    }
        alarm_list.append(alarm_dict) # 依次将 alarm_dict 添加到 alarm_list 中
    logging.info(alarm_list) # 在 App 日志中打印 alarm_list
    return alarm_list # 将 alarm_list 发送给 App，App 将自行按照采集时间顺序上传至
MQTT 服务器。如果发送失败则缓存数据等待连接恢复后按时间顺序上传至 MQTT 服务器
```

- 发布示例 3：使用 `mqtt_publish` 上传变量数据并使用 `save_data` 存储上传失败的变量数据

  本示例实现了使用 `mqtt_publish` 方法将变量数据上传至 MQTT 服务器，当 MQTT 连接异常导致变量数据上传失败时，使用 `save_data` 方法存储主题、qos 和变量数据至数据库，存储的变量数据会在 MQTT 连接正常时按照先存先传的方式通过存储数据中的主题和 Qos 上传至 MQTT 服务器。发布和代码配置示例如下：

```
from common.Logger import logger
import json
from datetime import datetime
"""
在网关中打印日志通常有两种办法。
1.import logging: 使用 logging.info(XXX) 打印日志，该方法的日志显示不受全局参数页面中的
日志等级参数控制。
2.from common.Logger import logger: 使用 logger.info(XXX) 打印日志，该方法的日志显示受
全局参数页面中的日志等级参数控制。
"""

def vars_cache_test(data_collect, wizard_api): # 定义发布主函数
```

(continues on next page)

(continued from previous page)

```
    value_list = [] # 定义数据列表
    utc_time = datetime.utcfromtimestamp(data_collect["timestamp"]) # 转换 LINUX 时间
戳为 UTC 时间
    for device, val_dict in data_collect['values'].items(): # 遍历 values 字典，该字
典中包含设备名称和设备下的变量数据
        value_dict = { # 自定义数据字典
                      "DeviceSN": device,
                      "Time": utc_time.strftime('%Y-%m-%dT%H:%M:%S.%fZ'),
                      "Data": {}
                       }
        for id, val in val_dict.items(): # 遍历变量数据，为 Data 字典赋值
            value_dict["Data"][id] = val["raw_data"]
        value_list.append(value_dict) # 依次将 value_dict 添加到 value_list 中
    if not wizard_api.mqtt_publish("v1/xxx/yyy", json.dumps(value_list), 1): # 调用
wizard_api 模块中的 mqtt_publish 方法将 value_list 数据通过主题 "v1/xxx/yyy"，qos 等级
1 发送至 MQTT 服务器并检测是否发送成功
        value_list = {"topic": "v1/xxx/yyy", "qos": 1, "payload": value_list}
        wizard_api.save_data(value_list) # 发送失败则存储数据，等待连接恢复后将按时间
顺序上传存储数据
        logger.info("Save data:%s" %value_list)
    logger.info(value_list) # 在 App 日志中打印 value_list
```

- 发布示例 4：使用 mqtt_publish 上传变量数据并使用 save_data 存储上传失败的变量数据

  本示例实现了使用 mqtt_publish 方法将变量数据上传至 MQTT 服务器，当 MQTT 连接异常导致变量数据上传失败时，使用 save_data 方法存储变量数据和分组名称，存储的变量数据会在 MQTT 连接正常时按照先存先传的方式将变量数据按照分组在云服务中关联的主题和 Qos 上传至 MQTT 服务器（请勿将 mqtt_publish 或 save_data 方法与 return 命令同时使用）。发布和代码配置示例如下：

```
from common.Logger import logger
import json
from datetime import datetime
"""
在网关中打印日志通常有两种办法。
1.import logging：使用 logging.info(XXX) 打印日志，该方法的日志显示不受全局参数页面中的
日志等级参数控制。
2.from common.Logger import logger：使用 logger.info(XXX) 打印日志，该方法的日志显示受
全局参数页面中的日志等级参数控制。
"""
```

(continues on next page)

(continued from previous page)

```
def vars_cache_test(data_collect, wizard_api): # 定义发布主函数
    value_list = [] # 定义数据列表
    utc_time = datetime.utcfromtimestamp(data_collect["timestamp"]) # 转换 LINUX 时间戳为 UTC 时间
    for device, val_dict in data_collect['values'].items(): # 遍历 values 字典，该字典中包含设备名称和设备下的变量数据
        value_dict = { # 自定义数据字典
                    "DeviceSN": device,
                    "Time": utc_time.strftime('%Y-%m-%dT%H:%M:%S.%fZ'),
                    "Data": {}
                     }
        for id, val in val_dict.items(): # 遍历变量数据，为 Data 字典赋值
            value_dict["Data"][id] = val["raw_data"]
        value_list.append(value_dict) # 依次将 value_dict 添加到 value_list 中
    if not wizard_api.mqtt_publish("v1/xxx/yyy", json.dumps(value_list), 1): # 调用 wizard_api 模块中的 mqtt_publish 方法将 value_list 数据通过主题 “v1/xxx/yyy”，qos 等级 1 发送至 MQTT 服务器并检测是否发送成功
        wizard_api.save_data(value_list,'default') # 发送失败则存储数据，等待连接恢复后将按时间顺序上传存储数据
        logger.info("Save data:%s" %value_list)
    logger.info(value_list) # 在 App 日志中打印 value_list
```

- 发布示例 5：使用 get_tag_config 获取设备、变量和告警点表

  本示例实现了每次重启 App 时使用 get_tag_config 方法获取设备、变量和告警点表并分别上传至 MQTT 服务器。发布和代码配置示例如下：

```
from common.Logger import logger
import json
"""
在网关中打印日志通常有两种办法。
1.import logging：使用 logging.info(XXX) 打印日志，该方法的日志显示不受全局参数页面中的日志等级参数控制。
2.from common.Logger import logger：使用 logger.info(XXX) 打印日志，该方法的日志显示受全局参数页面中的日志等级参数控制。
"""

IS_UPLOAD_CONFIG = True  # 定义变量用于判断是否需要获取并上传点表
```

(continues on next page)

(continued from previous page)

```
def upload_tagconfig(recv, wizard_api):  # 定义发布主函数
    global IS_UPLOAD_CONFIG  # 声明变量为全局变量
    if IS_UPLOAD_CONFIG:  # 判断是否需要获取并上传点表
        wizard_api.get_tag_config(tagconfig)  # 调用 wizard_api 模块中的 recall_data
方法，定义该方法的回调函数名称为 tagconfig
        IS_UPLOAD_CONFIG = False  # 获取并上传点表后不再上传点表

def tagconfig(config, tail, wizard_api): # 定义获取点表的回调函数 tagconfig
    logger.info(config) # 打印点表信息，包含设备、分组、变量和告警点表
    deviceConfiguration_list = [] # 定义设备点表列表
    for device in config['devices']: # 遍历设备点表
        deviceInfo = {} # 定义设备信息字典
        if device['protocol'] == "ModbusTCP": # 判断设备通讯协议是否为 ModbusTCP
            deviceInfo["Device"] = device['device_name']
            deviceInfo["PLCProtocol"] = device['protocol']
            deviceInfo["IP Address"] = device['ip']
            deviceInfo["Port"] = device['port']
            deviceInfo["SlaveAddress"] = device['slave']
            deviceInfo["Endian"] = device['byte_order']
            deviceConfiguration_list.append(deviceInfo) # 依次将 deviceInfo 添加到
deviceConfiguration_list 中
        elif device['protocol'] == "ModbusRTU": # 判断设备通讯协议是否为 ModbusRTU
            deviceInfo["Device"] = device['device_name']
            deviceInfo["PLCProtocol"] = device['protocol']
            deviceInfo["Port"] = device['serial']
            deviceInfo["Baudrate"] = device['baudrate']
            deviceInfo["DataBits"] = device['bytesize']
            deviceInfo["Parity"] = device['parity']
            deviceInfo["StopBits"] = device['stopbits']
            deviceInfo["SlaveAddress"] = device['slave']
            deviceInfo["Endian"] = device['byte_order']
            deviceConfiguration_list.append(deviceInfo)
        elif device['protocol'] == "ISO-on-TCP": # 判断设备通讯协议是否为 ISO-on-TCP
            deviceInfo["Device"] = device['device_name']
            deviceInfo["PLCProtocol"] = device['protocol']
            deviceInfo["IP Address"] = device['ip']
            deviceInfo["Port"] = device['port']
            deviceInfo["Rack"] = device['rack']
            deviceInfo["Slot"] = device['slot']
```

(continues on next page)

(continued from previous page)

```
            deviceConfiguration_list.append(deviceInfo)
    logger.info(deviceConfiguration_list)
    wizard_api.mqtt_publish("Config/DeviceInfo", json.dumps(deviceConfiguration_
↪list), 1) # 调用 wizard_api 模块中的 mqtt_publish 方法将 deviceConfiguration_list
数据通过主题 "Config/DeviceInfo", qos 等级 1 发送至 MQTT 服务器



    tagConfiguration_list = [] # 定义变量点表列表
    device_group_info_dict = {} # 定义设备下的分组信息字典
    group_info_dict = {} # 定义分组信息字典
    for groupinfo in config['groups']: # 遍历点表中的分组
        group_info_dict[groupinfo["group_name"]] = groupinfo # 建立组名与分组信息的对
应关系
    for device in config['devices']: # 遍历点表中的设备
        group_list= [] # 定义设备下的分组列表
        for var in config['vars']: # 遍历点表中的变量
            if device['device_name'] == var['device'] and var['group'] not in group_
↪list: # 判断设备下存在变量，且该变量所属的分组未存在与 group_list 中，则将该分组添加到
group_list 中
                group_list.append(var['group'])
        device_group_info_dict[device['device_name']] = group_list # 建立设备与设备下
的分组列表的对应关系
    for device, group_list in device_group_info_dict.items(): # 遍历设备下的分组信息
字典
        if group_list == []: # 如果设备下的分组列表为空，即该设备下未定义变量，则跳过该
设备
            continue
        tagConfiguration = {} # 定义变量点表字典
        tagConfiguration["Device"] = device # 添加设备信息
        tagConfiguration["Collections"] = []
        for group in group_list: # 遍历设备下的分组并添加分组信息
            group_info = {}
            group_info["CollectionName"] = group
            group_info["SampleRate"] = group_info_dict[group]["polling_interval"]
            group_info["PublishInterval"] = group_info_dict[group]["upload_interval
↪"]
            group_info["TagData"] = []
            tagConfiguration["Collections"].append(group_info)
            for var in config['vars']: # 遍历点表中的变量
                if var['device'] != device or var['group'] != group: # 如果该变量不属
于当前设备和分组，则跳过该变量
```

(continues on next page)

(continued from previous page)

```
                    continue
                index_number = tagConfiguration["Collections"].index(group_info) #
获取该分组在 tagConfiguration["Collections"] 中的索引
                data_info = {} # 定义变量信息字典
                data_info["Tag"] = var["var_name"]
                data_info["Address"] = var["address"]
                data_info["ValueType"] = var["data_type"]
                data_info["AccessLevel"] = var["read_write"]
                data_info["Mode"] = var["mode"]
                data_info["Unit"] = var["unit"]
                data_info["Description"] = var["desc"]
                tagConfiguration["Collections"][index_number]["TagData"].
↪append(data_info) # 将变量信息添加至指定分组的 TagData 中
        tagConfiguration_list.append(tagConfiguration) # 依次将设备点表添加至
tagConfiguration_list 中
    logger.info(tagConfiguration_list) # 打印变量点表
    for tagConfiguration in tagConfiguration_list: # 遍历每个设备下的变量点表并依次上
传至 MQTT 服务器
        wizard_api.mqtt_publish("Config/TagConfiguration", json.
↪dumps(tagConfiguration), 1) # 调用 wizard_api 模块中的 mqtt_publish 方法将
tagConfiguration_list 中的数据依次通过主题 "Config/TagConfiguration", qos 等级 1 发送
至 MQTT 服务器


    alarmConfiguration_list = [] # 定义告警点表
    for alarm in config['warning']: # 遍历点表中的告警
        alarmInfo = {} # 定义告警信息字典
        alarmInfo['Warn_name'] = alarm['warn_name']
        alarmInfo['Group'] = alarm['group']
        alarmInfo['Alarm_content'] = alarm['alarm_content']
        alarmInfo['Condition1'] = alarm['condition1']
        alarmInfo['Operand1'] = alarm['operand1']
        alarmInfo['Combine_method'] = alarm['combine_method']
        alarmInfo['Condition2'] = alarm['condition2']
        alarmInfo['Operand2'] = alarm['operand2']
        alarmInfo['Device'] = alarm['device']
        alarmInfo['Var_name'] = alarm['var_name']
        alarmInfo['Address'] = alarm['address']
        alarmConfiguration_list.append(alarmInfo) # 依次将告警信息添加至
alarmConfiguration_list 中
```

(continues on next page)

(continued from previous page)

```
    logger.info(alarmConfiguration_list) # 打印告警点表
    wizard_api.mqtt_publish("Config/AlarmInfo", json.dumps(alarmConfiguration_list),
↪ 1) # 调用 wizard_api 模块中的 mqtt_publish 方法将 alarmConfiguration_list 数据通过
主题 "Config/AlarmInfo", qos 等级 1 发送至 MQTT 服务器
```

- 发布示例 6：使用 get_global_parameter 获取全局参数设置

  本示例实现了获取 "全局参数" 中设置的参数 device_id，并通过通配符 ${device_id} 的配置方式配置 MQTT 主题。发布和代码配置示例如下：

```
import logging
"""
在网关中打印日志通常有两种办法。
1.import logging: 使用 logging.info(XXX) 打印日志，该方法的日志显示不受全局参数页面中的
日志等级参数控制。
2.from common.Logger import logger: 使用 logger.info(XXX) 打印日志，该方法的日志显示受
全局参数页面中的日志等级参数控制。
"""

def vars_upload_test(data_collect, wizard_api): # 定义发布主函数
    global_parameter = wizard_api.get_global_parameter() # 定义全局参数变量
    logging.info(global_parameter) # 打印全局参数变量
    value_list = [] # 定义数据列表
    for device, val_dict in data_collect['values'].items(): # 遍历 values 字典，该字
典中包含设备名称和设备下的变量数据
        value_dict = { # 自定义数据字典
                      "Device": device,
                      "DeviceID": global_parameter["device_id"], # 获取全局变量中定义
的设备 ID
                      "timestamp": data_collect["timestamp"],
                      "Data": {}
                      }
        for id, val in val_dict.items(): # 遍历变量数据，为 Data 字典赋值
            value_dict["Data"][id] = val["raw_data"]
        value_list.append(value_dict) # 依次将 value_dict 添加到 value_list 中
    logging.info(value_list) # 在 App 日志中打印 value_list，数据格式为 [{'Device':
↪'S7-1200', 'DeviceID': '1', 'timestamp': 1589538347.5604711, 'Data': {'Test1':␣
↪False, 'Test2': 12}}]
```

(continues on next page)

(continued from previous page)

```
    return value_list # 将 value_list 发送给 App，App 将自行按照采集时间顺序上传至
MQTT 服务器。如果发送失败则缓存数据等待连接恢复后按时间顺序上传至 MQTT 服务器
```

订阅

自定义订阅中包含以下项：

- 名称：自定义订阅名称
- 主题：订阅主题，与 MQTT 服务器发布的数据主题保持一致
- Qos(MQTT)：订阅 Qos，建议与 MQTT 服务器的 Qos 保持一致
- 主函数：主函数名称，即入口函数名称，与脚本中的入口函数名称保持一致
- 脚本：使用 Python 代码自定义组包和处理逻辑，订阅中的主函数参数包括：
  - 参数 1：该参数为接收到的主题，数据类型为 string
  - 参数 2：该参数为接收到的数据，数据类型为 string
  - 参数 3：该参数为 Device Supervisor 提供的 api 接口，参数说明见 *Device Supervisor 的 api 接口说明*

以下是三个常见的自定义订阅方法示例：

- 订阅示例 1：下发变量名称和变量值写入 PLC 数据且不返回写入结果

  本示例实现了从 MQTT 服务器下发指定命令修改变量数值，发布和代码配置示例如下：

```
import logging
import json

def ctl_test(topic, payload, wizard_api): # 定义订阅主函数
    logging.info(topic) # 打印订阅主题，假定 topic 为 write/plc
    logging.info(payload) # 打印订阅数据，假定 payload 数据为{"method":"setValue",
↪"TagName":"SP1", "TagValue":12.3}
    payload = json.loads(payload) # 反序列化订阅数据
    if payload["method"] == "setValue": # 检测是否为写入数据
        message = {payload["TagName"]:payload["TagValue"]} # 定义下发消息，包括下发的
变量名称和变量值
        wizard_api.write_plc_values(message) # 调用 wizard_api 模块中的 write_plc_
↪values 方法，将 message 字典中的数据下发至指定变量
```

- 订阅示例 2：下发设备名称，变量名称和变量值写入 PLC 数据且不返回写入结果

本示例实现了从 MQTT 服务器下发指定命令修改变量数值，发布和代码配置示例如下：

```
import logging
import json

def ctl_test(topic, payload, wizard_api): # 定义订阅主函数
    logging.info(topic) # 打印订阅主题
    logging.info(payload) # 打印订阅数据
    # 假定 payload 数据为{"method":"setValue","Device":"Modbus_test", "TagName":"SP1
↪", "TagValue":12.3}
    payload = json.loads(payload) # 反序列化订阅数据
    data_dict = {payload["TagName"]:payload["TagValue"]} # 定义下发的数据字典，包含下
发的变量名称和变量值
    var_device = payload["Device"] # 定义设备名称
    if payload["method"] == "setValue": # 检测是否为写入数据
        message = {var_device:data_dict} # 定义下发消息，包括设备名称和下发的数据字典
        wizard_api.write_plc_values(message) # 调用 wizard_api 模块中的 write_plc_
↪values 方法，将 message 字典中的数据下发至指定变量
```

- 订阅示例 3：写入变量数据并返回写入结果

  本示例实现了从 MQTT 服务器下发指定命令修改变量数值并返回修改结果，发布和代码配置示例如下：

```
import logging
import json

def ctl_test(topic, payload, wizard_api): # 定义订阅主函数
    logging.info(topic) # 打印订阅主题，假定 topic 为 request/v1
    logging.info(payload) # 打印订阅数据
    # 假定 payload 数据为{"method":"setValue","Device":"Modbus_test", "TagName":"SP1
↪", "TagValue":12.3}
    payload = json.loads(payload) # 反序列化订阅数据
    data_dict = {payload["TagName"]:payload["TagValue"]} # 定义下发的数据字典，包含下
发的变量名称和变量值
    var_device = payload["Device"] # 定义设备名称
    if payload["method"] == "setValue": # 检测是否为写入数据
        message = {var_device:data_dict} # 定义下发消息，包括设备名称和下发的数据字典
        ack_tail = [topic.replace('request', 'response'), message] # 定义确认数据，包
括响应的主题和消息
```

(continues on next page)

(continued from previous page)

```
        logging.info(message)
        wizard_api.write_plc_values(message, ack, ack_tail, timeout = 0.5) # 调用
wizard_api 模块中的 write_plc_values 方法，将 message 字典中的数据下发至指定变量；定义
该方法的回调函数名称为 ack 并将 ack_tail 传递给回调函数 ack

def ack(send_result, ack_tail, wizard_api): # 定义回调函数 ack
    topic = ack_tail[0] # 定义响应主题
    if isinstance(send_result,tuple): # 检测 send_result 的数据类型是否为元组，为元组
则说明下发超时
        resp_data = {"Status":"timeout", "Data":ack_tail[1]} # 定义下发超时的响应数据
    else:
        resp_data = {"Status":send_result[0]["result"], "Data":ack_tail[1]} # 定义下
发未超时的响应数据
    wizard_api.mqtt_publish(topic, json.dumps(resp_data), 0) # 调用 wizard_api 模块中
的 mqtt_publish 将响应数据发送给 MQTT 服务器
```

- 订阅示例 4：立即召回数据

  本示例实现了从 MQTT 服务器下发指定命令时，立即读取所有变量数值并发送至 MQTT 服务器，发布和代码配置示例如下：

```
from common.Logger import logger
import json

def recall_test(topic, payload, wizard_api): # 定义订阅主函数
    logger.info(topic) # 打印订阅主题，假定 topic 为 recall/v1
    payload = json.loads(payload) # 反序列化订阅数据
    logger.info(payload) # 打印订阅数据，假定 payload 数据为{"command":"Upload␣
↪immediately"}
    if payload["command"] == "Upload immediately": # 检测是否需要数据召回
        wizard_api.recall_data(recall) # 调用 wizard_api 模块中的 recall_data 方法，
定义该方法的回调函数名称为 recall

def recall(data_collect, tail, wizard_api): # 定义回调函数 recall
    logger.info(data_collect) # 打印读取到的数据
    value_list = [] # 定义数据列表
    for device, val_dict in data_collect["values"].items(): # 遍历 values 字典，该字
典中包含设备名称和设备下的变量数据
        value_dict = { # 自定义数据字典
```

(continues on next page)

(continued from previous page)

```
                    "DeviceSN": device,
                    "timestamp": data_collect["timestamp"],
                    "Data": []
                     }
        for id, val in val_dict.items(): # 遍历变量数据，为 Data 列表赋值
            var_dict = {} # 定义变量字典
            var_dict[id] = val["raw_data"]
            value_dict["Data"].append(var_dict) # 依次将变量字典添加到 value_dict 中
        value_list.append(value_dict) # 依次将数据字典添加到 value_list 中
    logger.info(value_list) # 打印 value_list
    wizard_api.mqtt_publish("v1/xxx/yyy", json.dumps(value_list), 1) # 调用 wizard_
↪api 模块中的 mqtt_publish 方法将 value_list 数据通过主题 "v1/xxx/yyy"，qos 等级 1 发
送至 MQTT 服务器
```

Device Supervisor 的 api 接口说明

Device Supervisor 提供的 api 接口，包含以下方法：

- `mqtt_publish`：MQTT 发布消息方法，用于将指定数据通过相应的主题发送到 MQTT 服务器并返回发送结果：发送成功（True），发送失败（False），使用示例请参考*发布示例 3*。该方法包含以下参数：
  - 参数 1：MQTT 主题，数据类型为 `string`。通过至该主题发送数据到 MQTT 服务器
  - 参数 2：需要发送的数据
  - 参数 3：qos 等级（包括 0/1/2 三种等级）
- `save_data`：存储数据至数据库方法，被存储的数据将在 MQTT 连接正常时按先存先传的方式上传至 MQTT 服务器，使用示例请参考*发布示例 3* 和*发布示例 4*。该方法包含以下参数：
  - 参数 1：需要存储的数据。如果调用 `save_data` 时只提供了参数 1，则参数 1 的数据类型为 `dict`，在储存的数据中需具备以 `topic`、`qos` 和 `payload` 为键的键值对，当 MQTT 连接正常后将以存储数据中的 `topic` 和 `qos` 将 `payload` 发送至 MQTT 服务器。
  - 参数 2（可选参数 `group`）：需要存储的数据的分组名称，数据类型为 `string`。如果调用 `save_data` 时提供参数 2，则将以该分组在云服务中关联的主题和 qos 上传参数 1 至 MQTT 服务器。
- `write_plc_values`：下发数据至指定变量方法并支持返回修改结果，使用示例请参考*订阅示例 1*，*订阅示例 2* 和*订阅示例 3*。该方法包含以下参数：
  - 参数 1：下发数据，该数据可以有两种形式：
    - 形式 1：传入一个以变量名称和变量值为键值对的 `dict`。使用此方法修改变量数值时，需要确保该变量的名称在设备列表中唯一，数据格式示例如下：

```
{
    "SP1": 12.3,  # 变量名称和变量值的键值对
    "SP2": 12.4
}
```

* 形式 2：传入一个包含设备名称、变量名称和变量值的 dict，数据格式示例如下：

```
{
    "S7-1200":  # 设备名称
    {
        "SP1": 12.3,  # 变量名称和变量值的键值对
        "SP2": 12.4
    }
}
```

  - 参数 2（可选参数 callback）：返回修改结果的回调函数名称，回调函数说明见 *write_plc_values 回调函数说明*
  - 参数 3（可选参数 tail）：已有参数 2 时，可将需要传递给返回修改结果的回调函数的数据赋值给参数 3
  - 参数 4（可选参数 timeout）：写入超时时间，数据类型为整数。默认为 60 秒

- get_tag_config：获取点表配置方法，点表配置包括 PLC、变量、分组和告警配置，使用示例请参考*发布示例 5*。该方法包含以下参数：
  - 参数 1：获取点表配置的回调函数的名称，回调函数说明见 *get_tag_config 回调函数说明*
  - 参数 2（可选参数 tail）：可将需要传递给获取点表配置的回调函数的数据赋值给参数 2
  - 参数 3（可选参数 timeout）：获取点表超时时间，数据类型为整数。默认为 60 秒
- recall_data：立即读取所有变量数值方法，使用示例请参考*订阅示例 4*。该方法包含以下参数：
  - 参数 1：立即读取所有变量数值的回调函数的名称，回调函数说明见 *recall_data 回调函数*
  - 参数 2（可选参数 tail）：可将需要传递给立即读取所有变量数值的回调函数的数据赋值给参数 2
  - 参数 3（可选参数 timeout）：立即读取所有变量的超时时间，数据类型为整数。默认为 60 秒
- get_global_parameter：获取全局参数设置方法，使用示例请参考*发布示例 6*。该方法会返回一个全局参数设置的字典，数据格式如下：

```
{
    'gateway_sn': 'GT902XXXXXXXXXX', # 系统参数，网关序列号
    'log_level': 'INFO', # 系统参数，日志等级
    'catch_recording': 100000, # 系统参数，最大可缓存的变量数据的 MQTT 消息数量
```

(continues on next page)

(continued from previous page)

```
    'warning_recording': 2000, # 系统参数，最大可缓存的告警数据的 MQTT 消息数量
    'device_id': '1' # 自定义全局参数
}
```

回调函数说明

- write_plc_values 回调函数说明 write_plc_values 回调函数包含以下参数，使用示例请参考订阅示例 *3*:
  - 参数 1: write_plc_values 方法的写入结果
    * 写入超时时返回值为

```
("error", -110, "timeout")
```

    * 写入成功时返回值格式为:

```
[
{
    'value': 12, # 写入值
    'device': 'S7-1200', # 写入设备
    'var_name': 'Test2', # 写入变量的名称
    'result': 'OK', # 写入结果。写入成功: OK，写入失败: Failed
    'error': '' # 写入错误，当写入成功时该项为空
}]
```;

    * 写入失败时返回值格式为:

```
[
{
    'value': 12.3,
    'device': 'Modbus_test',
    'var_name': 'SP1',
    'result': 'Failed',
    'error': "Device 'Modbus_test' not found."
}]
```

  - 参数 2: write_plc_values 方法中配置的参数 3，如果未在 write_plc_values 中配置参数 3，则该参数为 None
  - 参数 3: 该参数为 Device Supervisor 提供的 api 接口，参数说明见 *Device Supervisor 的 api 接口说明*

- get_tag_config 回调函数说明 get_tag_config 回调函数包含以下参数，使用示例请参考发布示例 *5*:
  - 参数 1: get_tag_config 方法返回的点表配置。获取点表超时时返回值为 ("error", -110, "timeout")，正常返回点表配置时数据格式如下:

```
{
    'devices': [ # 设备点表
    {
        'protocol': 'ISO-on-TCP', # 设备协议
        'device_name': 'S7-1200', # 设备名称
        'ip': '10.5.16.73', #IP 地址
        'port': 102, # 端口号
        'rack': 0, # 机架号
        'slot': 0, # 槽号
        'id': '6358f50294dc11ea8d890018050ff046' # 设备 ID
    }],
    'groups': [ # 分组点表
    {
        'group_name': 'warning', # 分组名称
        'polling_interval': 10, # 采集间隔
        'upload_interval': '', # 上报间隔间隔
        'group_type': 'alarm', # 分组类型。collect: 采集组, alarm: 告警组
        'id': '84c371902eb911eabab11a4f32d1ee44' # 分组 ID
    }],
    'warning': [ # 告警点表
    {
        'warn_name': 'Warn1', # 告警名称
        'group': 'warning', # 告警所属分组
        'quotes': 1, # 告警变量来源。0: 直接使用地址, 1: 引用地址
        'device': 'S7-1200', # 告警变量所属设备
        'alarm_content': '速度超过 30! ', # 告警描述
        'condition1': 'Gt', # 告警条件 1。Eq: 等于,Neq: 不等于,Gt: 大于,Gne: 大于等
于,Lne: 小于等于,Lt: 小于
        'operand1': '30', # 告警阈值 1
        'combine_method': 'And', # 告警条件连接方式。None: 空, And: &&, Or: ||
        'condition2': 'Lt', # 告警条件 2
        'operand2': '50', # 告警阈值 1
        'var_name': 'Test2', # 告警变量名称
        'var_id': '96c93c3094dd11eabd400018050ff046', # 告警变量 ID
        'size': 1, # 告警变量的数据类型为 STRING 时的字符串长度
        'float_repr': 2, # 告警变量的数据类型为 FLOAT 时变量小数点后的数据长度
```

(continues on next page)

(continued from previous page)

```
        'id': '9165ed78943e11ea8a000018050ff046', # 告警 ID
        'address': 'DB6.2', # 告警变量地址
        'protocol': 'ISO-on-TCP', # 告警设备通讯协议
        'data_type': 'WORD', # 告警变量数据类型
        'register_type': 'DB', # 告警变量寄存器类型
        'register_addr': 2, # 告警变量寄存器地址
        'read_write': 'read/write', # 告警读写权限。read: 只读、write: 只写、
↪'read/write: 可读可写
        'mode': 'realtime', # 告警变量采集模式
        'unit': '', # 告警变量单位
        'desc': '', # 告警变量描述
        'dbnumber': 6, # 告警变量寄存器类型为 DB 时变量的 DB 号
        'register_bit': '' # 告警变量数据类型为 BOOL 或 BIT 时变量的位偏移
    }],
    'vars': [ # 变量点表
    {
        'device': 'S7-1200', # 变量所属设备的名称
        'protocol': 'ISO-on-TCP', # 变量所示设备的通讯协议
        'data_type': 'BOOL', # 变量数据类型
        'register_type': 'I', # 变量寄存器类型
        'var_name': 'Test1', # 变量名称
        'register_addr': 0, # 变量寄存器地址
        'read_write': 'read/write', # 变量读写权限。read: 只读、write: 只写、
↪'read/write: 可读可写
        'mode': 'realtime', # 变量采集模式
        'unit': '', # 变量单位
        'desc': '', # 变量描述
        'group': 'default', # 变量所属分组
        'register_bit': 0, # 变量数据类型为 BOOL 或 BIT 时变量的位偏移
        'size': 1, # 变量的数据类型为 STRING 时的字符串长度
        'float_repr': 2, # 变量的数据类型为 FLOAT 时变量小数点后的数据长度
        'dbnumber': 0, # 变量寄存器类型为 DB 时变量的 DB 号
        'id': 'a1d9439a94dc11eaa2830018050ff046', # 变量 ID
        'address': 'I0.0' # 变量地址
    },
    {
        'device': 'S7-1200',
        'protocol': 'ISO-on-TCP',
        'data_type': 'WORD',
        'register_type': 'DB',
```

(continues on next page)

(continued from previous page)

```
        'var_name': 'Test2',
        'register_addr': 2,
        'read_write': 'read/write',
        'mode': 'realtime',
        'unit': '',
        'desc': '',
        'group': '2222',
        'dbnumber': 6,
        'size': 1,
        'float_repr': 2,
        'register_bit': '',
        'id': '96c93c3094dd11eabd400018050ff046',
        'address': 'DB6.2'
    }]
}
```

  - 参数 2：get_tag_config 方法中配置的参数 3，如果未在 get_tag_config 中配置参数 3，则该参数为 None
  - 参数 3：该参数为 Device Supervisor 提供的 api 接口，参数说明见 *Device Supervisor 的 api 接口说明*

- recall_data 回调函数说明 recall_data 回调函数包含以下参数订阅示例 *4*：
  - 参数 1：recall_data 方法返回的变量数据。获取变量数据超时时返回值为 ("error", -110, "timeout")，正常返回变量数据时数据格式如下：

```
{
    'timestamp': 1589507333.2521989,  # 数据产生时间戳
    'values':  # 数据字典，包括 PLC 名称，变量名称和变量值
    {
        'S7-1200':  #PLC 名称
        {
            'Test1':  # 变量名称
            {
                'raw_data': False,  # 变量值
                'status': 1  # 采集状态，非 1 即采集异常
            },
            'Test2':
            {
                'raw_data': 33,
                'status': 1
```

(continues on next page)

(continued from previous page)

```
            }
        }
    }
}
```

- 参数 2：recall_data 方法中配置的参数 3，如果未在 recall_data 中配置参数 3，则该参数为 None
- 参数 3：该参数为 Device Supervisor 提供的 api 接口，参数说明见 *Device Supervisor 的 api 接口说明*

全局参数

你可以访问“边缘计算 > 设备监控 > 全局参数”页面配置 Device Supervisor 的通用设置。

- 全局参数你可以在全局参数中设置 Device Supervisor 的系统参数或者自行添加通配符参数
  - catch_recording：最大可缓存的变量数据的 MQTT 消息数量，默认为 100000
  - log_level：日志等级，设置不同的日志等级可以在 Device Supervisor 的日志中看到不同等级的日志信息。默认为 INFO
  - warning_recording：最大可缓存的告警数据的 MQTT 消息数量，默认为 2000
  - 自定义通配符：你可以自行添加全局参数作为云服务中的通配符使用。使用方法为 ${参数名称}，如下图所示：

- 串口设置你可以在串口设置中配置 RS485 和 RS232 串口的通讯参数，如下图所示：

其他网关操作

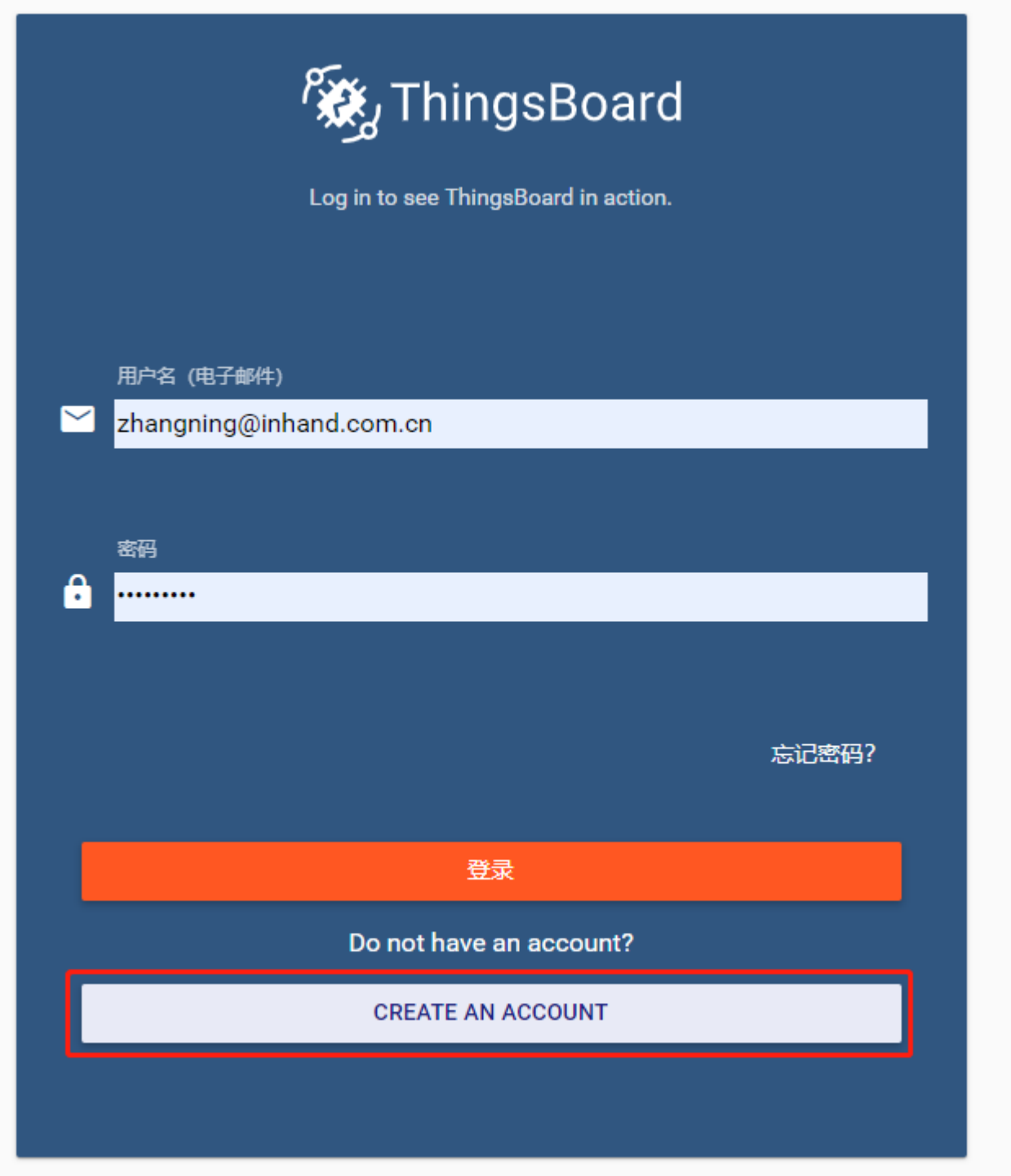
关于网关的其他常用操作请查看IG501 快速使用手册或IG902 快速使用手册。

Thingsboard 参考流程

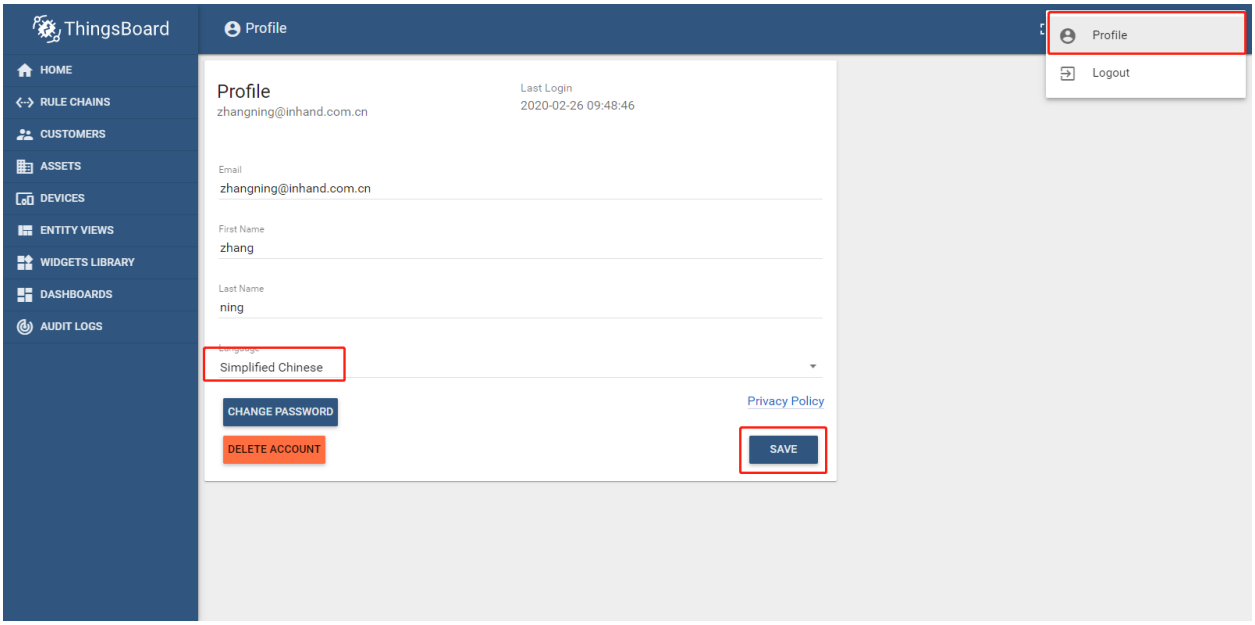
- 添加设备和资产
- 传输 *PLC* 数据到 *Thingsboard* 设备
- 配置可视化仪表板

添加设备和资产

访问 `https://demo.thingsboard.io/login`，输入登录账号和密码。如果未注册过账号则需要先注册账号后再登录（注册账号时需要能够访问海外网络，否则可能无法正常注册）。



登录后，进入属性页面修改语言为简体中文。



- 添加一个资产

  添加成功后如下图所示：

- 添加一个设备

  建立资产与设备的关联。

  添加完成后如下图所示：

传输 PLC 数据到 Thingsboard 设备

资产和设备配置完成后，复制已添加设备的访问令牌并粘贴至网关的云服务页面的用户名参数中以将数据传输至 Thingsboard 中的 `S7-1200` 设备。

随后可在设备的最新遥测中查看已上传的数据。

配置可视化仪表板

- 添加仪表板
- 添加趋势图
- 添加开关
- 添加仪表板

  点击“添加仪表板”，选择“创建新的仪表板”。在“添加仪表板”配置并添加一个仪表板。

  添加完成后单击仪表板名称并选择“打开仪表板”。

  点击“进入编辑模式”。

  点击“实体别名”为仪表板添加实体别名。

在“添加别名”页面参考下图进行配置:

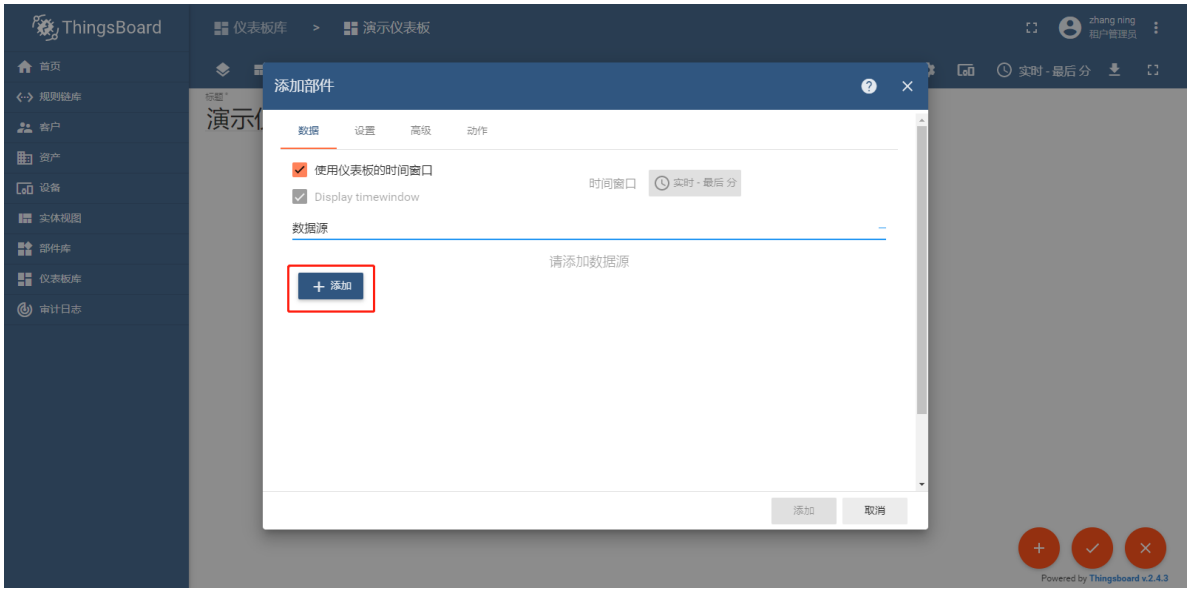
配置完成后保存配置即可。

- 添加趋势图

  在仪表板中点击“进入编辑模式”并点击添加新的部件

在部件中选择“Charts”并点击“Timeseries-Flot”。

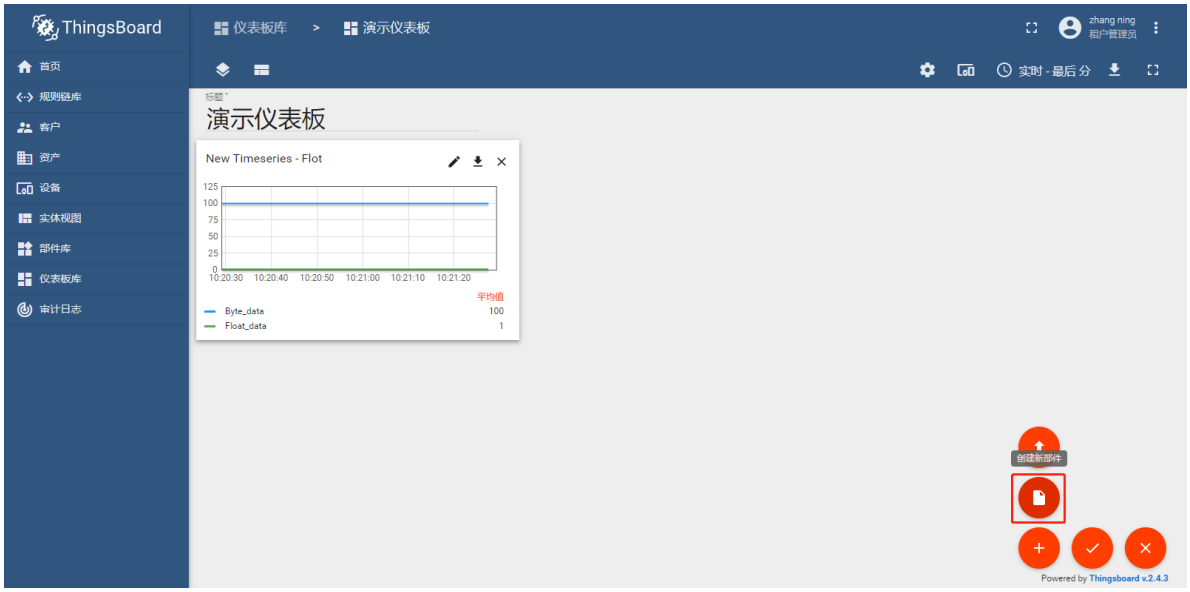
在图表的“数据”页面为趋势图添加展示数据。



添加完成后如下图所示：

- 添加开关

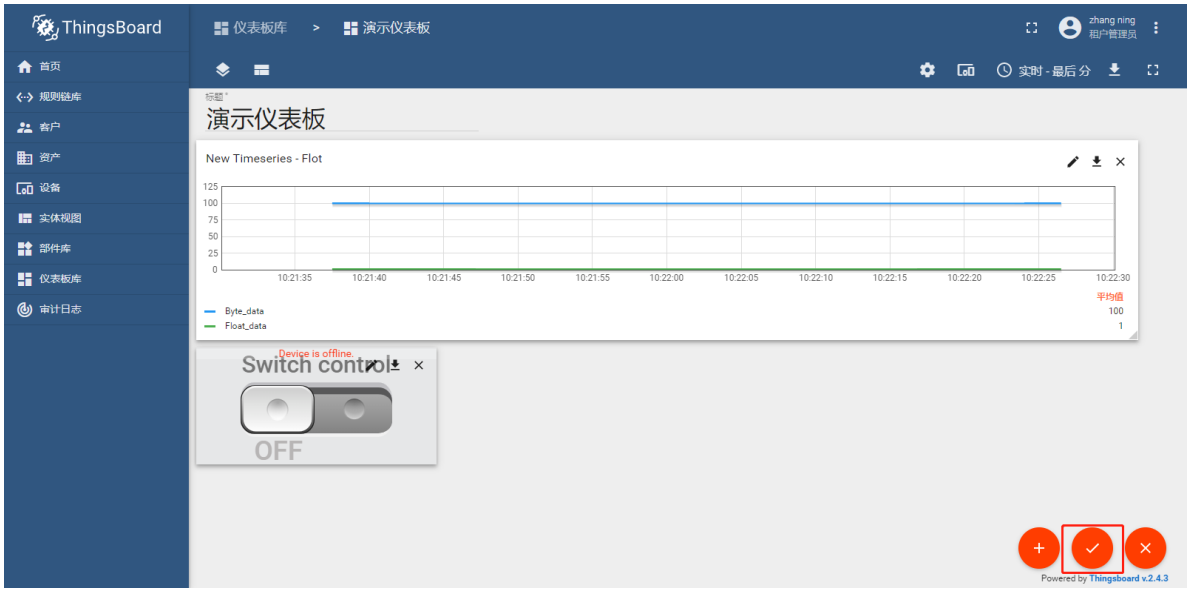
  点击“添加新的部件”，选择“创建新部件”以添加一个控制开关。



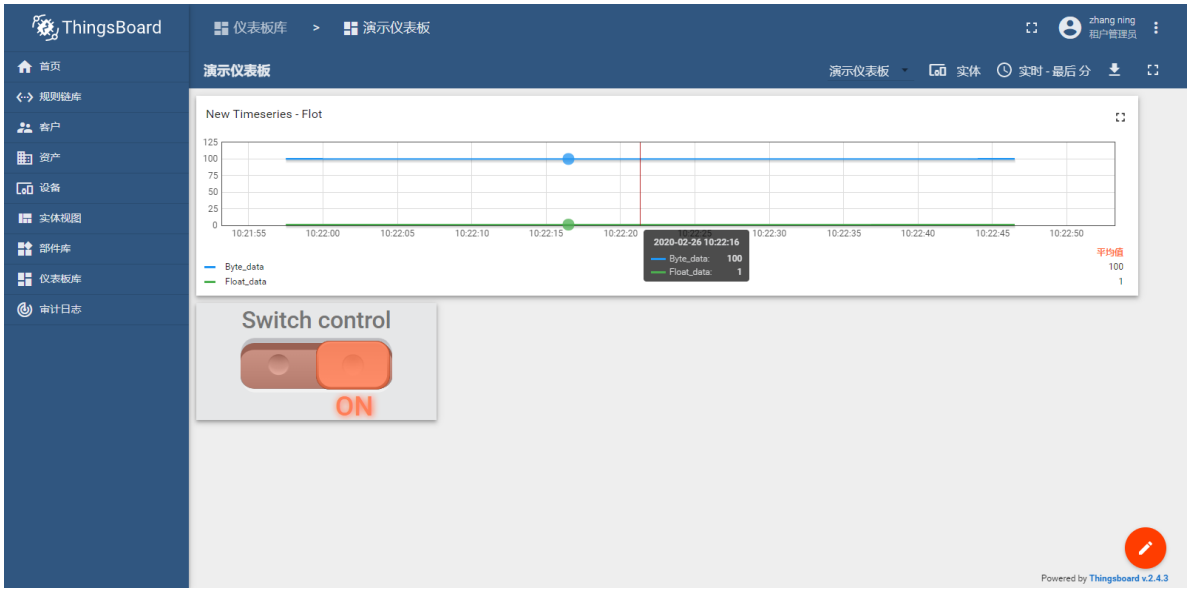
在部件中选择“Control widgets”并点击“Switch control”。

随后选择目标设备。

配置完成后调整部件的大小和布局并保存。



随后可以通过开关下发控制命令以及通过趋势图查看数据趋势。



1.1.6 FAQ

- 查看云服务脚本是否正确打开 Device Supervisor App 日志。脚本编写完成并点击“确定”后，通过日志中的 `Build module: < 主函数名称 >, type: <publish/subscribe>` 信息查看脚本是否构建成功。

  脚本构建成功如下图所示：

  脚本构建失败如下图所示：

- 查看 App 的云服务输出是否正确您可以使用使用 `logger` 和 `logging` 输出重要日志。下图是在运行脚本中的第 6 行使用了 `logging.info` 方法，在日志中可以通过搜索 `<string> 6` 查看输出结果是否符合预期。

1.2 MobiusPi Python 开发快速入门

北京映翰通网络技术股份有限公司的 InGateway 系列产品包含两个大的产品系列，InGateway902 和 InGateway501 系列，以下文档中将 InGateway501 简称为 “IG501”；InGateway902 简称为 “IG902”。MobiusPi 是 InGateway 系列产品二次开发平台的名称，本文档旨在为用户说明如何利用 MobiusPi 进行基于 Python 的二次开发。

- *1. 搭建 MobiusPi 开发环境*
  - *1.1 准备硬件设备及其网络环境*
    - *1.1.1 接通电源并使用网线连接 PC*
    - *1.1.2 设置 LAN 网络参数*
    - *1.1.3 设置 WAN 网络参数*
    - *1.1.4 更新软件版本*
    - *1.1.5 启用 MobiusPi 的调试模式*
  - *1.2 安装 PC 上需要的软件*
    - *1.2.1 安装 Python 解释器*
    - *1.2.2 安装 Visual Studio Code 软件*
    - *1.2.3 安装 OpenSSH*
  - *1.3 准备 VS Code 开发环境*
    - *1.3.1 安装 VS Code 扩展插件*
    - *1.3.2 配置 Python 解释器版本*
    - *1.3.3 配置工程模板*
      - *1.3.3.1 使用映翰通标准工程模板*
      - *1.3.3.2 自定义工程模板*
- *2. 编写第一个 MobiusPi App：Hello World*
  - *2.1 使用模板创建工程*
  - *2.2 编码*
  - *2.3 调试*

1.2.1 1. 搭建 MobiusPi 开发环境

在进行开发前，您需要具备以下条件：

- InGateway
    - InGateway501
        * 固件版本：V2.0.0.r12351 及以上
        * SDK 版本：py2sdk-V1.3.4 及以上
        * 联网并已启用调试模式
    - InGateway902
        * 固件版本：V2.0.0.r12537 及以上
        * SDK 版本：py3sdk-V1.3.5 及以上
        * 联网并已启用调试模式
- PC
    - Python 2.7.X/3.7.X 解释器
    - Visual Studio Code 软件
        * Python 插件
        * Project Templates 插件
        * SFTP 插件
    - OpenSSH

如果您的 MobiusPi 和 PC 已经满足了以上所有条件，则可以跳过这一小节。否则请参考以下说明准备开发环境。

- *1.1 准备硬件设备及其网络环境*
- *1.2 安装 PC 上需要的软件*
- *1.3 准备 VS Code 开发环境*

1.1 准备硬件设备及其网络环境

- *1.1.1 接通电源并使用网线连接 PC*
- *1.1.2 设置 LAN 网络参数*
- *1.1.3 设置 WAN 网络参数*
- *1.1.4 更新软件版本*
- *1.1.5 启用 MobiusPi 的调试模式*

1.1.1 接通电源并使用网线连接 PC

- 准备 IG501 硬件设备

  接通 IG501 的电源并按照拓扑使用以太网线连接 PC 和 IG501。

- 准备 IG902 硬件设备

  接通 IG902 的电源并按照拓扑使用以太网线连接 PC 和 IG902。

1.1.2 设置 LAN 网络参数

- 设置 IG501LAN 网络参数，请参考在局域网访问 IG501。

- 设置 IG902LAN 网络参数，请参考在局域网访问 IG902。

1.1.3 设置 WAN 网络参数

- 设置 IG501 WAN 网络参数，请参考IG501 连接 Internet。
- 设置 IG902 WAN 网络参数，请参考IG902 连接 Internet。

1.1.4 更新软件版本

如需获取 MobiusPi 最新软件版本及其功能特性信息，请联系客服。如需更新软件版本，请参考如下链接：

- 更新 IG501 软件版本
- 更新 IG902 软件版本

1.1.5 启用 MobiusPi 的调试模式

开发过程中，为了在 MobiusPi 上运行并调试 Python 代码，需要启用 MobiusPi 的调试模式。

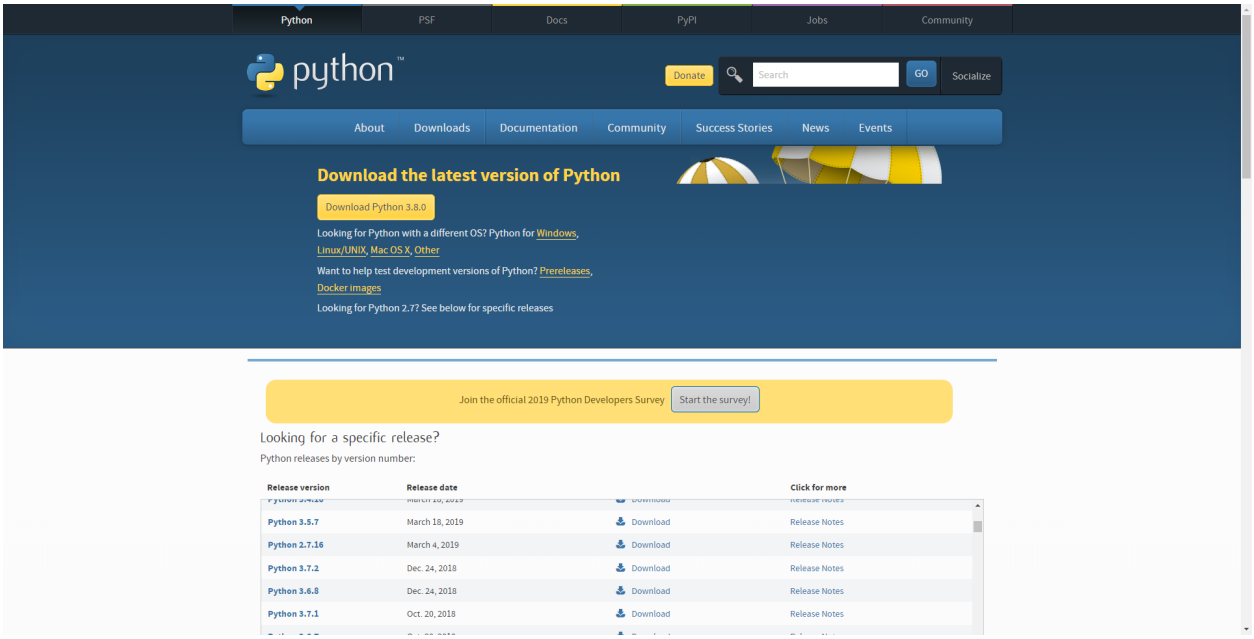
- 启用 IG501 调试模式
- 启用 IG902 调试模式

1.2 安装 PC 上需要的软件

- *1.2.1 安装 Python 解释器*
- *1.2.2 安装 Visual Studio Code 软件*
- *1.2.3 安装 OpenSSH*

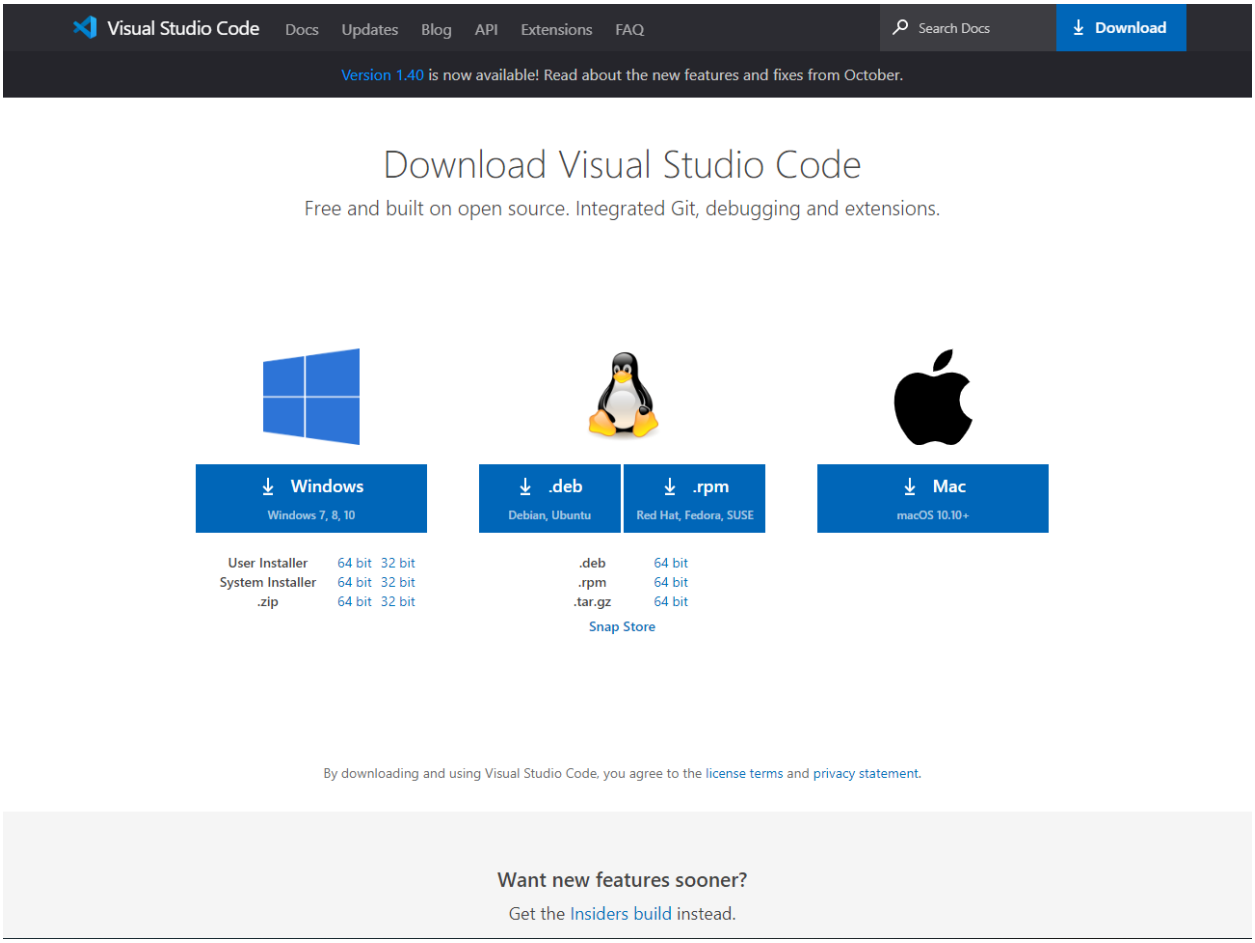
1.2.1 安装 Python 解释器

在开发过程中 PC 需具备 Python2.7.X 或 3.7.X 解释器（建议使用 3.7.X 解释器），您可以访问 https://www.python.org/downloads/ 下载相应的安装包并安装至 PC。

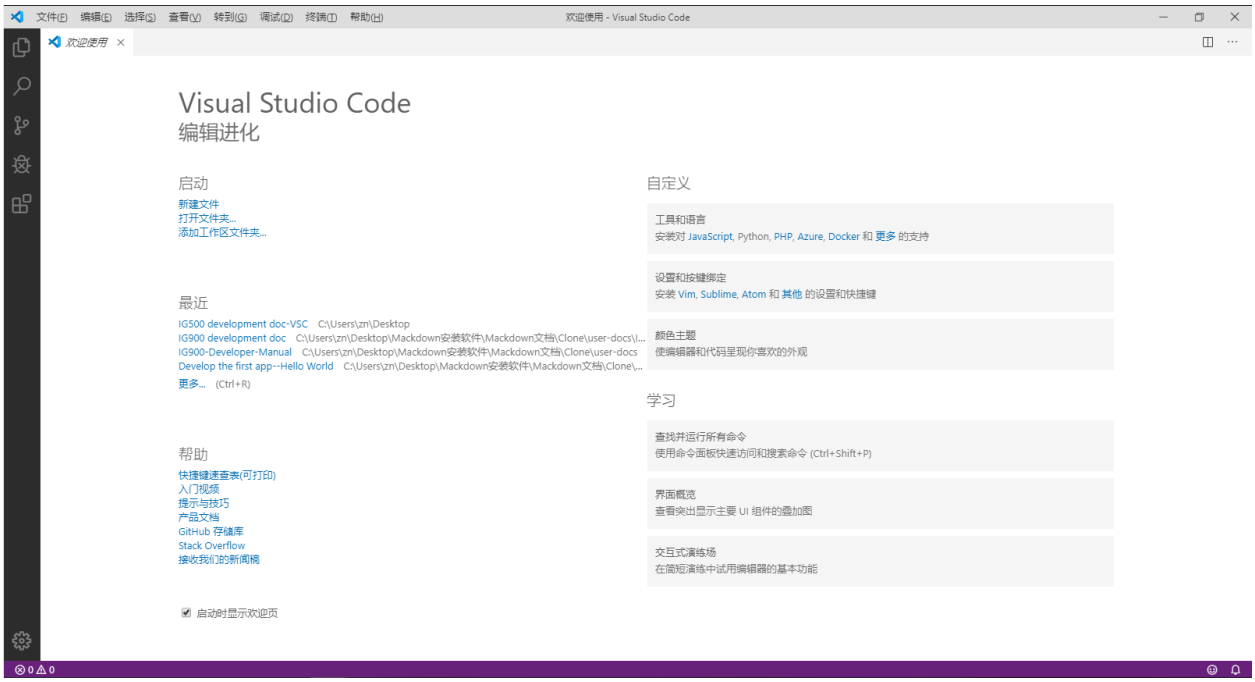


1.2.2 安装 Visual Studio Code 软件

您可以访问：https://code.visualstudio.com/Download 获取相应的 Visual Studio Code 软件（以下简称 VS Code）。



下载完成后运行安装程序，安装成功后打开 VS Code 软件，软件页面如下图。



1.2.3 安装 OpenSSH

您可以在命令提示符中输入 `ssh` 命令查看 PC 是否支持 SSH 协议，当 PC 支持 SSH 协议时如下图所示：

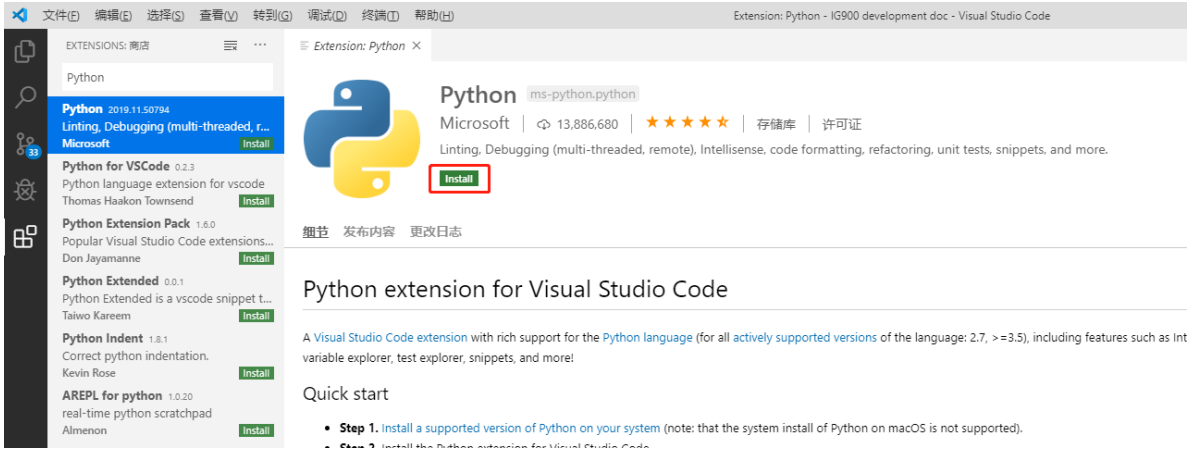
如果 PC 不支持 SSH 协议，您可以访问：https://www.openssh.com 获取相应的 OpenSSH 工具并安装至 PC。

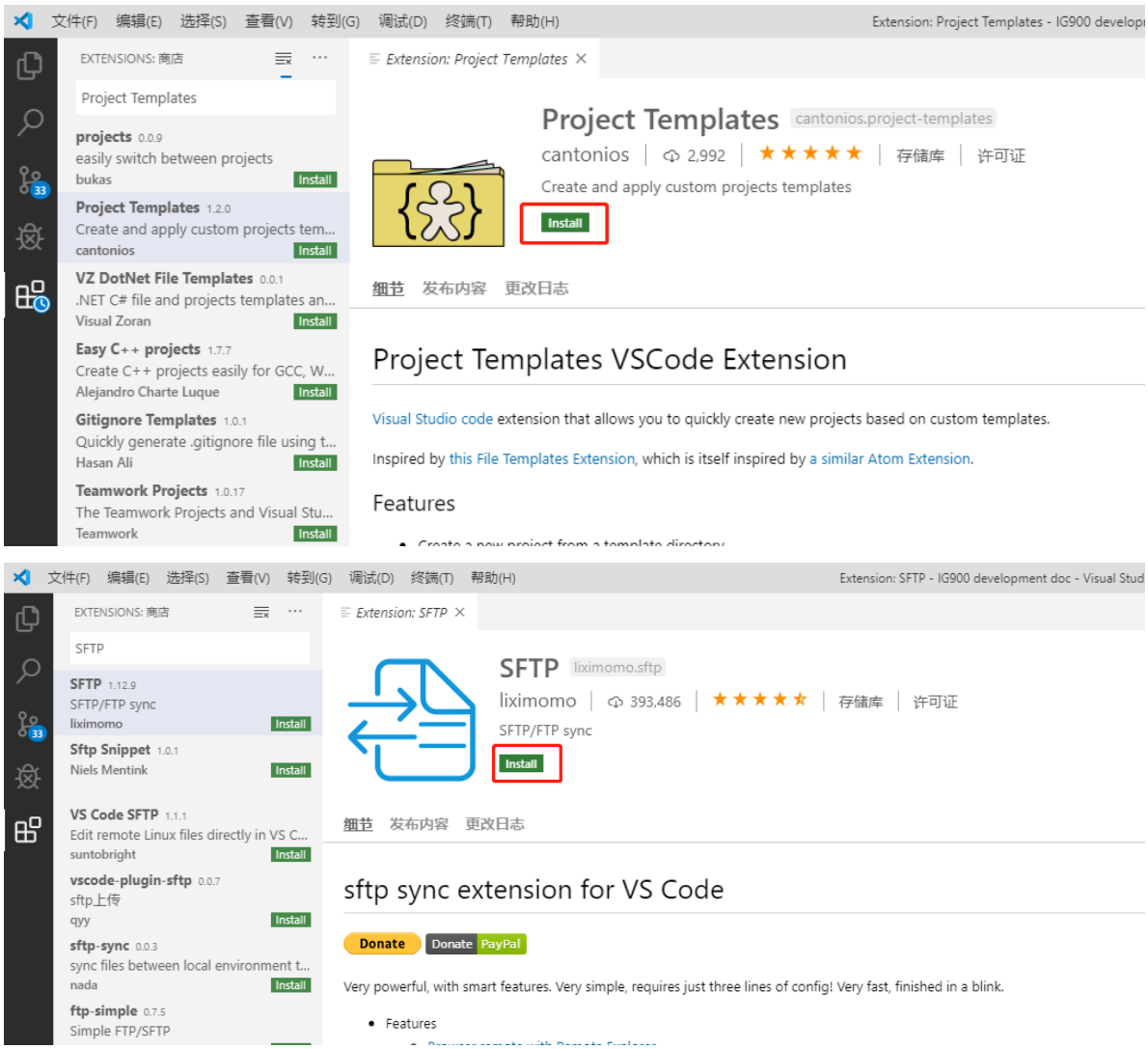
1.3 准备 VS Code 开发环境

1.3.1 安装 VS Code 扩展插件

为了在 MobiusPi 上开发并调试 Python 代码，您要在 VS Code IDE 的 “Extensions” 中安装以下必需的扩展插件：

- `Python`：一个拥有丰富的功能特性的 VS Code Python 扩展插件，包括诸如 IntelliSense、linting、调试、代码导航、代码格式化、Jupyter 笔记本支持、重构、变量资源管理器、测试资源管理器、代码片段等功能！想要了解跟多详细信息，请访问该插件的官方网页。
- `Project Templates`：一个支持基于自定义模板快速创建新项目的 VS Code 扩展插件。我们会发布若干 Python App 模板，您可以使用 Project Templates 导入模板，并快速初始化项目。
- `SFTP`：您可以使用 SFTP Sync 插件将代码上传至 MobiusPi 中。

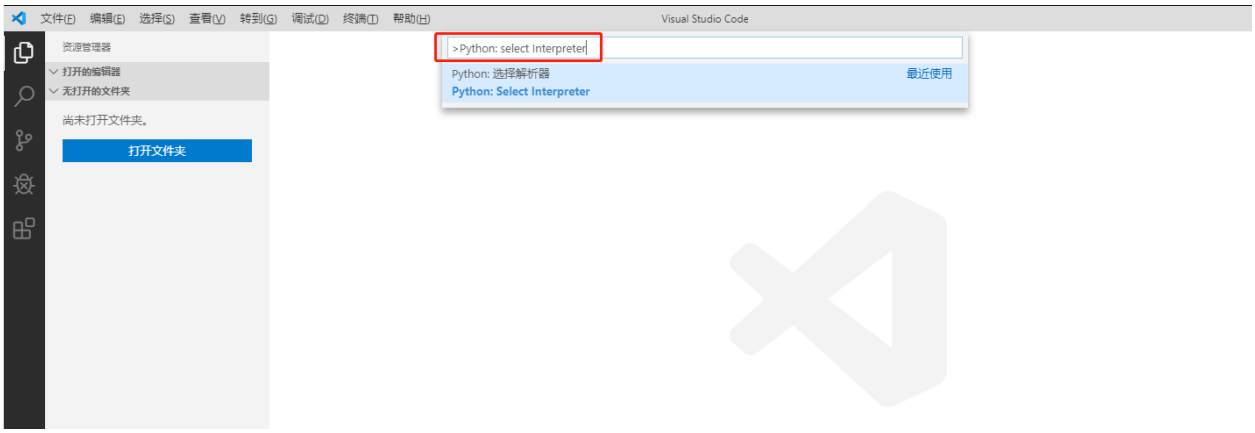




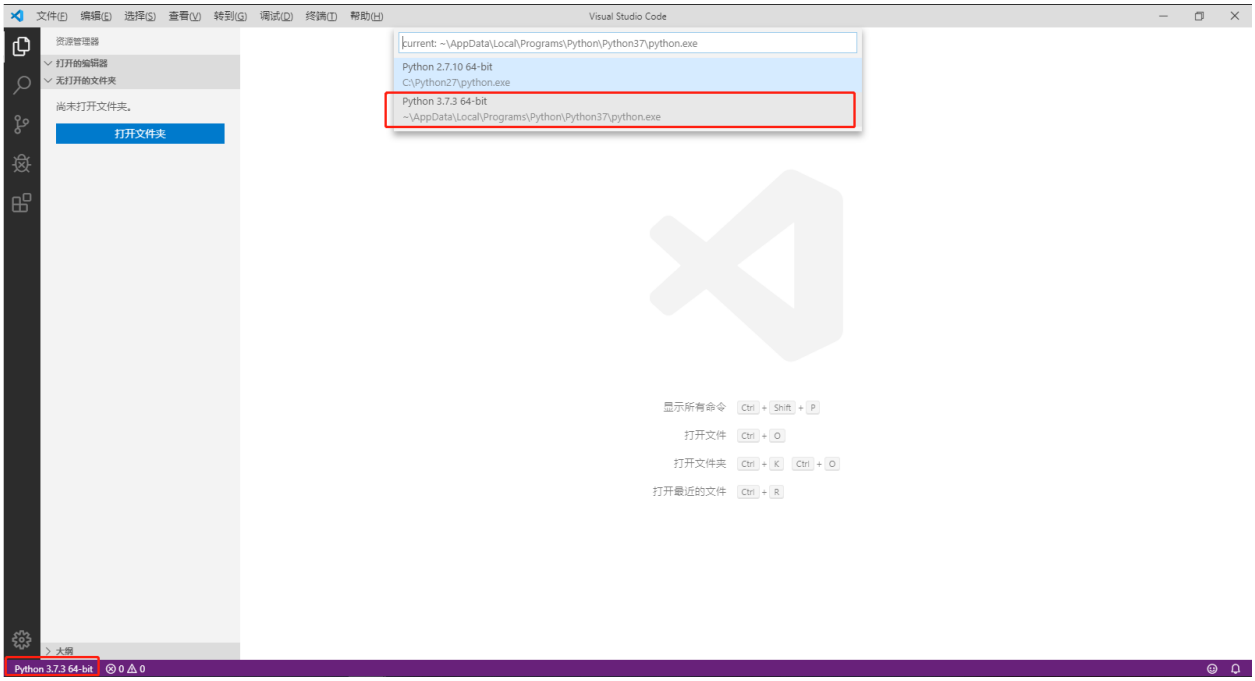
至此，MobiusPi 二次开发平台所必需的插件安装完成。想要了解更多 VS Code 插件，请访问Visual Studio Code 官网。

1.3.2 配置 Python 解释器版本

使用快捷键：`Ctrl+Shift+P` 弹出命令面板，在命令面板中输入 `>Python: select Interpreter`。



根据需要选择相应的 Python 解释器，本教程使用 3.7.X 版本的 Python 解释器（所选择的 Python 解释器版本应同“边缘计算 > Python 边缘计算”页面的 Python 解释器一致）。选择后在 VS Code 左下角可以看到已选择的 Python 解释器版本。

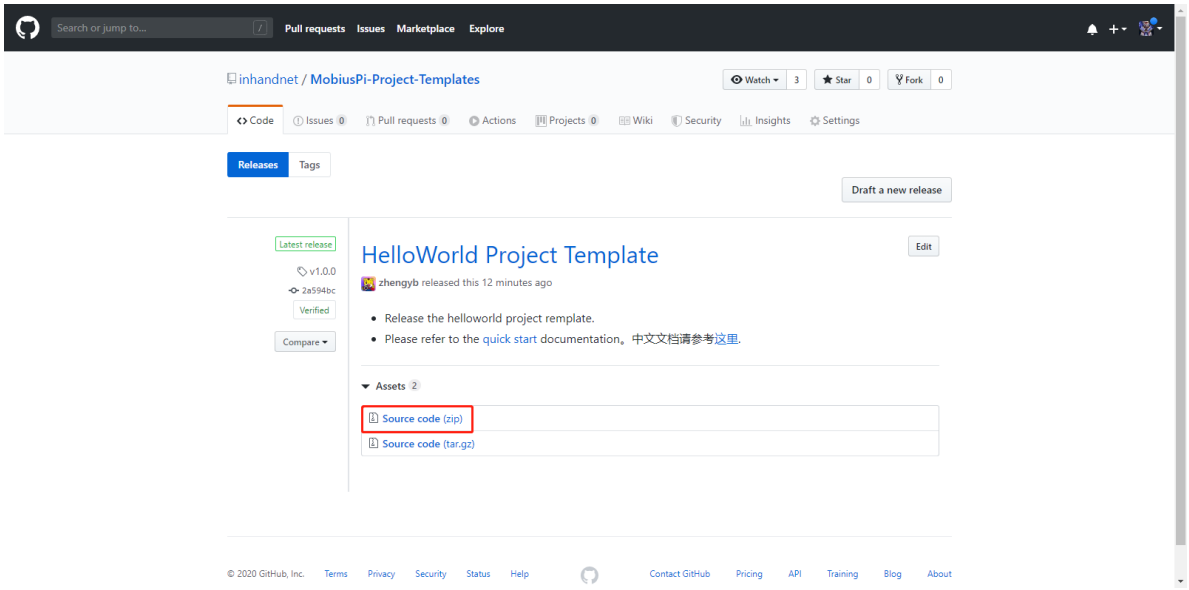


1.3.3 配置工程模板

1.3.3.1 使用映翰通标准工程模板

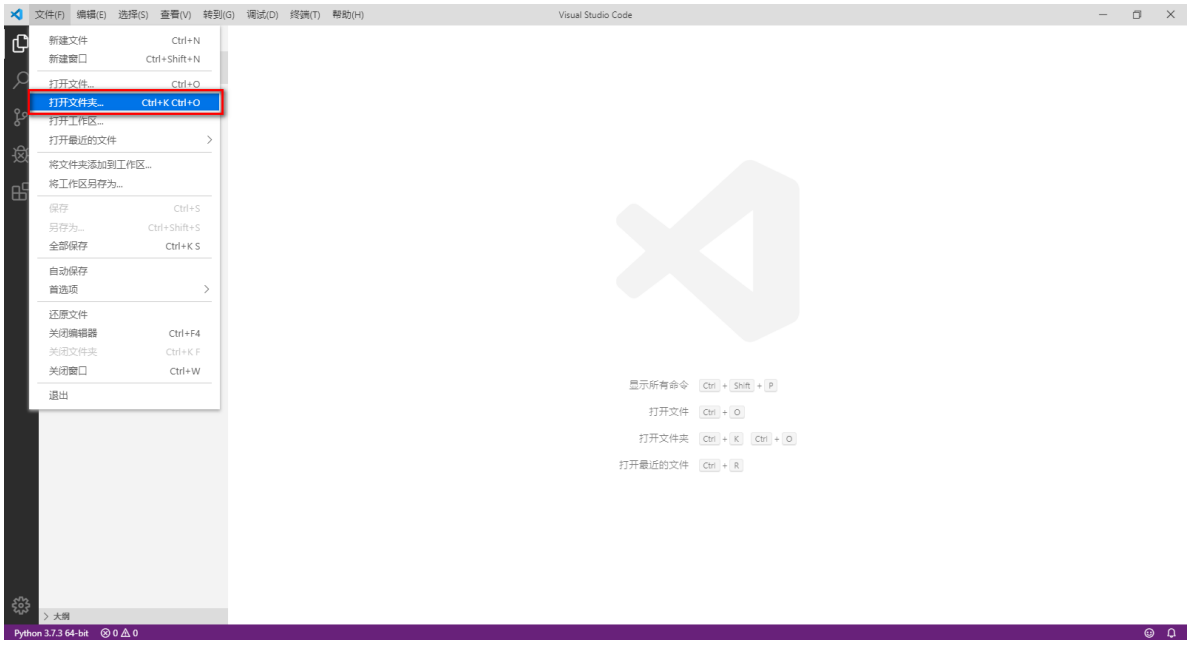
- 步骤 1：您可以从这里下载 MobiusPi 工程模板。

  MobiusPi 提供多种工程模板以方便您快速初始化工程目录。各工程模板的详细说明请参考README.md。本教程使用标准工程模板“helloworld-template”进行演示说明。



- 步骤 2：打开工程模板。

  解压下载后的工程模板压缩包，使用 VS Code 打开解压文件夹中的 helloworld-template 文件夹，点击“文件 > 打开文件夹”并选择解压后的“helloworld-template”文件夹。



打开工程模板文件夹“helloworld-template”后如下图所示，工程模板中包含以下内容：

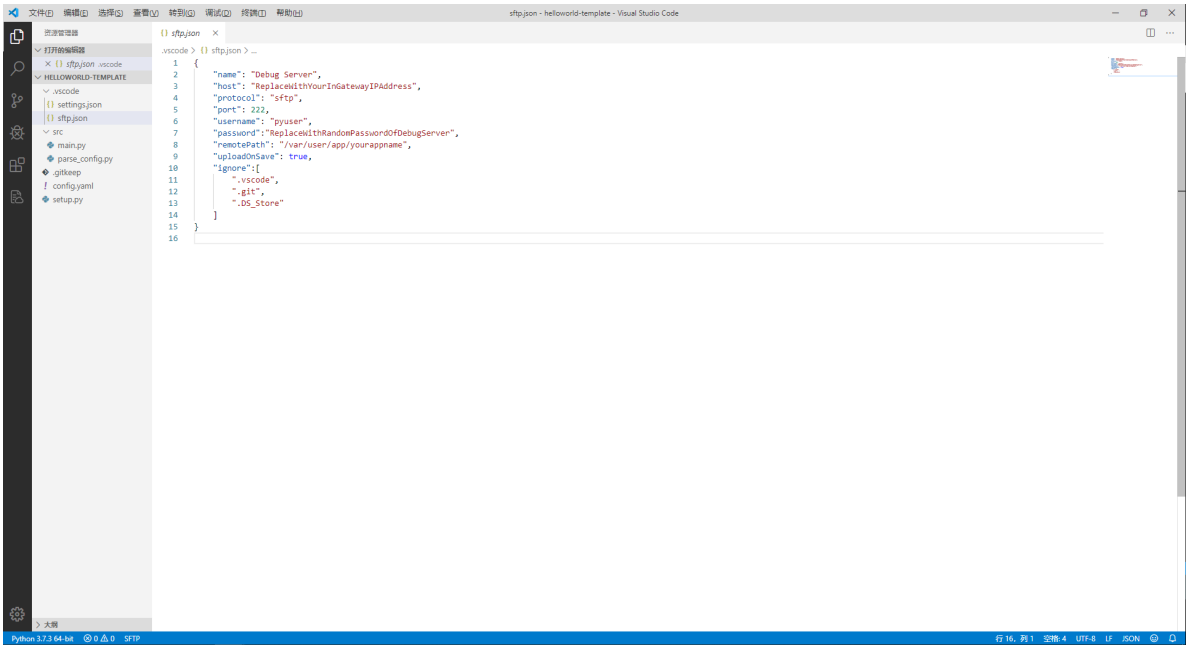
```
.vscode
   sftp.json
build
lib
src
```

(continues on next page)

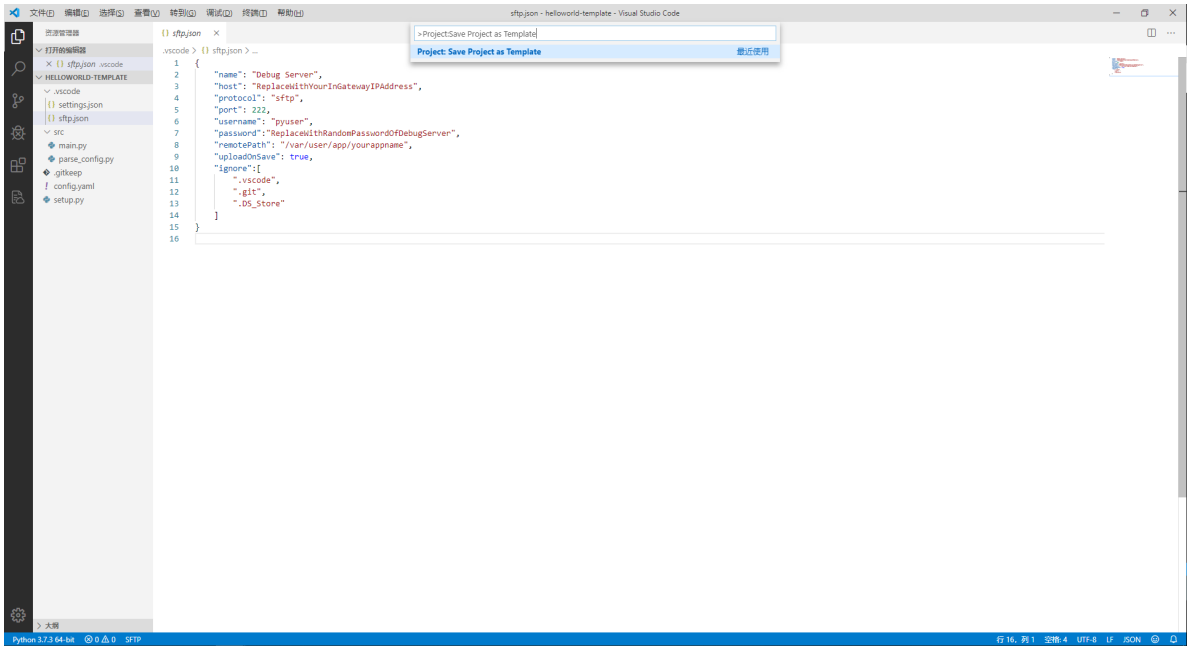
(continued from previous page)

```
    main.py
    parse_config.py
config.yaml
setup.py
```

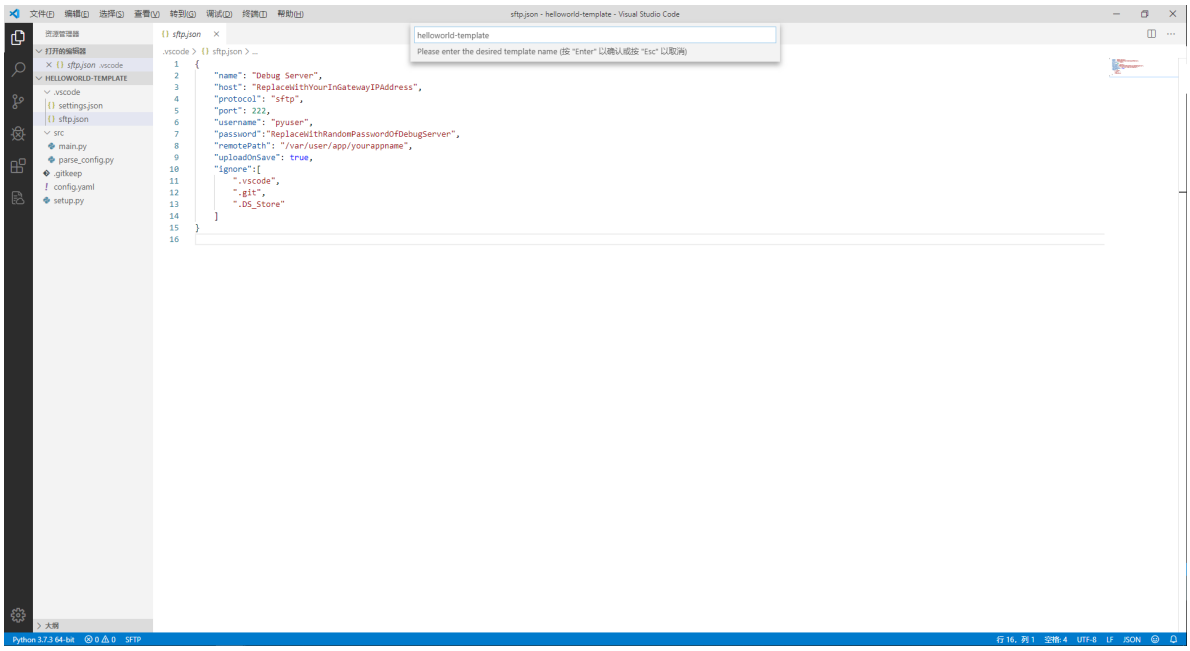
- `.vscode`：VS Code 配置文件夹
  - `sftp.json`：SFTP 插件的配置文件，用于与 MobiusPi 建立 SFTP 连接
- `build`：App 发布包文件夹
- `lib`：App 第三方依赖库文件夹
- `src`：App 源码文件夹
  - `main.py`：App 入口
  - `parse_config.py`：解析 App 配置文件
- `config.yaml`：App 配置文件
- `setup.py`：App 版本、SDK 版本等信息说明



- 步骤 3：在命令面板中输入 `>Project:Save Project as Template` 命令以将当前工程文件存为模板。



定义您模板的名称，如：helloworld-template。



1.3.3.2 自定义工程模板

- 步骤 1：新建一个工程模板文件夹，文件夹中必须包含以下内容。其他文件可根据您的实际情况自行添加

```
.vscode
  sftp.json
```

(continues on next page)

(continued from previous page)

```
build
src
   main.py
setup.py
```

  - `.vscode`：VS Code 配置文件夹（在 VS Code 的命令面板中输入 `>SFTP:Config` 命令可快速创建.vscode 文件夹和 sftp.json 文件）
    * `sftp.json`：SFTP 插件的配置文件，用于与 MobiusPi 建立 SFTP 连接
  - `build`：App 发布包文件夹
  - `src`：App 源码文件夹
    * `main.py`：App 入口
  - `setup.py`：App 版本、SDK 版本等信息说明，建议参照标准模板定义

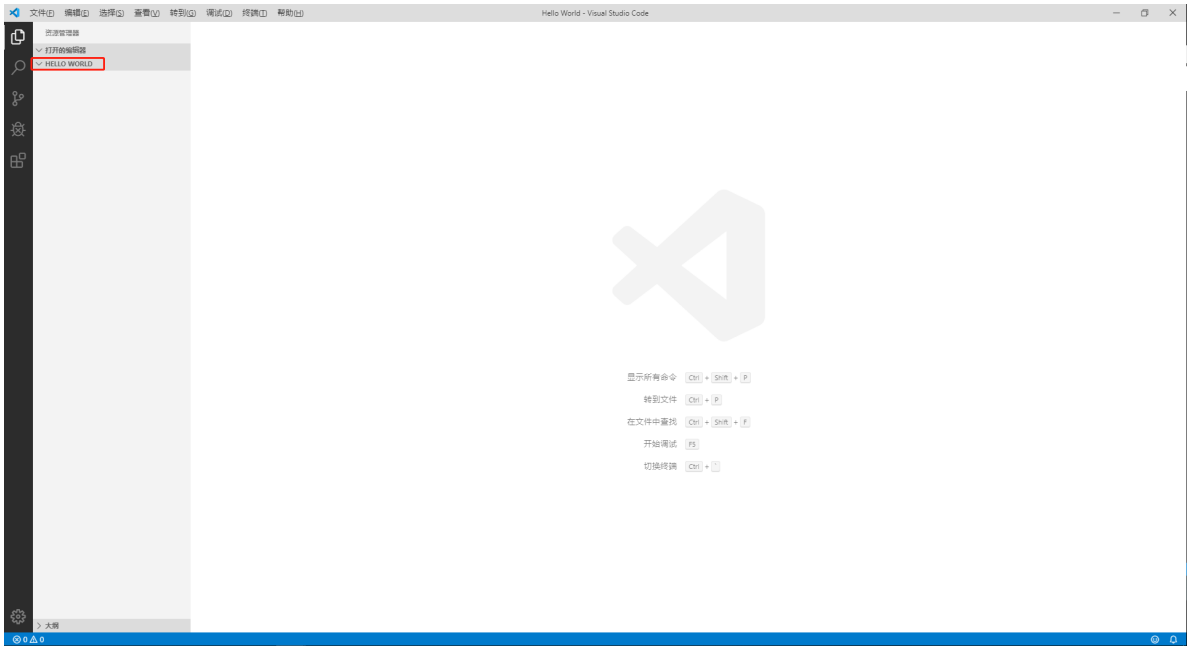
- 步骤 2：使用 VS Code 打开自定义工程模板文件夹，点击“文件 > 打开文件夹”并选择的自定义工程模板文件夹。
- 步骤 3：在命令面板中输入 `>Project:Save Project as Template` 命令以将当前工程文件存为模板。

1.2.2 2. 编写第一个 MobiusPi App：Hello World

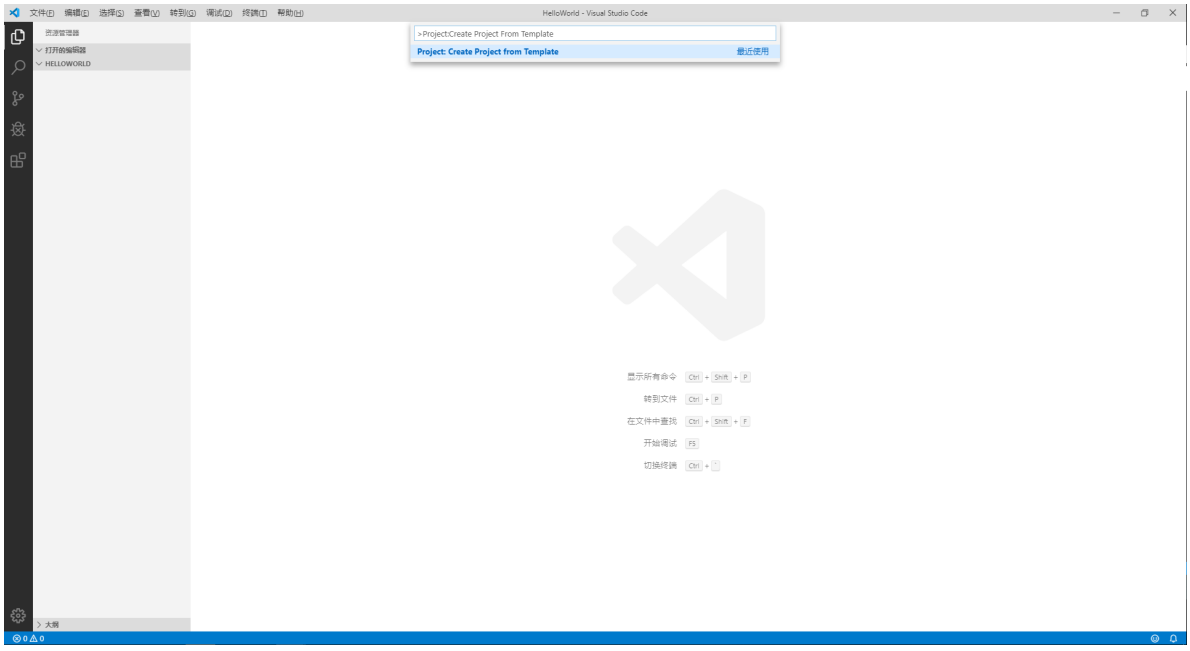
本教程以开发一个“HelloWorld”App 为例说明如何通过 VS Code 实现 MobiusPi Python App 的开发。该 App 具备在 MobiusPi 中每 10 秒打印一条“hello world!”日志以及导入配置文件修改日志内容的功能。

2.1 使用模板创建工程

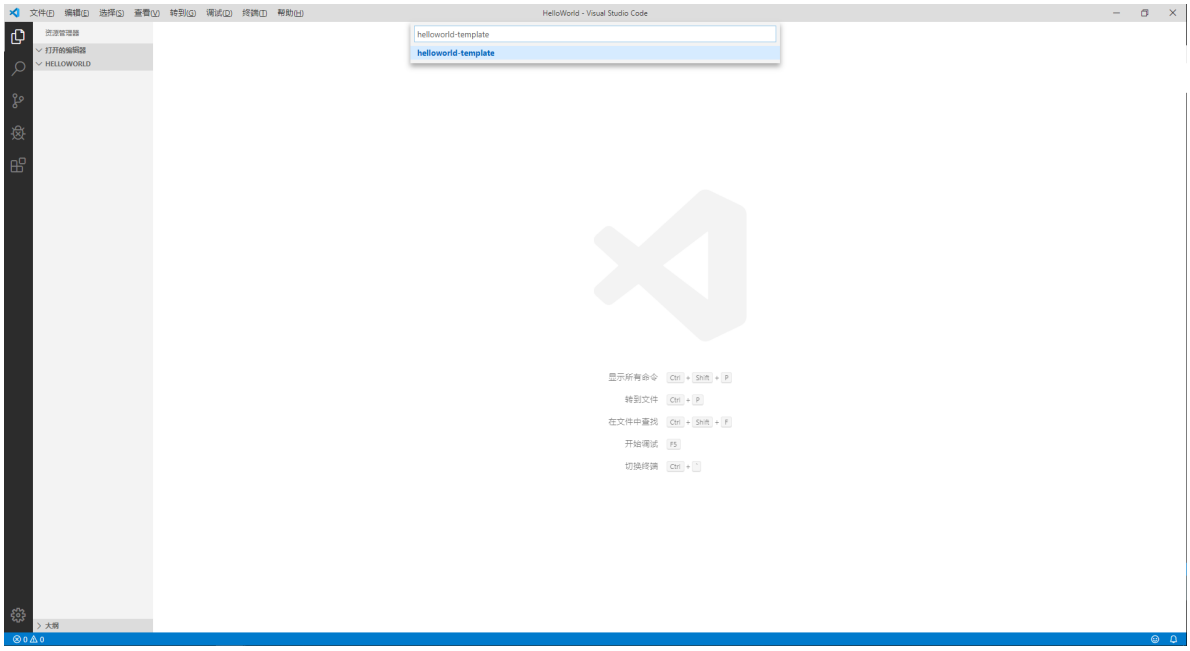
- 步骤 1：使用 VS Code 打开 Python App 的工程文件夹，打开后如下图所示：



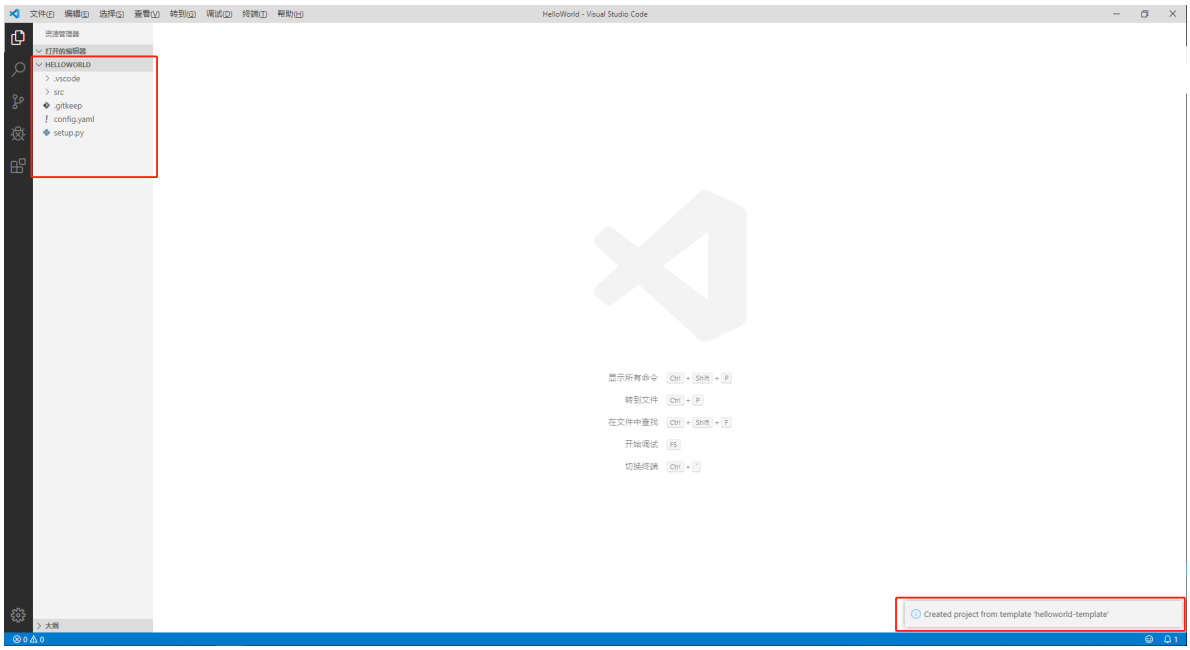
- 步骤 2：在命令面板中输入 `>Project:Create Project From Template` 命令以通过已有模板快速创建工程目录。



- 步骤 3：输入 helloworld-template 模板的名称并回车。



选择模板后 VS Code 会自动在当前工程文件夹下增加模板中包含的文件。



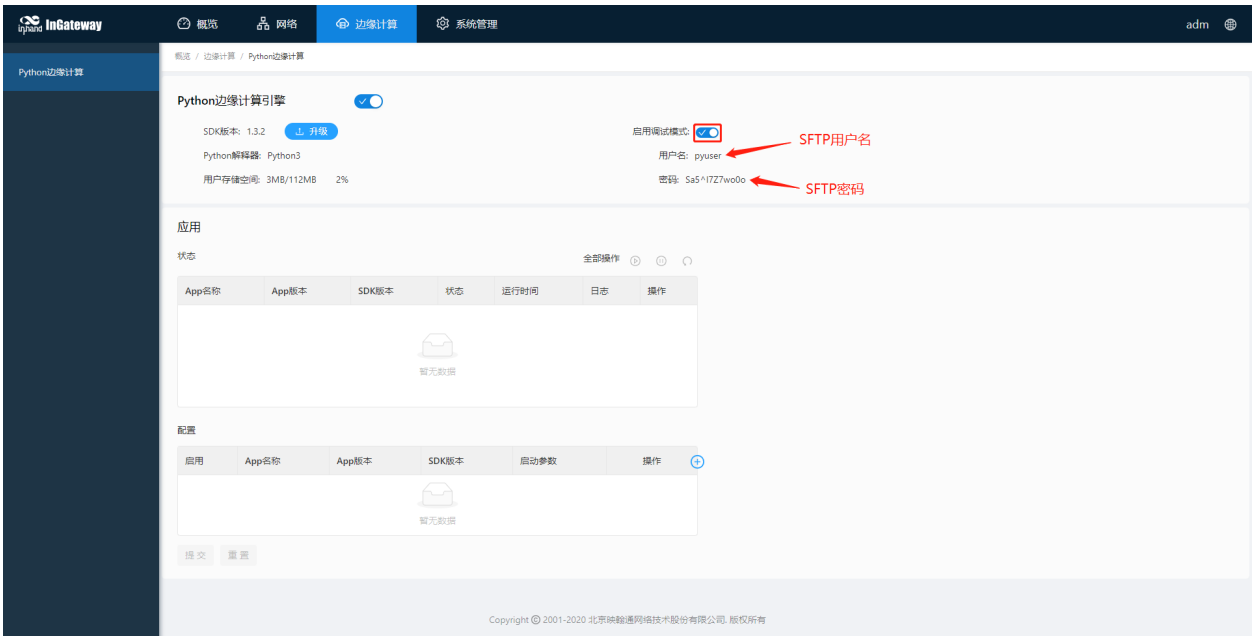
2.2 编码

标准工程模板“helloworld-template”已经实现了在 MobiusPi 中每 10 秒打印一条“hello world!”日志以及导入配置文件修改日志内容的功能。如需修改 App 名称，请您参考下图修改 `main.py` 和 `setup.py` 中的代码。注意：Python App 名称中不能包含空格。

2.3 调试

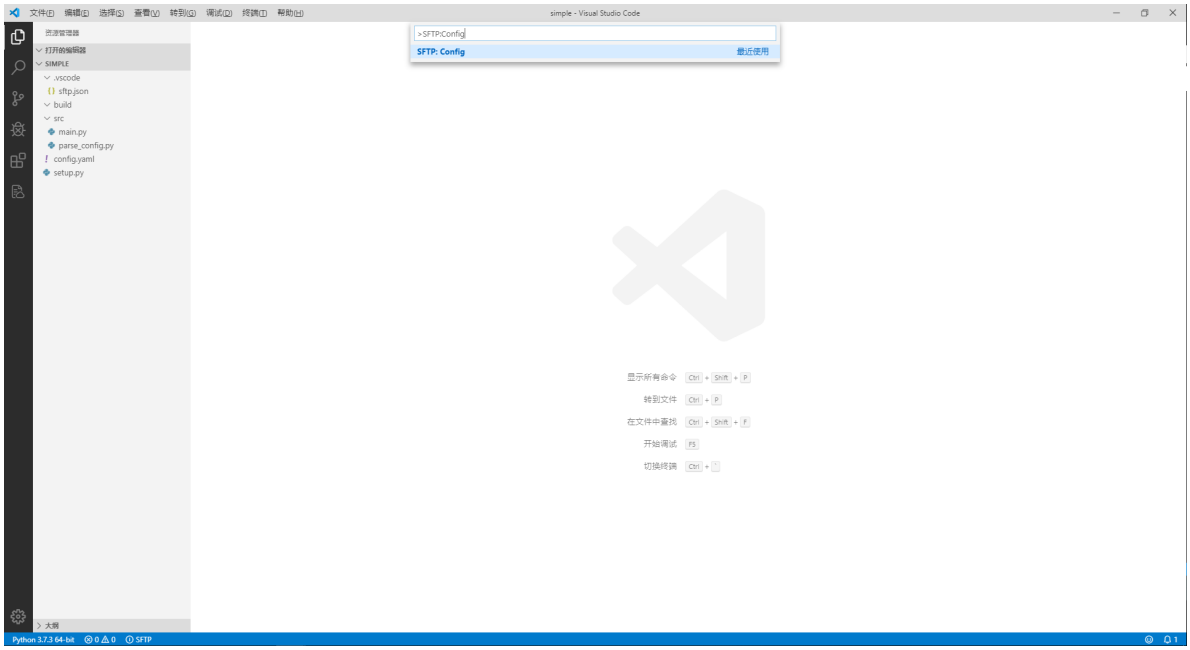
2.3.1 建立 SFTP 连接

远程调试代码时需要先将本地代码上传到远程服务器（即 MobiusPi）。上传本地代码前需要确认 MobiusPi 的调试模式已启用，启用调试模式后如下图所示：



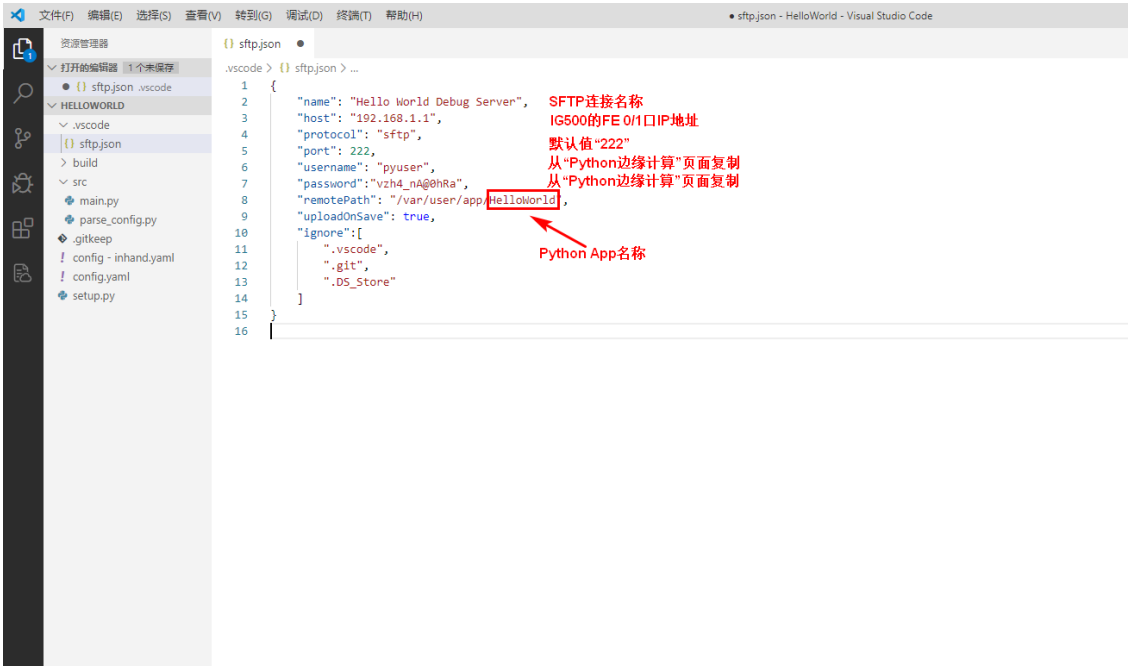
- 步骤 1：打开 `sftp.json` 文件

  在命令面板中输入 `>SFTP:Config` 命令后打开 `sftp.json` 文件。



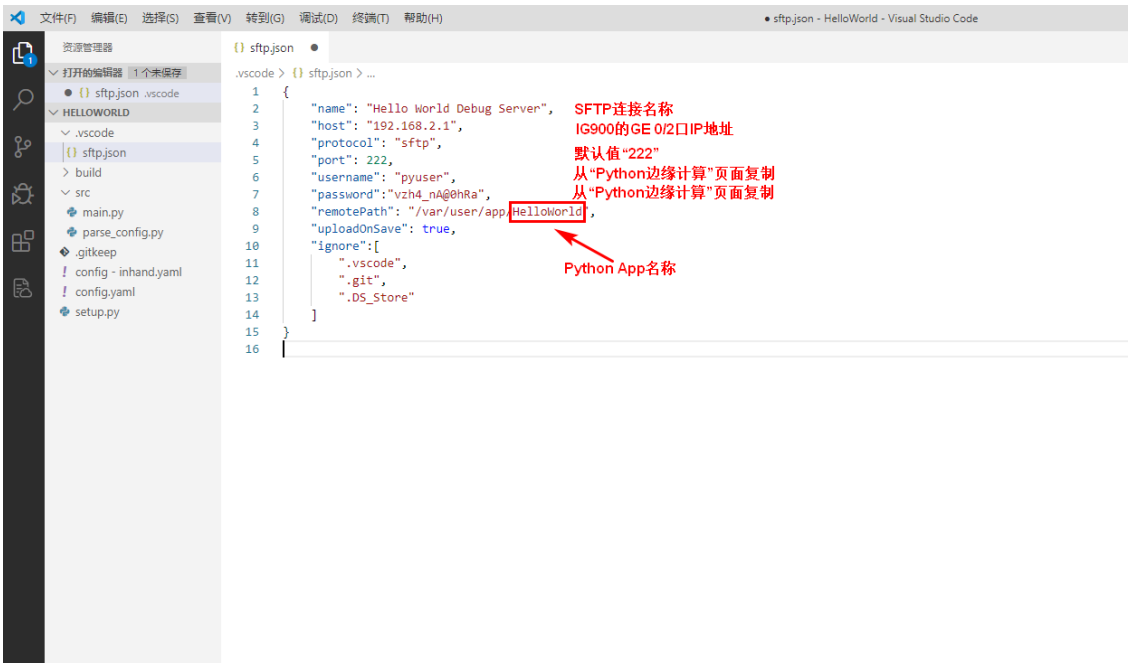
- 步骤 2：配置 SFTP 连接
  - 配置 IG501 SFTP 连接

    在 `sftp.json` 文件中根据“边缘计算 > Python 边缘计算”页面的连接参数配置 SFTP 连接。注意：Python App 名称应与 mian.py 中的 App 名称保持一致。

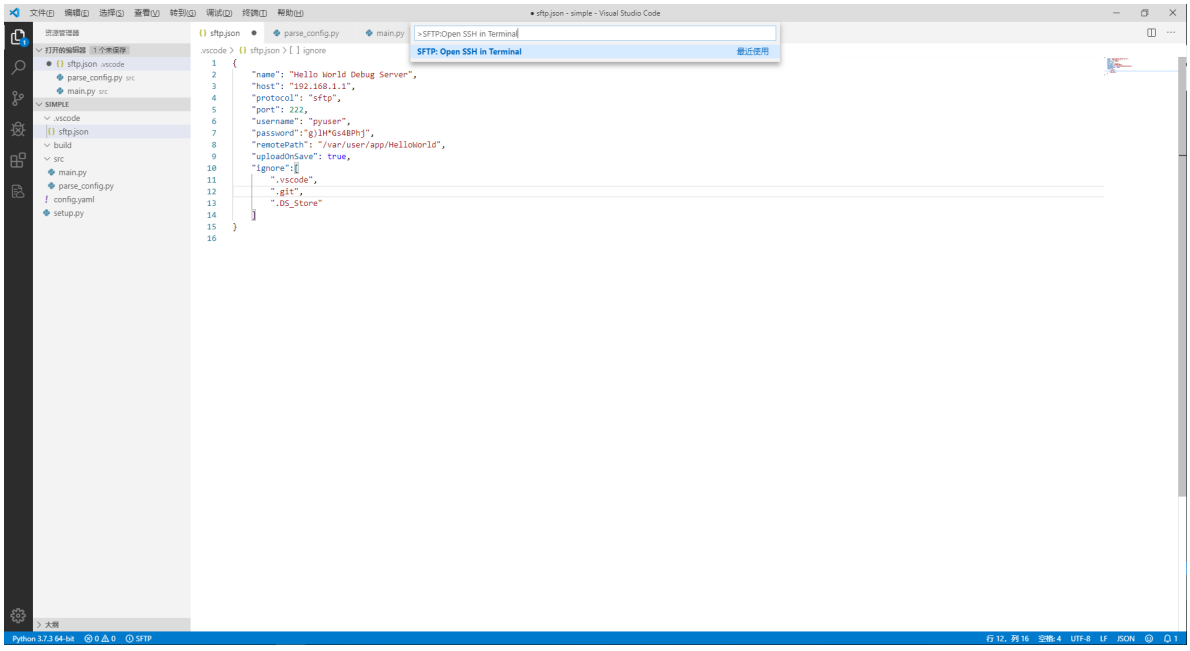


  - 配置 IG902 SFTP 连接

    在 `sftp.json` 文件中根据“边缘计算 > Python 边缘计算”页面的连接参数配置 SFTP 连接。注意：Python App 名称应与 mian.py 中的 App 名称保持一致。

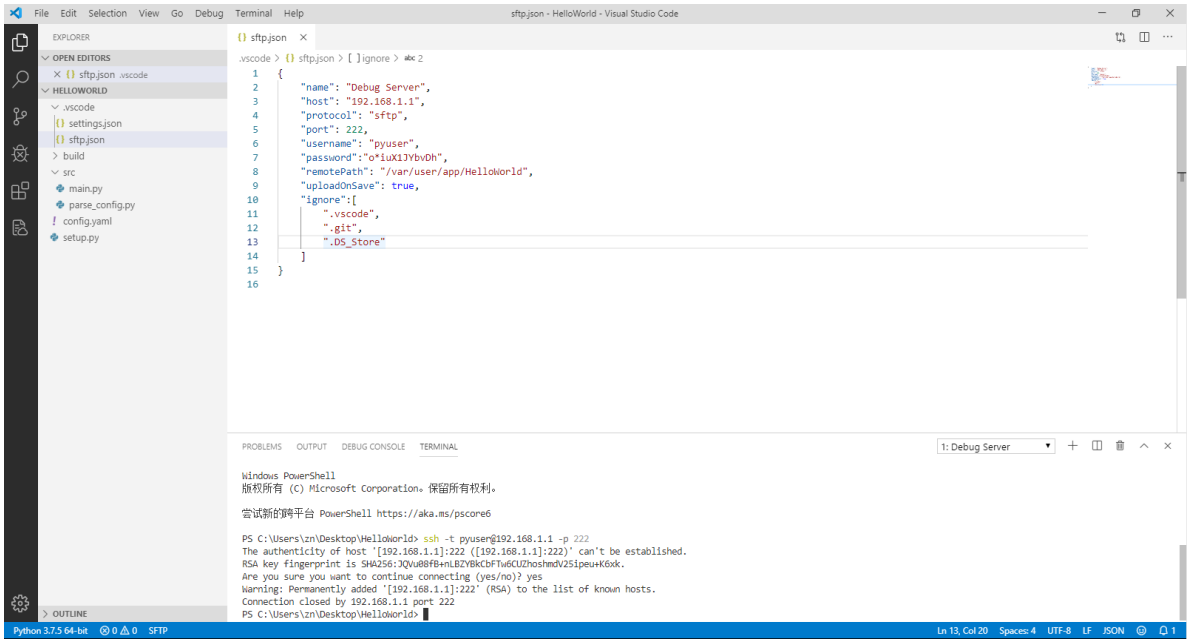


- 步骤 3：配置完成并保存后在命令面板中输入 `>SFTP:Open SSH in Terminal` 以连接远端的服务器。

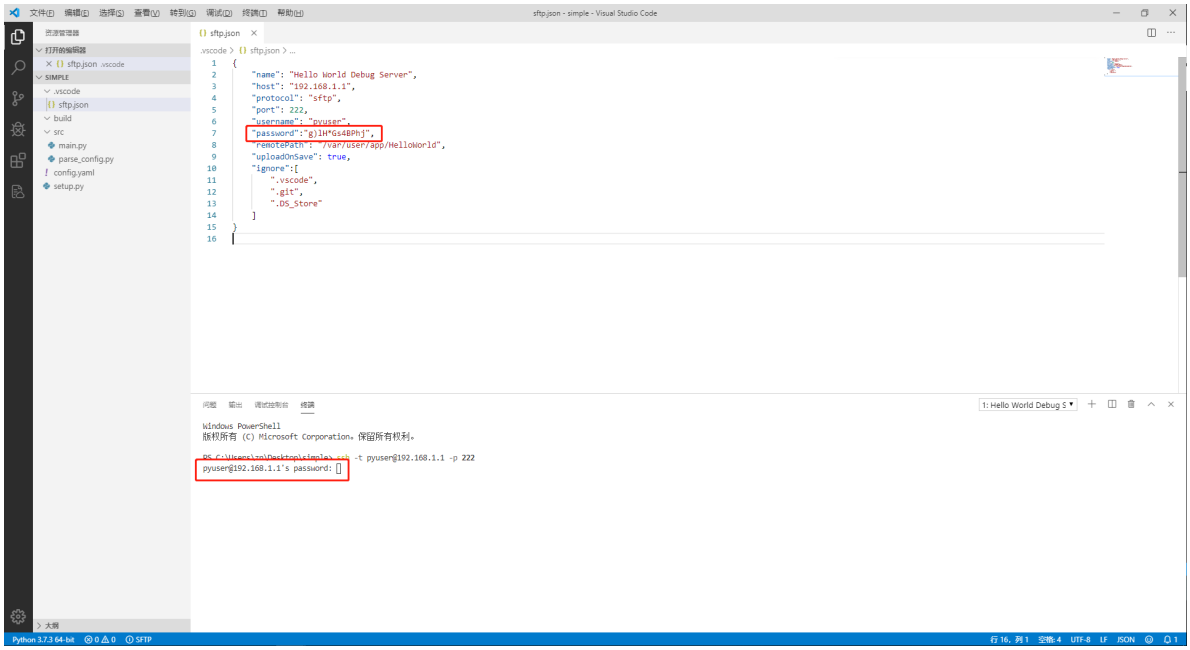


- 步骤 4：随后命令面板会提示您需要选择要连接的 SFTP 服务器，选择 `sftp.json` 中的 SFTP 服务器并回车即可。

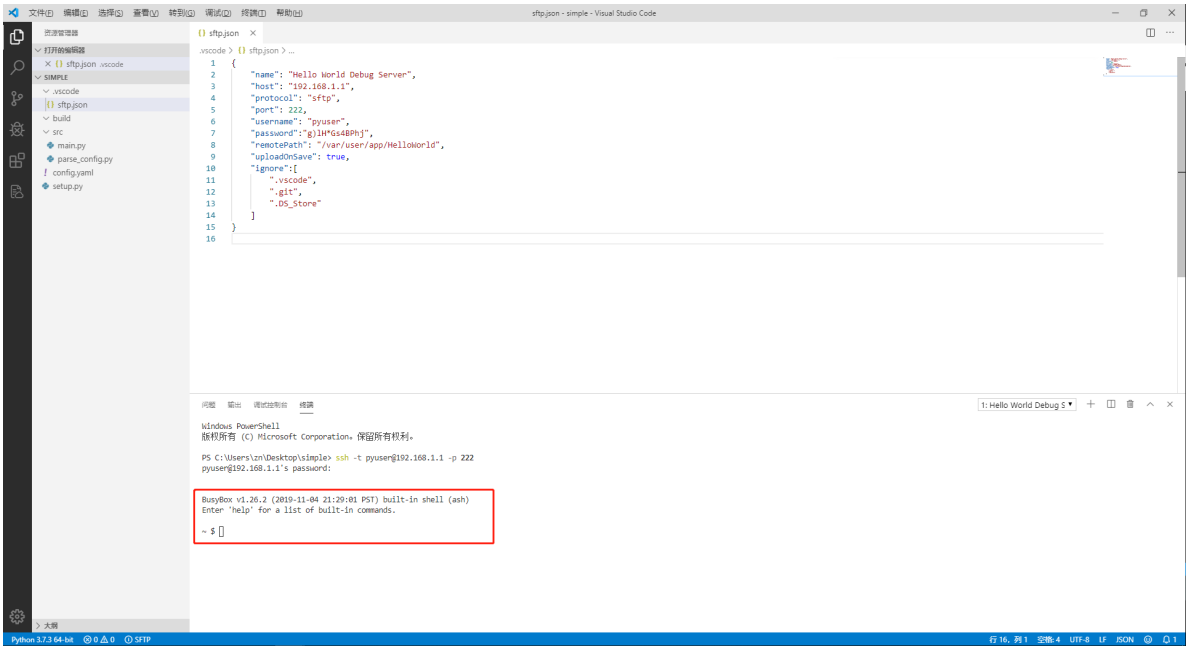
- 步骤 5：首次连接时“终端”窗口会提示您是否要继续连接，此时输入“yes”并回车。随后再次在命令面板中输入 `>SFTP:Open SSH in Terminal` 和 SFTP 服务器的 IP 地址。



- 步骤 6：“终端”窗口会提示您需要输入密码，您只需要将 `sftp.json` 文件中“password”项复制粘贴到此处即可。



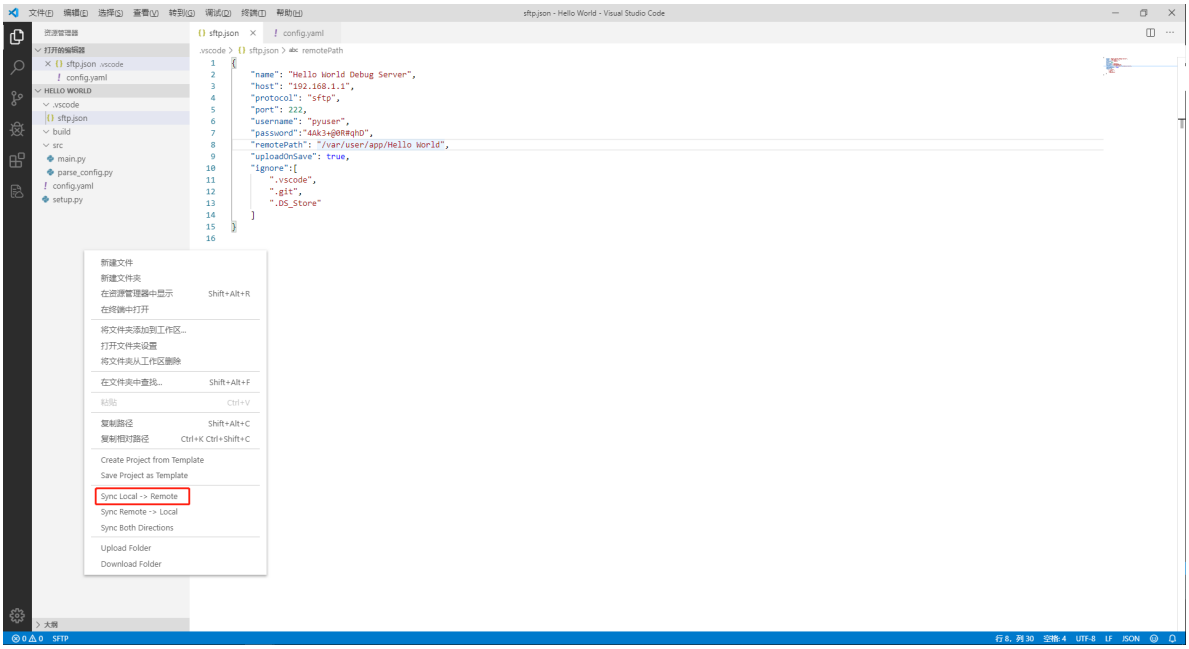
成功与 MobiusPi 建立 SFTP 连接后如下图所示：



2.3.2 调试代码

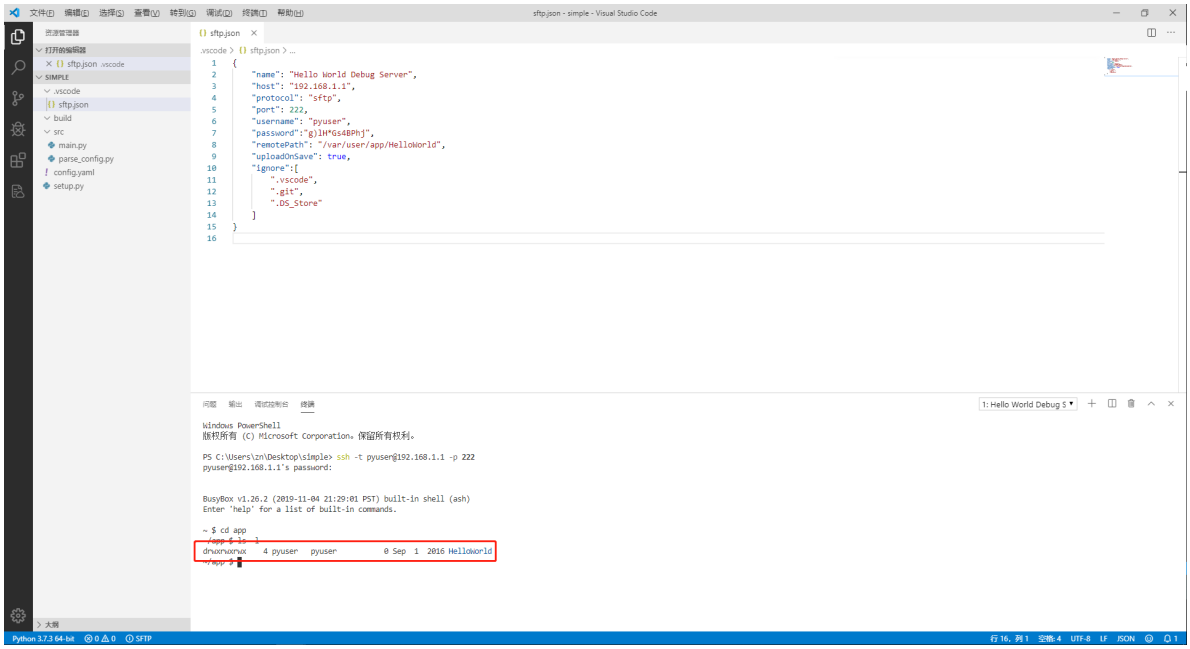
- 步骤 1：同步代码

SFTP 连接成功后，在左侧空白处右键选择 “Sync Local->Remote” 将代码同步到远程服务器，同步成功后本地修改或者删除代码时都会自动和远端服务器同步。



可以在 TERMINAL 窗口查看远程服务器是否已接收到相应的 App 代码。在 TERMINAL 窗口输入以下命令查看已上传的 App 文件夹信息：

```
cd app
ls -l
```



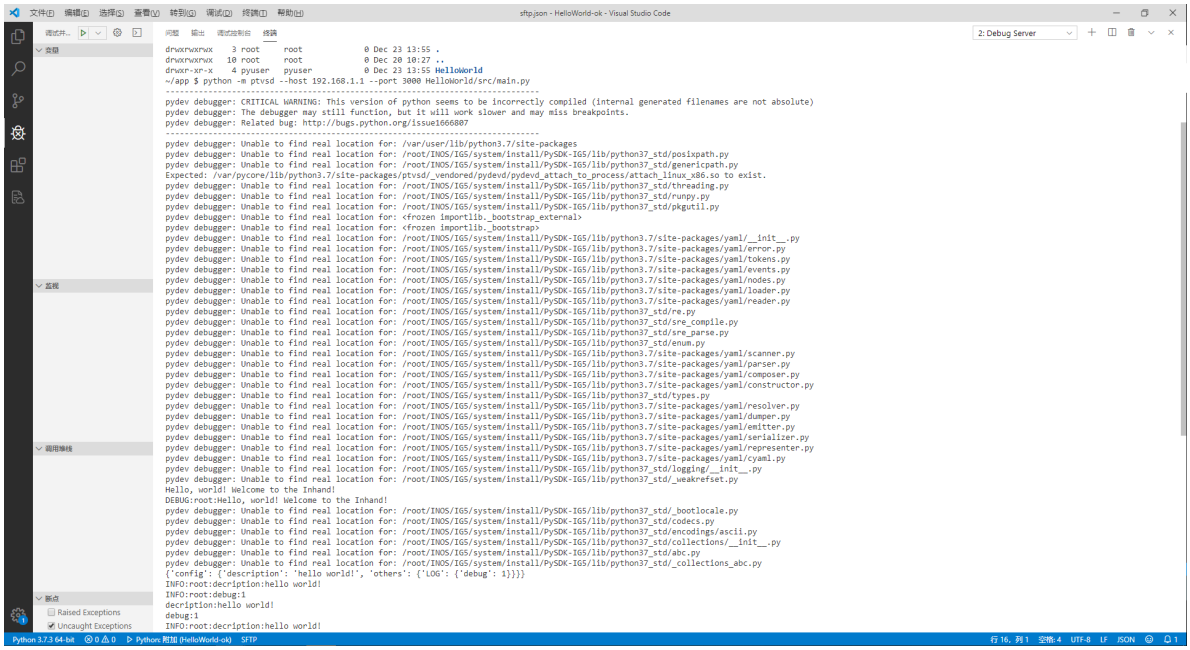
- 步骤 2：在终端窗口调试脚本

  同步代码后在终端窗口输入如下命令在 IG501 中立即执行脚本，执行脚本后在终端窗口查看执行结果：打印“hello world!”。

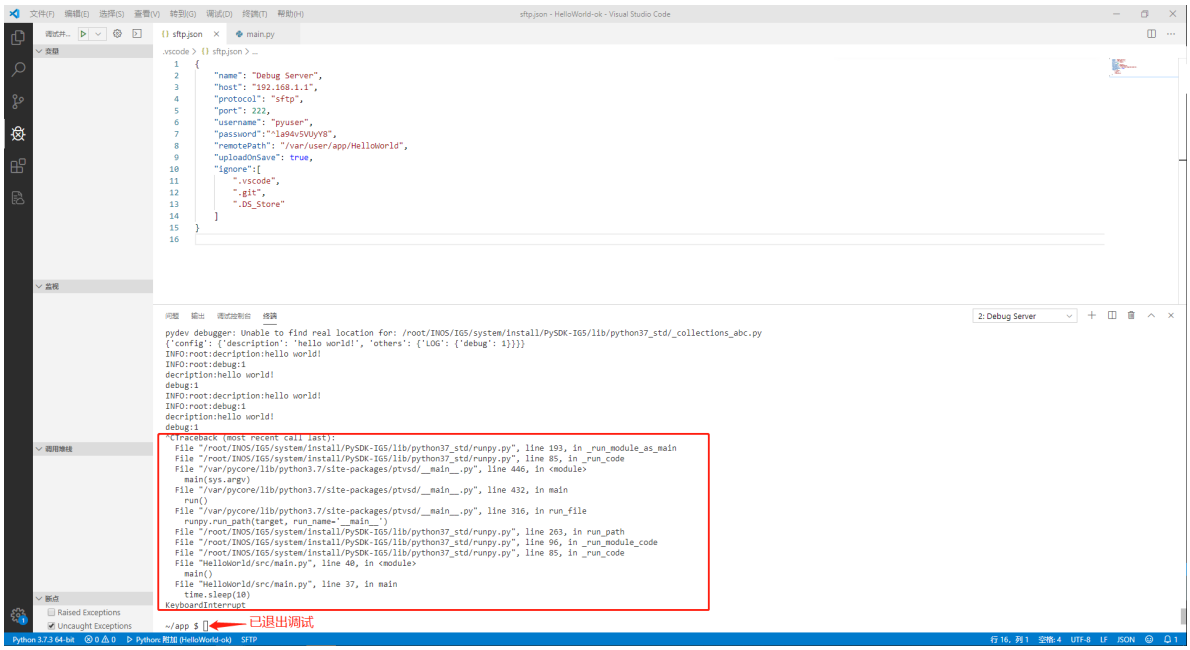
```
python -m ptvsd --host 192.168.1.1 --port 3000 HelloWorld/src/main.py
```

  - `192.168.1.1` 为 `sftp.json` 中“host”项配置的 IP 地址
  - `3000` 为建议的调试端口号
  - `HelloWorld/src/main.py` 为 mian. py 的执行路径，请根据您的当前位置适当调整

  MobiusPi 的 Python 开发环境内置了 ptvsd 依赖库用于远程调试代码，想要了解更多 ptvsd 插件的用法，请访问ptvsd 使用说明。



- 步骤 3：调试完成后在终端使用 `Ctrl + C` 快捷键终止调试。

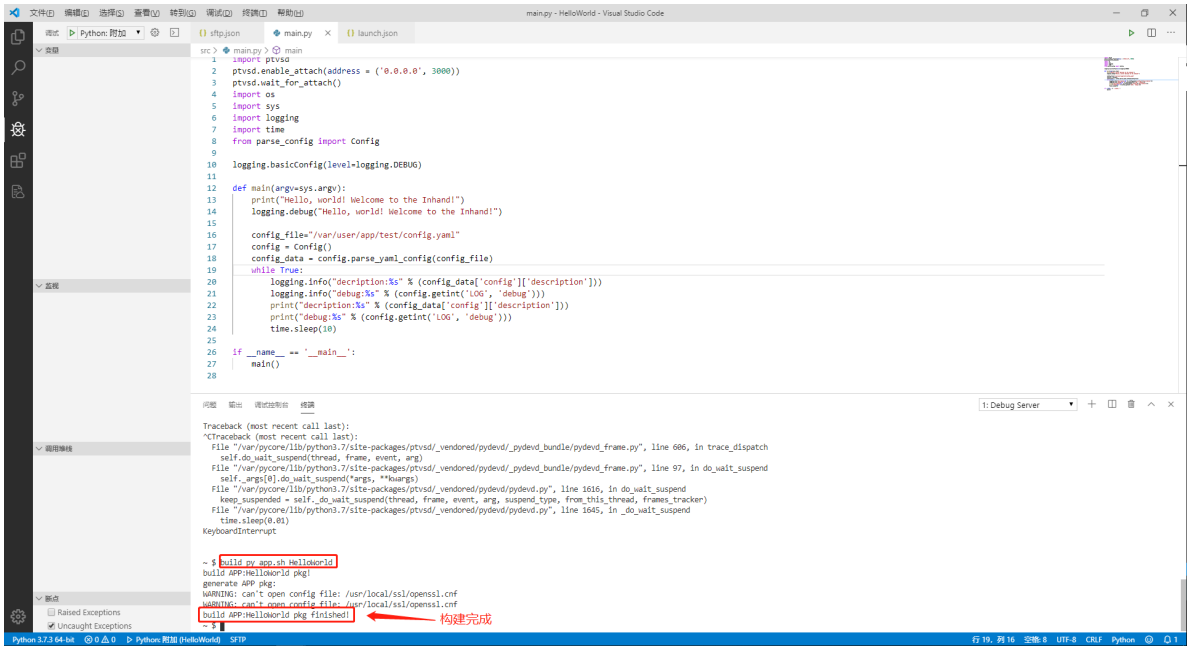


2.4 构建 App 发布包

调试完毕后可以构建 App 发布包以便于将 App 快速部署至其他 MobiusPi。

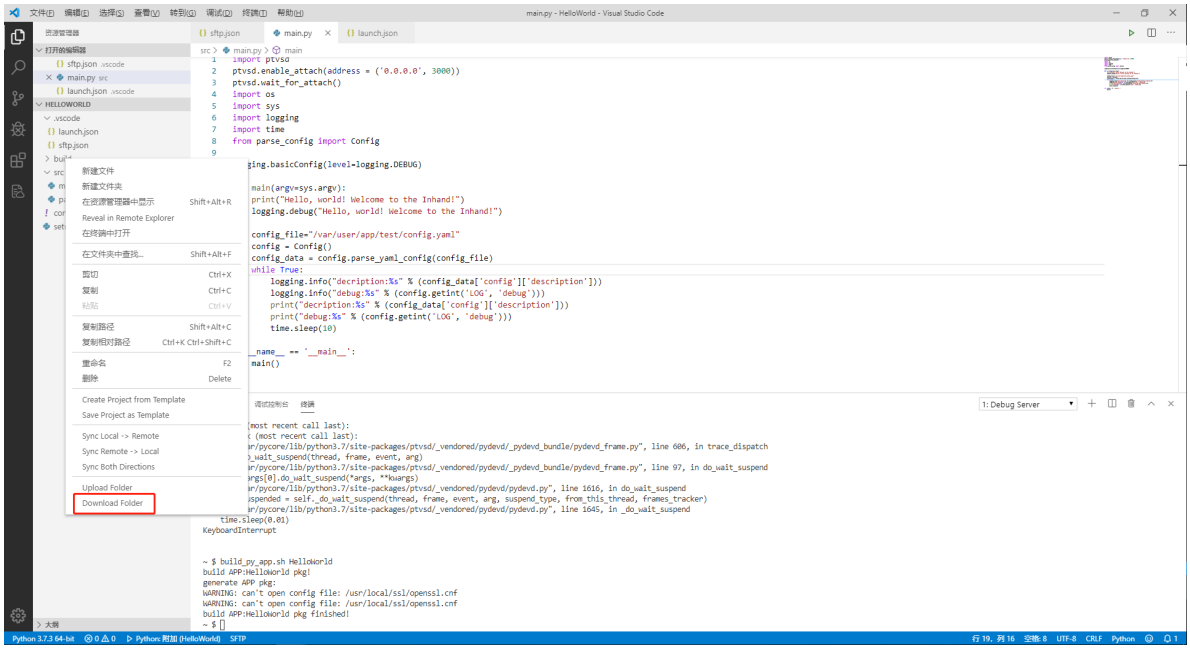
- 步骤 1：构建 App 发布包

  在“终端”窗口执行 `build_py_app.sh HelloWorld` 命令构建 App 发布包。（即 `build_py_app.sh` Python App 名称）



- 步骤 2：下载 App 发布包

执行完成后在远程服务器的 build 目录下会自动生成 App 发布包。右键本地的“build”文件夹并选择“Download Folder”将构建好的 App 发布包下载到本地以便于后续部署。



下载完成后在 build 目录下可以看到 HelloWorld App 发布包。

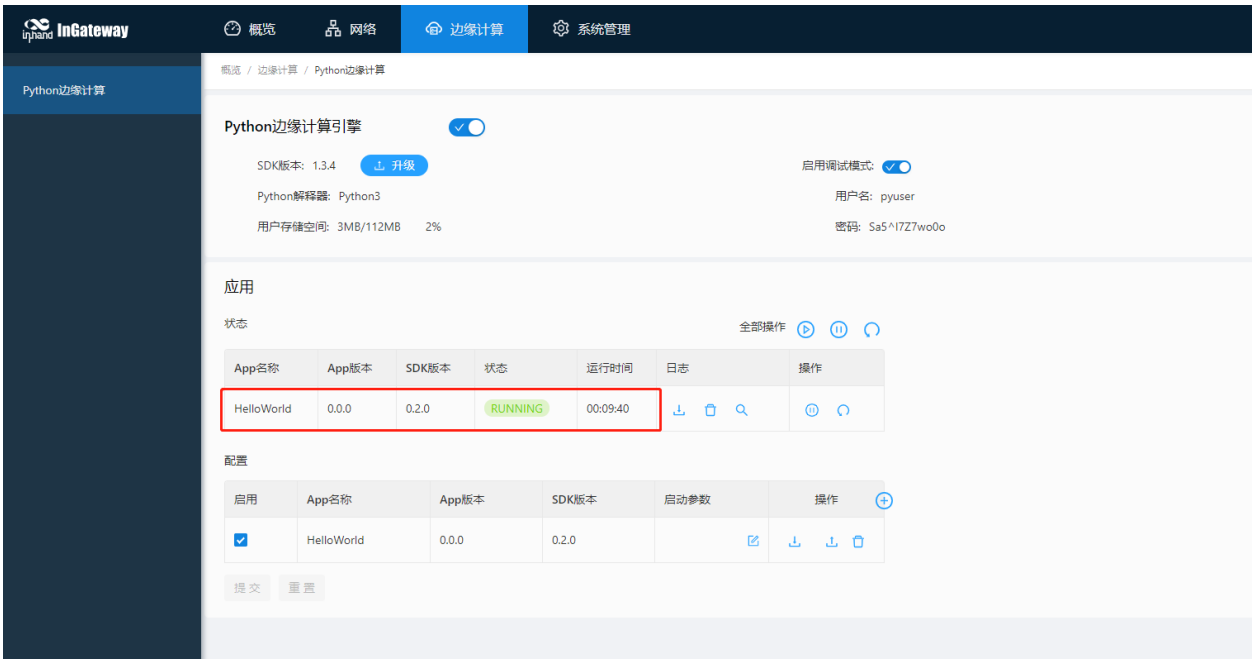
2.5 通过 MobiusPi Web 页面部署 App

执行 `main.py` 脚本或构建 App 发布包命令后会自动在已连接的 MobiusPi 上生成对应的 App, 此 App 无法正常启动。请参考如下链接部署 App 至 MobiusPi:

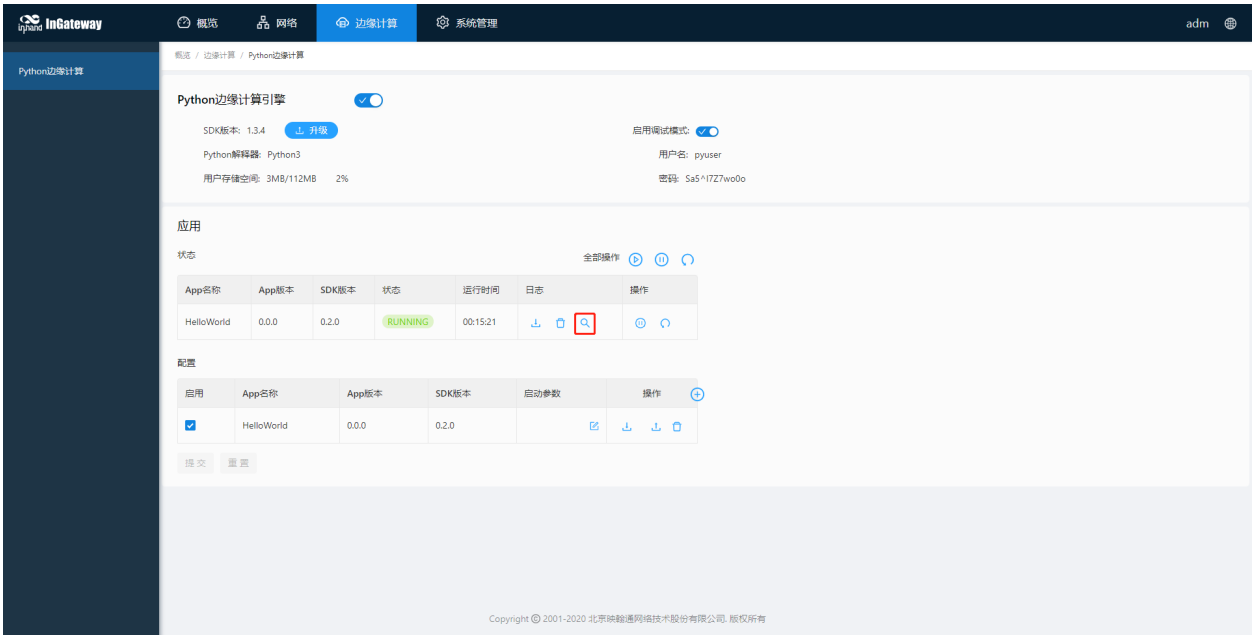
- IG501 部署 App
- IG902 部署 App

2.6 查看 App 运行状态

在 MobiusPi 的 “边缘计算 > Python 边缘计算” 页面可查看 App 的运行状态。



点击 “查看日志” 可查看 App 的运行日志。




```
DEBUG:root:Hello, world! Welcome to the Inhand!
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
```

2.7 为 App 更新配置文件

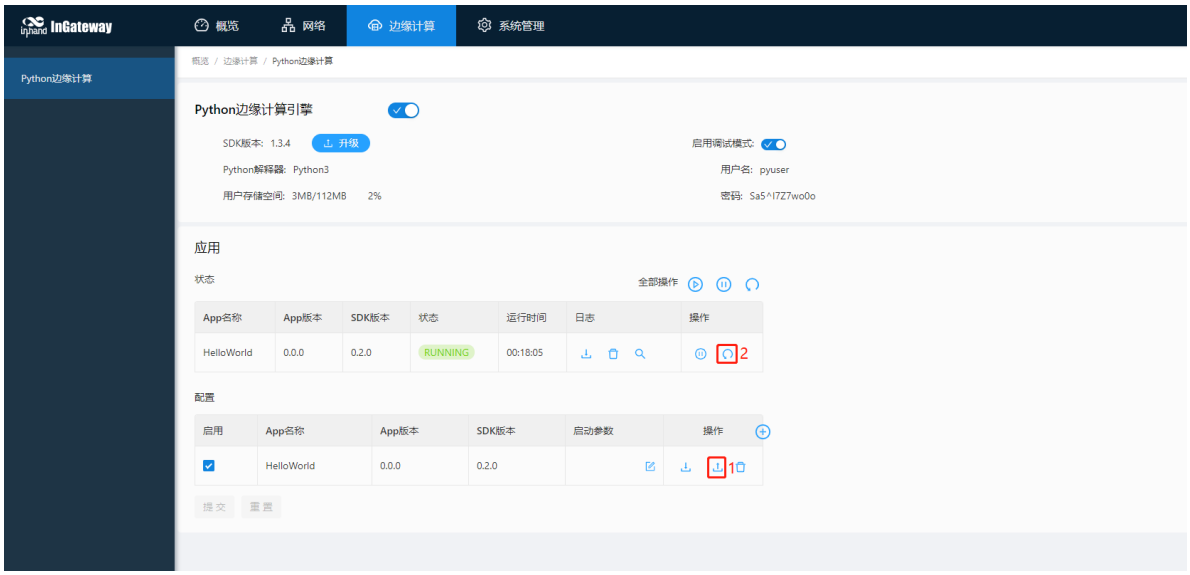
- 步骤 1：修改配置文件

将 App 的“config.yaml”中的配置 `description: "hello world!"` 修改为:`description: "hello inhand!"`

```
config:
    description: "hello inhand!"
    others:
        LOG:
            #Enable debug feature to write all logs to the console
            #Default: debug=0
            debug: 1
```

- 步骤 2：导入配置文件并重启 App

  在 MobiusPi 的“边缘计算 > Python 边缘计算”页面为“HelloWorld”App 导入修改后的配置文件并重启 App。



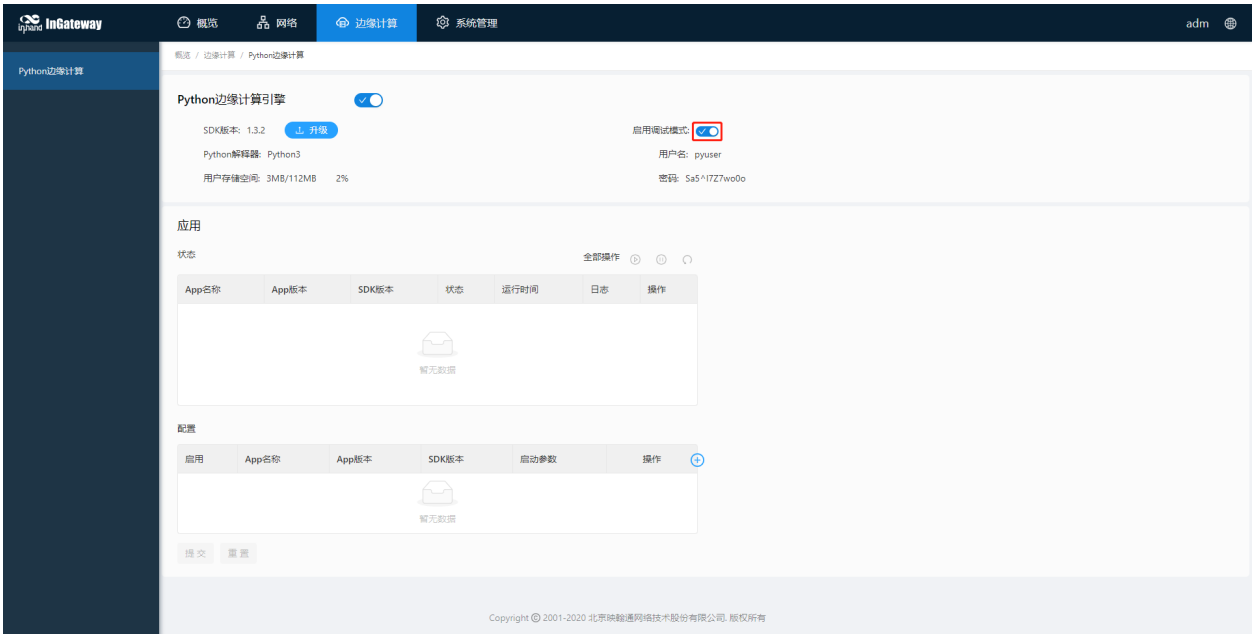
重启后“HelloWorld”App 将按照更新后的配置文件运行，即每 10 秒打印一条”hello inhand!”日志。

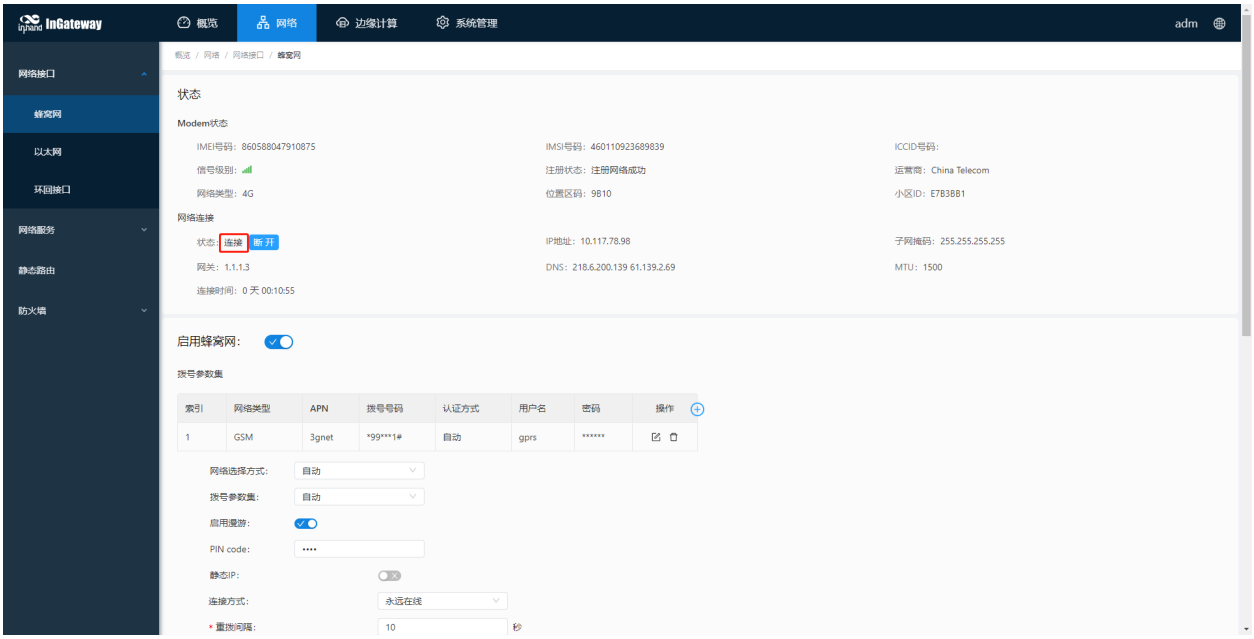
```
DEBUG:root:Hello, world! Welcome to the Inhand!
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
DEBUG:root:Hello, world! Welcome to the Inhand!
INFO:root:decription:hello inhand!
INFO:root:debug:1
INFO:root:decription:hello inhand!
INFO:root:debug:1
INFO:root:decription:hello inhand!
INFO:root:debug:1
INFO:root:decription:hello inhand!
INFO:root:debug:1
INFO:root:decription:hello inhand!
```

2.8 附录

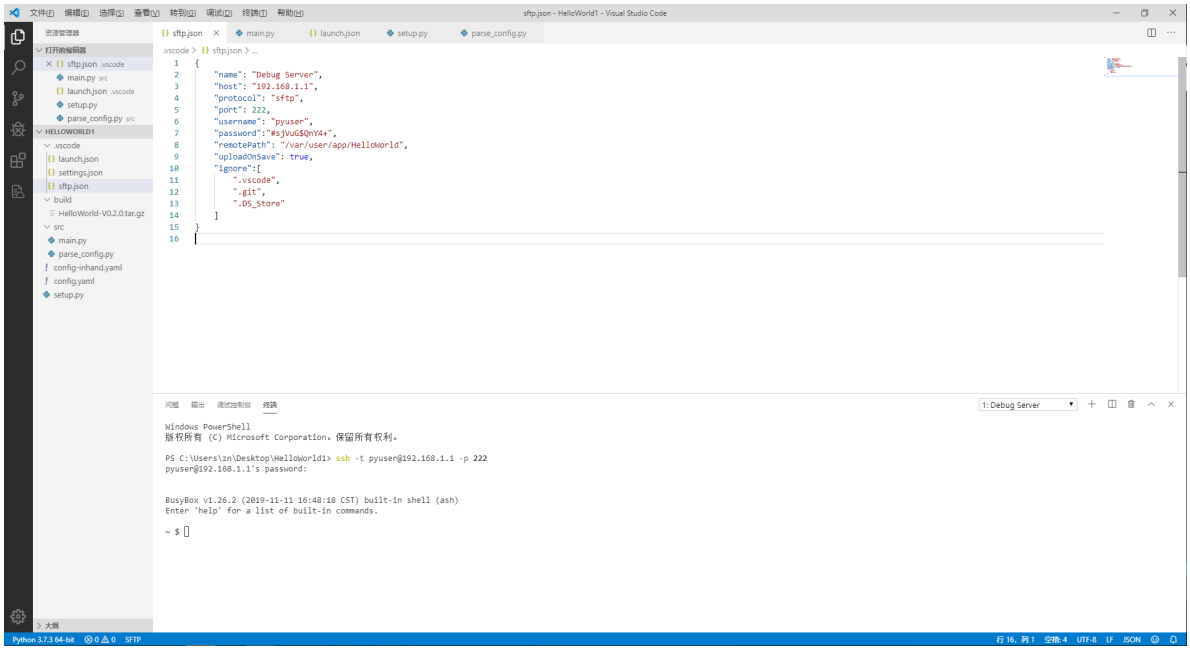
2.8.1 为指定 App 安装第三方依赖库

为指定 App 安装第三方依赖库时需启用 MobiusPi 的调试模式并且接入互联网，以下以 HelloWorld App 安装 `xlrd` 依赖库为例说明如何为指定 App 安装第三方依赖库：



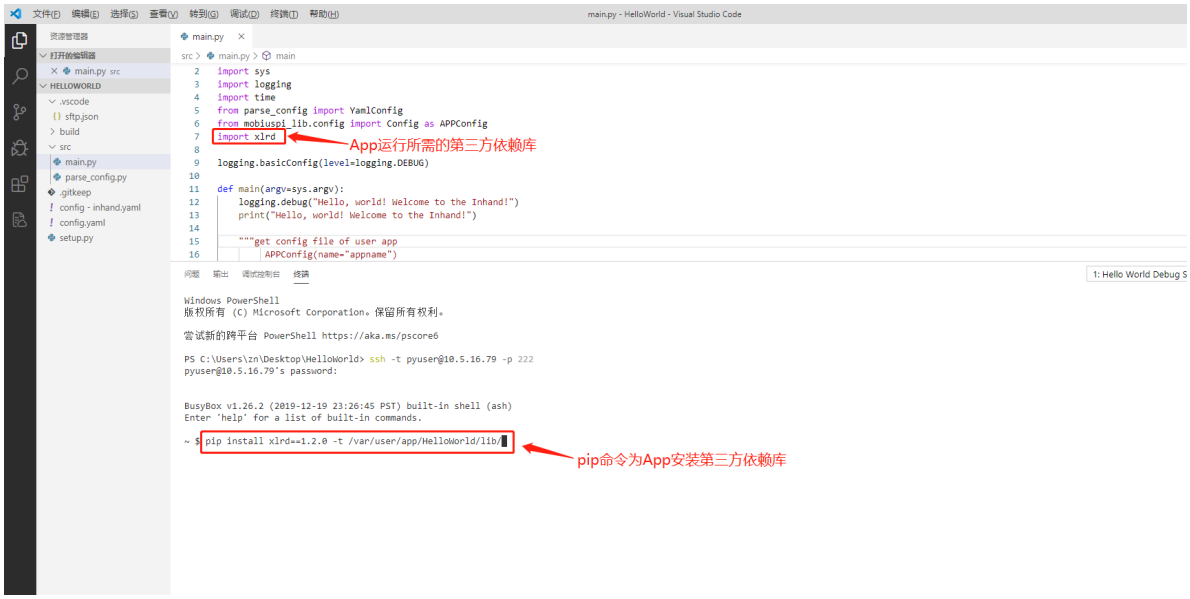


- 步骤 1：使用 VS Code 与 MobiusPi 建立 SFTP 连接，详见*建立 SFTP 连接*。

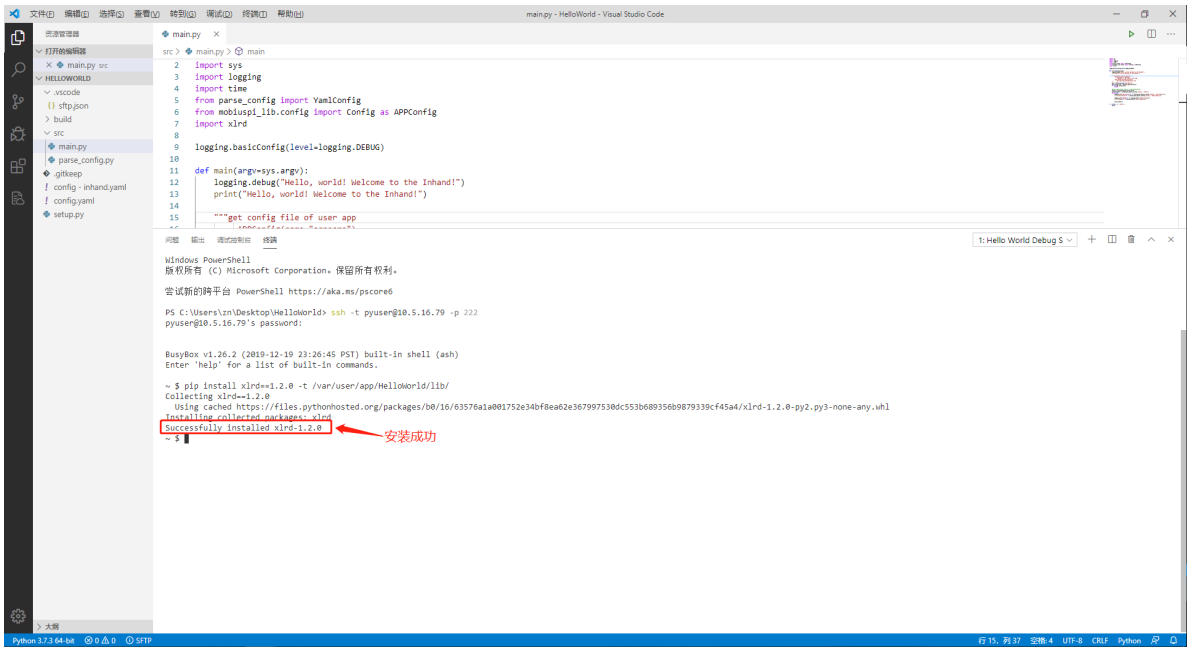


- 步骤 2：执行 `pip install` + “依赖库名称” + “== 版本号” + `-t` + “App 的 lib 文件夹路径” 命令并回车以安装相应依赖库至指定 App。（不加版本号时 pip 会自动安装最新版的依赖库）

```
pip install xlrd==1.2.0 -t /var/user/app/HelloWorld/lib/
```

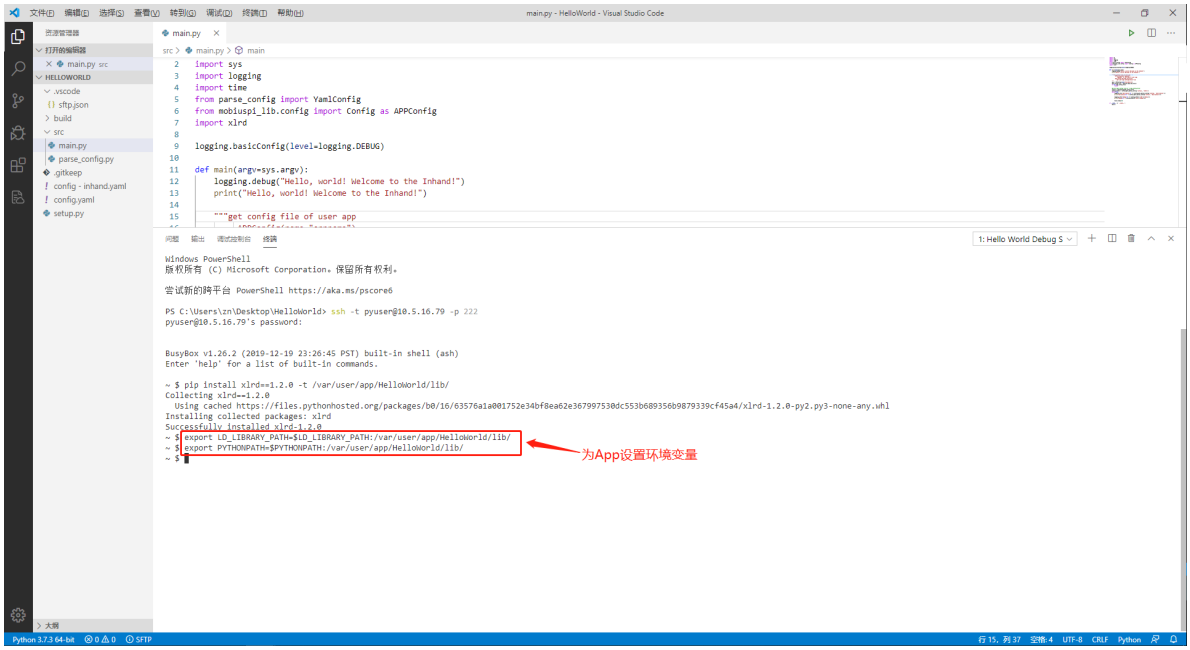


- 步骤 3：随后会自动下载并安装相应的依赖库，安装成功后如下图所示：



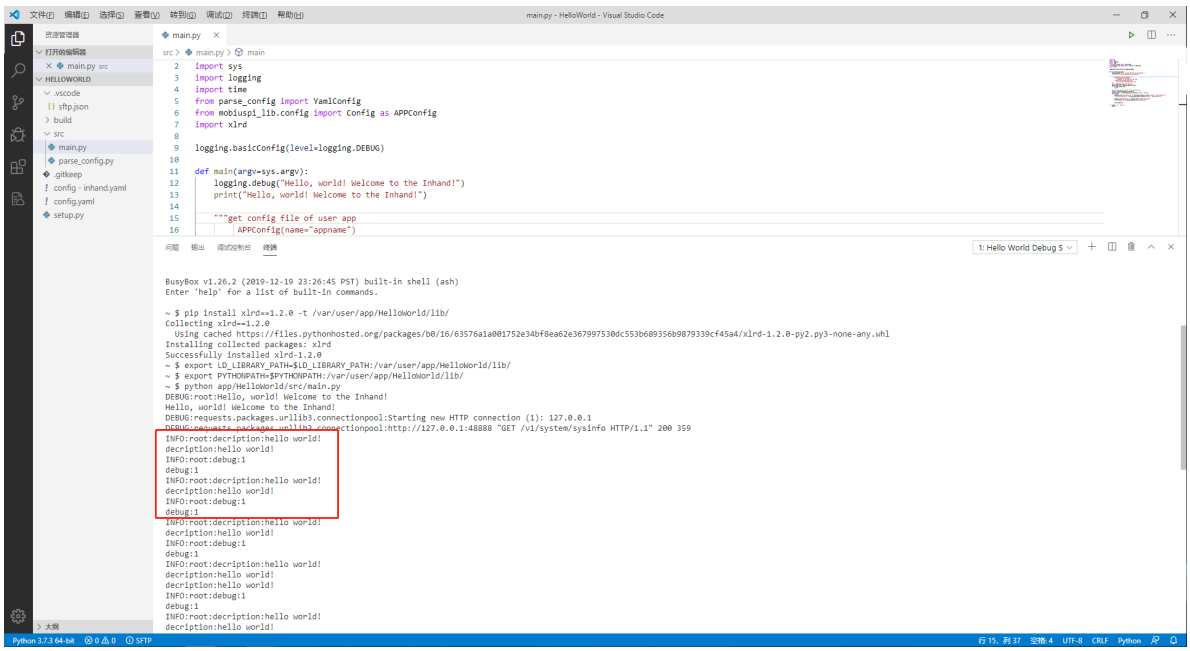
- 步骤 4：使用 `export` 命令为 App 设置环境变量。在终端窗口执行以下命令（“HelloWorld” 项需要根据实际的 App 名称调整）

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/var/user/app/HelloWorld/lib/
export PYTHONPATH=$PYTHONPATH:/var/user/app/HelloWorld/lib/
```



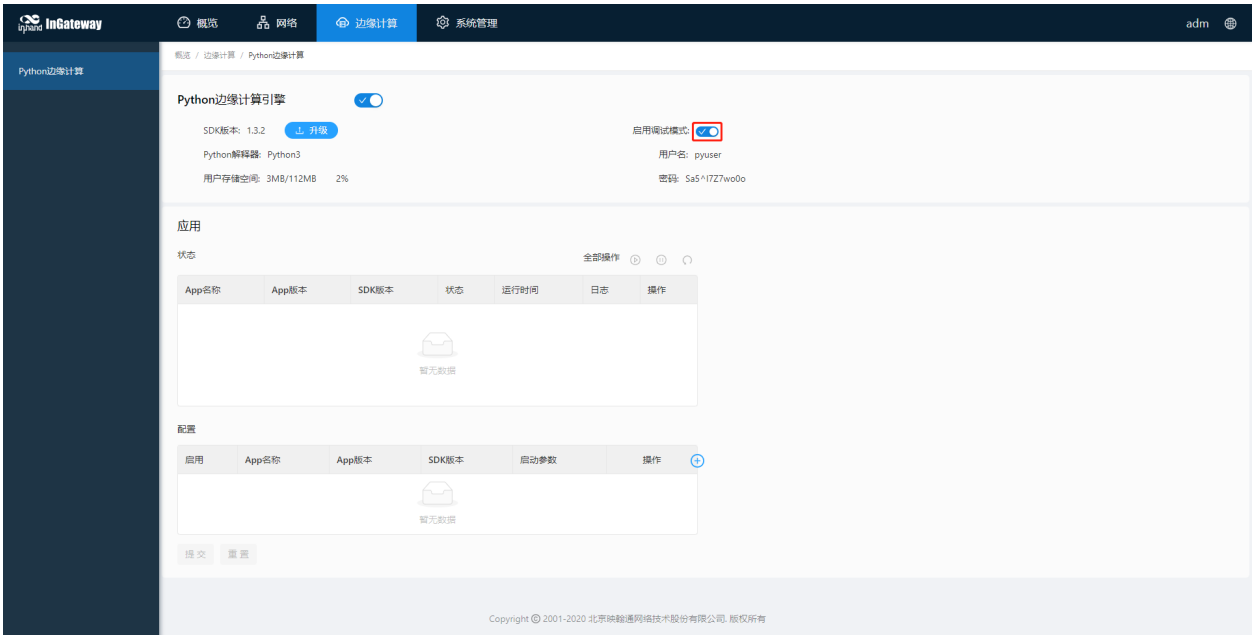
为指定 App 安装第三方依赖库后，调试该 App 前必须要配置相应的环境变量，否则调试时该 App 将不能正常运行（安装多个第三方依赖库时仅需要添加一次环境变量即可；重新建立 SFTP 连接后需要再次配置环境变量）。在 MobiusPi 中启用 App 后会自动为该 App 添加 App 的 lib 文件夹下第三方依赖库的环境变量，无须手动配置。

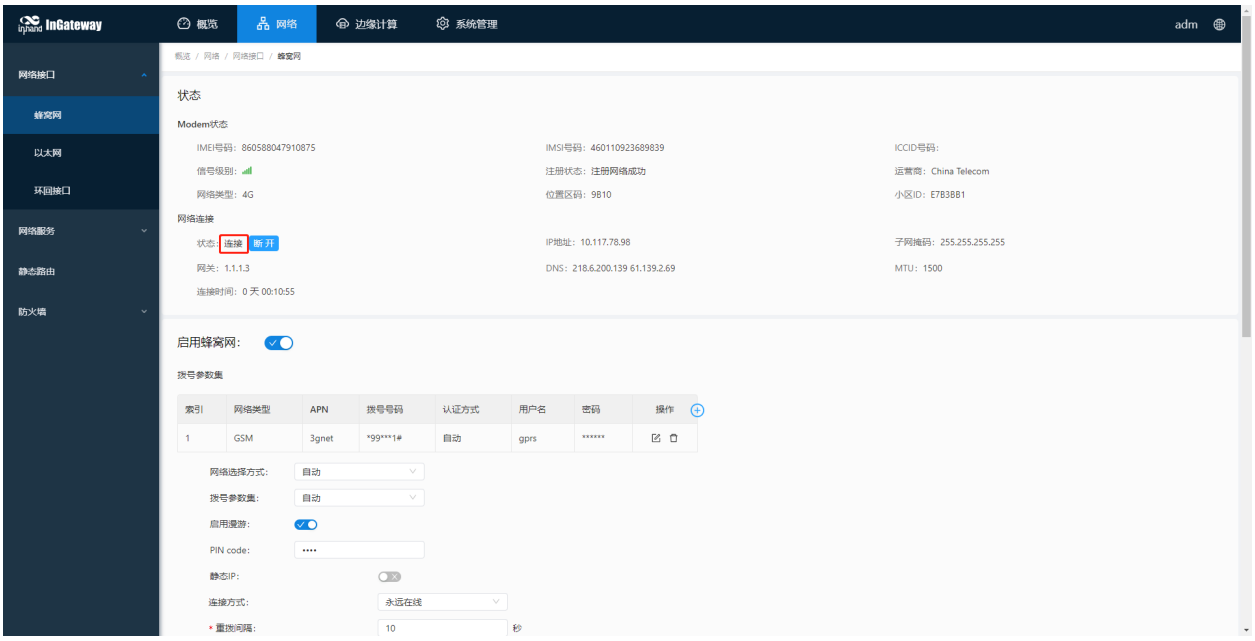
- 步骤 5：运行代码以确认 App 运行正常



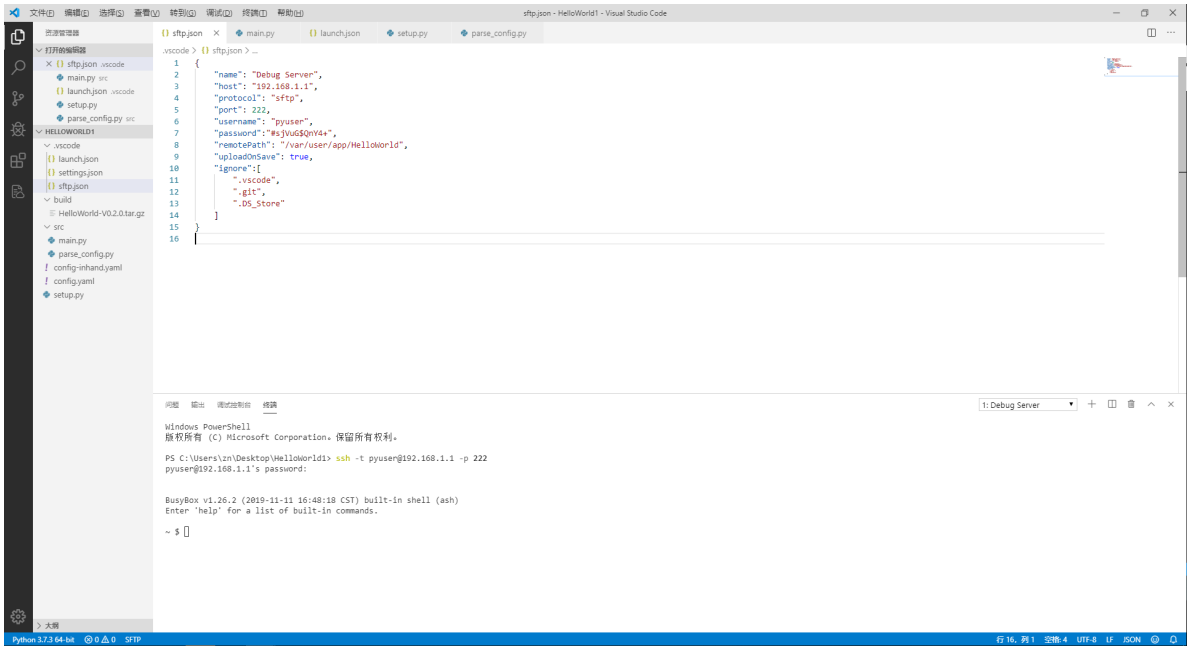
2.8.2 安装第三方依赖库至 SDK

安装第三方依赖库至 SDK 时需启用 MobiusPi 的调试模式并且接入互联网，以下以安装 `xlrd` 依赖库至 SDK 为例说明如何安装第三方依赖库至 SDK：





- 步骤 1：使用 VS Code 与 MobiusPi 建立 SFTP 连接，详见*建立 SFTP 连接*。



- 步骤 2：执行 `pip install` + “依赖库名称” + “== 版本号” + `--user` 命令并回车以安装相应依赖库至 SDK。（不加版本号时 pip 会自动安装最新版的依赖库）

```
pip install xlrd==1.2.0 --user
```

- 步骤 3：随后会自动下载并安装相应的依赖库，安装成功后如下图所示：

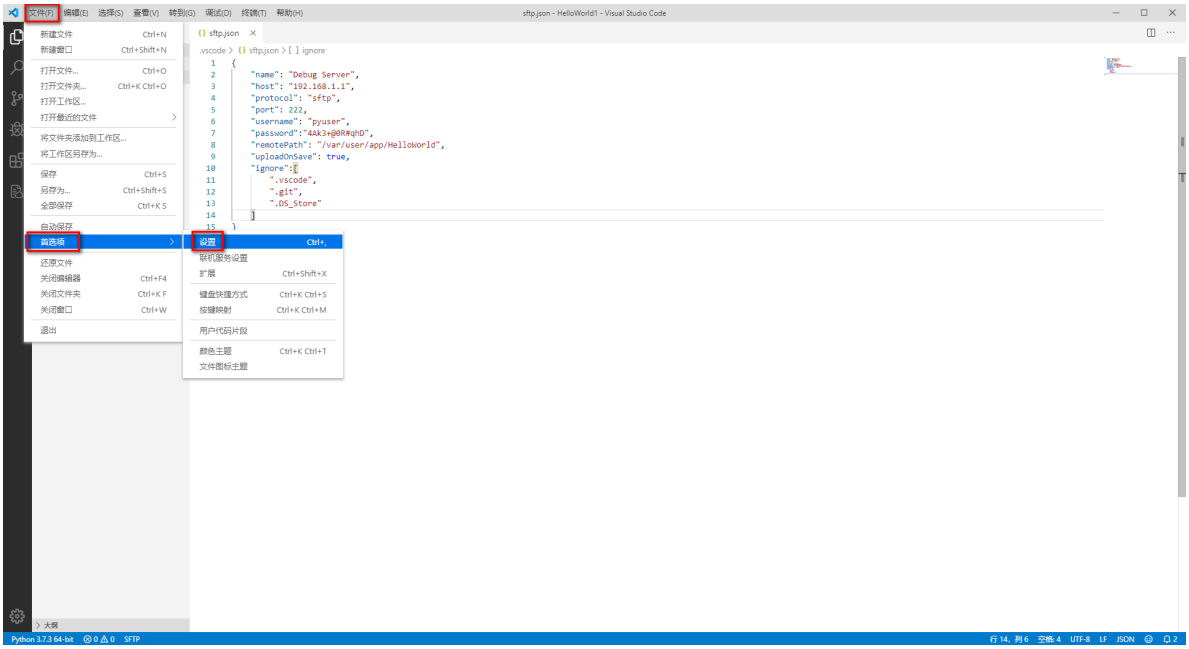
- 步骤 4：运行代码以确认 App 运行正常

注意：使用此方法安装第三方依赖库后，若导入 App 发布包到其他未在 SDK 中安装此 App 运行所需的第三方依赖库的 MobiusPi 上时，App 可能无法正常运行。

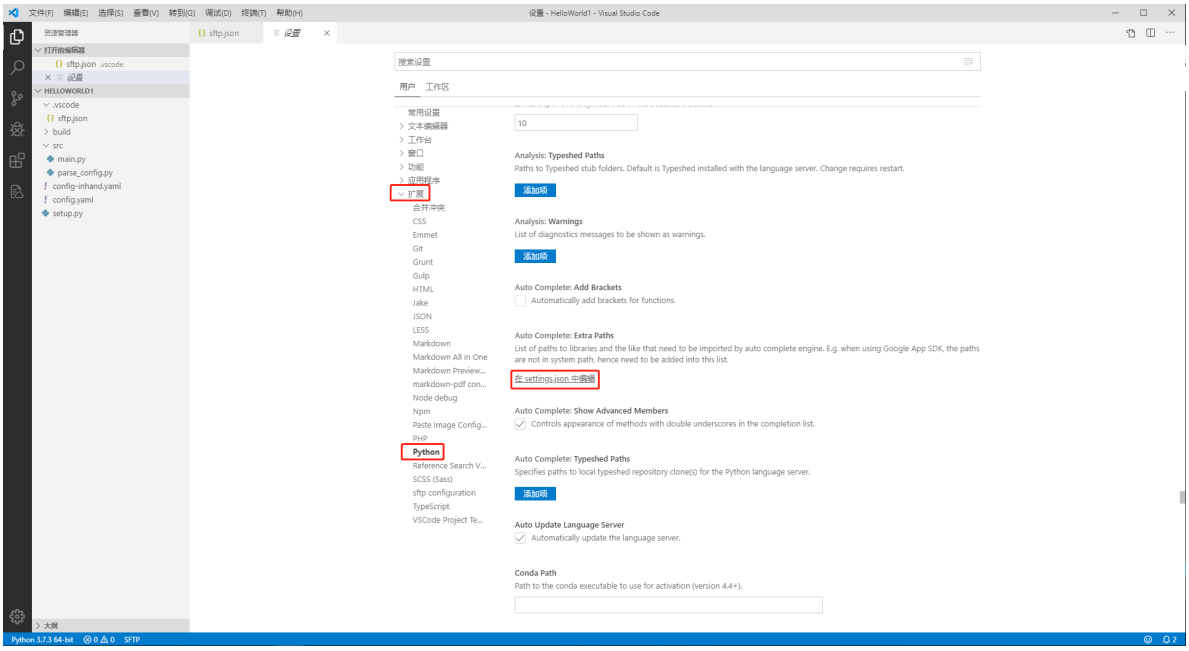
2.8.3 启用代码自动补全

为了提高编码效率，您可以通过 Python 扩展插件实现代码自动补全功能。

- 步骤 1：点击“文件 > 首选项 > 设置”进入设置页面。



- 步骤 2：选择设置页面中的“扩展 > Python”项，找到“Auto Complete: Extra Paths”项并点击“在 setting.json 中编辑”。



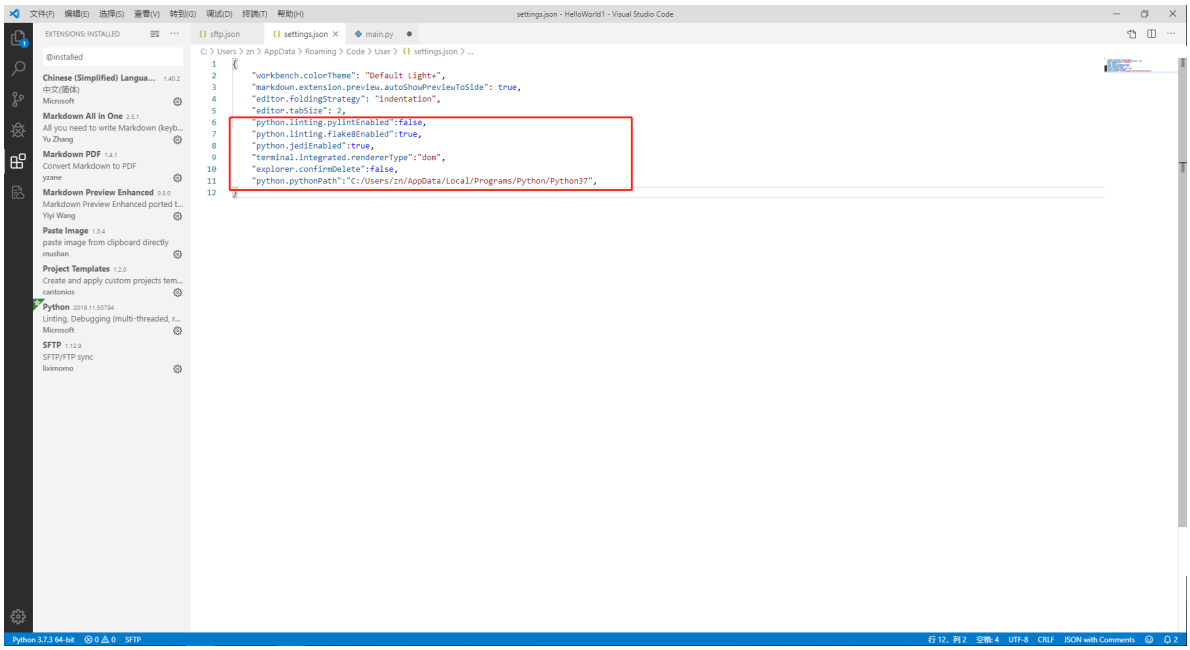
在“settings.json”中增加如下配置项并保存（`python.pythonPath` 为 python 解释器的安装路径）

```
"python.linting.pylintEnabled":false,
"python.linting.flake8Enabled":true,
"python.jediEnabled":true,
"terminal.integrated.rendererType":"dom",
"explorer.confirmDelete":false,
```

(continues on next page)

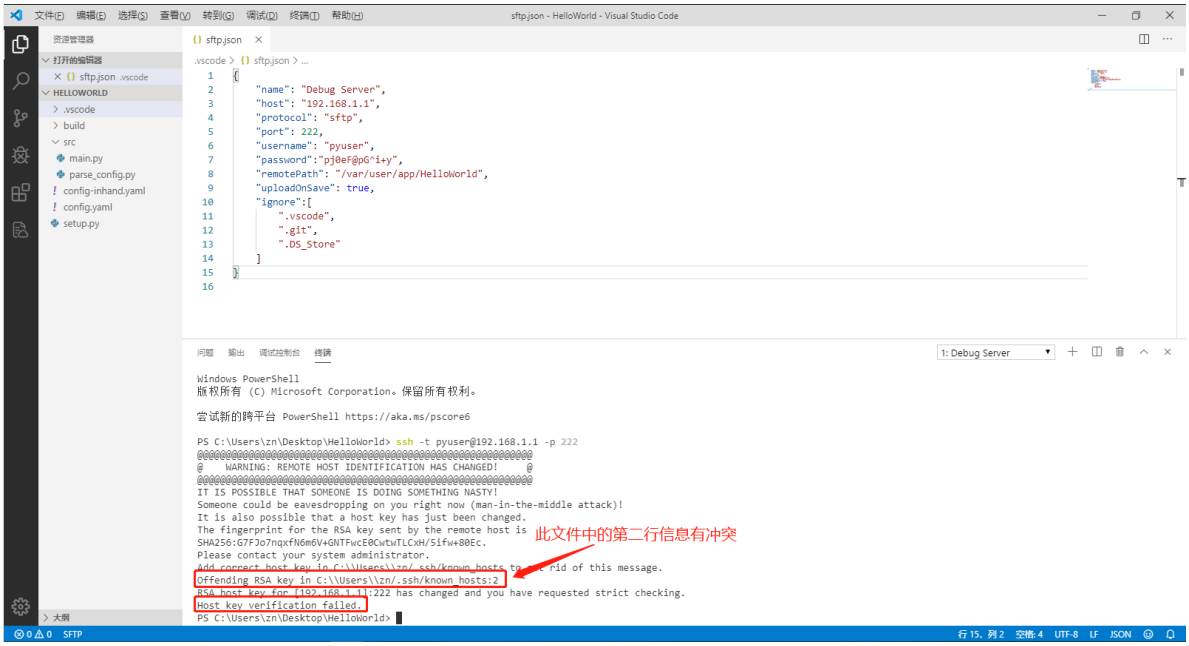
(continued from previous page)

```
"python.pythonPath":"C:/Users/zn/AppData/Local/Programs/Python/Python37",
```



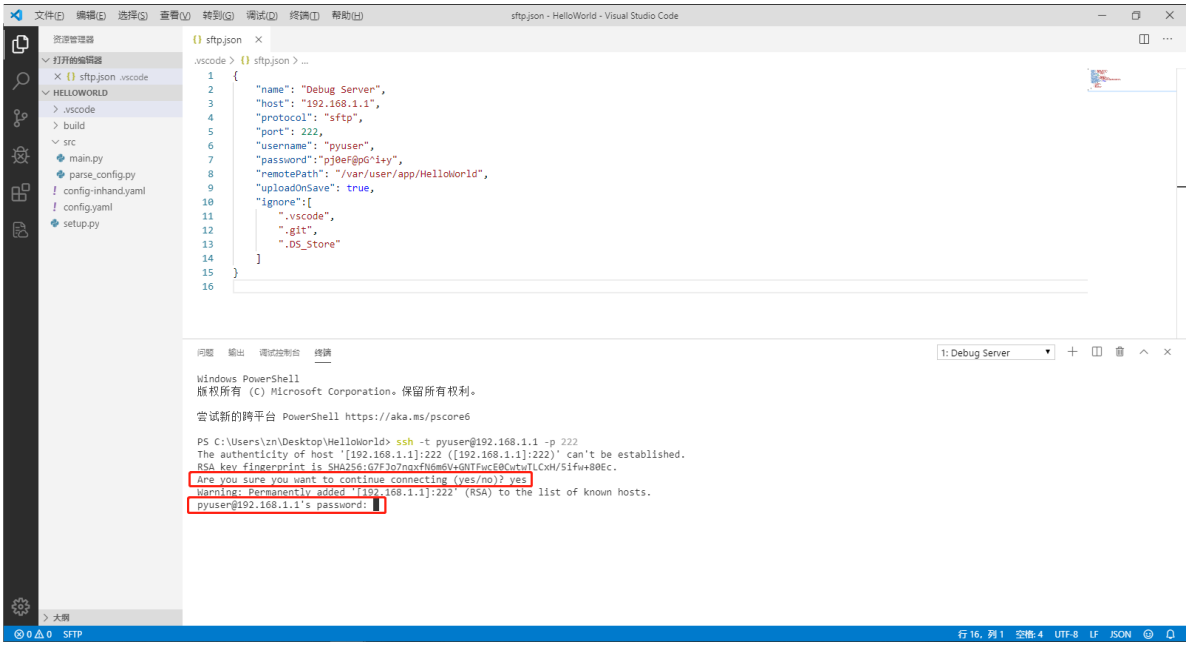
1.2.3 FAQ

- 建立 *SFTP* 连接时提示远程主机标识更新，认证失败
- 代码同步到远程服务器时提示“配置的身份验证方法失败”
- 在开发过程中如何调用 *IG902* 的串口和网口
- 与 *MobiusPi* 建立 *SFTP* 连接时提示“*SSH* 错误”
- Q1: 建立 SFTP 连接时提示远程主机标识更新，认证失败

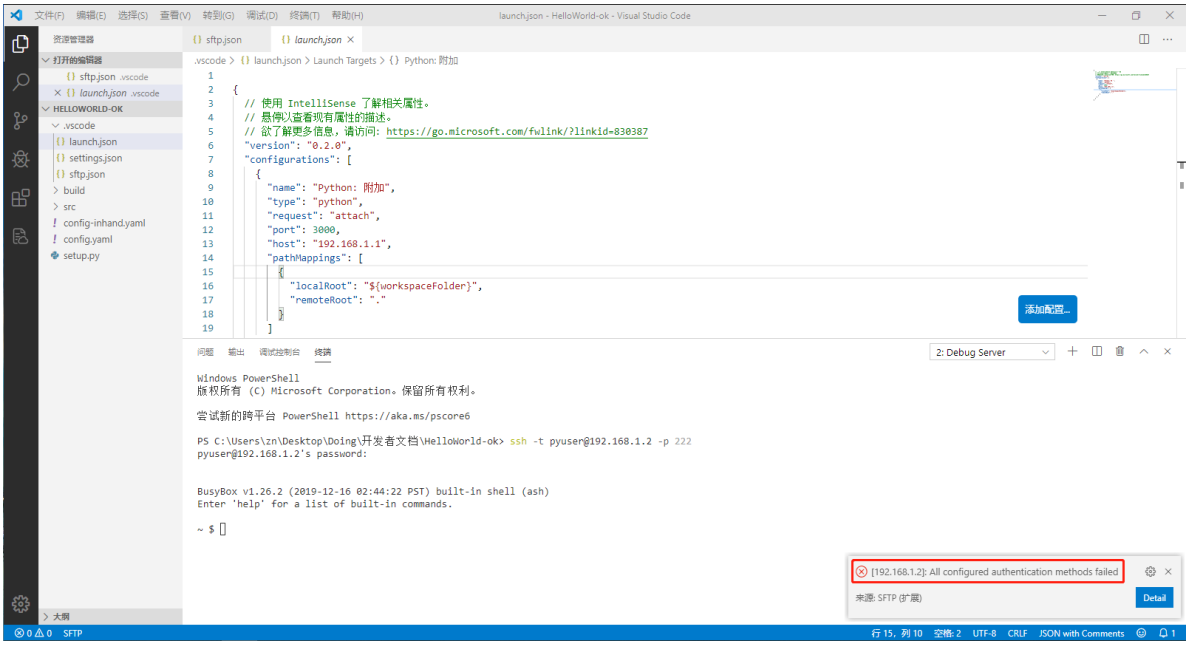


A1：出现此问题的原因是因为 MobiusPi 的密钥更新了，但是 PC 上的密钥还未更新导致认证失败。此时您只需要删掉密钥文件中有冲突的那一行即可。(按住 Ctrl 并单击冲突项可以快速访问链接)

删除后再次使用 `>SFTP:Open SSH in Terminal` 命令即可成功建立 SFTP 连接。



- Q2: 建立 SFTP 连接后，在左侧空白处右键选择”Sync Local->Remote”将代码同步到远程服务器时提示“配置的身份验证方法失败”



A2: 确保“sftp.json”文件中的 password 项与 MobiusPi 的密码一致。一致后重新同步代码。

- Q3：在开发过程中如何调用 IG902 的串口和网口

A3: RS485 串口名称为：`/dev/ttyO3`；RS232 串口名称为：`/dev/ttyO1`。串口和网口均可以使用 Python 标准的串口/网口使用方法进行调用，如使用 `pyserial` 库调用串口。

- Q4：与 MobiusPi 建立 SFTP 连接时提示“SSH 错误”，如下图所示：

A4：请安装 OpenSSH 工具以支持 SSH 协议。您可以访问：https://www.openssh.com 获取相应的 OpenSSH 工具。

1.3 Quick Start for MobiusPi Python Development

The InGateway series of Beijing InHand Networks Technology Co., Ltd. (InHand) consists of InGateway902 (referred to as **IG902** hereinafter) and InGateway501 (referred to as **IG501** hereinafter). MobiusPi is a secondary development platform for the InGateway series. This document describes how to perform Python-based secondary development on MobiusPi.

1.3.1 1. Build a MobiusPi development environment

Before starting development, ensure that you get the following items ready:

- InGateway
  - InGateway501
    * Firmware version: V2.0.0.r12351 or later
    * SDK version: py2sdk-V1.3.4 or later
    * The network is connected and the debugging mode is enabled.
  - InGateway902
    * Firmware version: V2.0.0.r12537 or later
    * SDK version: py3sdk-V1.3.5 or later
    * The network is connected and the debugging mode is enabled.
- PC
  - Python 2.7.X/3.7.X interpreter
  - Visual Studio Code software
    * Python plug-in
    * Project Templates plug-in
    * SFTP plug-in
  - OpenSSH

If your MobiusPi and PC have met all the above items, skip this section. Otherwise, prepare a development environment as follows.

1.1 Prepare the hardware and network environment

1.1.1 Connect the power supply and use a network cable to connect the platform to the PC

- Prepare the IG501 hardware

  Power on IG501 and connect the PC and IG501 through an Ethernet cable according to the topology.

- Prepare the IG902 hardware

  Power on IG902 and connect the PC and IG902 through an Ethernet cable according to the topology.

1.1.2 Configure the LAN parameters

- Configure the IG501 LAN parameters by referring to Access IG501 in a LAN.
- Configure the IG902 LAN parameters by referring to Access IG902 in a LAN.

1.1.3 Configure the WAN parameters

- Configure the IG501 WAN parameters by referring to Connect IG501 to the Internet.
- Configure the IG902 WAN parameters by referring to Connect IG902 to the Internet.

1.1.4 Update the software version

If you want to get the latest MobiusPi and its functional characteristics, contact Customer Services. To update the software version, see the following links:

- Update the IG501 software version
- Update the IG902 software version

1.1.5 Enable the debugging mode for MobiusPi

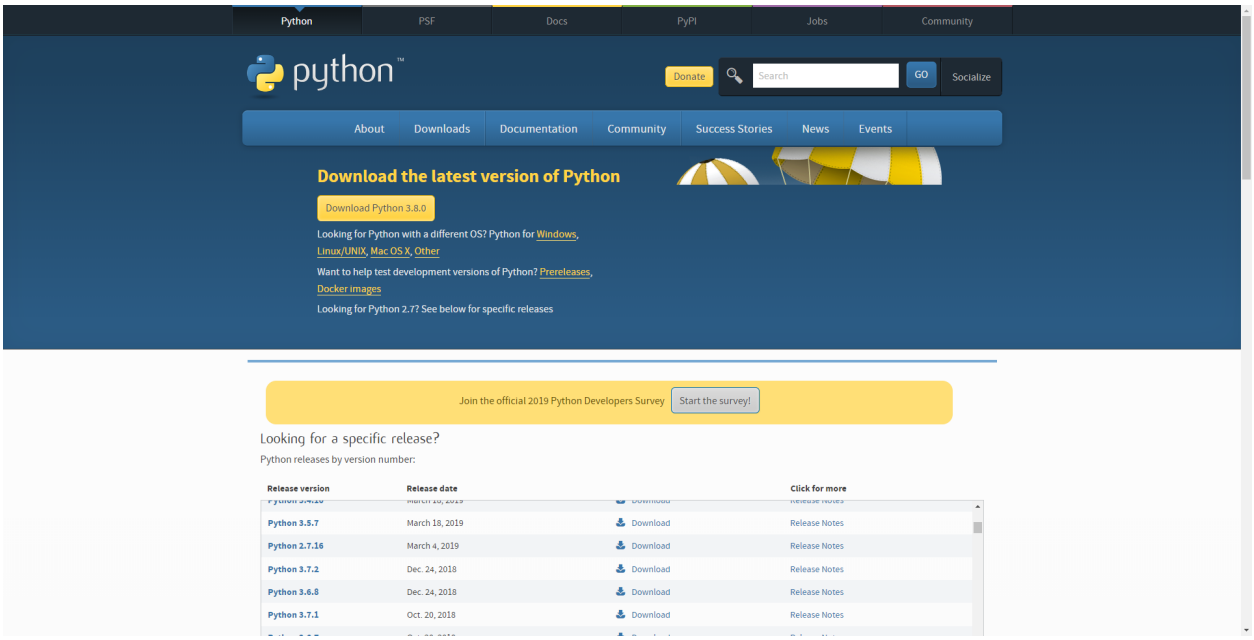
To run and debug Python code on MobiusPi during development, you need to enable the debugging mode for MobiusPi.

- Enable the IG501 debugging mode
- Enable the IG902 debugging mode

1.2 Install required software on the PC

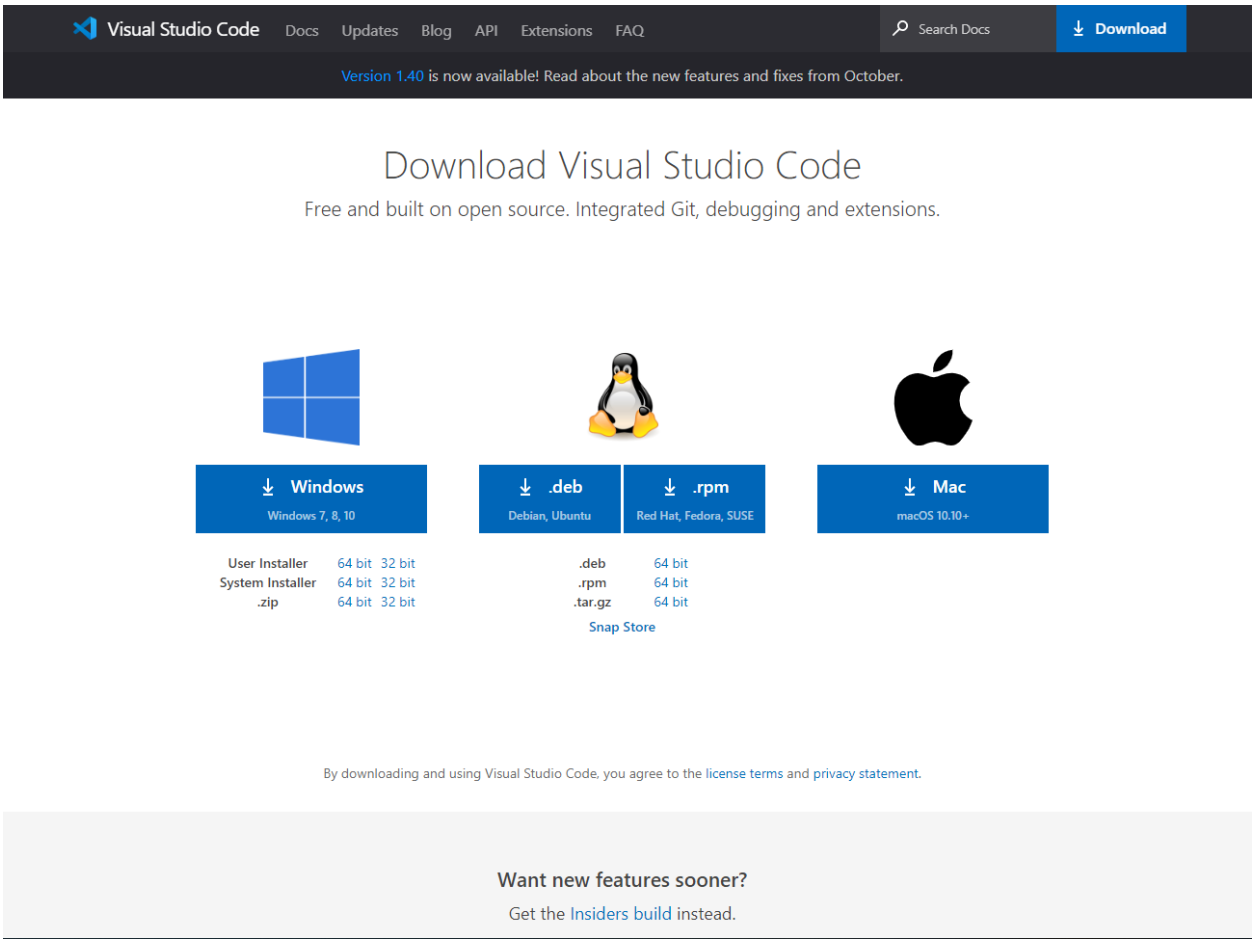
1.2.1 Install the Python interpreter

The PC shall be installed with the Python2.7.X or 3.7.X interpreter (the 3.7.X interpreter is recommended). You can download the install package from https://www.python.org/downloads/ and install it to the PC.

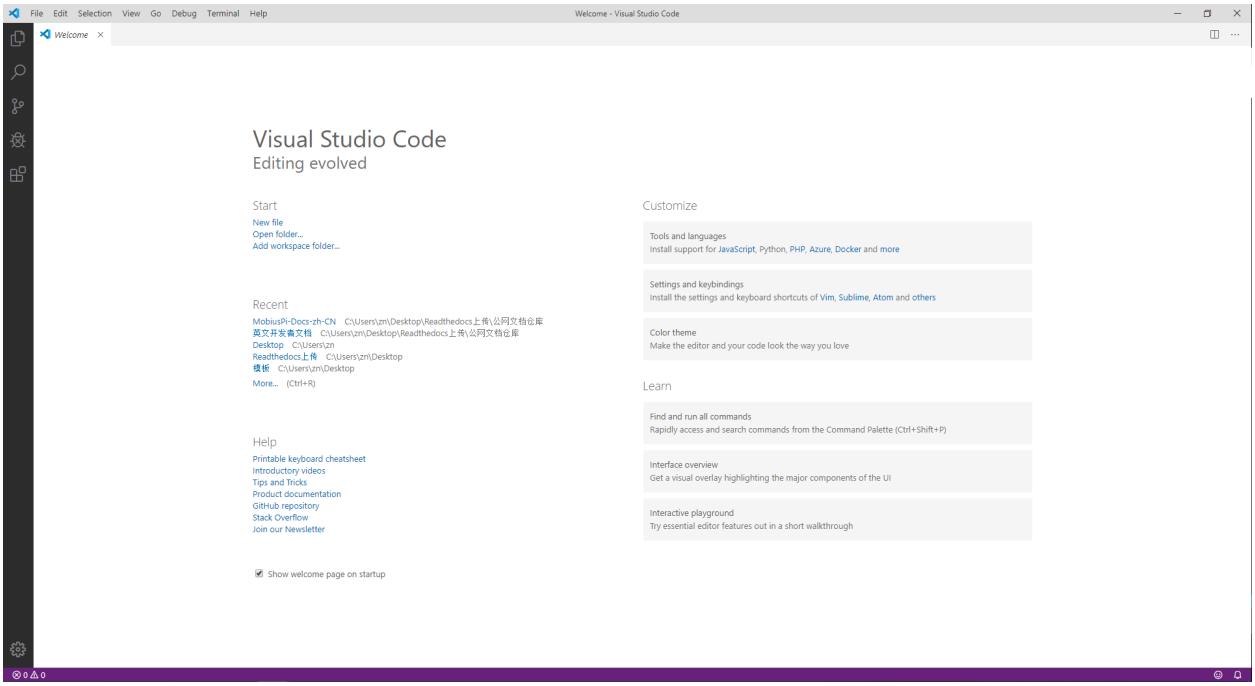


1.2.2 Install Visual Studio Code

You can download Visual Studio Code (VS Code) from https://code.visualstudio.com/Download.



After VS Code is downloaded and installed, run the software, and its page is as follows.



1.2.3 Install OpenSSH

You can enter the `ssh` command in Command Prompt to check whether the PC supports the SSH protocol. If the PC supports the SSH protocol, the following figure is displayed:

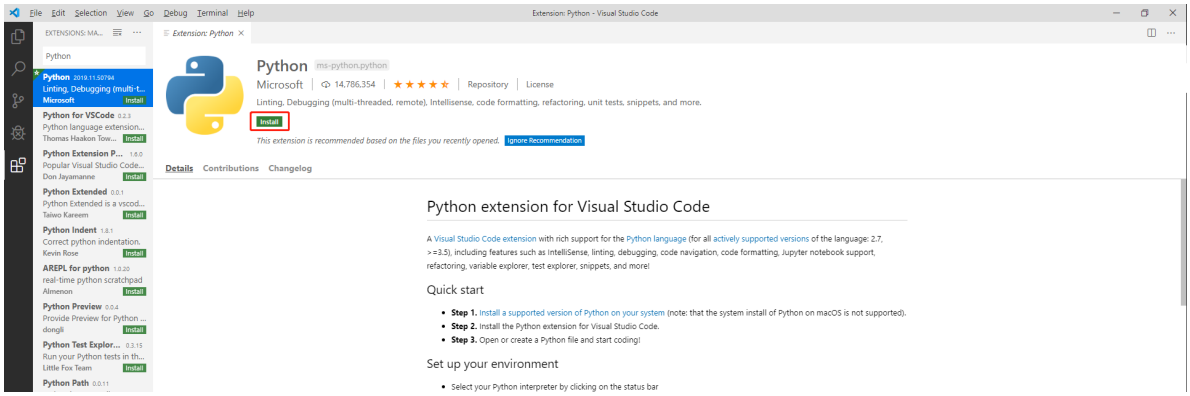
If the PC does not support the SSH protocol, download OpenSSH from https://www.openssh.com and install it to the PC.

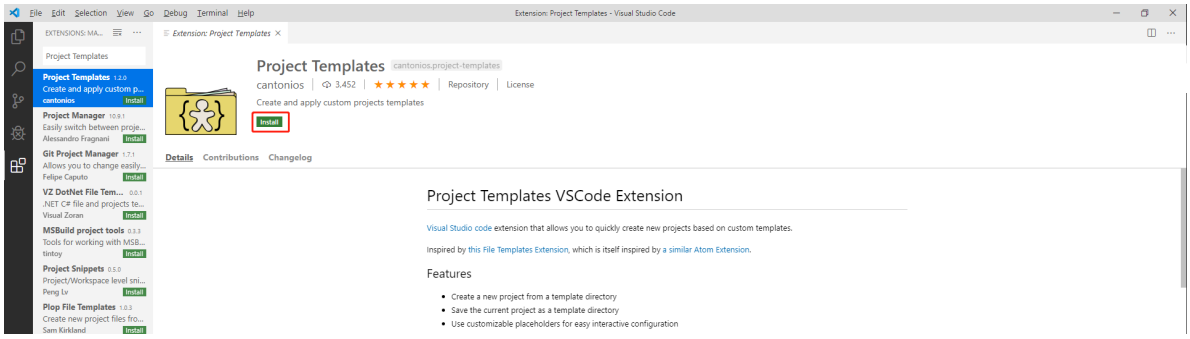
1.3 Prepare the VS Code development environment

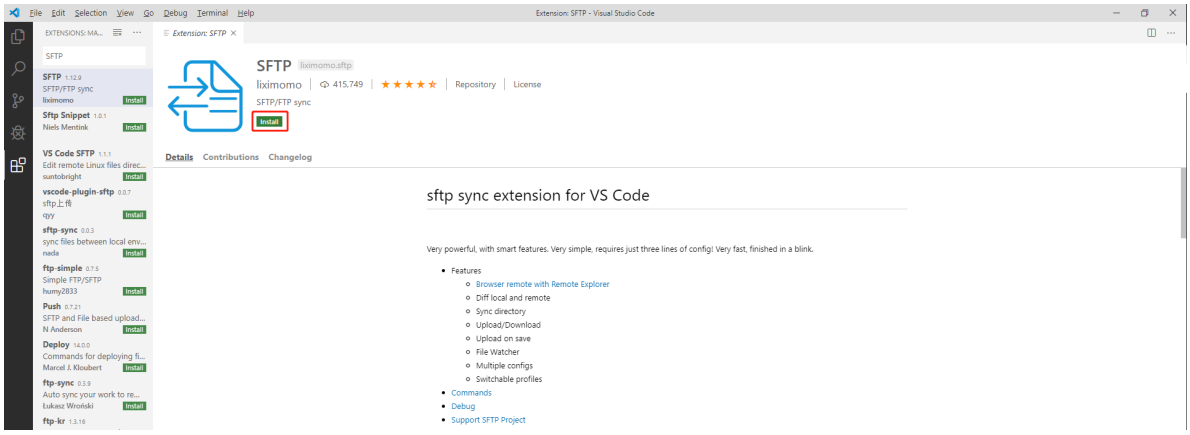
1.3.1 Install the VS Code extension

To develop and debug the Python code on MobiusPi, you must install the following extensions in Extensions of VS Code IDE:

- `Python` is a VS Code Python extension. It provides rich features, including IntelliSense, Linting, debugging, code navigation and formatting, support to Jupyter notebook, restructuring, variable resource manager, test resource manager, and code snippets. For more information, see its official website.
- `Project Templates` is a VS Code extension that allows you to quickly create new projects based on a custom template. InHand will release multiple Python App templates. You can use Project Templates to import a template and initialize your project quickly.
- `SFTP` allows you to use the SFTP Sync plug-in to upload the code to MobiusPi.



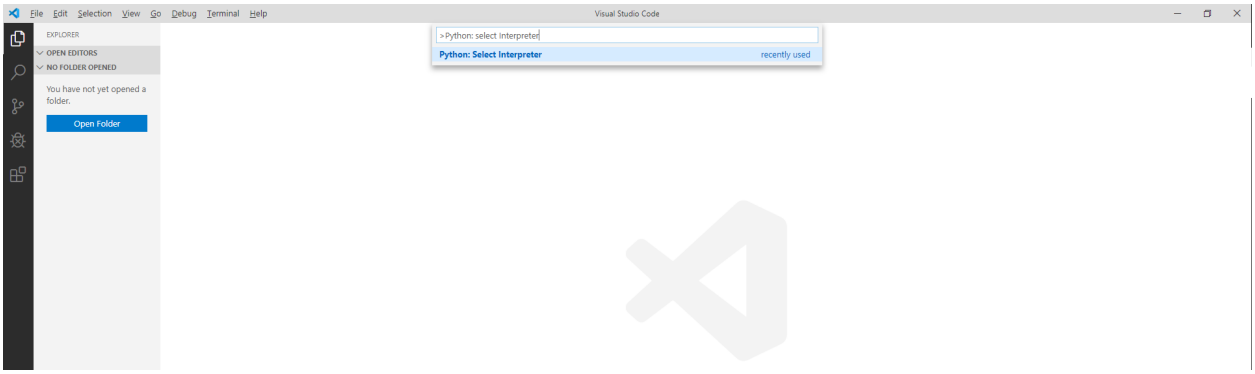




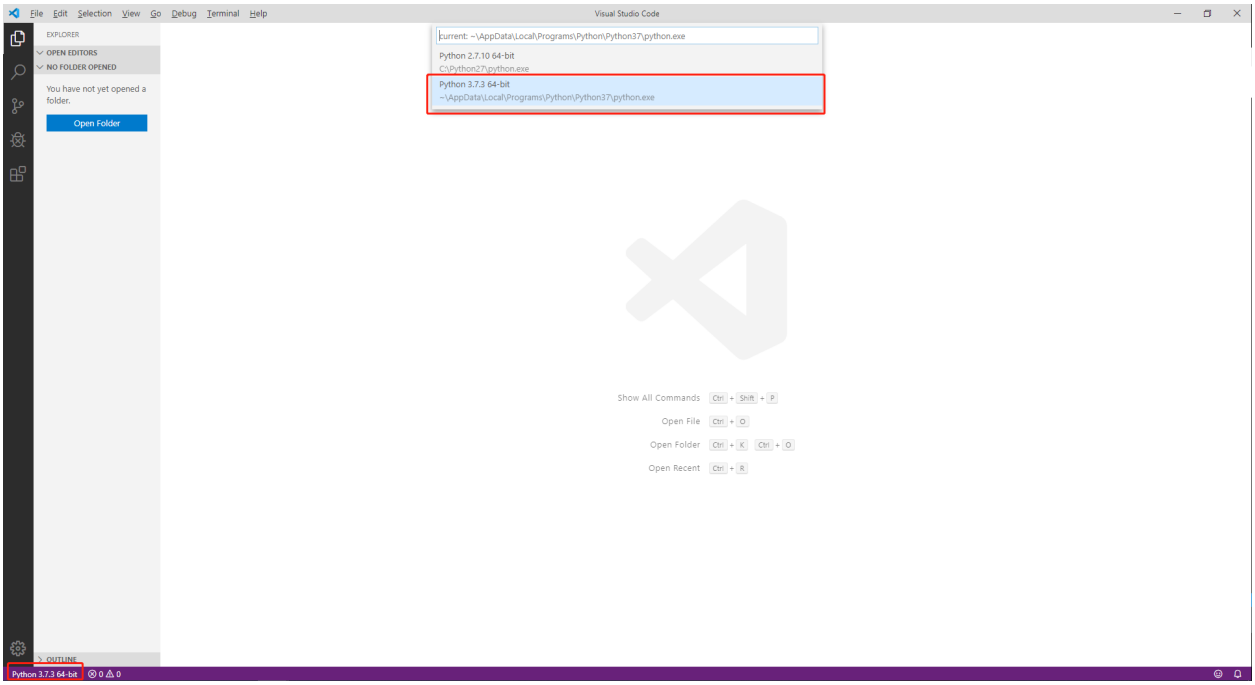
Now, all plug-ins required by MobiusPi have been installed. For more information about VS Code plug-ins, see the Visual Studio Code official website.

1.3.2 Configure the Python interpreter version

Press `Ctrl+Shift+P`. On the displayed Command Palette, enter `>Python: select Interpreter`.



Select a required Python interpreter. In this document, the Python 3.7.X interpreter is used. The selected 3.7.X version shall be consistent with that on the **Edge Computing > Python Edge Computing** page. Then, the selected Python interpreter version is displayed in the lower-left corner of VS Code.



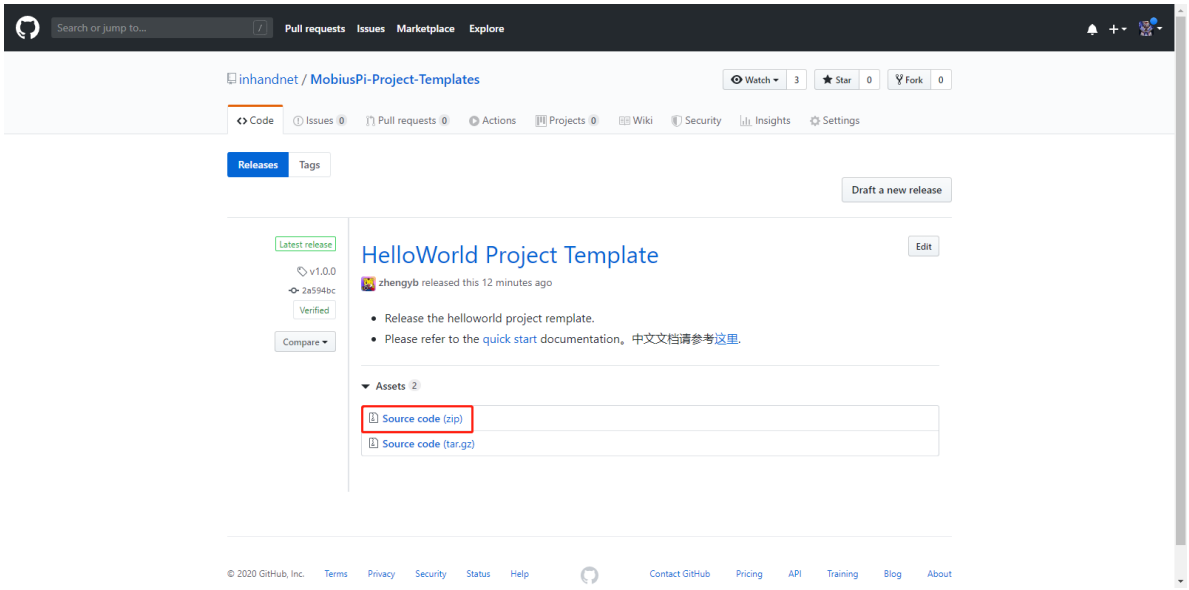
1.3.3 Configure the project template

1.3.3.1 Apply the InHand standard project template

- Step 1: Click Here to download a MobiusPi project template.

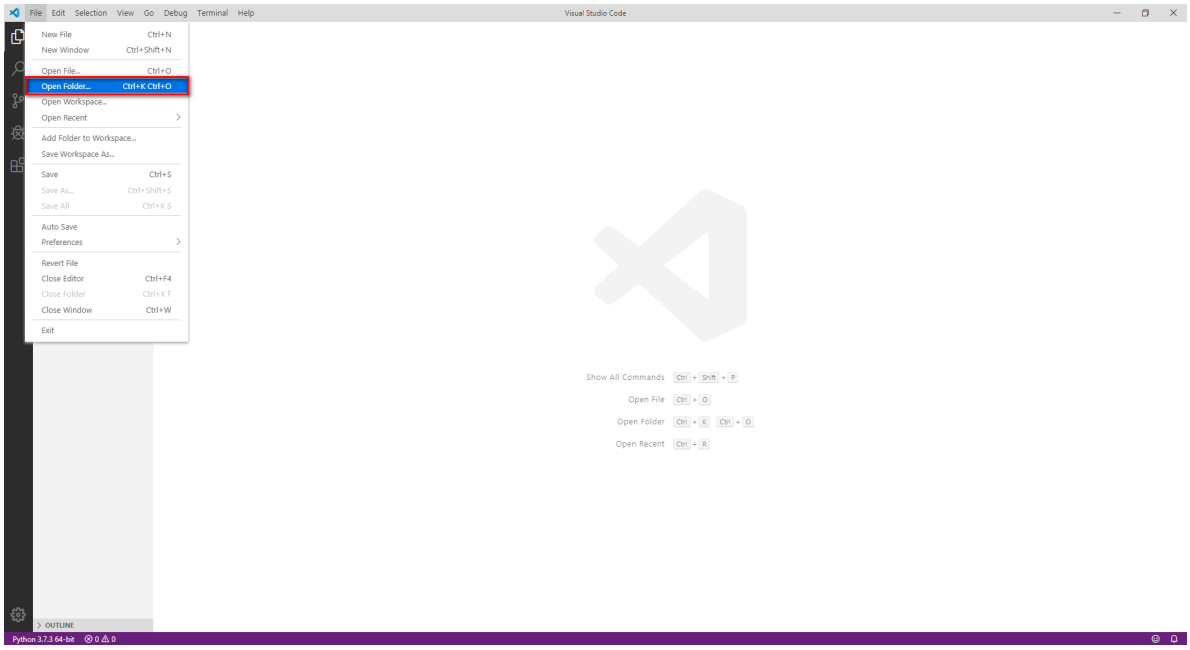
  MobiusPi provides multiple project templates for you to quickly initialize your project directory. For more information about each project template, see README.md. In this document, the standard

project template **helloworld-template** is used as an example.



- Step 2: Open the project template.

  Decompress the downloaded project template package, use VS Code to open the helloworld-template folder in the decompressed package, choose **Files > Open Folder**, and select the decompressed **helloworld-template** folder.



The following figure shows the opened project template folder **helloworld-template**. The project template contains the following information:

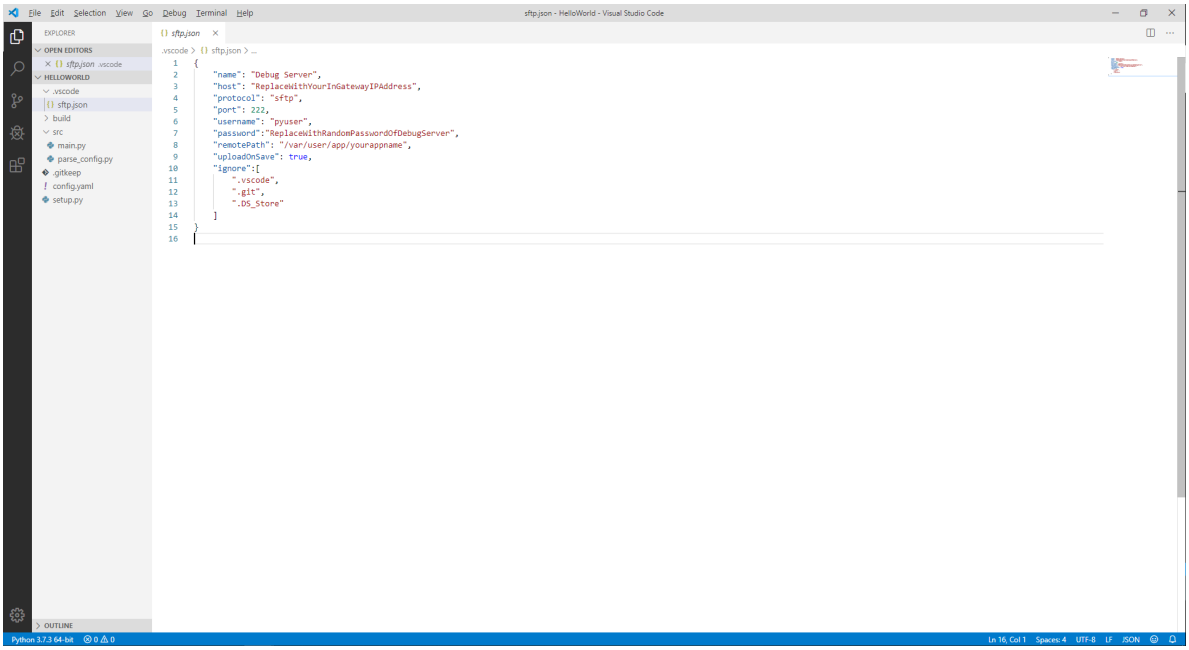
```
.vscode
```

(continues on next page)

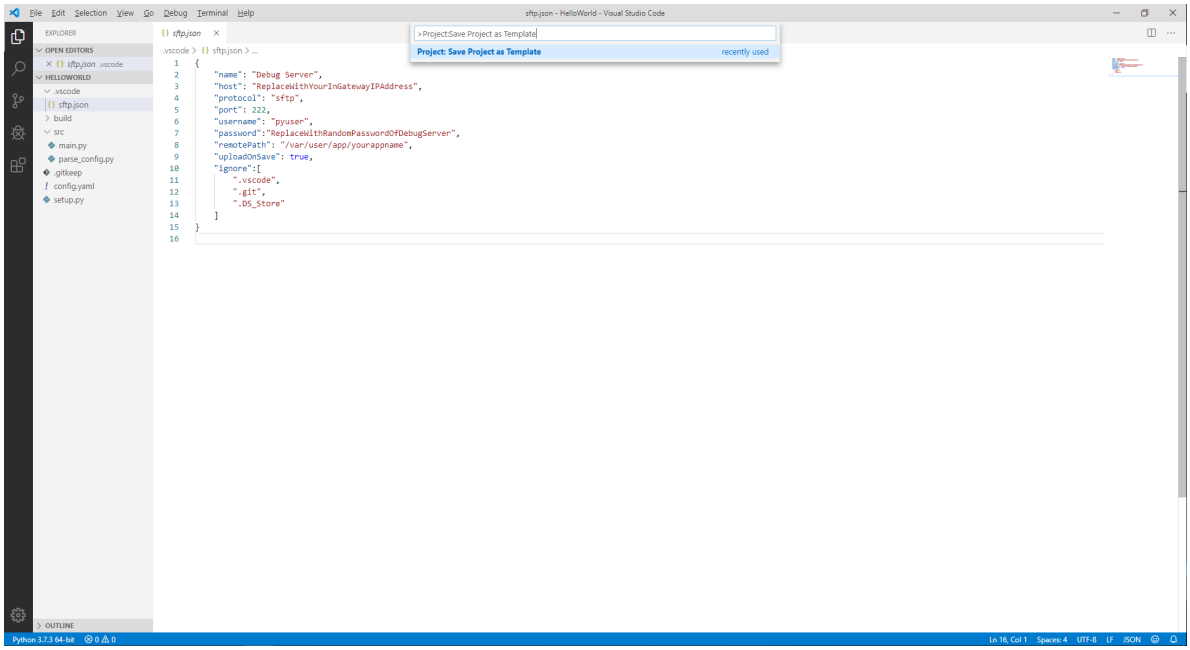
(continued from previous page)

```
    sftp.json
build
lib
src
    main.py
    parse_config.py
config.yaml
setup.py
```

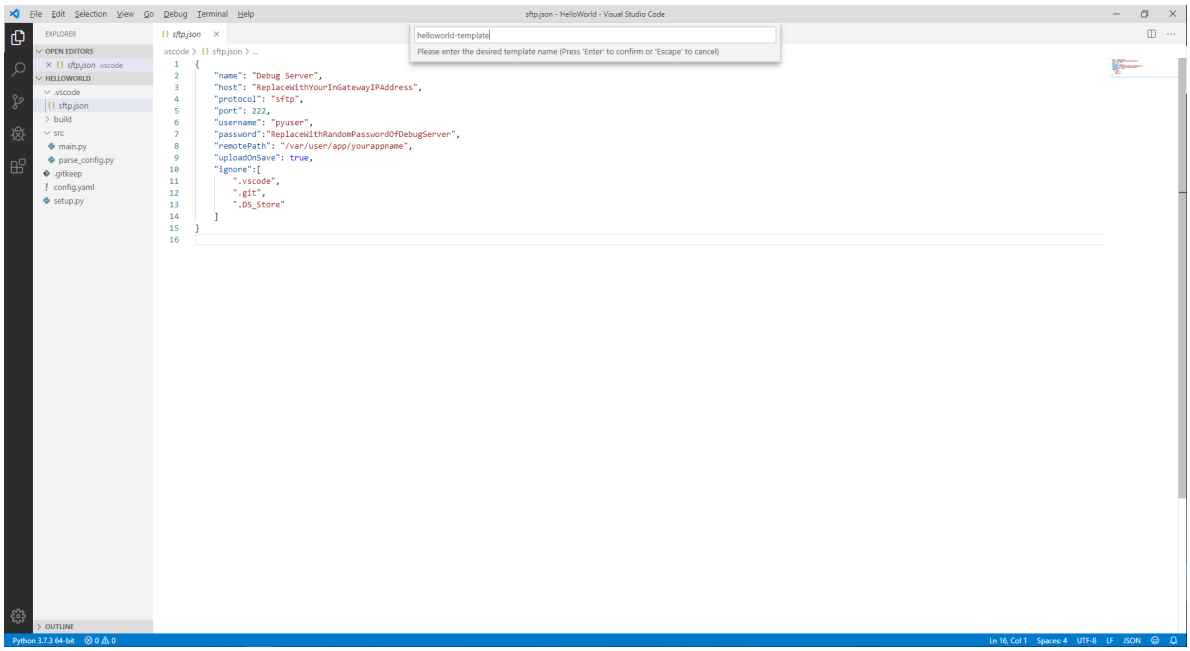
- `.vscode`: The VS Code configuration folder.
  - `sftp.json`: The configuration file of the SFTP plug-in, which is used to connect to MobiusPi over SFTP.
- `build`: The App release package folder.
- `lib`: The folder of the App third-party dependency library.
- `src`: The App source code folder.
  - `main.py`: The App entry.
  - `parse_config.py`: The App parsing configuration file.
- `config.yaml`: The App configuration file.
- `setup.py`: The App version, SDK version, and other information.



- Step 3: In the Command Palette, enter the `>Project:Save Project as Template` command to save the current project file as a template.



Enter a name for your template, such as helloworld-template.



1.3.3.2 Custom project template

- Step 1: Create a project template folder which must contain the following information. You can add other files as needed.

```
.vscode
  sftp.json
```

(continues on next page)

(continued from previous page)

```
build
src
   main.py
setup.py
```

  - `.vscode`: The VS Code configuration file. You can enter the `>SFTP:Config` command in the VS Code Command Palette to quickly create the .vscode folder and the sftp.json file.
    * `sftp.json`: The configuration file of the SFTP plug-in, which is used to connect to MobiusPi over SFTP.
  - `build`: The App release package folder.
  - `src`: The App source code folder.
    * `main.py`: The App entry.
  - `setup.py`: The App version, SDK version, and other information. We recommend that you define the information according to the standard template.

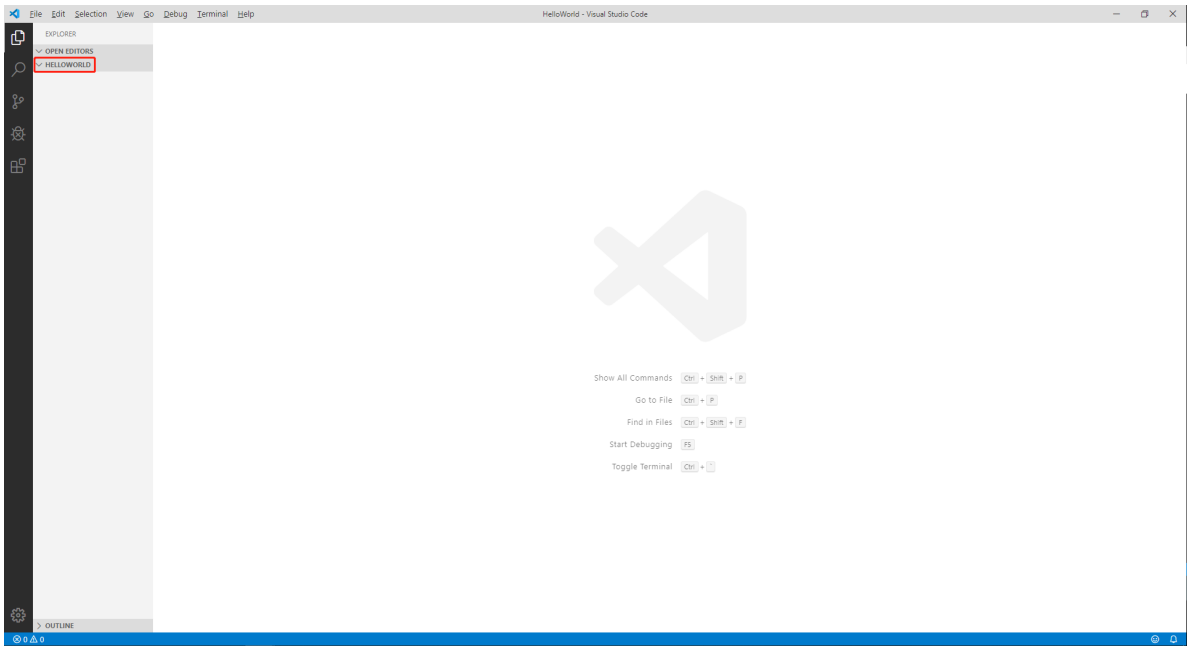
- Step 2: Use VS Code to open the custom project template folder, choose **Files > Open Folder**, and select the custom project template folder.
- Step 3: In the Command Palette, enter the `>Project:Save Project as Template` command to save the current project file as a template.

1.3.2 2. Compile the first MobiusPi App: Hello World

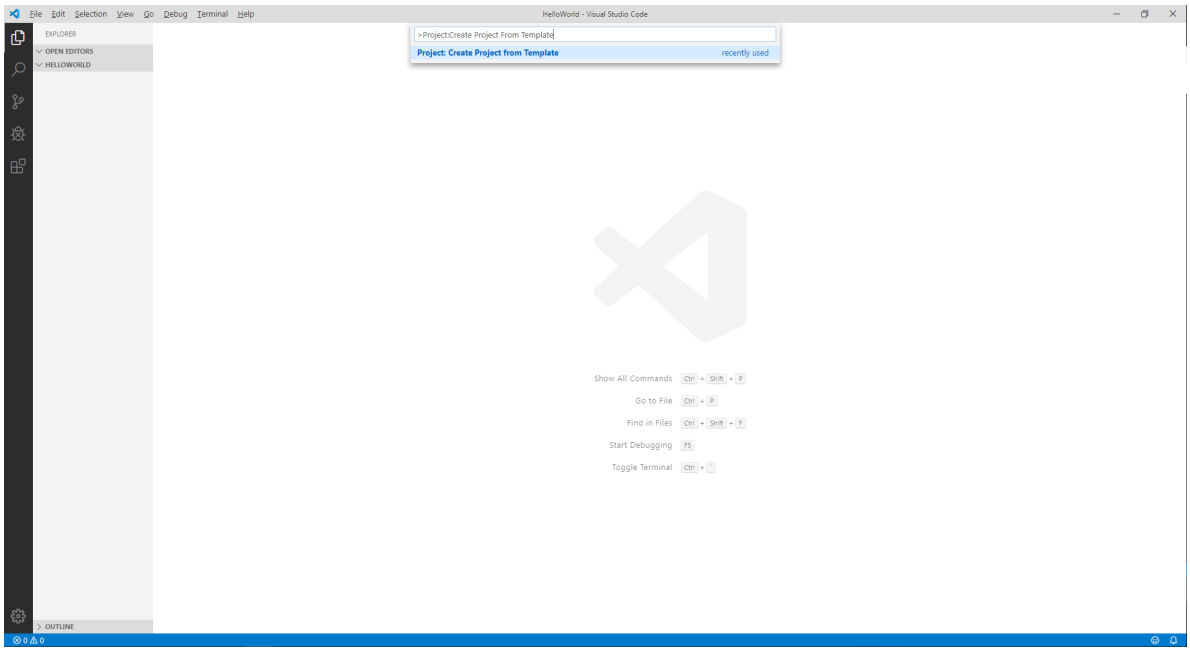
This document takes the **HelloWorld** App as an example to describe how to develop MobiusPi Python Apps in VS Code. This App can print a "hello world!" log every 10s in MobiusPi and allows you to import the configuration file and modify the log.

2.1 Use a template to create a project

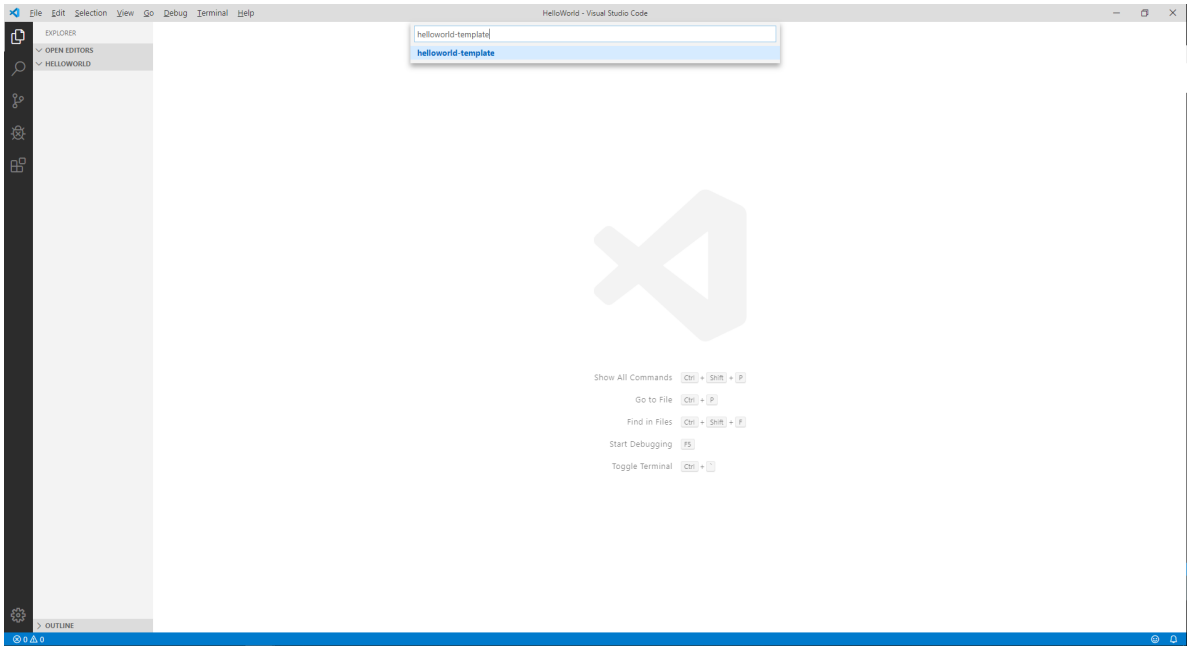
- Step 1: Use VS Code to open the project folder of the Python App. The following figure is displayed:



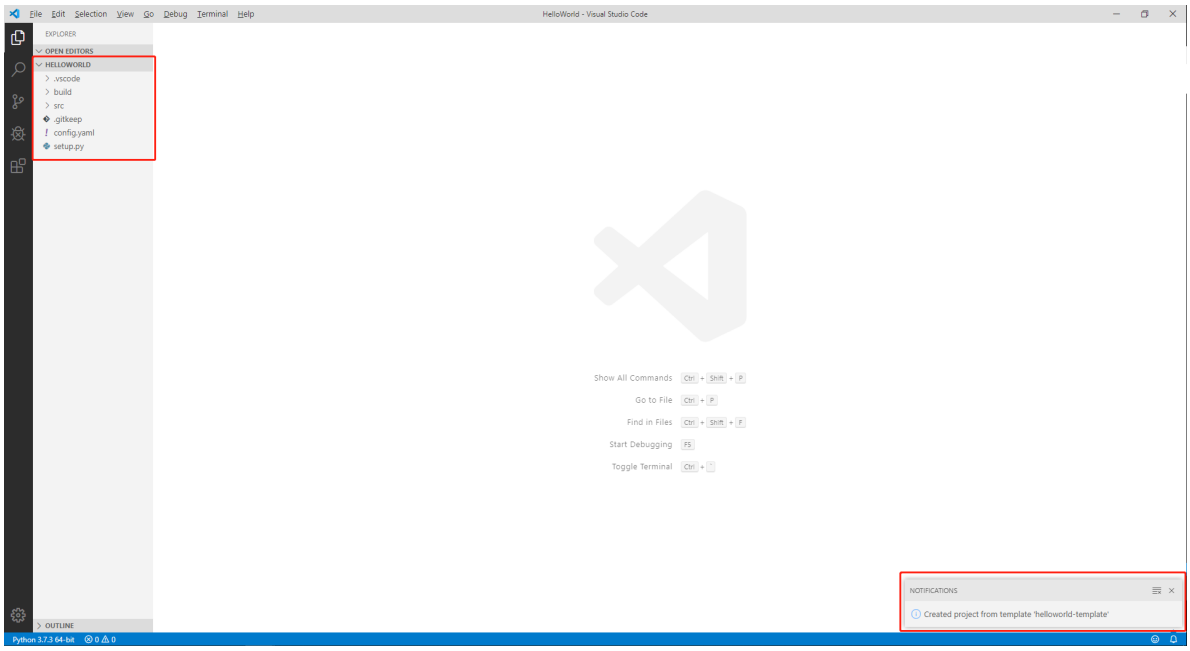
- Step 2: In the Command Palette, enter the `>Project:Create Project From Template` command to quickly create a project directory based on the existing template.



- Step 3: Enter the name of the helloworld-template, and press Enter.



After the template is selected, VS Code automatically adds the file that you add in the template to the folder of the current project.



2.2 Encoding

The standard project template **helloworld-template** can print a "hello world!" log every 10s in MobiusPi and allows you to import the configuration file and modify the log. To modify the App name, modify the code in `main.py` and `setup.py` by referring to the following figure. Note: The Python App name cannot contain any space.

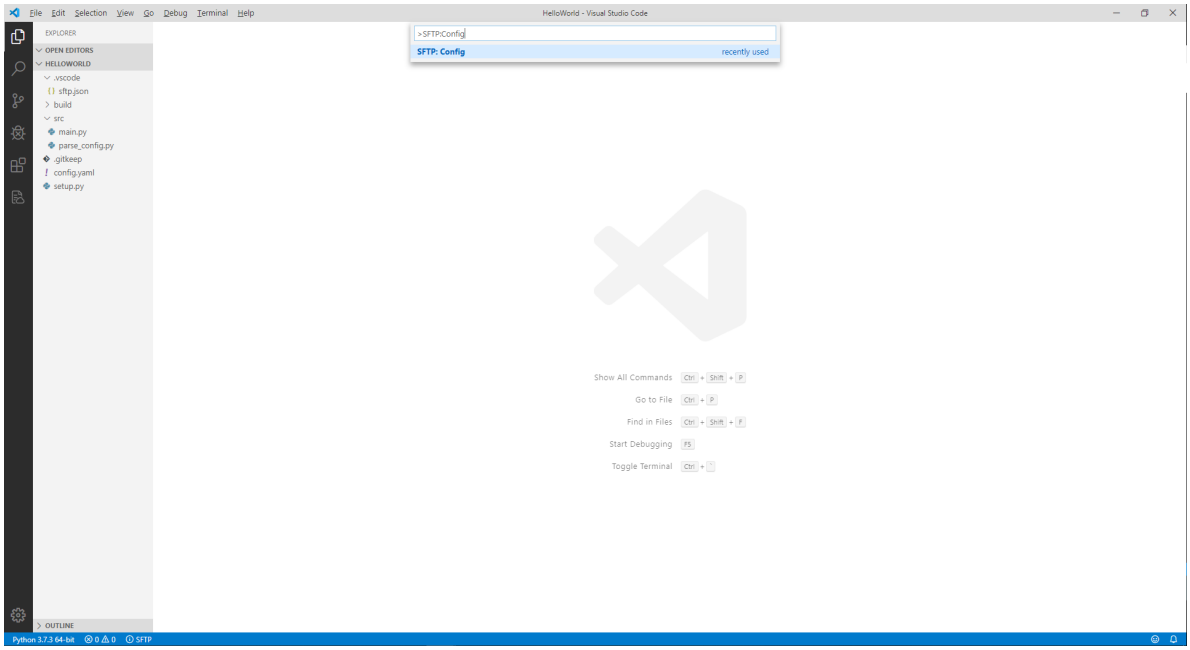
2.3 Debugging

2.3.1 Create an SFTP connection

To debug the code remotely, upload the local code to the remote server (MobiusPi). Before uploading the local code, ensure that the debugging mode has been enabled for MobiusPi. After the debugging mode is enabled, the following figure is displayed:

- Step 1: Open the `sftp.json` file.

  In the Command Palette, enter the `>SFTP:Config` command and then open the `sftp.json` file.



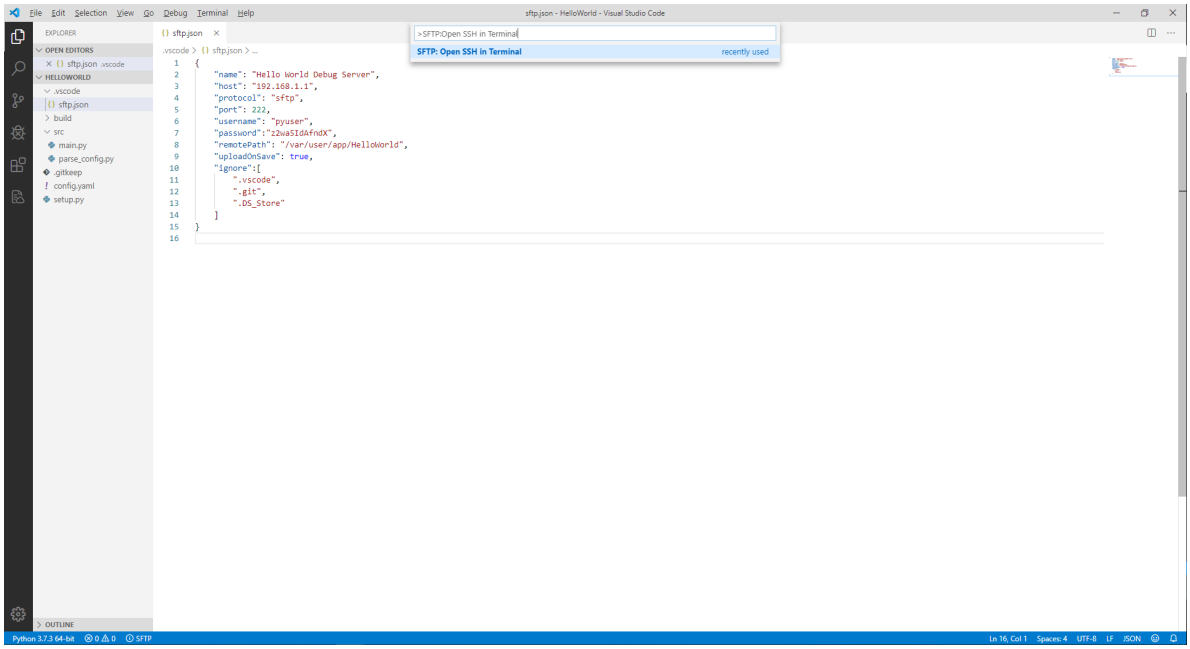
- Step 2: Configure the SFTP connection.
  - Configure the SFTP connection for IG501

    In the `sftp.json` file, configure the SFTP connection according to the connection parameters on the **Edge Computing > Python Edge Computing** page. Note: The Python App name must be same to that in `mian.py`.

– Configure the SFTP connection for IG902

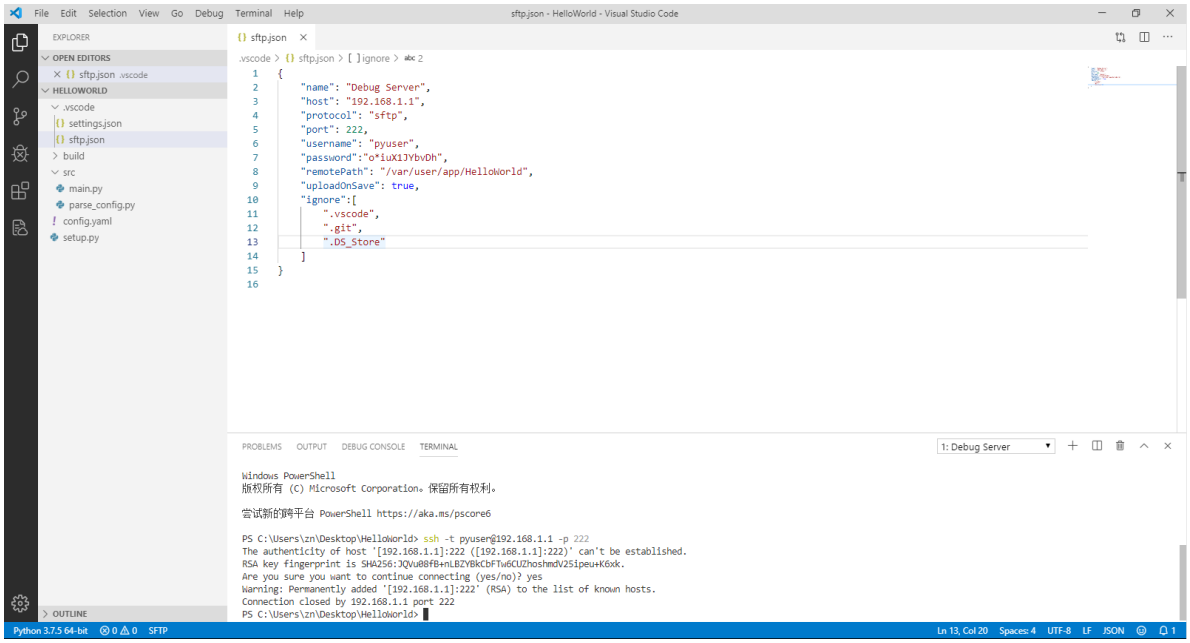
In the `sftp.json` file, configure the SFTP connection according to the connection parameters on the **Edge Computing > Python Edge Computing** page. Note: The Python App name must be same to that in `mian.py`.

- Step 3: After the configuration is completed and saved, enter the `>SFTP:Open SSH in Terminal` command in the Command Palette to connect to the remote server.

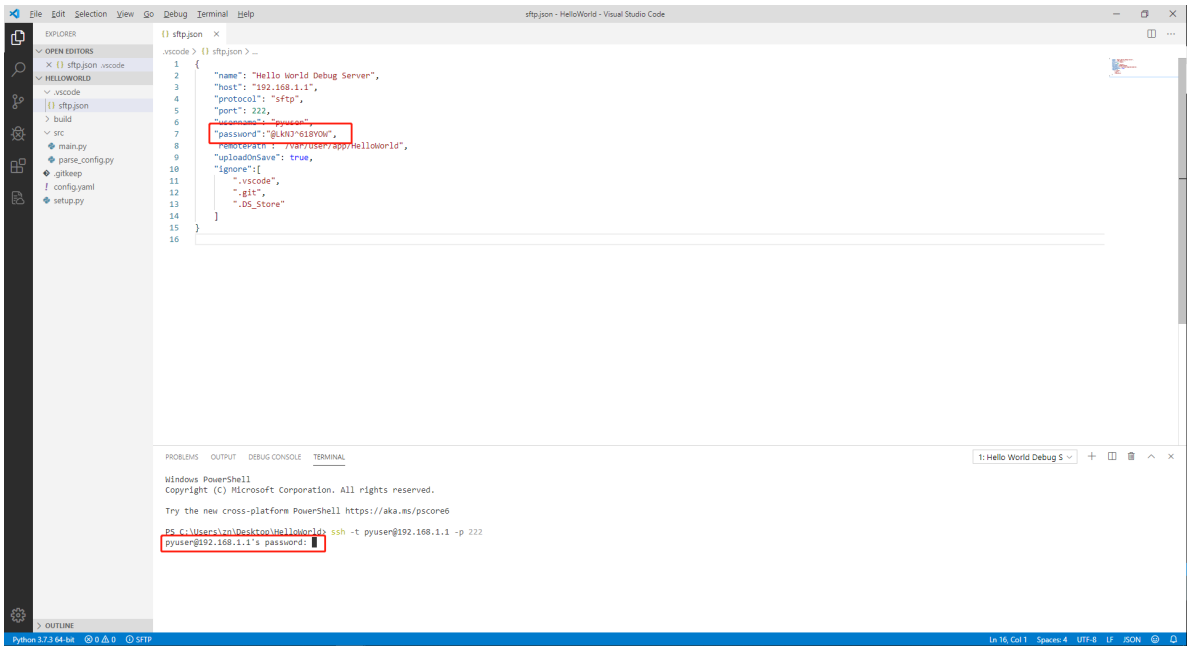


- Step 4: In the Command Palette, you are requested to select the SFTP server to be connected. Select the SFTP server in `sftp.json` and press Enter.

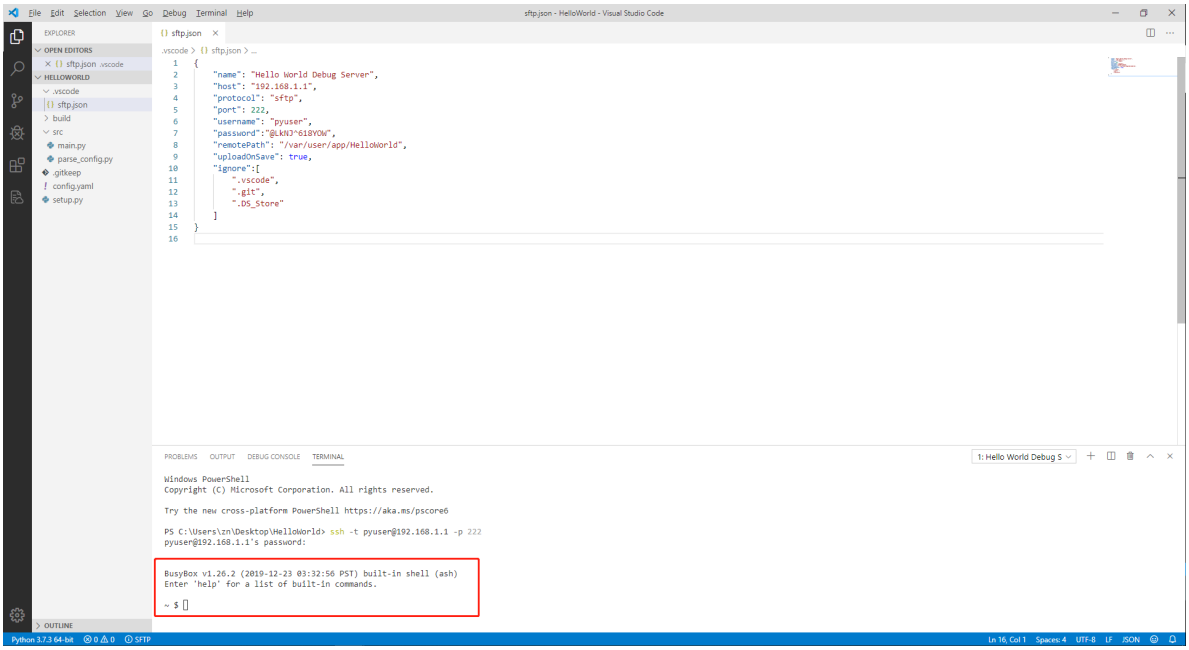
- Step 5: If this is the first time you connect to the server, the **TERMINAL** window prompts you to confirm whether you want to connect to the server. Enter **Yes** and press Enter. Then, enter the `>SFTP:Open SSH in Terminal` command and the IP address of the SFTP server in the Command Palette again.



- Step 6: The **TERMINAL** window prompts you to enter the password. You only need to copy the password in the `sftp.json` file, and paste it to the required place.



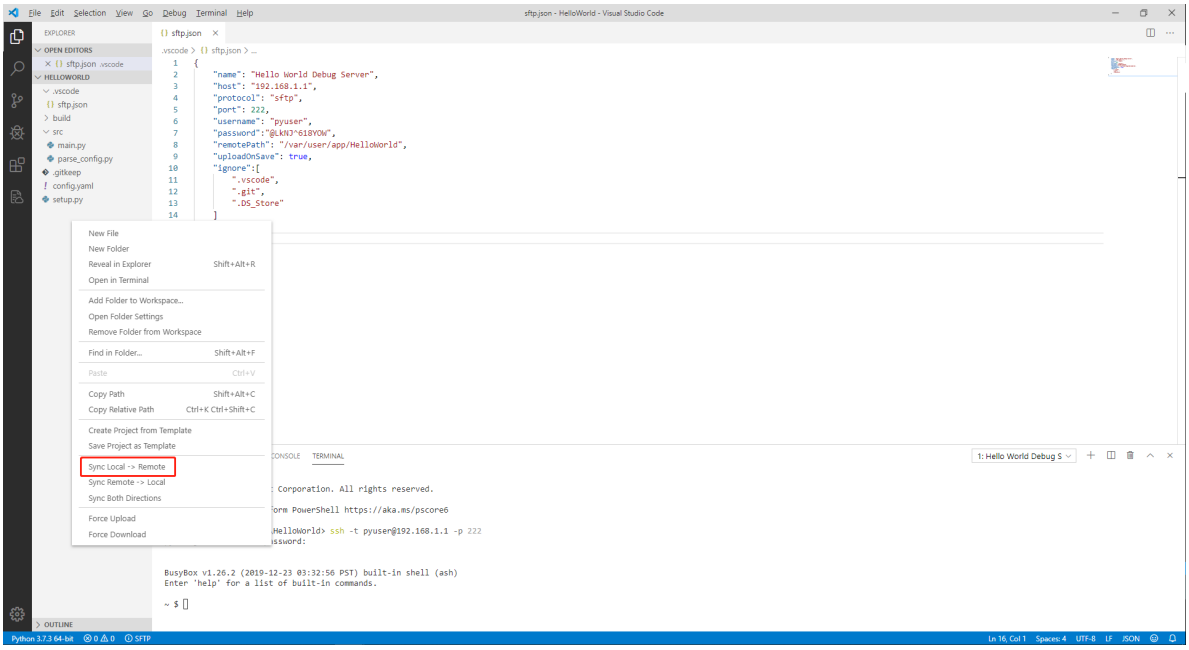
After an SFTP connection with MobiusPi is created, the page is shown as follows:



2.3.2 Debug the code

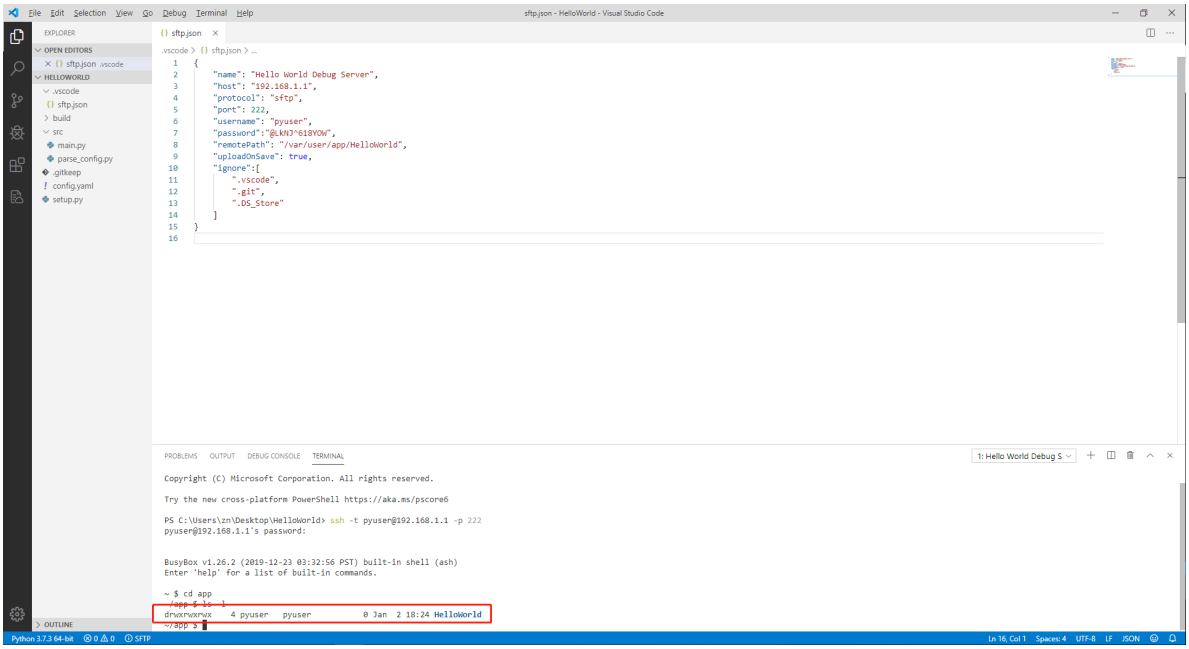
- Step 1: Synchronize the code.

  After the SFTP connection is created, right-click the blank area on the left, and choose **Sync Local > Remote** to synchronize the code to the remote server. Later, if you modify or delete code on the local PC, the code changes will be automatically synchronized to the remote server.



You can check whether the remote server has received the App code in the TERMINAL window. In the TERMINAL window, enter the following command to view uploaded App folder information:

```
cd app
ls -l
```



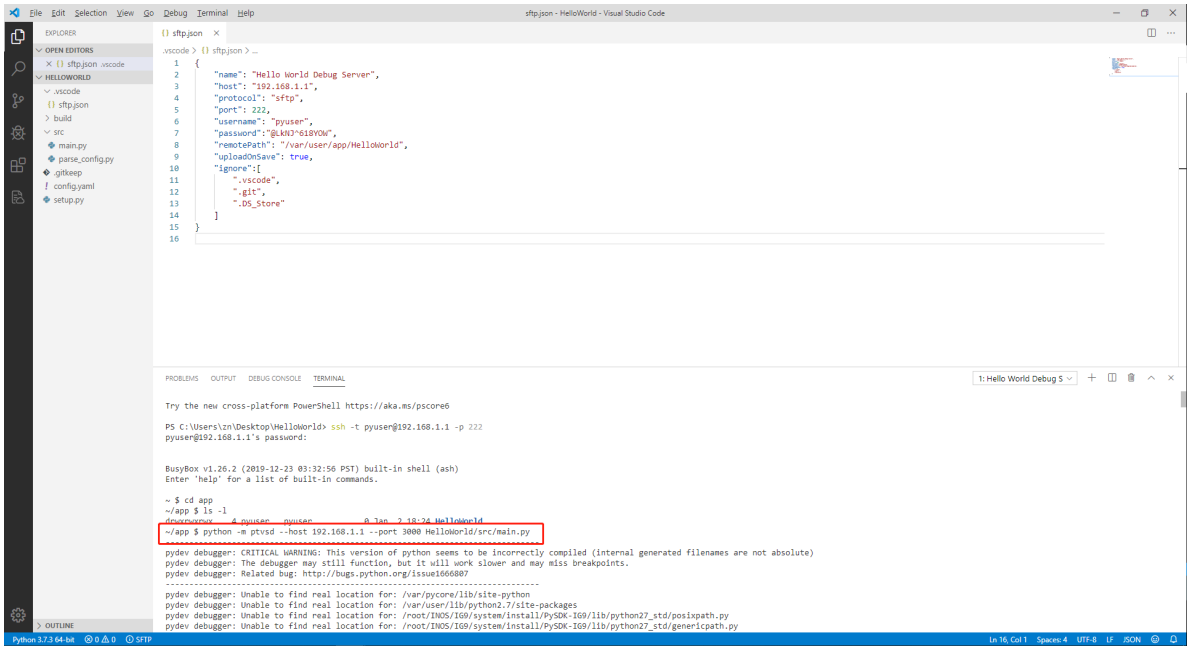
- Step 2: Debug the script in the TERMINAL window.

  After the code is synchronized, in the TERMINAL window, enter the following command to execute the script on IG501 immediately. After the script is executed, you can view the execution result in the TERMINAL window: the log "hello world!" is printed.

```
python -m ptvsd --host 192.168.1.1 --port 3000 HelloWorld/src/main.py
```

  - `192.168.1.1` is the IP address of the **host** configured in `sftp.json`.
  - `3000` is the recommended debugging port number.
  - `HelloWorld/src/main.py` is the execution path of `mian.py`, which can be adjusted as needed.

  MobiusPi' s Python development environment provides the ptvsd dependency library for remote code debugging. For more information about the ptvsd plug-in, see ptvsd user manual.



- Step 3: After debugging, press `Ctrl + C` on the terminal to terminate the debugging process.

2.4 Build an App release package

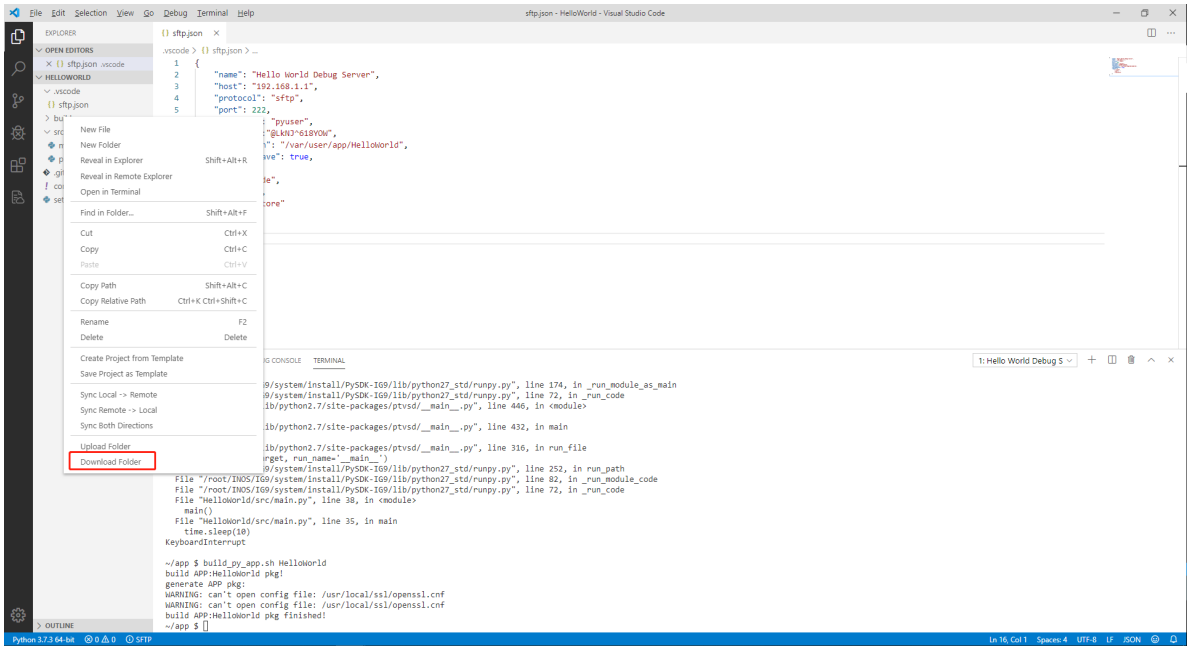
After debugging, you can build an App release package to quickly deploy the App to other MobiusPi devices.

- Step 1: Build an App release package.

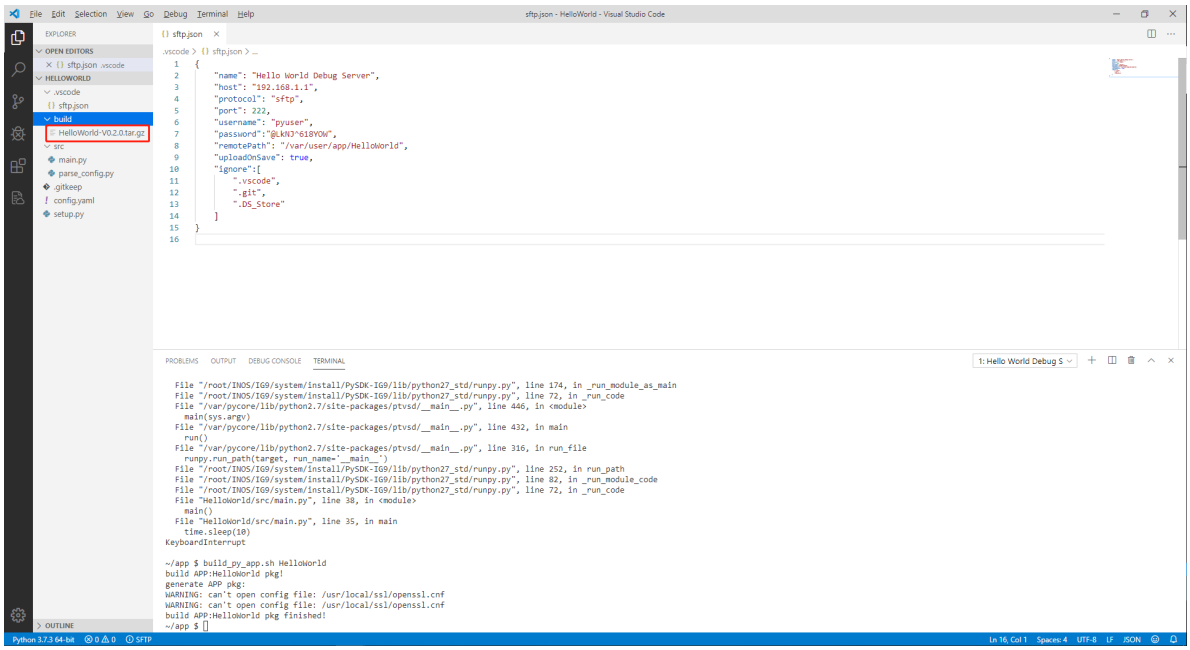
  In the **TERMINAL** window, run the `build_py_app.sh HelloWorld` command to build an App release package. (That is, the `build_py_app.sh` Python App name.)

- Step 2: Download the App release package.

  After the command is executed, the App release package is automatically generated in the build directory on the remote server. Right-click the **build** folder and select **Download Folder** to download the built App release package to the local PC for subsequent deployment.



After the package is downloaded, you can view the HelloWorld App release package in the build directory.



2.5 Deploy the App on the MobiusPi web page

Run the `main.py` script. Or, after the App release package is built, the App is automatically generated on the connected MobiusPi, but this App cannot be started. Refer to the following links to deploy the App to MobiusPi:

- Deploy an App on IG501

- Deploy an App on IG902

2.6 View the App running status

You can view the App running status on the **Edge Computing > Python Edge Computing** page in MobiusPi.

Click **View Logs** to view the App running logs.

```
DEBUG:root:Hello, world! Welcome to the Inhand!
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
```

2.7 Update the App configuration file

- Step 1: Modify the configuration file.

  In the **config.yaml** file of the App, modify `description: "hello world!"` to `description: "hello inhand!"`.

```
config:
    description: "hello inhand!"
    others:
        LOG:
            #Enable debug feature to write all logs to the console
            #Default: debug=0
            debug: 1
```

- Step 2: Import the configuration file and restart the App.

  On the **Edge Computing > Python Edge Computing** page in MobiusPi, import the modified configuration file of the **HelloWorld** App, and restart the App.

After restart, the HelloWorld App runs with the updated configuration file and prints a log "hello inhand!" every 10s.

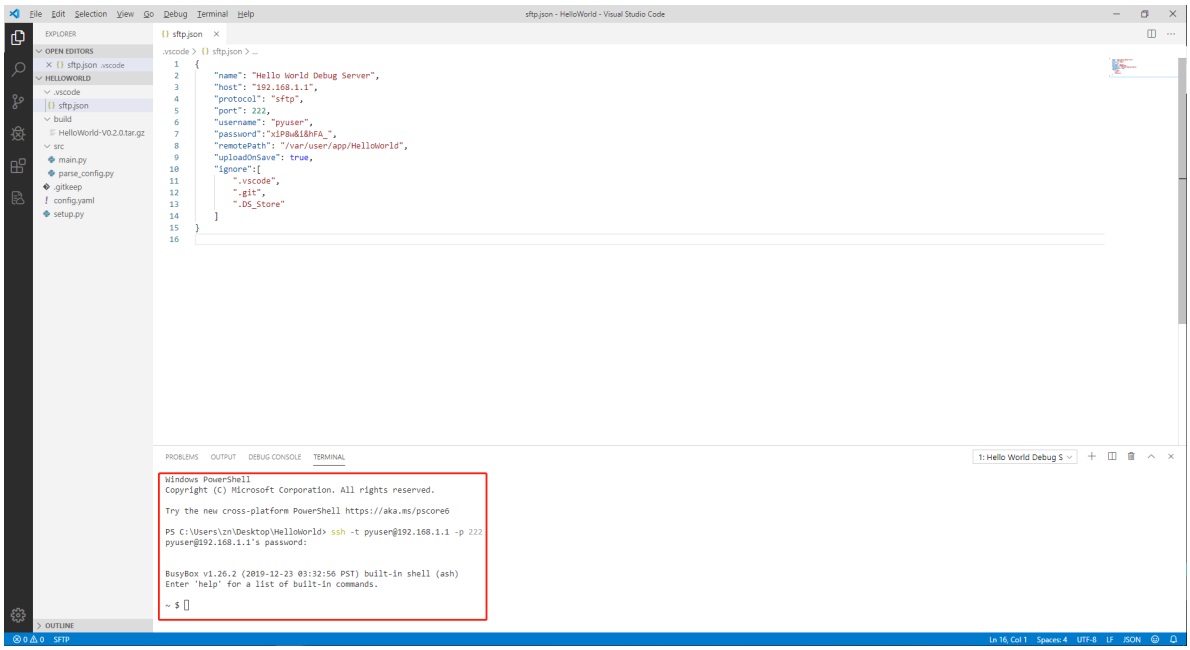
```
DEBUG:root:Hello, world! Welcome to the Inhand!
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
INFO:root:debug:1
INFO:root:decription:hello world!
DEBUG:root:Hello, world! Welcome to the Inhand!
INFO:root:decription:hello inhand!
INFO:root:debug:1
INFO:root:decription:hello inhand!
INFO:root:debug:1
INFO:root:decription:hello inhand!
INFO:root:debug:1
INFO:root:decription:hello inhand!
INFO:root:debug:1
INFO:root:decription:hello inhand!
```

2.8 Annex

2.8.1 Install a third-party dependency library for the specified App

To install a third-party dependency library for the specified App, you need to enable the debugging mode for MobiusPi and connect to the Internet. The following takes installing the `xlrd` dependency library to the HelloWorld App as an example to describe how to install a third-party dependency library to a specified App.

- Step 1: Use VS Code to connect to MobiusPi over SFTP. For more information, see *Create an SFTP connection*.



- Step 2: Run `pip install` + **dependency library name** + **==version number** + `-t` + **App' s lib folder path** and press Enter to install the dependency library to the specified App. (If the version number is not added, after the pip command is run, the latest dependency library is automatically installed.)

```
pip install xlrd==1.2.0 -t /var/user/app/HelloWorld/lib/
```

- Step 3: Then, the dependency library is automatically downloaded and installed. After the library is installed, the following figure is displayed:

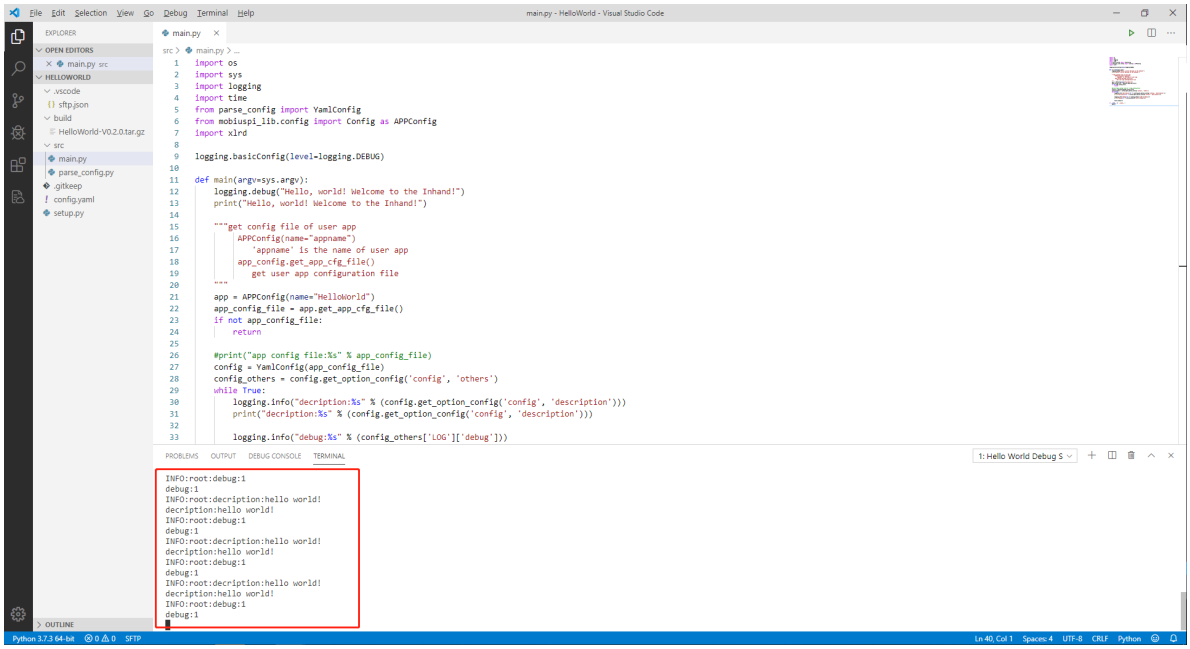
- Step 4: Run the `export` command to set an environment variable for the App. In the TERMINAL window, run the following command (replace **HelloWorld** with the actual App name).

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/var/user/app/HelloWorld/lib/
export PYTHONPATH=$PYTHONPATH:/var/user/app/HelloWorld/lib/
```

After the third-party dependency library is installed for the specified App, you must configure a corresponding environment variable before debugging this App; otherwise, this App cannot run during

debugging. When multiple third-party dependency libraries are installed, you just need to add the environment variable once. After an SFTP connection is created again, you need to configure the environment variable again. When the App is started in MobiusPi, the environment variable of the third-party dependency library is automatically added to the App' s lib folder.

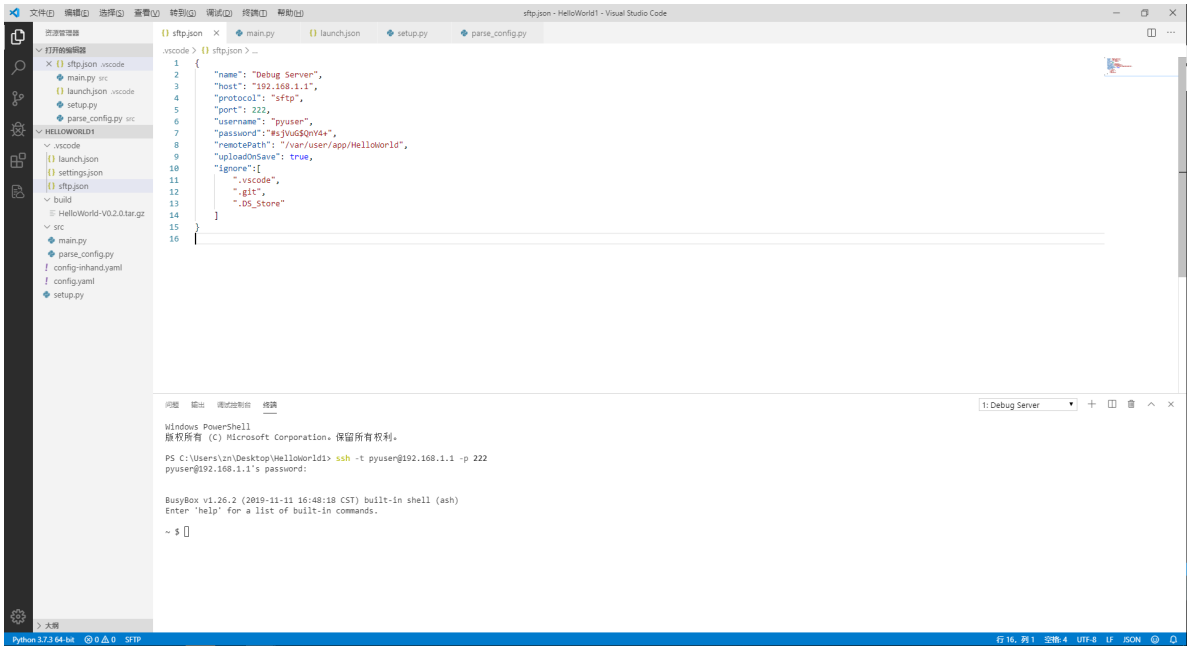
- Step 5: Run the code to check whether the App runs normally.



2.8.2 Install the third-party dependency library to SDK

To install a third-party dependency library to SDK, you need to enable the debugging mode for MobiusPi and connect to the Internet. The following takes installing the `xlrd` dependency library to SDK as an example to describe how to install a third-party dependency library to SDK.

- Step 1: Use VS Code to connect to MobiusPi over SFTP. For more information, see *Create an SFTP connection*.



- Step 2: Run `pip install` + **dependency library name** + **==version number** + `--user` and press Enter to install the dependency library to SDK. (If the version number is not added, after the pip command is run, the latest dependency library is automatically installed.)

```
pip install xlrd==1.2.0 --user
```

- Step 3: Then, the dependency library is automatically downloaded and installed. After the library is installed, the following figure is displayed:

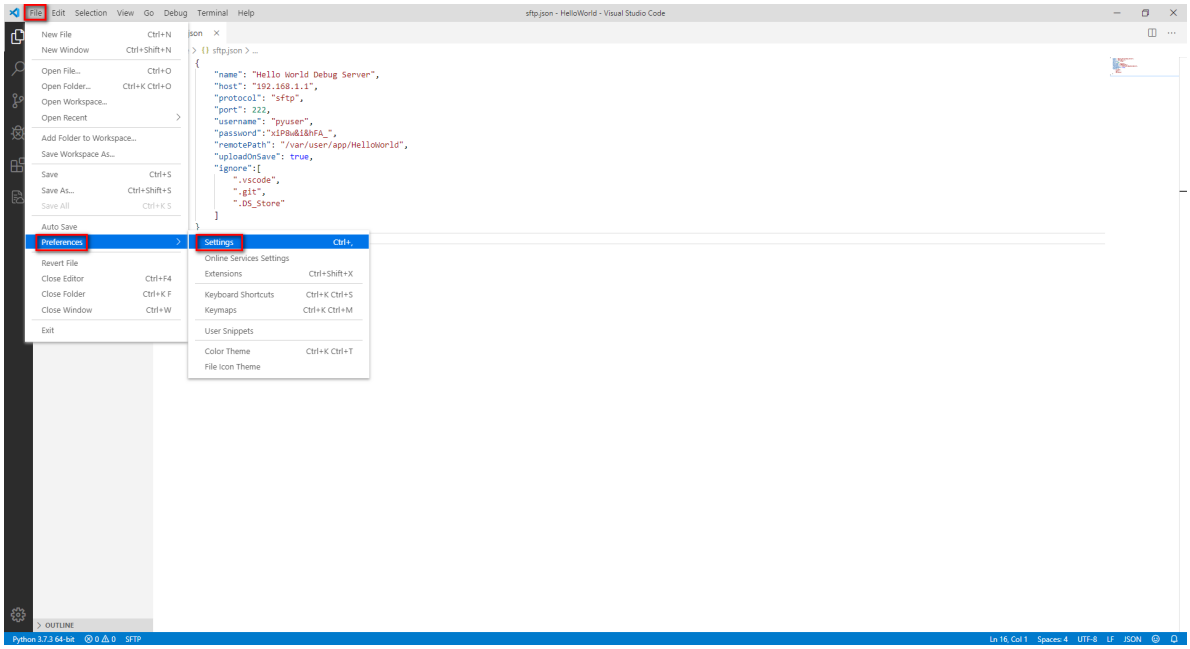
- Step 4: Run the code to check whether the App runs normally.

Note: After the third-party dependency library is installed in this way, if the App release package is imported to another MobiusPi project on which the third-party dependency library required for running this App is not installed to SDK, this App may cannot run.

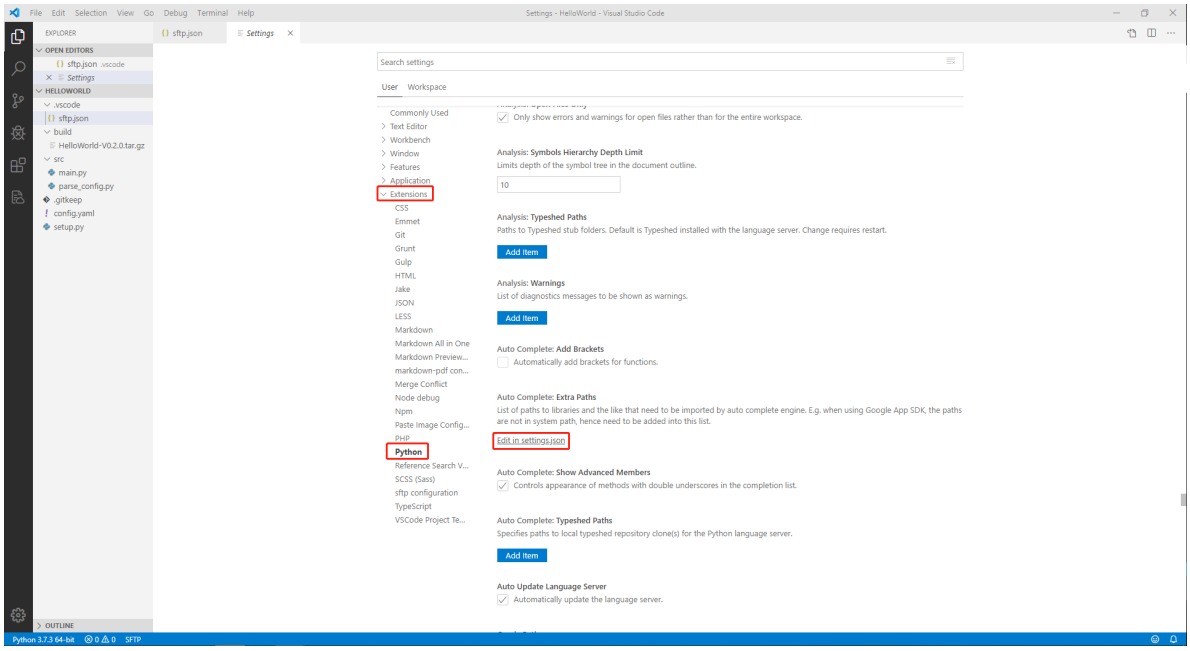
2.8.3 Enable automatic code completion

To improve the encoding efficiency, you can implement automatic code completion by using Python extensions.

- Step 1: Choose **Files** > **Preferences** > **Settings** to enter the Settings page.



- Step 2: On the Settings page, choose **Extensions > Python**, locate **Auto Complete: Extra Paths**, and then click **Edit in setting.json**.



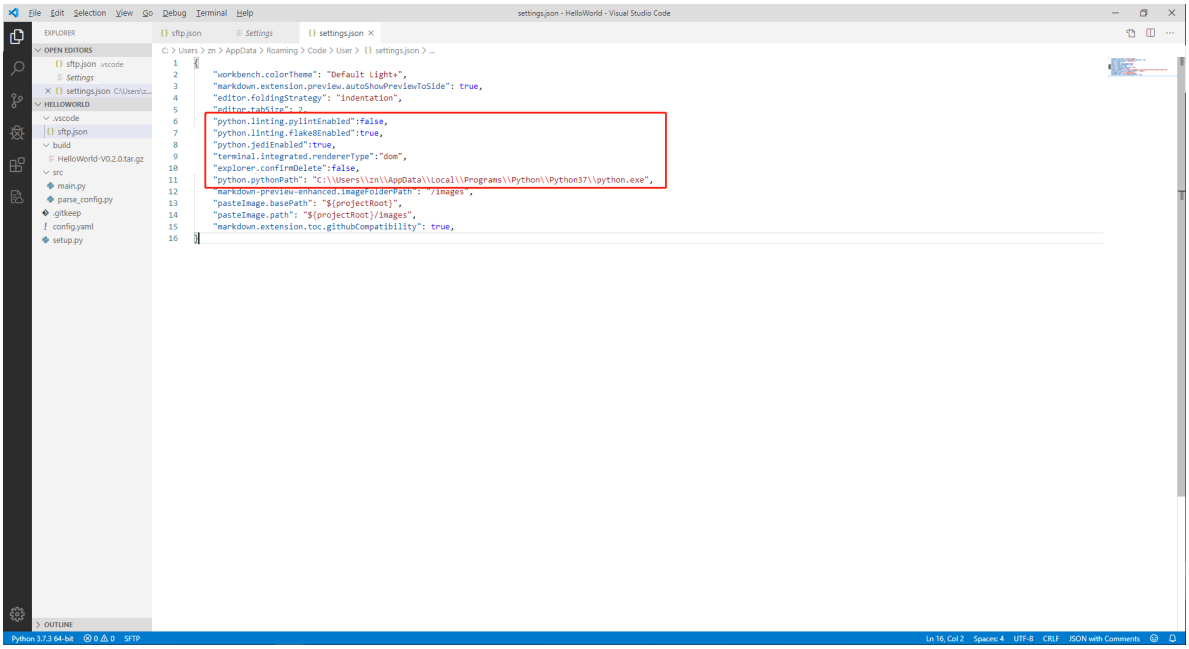
Add the following configuration items to **settings.json** and save the configuration (`python.pythonPath` is the installation path of the Python interpreter).

```
"python.linting.pylintEnabled":false,
"python.linting.flake8Enabled":true,
"python.jediEnabled":true,
"terminal.integrated.rendererType":"dom",
```

(continues on next page)

(continued from previous page)

```
"explorer.confirmDelete":false,
"python.pythonPath":"C:/Users/zn/AppData/Local/Programs/Python/Python37",
```

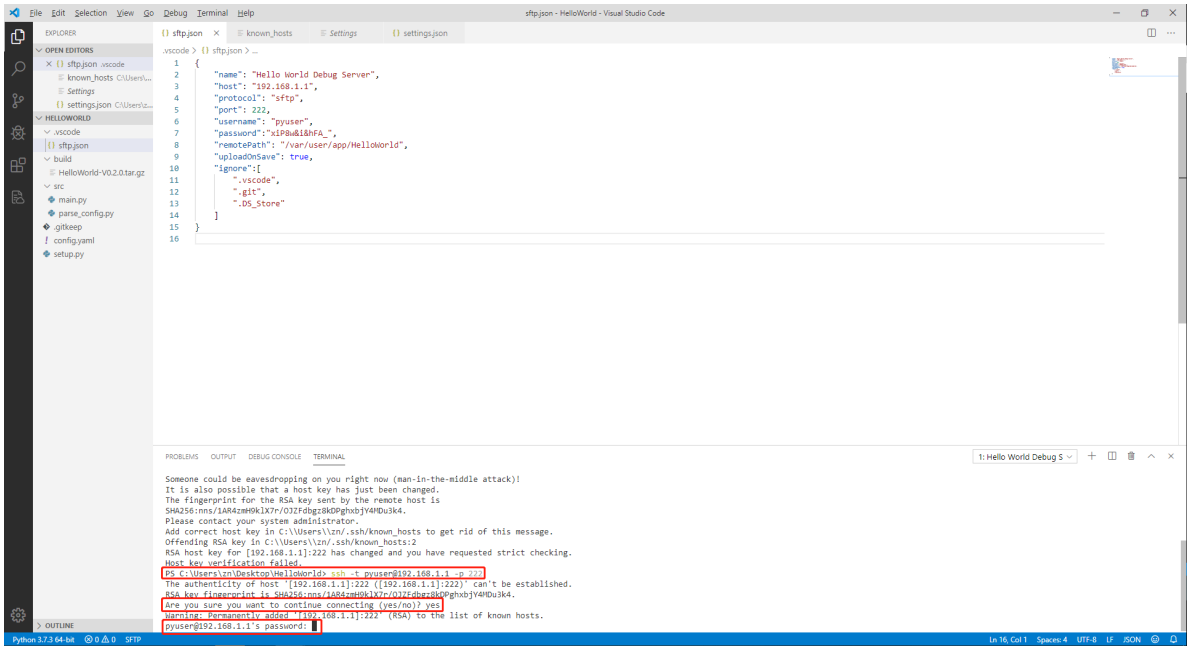


1.3.3 FAQ

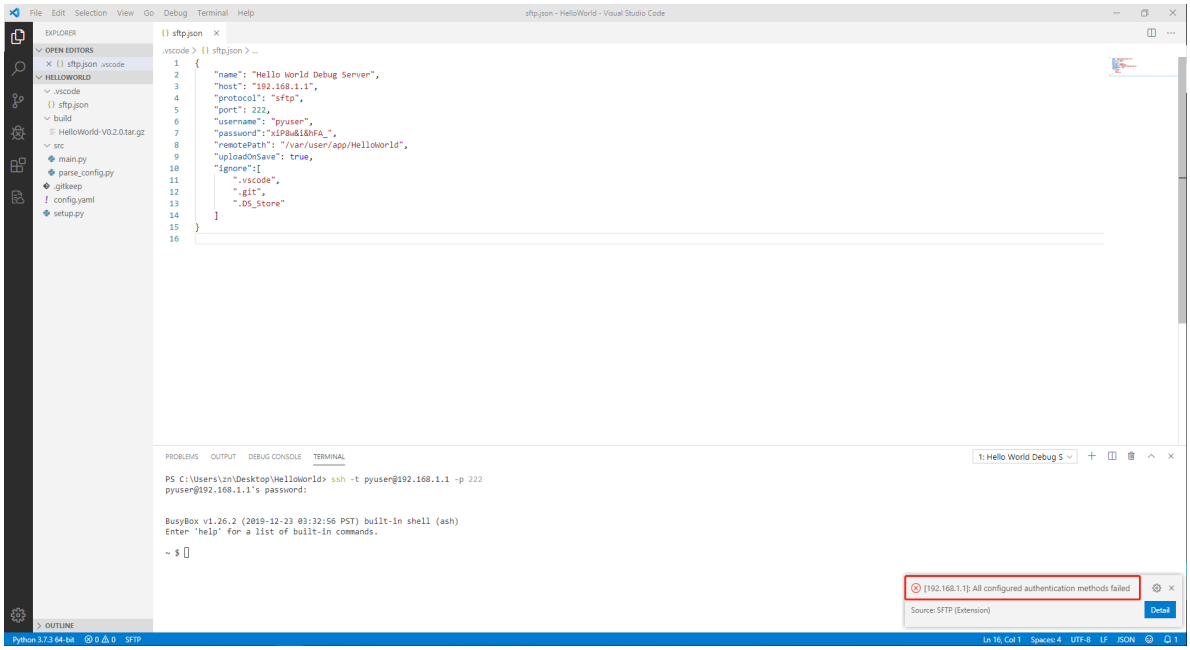
- *During SFTP connection building, it prompts "REMOTE HOST IDENTIFICATION HAS CHANGED and Host key verification failed"*
- *When synchronizing the code to the remote server, it prompts "All configured authentication methods failed"*
- *How to call the serial port and network port of IG902 during development*
- *When creating an SFTP connection with MobiusPi, it prompts "SSH error"*
- Q1: During SFTP connection building, it prompts "REMOTE HOST IDENTIFICATION HAS CHANGED and Host key verification failed" . What should I do?

  A1: The reason is that the MobiusPi key has been updated but the PC key does not, resulting in authentication failure. To solve the problem, just delete the row with key conflict from the key file. Press Ctrl and click the conflicting item to quickly access the link.

  After deletion, run the `>SFTP:Open SSH in Terminal` command. Then, an SFTP connection is created.



- Q2: After the SFTP connection is created, right-click on a blank area and choose **Sync Local > Remote** to synchronize the code to the remote server. It prompts "All configured authentication methods failed". What should I do?



A2: Ensure that the password in the **sftp.json** file is same to that on MobiusPi. After the passwords are the same, synchronize the code again.

- Q3: How to call the serial port and network port of IG902 during development?

A3: The RS485 serial port name is `/dev/ttyO3` and the RS232 serial port name is `/dev/ttyO1`. The serial port and network port can be called with the standard Python serial port/network port usage.

For example, use the `pyserial` library to call the serial port.

- Q4: When creating an SFTP connection with MobiusPi, it prompts "SSH error", as shown in the following figure. What should I do?

A4: Install OpenSSH to support the SSH protocol. Download OpenSSH from https://www.openssh.com.

1.4 MobiusPi API 手册

- 概述
- 安装
- *Python* 要求
- *1. cellular*
  - *Getting Started*
  - *Cellular*
  - *Option functions*
- *2. serial*
  - *Getting Started*
  - *Serial*
  - *Option functions*

1.4.1 概述

本文档描述了 `mobiuspi_lib` 库的源代码，该库实现了获取 MobiusPi 的运行状态和调用 MobiusPi 物理接口等功能。

1.4.2 安装

映翰通提供包含 `mobiuspi_lib` 库的软件开发工具包（SDK），如需获取 MobiusPi 的 SDK 及其功能特性信息，请联系客服。安装和升级 SDK 请参考IG902 更新软件版本。

1.4.3 Python 要求

MobiusPi Python SDK 适用于 Python 3.7 和 3.8，如果您使用其他版本的 Python，则可能导致代码运行异常。您可以打开命令提示符或启动 python IDE，使用 `python` 命令确认您的 Python 版本。本文档假设您使

用了 Python 3.7 和 3.8。

1.4.4 1. cellular

Getting Started

Here is a very simple example that shows how to get cellular info of device:

```
# import Cellular
from mobiuspi_lib.cellular import Cellular

# build Cellular instance
cellular = Cellular()

# get modem info
modem = cellular.get_modem()
print("get modem: %s" % modem)
```

Cellular

You can use the Cellular class as an instance, within a class or by subclass. The general usage flow is as follows:

- Create a Cellular instance
- Use `get_modem()` to get model info
- Use `get_network()` to get network info

Option functions

get_modem()

Request parameter

None

Description

You can use this function to get system info.

Returns

- Return type

  dict

- Return value


```
{
    "active_sim": "SIM 1",
    "imei_code": "811622048741556",
    "imsi_code": "411220441893359",
    "iccid_code": "84463317227780999882",
    "phone_number": "+8611162203133",
    "signal_level": 0,
    "dbm": 113,
    "rerp": 0,
    "rerq": 0,
    "register_status": 0,
    "operator": "CHN-CT",
    "apns": "",
    "network_type": "4G",
    "lac": "BB00",
    "cell_id": "DD788B81"
}
```


- Return structure
  - active_sim(string)：当前 SIM 卡
  - imei_code(string)：IMEI 号码
  - imsi_code(string)：IMSI 号码
  - iccid_code(string)：ICCID 号码
  - phone_number(string)：电话号码
  - signal_level(int)：信号值
  - dbm(int)：dBm 值
  - rerp(int)：RSRP，预留参数
  - rerq(int)：RSRQ，预留参数
  - register_status(int)：注册状态
    - 0：正在注册到网络

    * 1：注册网络成功
    * 5：注册网络成功，漫游状态
    * 6：尚未注册到网络
    * 7：未注册
  - operator(string)：运营商
  - apns(string)：APN，预留参数
  - network_type(string)：网络类型
  - lac(string)：位置区码
  - cell_id(string)：小区 ID

get_network()

Request parameter

None

Description

You can use this function to get network info.

Returns

- Return type

  list

- Return value

```
[
{
    'status': 0,
    'ip_addr': '0.0.0.0',
    'netmask': '0.0.0.0',
    'gateway': '0.0.0.0',
    'dns': '0.0.0.0',
    'mtu': 1200,
    'connect_time': 0
}]
```

- Return structure
  - status(int)：网络状态
    - 0：未连接
    - 1：已连接
  - ip_addr(string)：IP 地址
  - netmask(string)：子网源码
  - gateway(string)：网关
  - dns(string)：DNS
  - mtu(int)：MTU
  - connect_time(int)：连接时间，单位为秒

get_signal_level()

Request parameter

None

Description

You can use this function to get signal level info.

Returns

- Return type

  int

- Return value

```
28
```

get_dbm()

Request parameter

None

Description

You can use this function to get dbm info.

Returns

- Return type

  int

- Return value

```
57
```

get_active_sim()

Request parameter

None

Description

You can use this function to get active sim info.

Returns

- Return type

  str

- Return value

```
SIM 1
```

get_imei_code()

Request parameter

None

Description

You can use this function to get imei info.

Returns

- Return type

  str

- Return value

```
811622048741556
```

get_imsi_code()

Request parameter

None

Description

You can use this function to get imsi info.

Returns

- Return type

  str

- Return value

```
411220441893359
```

get_iccid_code()

Request parameter

None

Description

You can use this function to get iccid info.

Returns

- Return type

  str

- Return value

```
84463317227780999882
```

get_phone_number()

Request parameter

None

Description

You can use this function to get phone number info.

Returns

- Return type

  str

- Return value

```
+8611162203133
```

get_rerp()

Request parameter

None

Description

You can use this function to get rerp info.

Returns

- Return type

  int

- Return value

```
0
```

get_rerq()

Request parameter

None

Description

You can use this function to get rerq info.

Returns

- Return type

  int

- Return value

```
0
```

get_register_status()

Request parameter

None

Description

You can use this function to get register status info.

Returns

- Return type

  int

- Return value

```
0
```

- Return structure
  - 0：正在注册到网络
  - 1：注册网络成功
  - 5：注册网络成功，漫游状态
  - 6：尚未注册到网络
  - 7：未注册

get_operator()

Request parameter

None

Description

You can use this function to get operator info.

Returns

- Return type

  str

- Return value

```
CHN-CT
```

get_apns()

Request parameter

None

Description

You can use this function to get apns info

Returns

- Return type

  str

- Return value

```
```

get_network_type()

Request parameter

None

Description

You can use this function to get network type info.

Returns

- Return type

  str

- Return value

```
4G
```

get_lac()

Request parameter

None

Description

You can use this function to get lac info.

Returns

- Return type

  str

- Return value

```
BB00
```

get_cell_id()

Request parameter

None

Description

You can use this function to get cell id info.

Returns

- Return type

  str

- Return value

```
DD788B81
```

1.4.5 2. serial

Getting Started

Here is a very simple example that shows how to get the path of the 232/285 serial:

```
from mobiuspi_lib.serial import Serial

# you can use serial as an instance of Serial class
serial = Serial()

# get 232 path
path_232 = serial.get_serial232_path()
print("232 path: %s" % path_232 )

# get 485 path
path_485 = serial.get_serial485_path()
print("485 path: %s" % path_485
```

Serial

You can use the Serial class as an instance, within a class or by subclassing. The general usage flow is as follows:

- Create a Serial instance
- Use `get_serial232_path()` to get 232 path
- Use `get_serial485_path()` to get 484 path

Option functions

get_serial232_path()

Request parameter

None

Description

You can use this function to get 232 path.

Returns

- Return type

  str

- Return value

```
/dev/ttyO1
```

- Return structure
  - /dev/ttyO5：IG501 时返回此数值
  - /dev/ttyO1：IG902 时返回此数值

get_serial485_path()

Request parameter

None

Description

You can use this function to get 485 path.

Returns

- Return type

  str

- Return value

```
/dev/ttyO3
```

- Return structure
  - /dev/ttyO1：IG501 时返回此数值
  - /dev/ttyO3：IG902 时返回此数值

1.5 Modbus Example

- 概述

- 先决条件
- 环境准备
  - 配置 *Modbus PLC*
  - 配置开发环境
- 开始测试

1.5.1 概述

Modbus 是 master/slave 架构的串行通信协议，允许多个设备连接在同一个网络上进行通信。Modbus 已经成为工业领域通信协议事实上的业界标准，并且现在是工业电子设备之间常用的连接方式。Modbus 比其他通信协议使用的更广泛的主要原因有：

1. 公开发表并且无著作权要求
2. 易于部署和维护
3. 对供应商来说，修改移动本地的比特或字节没有很多限制

映翰通提供 `modbus` 示例以便于客户基于 InGateway 二次开发实现 modbus 数据采集。该示例主要基于 `modbus_tk` 实现了 InGateway 作为 Modbus master 通过 Modbus TCP 和 Modbus RTU 协议读写 Modbus slave 的 01、02、03 和 04 功能码的 Modbus 变量，支持的变量类型包括 bit、word、string 等数据类型。`modbus_tk` 的详细使用方法请访问modbus_tk。注意：请勿同时使用 `device_supervisor` 和 `demo` 示例，否则可能导致代码运行异常。

1.5.2 先决条件

在进行开发和测试前，你需要具备以下条件：

- InGateway
  - 固件版本：`2.0.0.r12513` 及以上（请联系客服获取）
  - SDK 版本：`1.3.5` 及以上（请联系客服获取）
- VS Code 软件
- Modbus PLC（本文档使用施耐德的 `TM241` PLC）及其编程软件（`TM241` 的编程软件为 `SoMachine`）

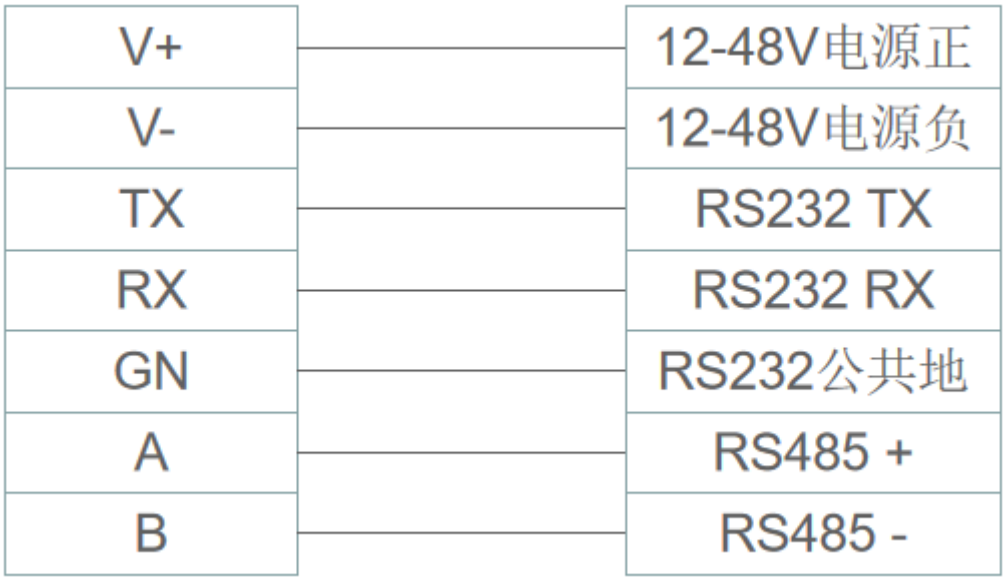
1.5.3 环境准备

- 配置 *Modbus PLC*
- 配置开发环境

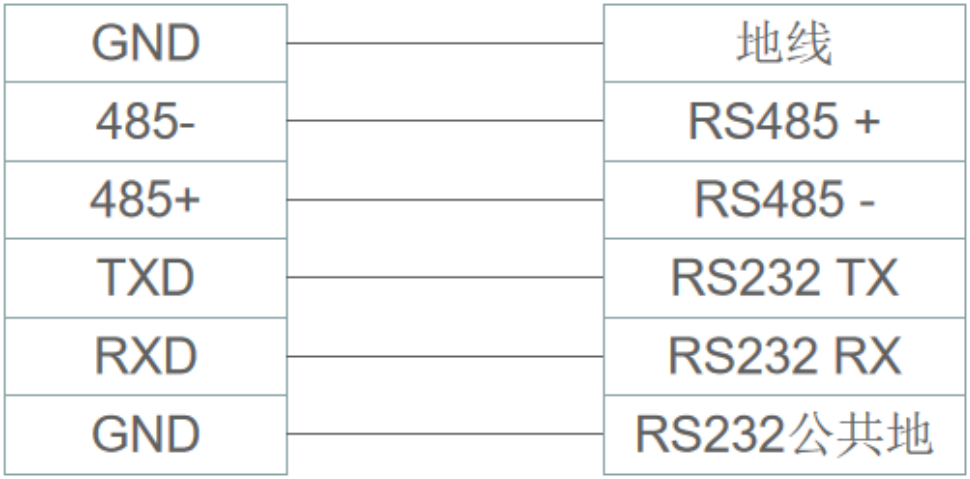
配置 Modbus PLC

如果你已经配置了相应的 Modbus slave，可以跳过这一小节。以下将说明如何配置 `TM241` 作为 Modbus slave 以便于后续测试验证。

- 步骤 1：建立 InGateway 与 `TM241` 连接
  - Modbus TCP：使用以太网线连接 `TM241` 与 InGateway 设置 `TM241` 与 InGateway 处于同一网段（IG501 的 FE 0/1 口的默认 ip 地址为 `192.168.1.1`；IG902 的 GE 0/2 口的默认 ip 地址为 `192.168.2.1`），本文档配置 `TM241` 的 IP 地址为 `10.5.16.155`。
  - Modbus RTU：使用串口线连接 `TM241` 与 InGatewayIG902 串口端子接线说明如下图：



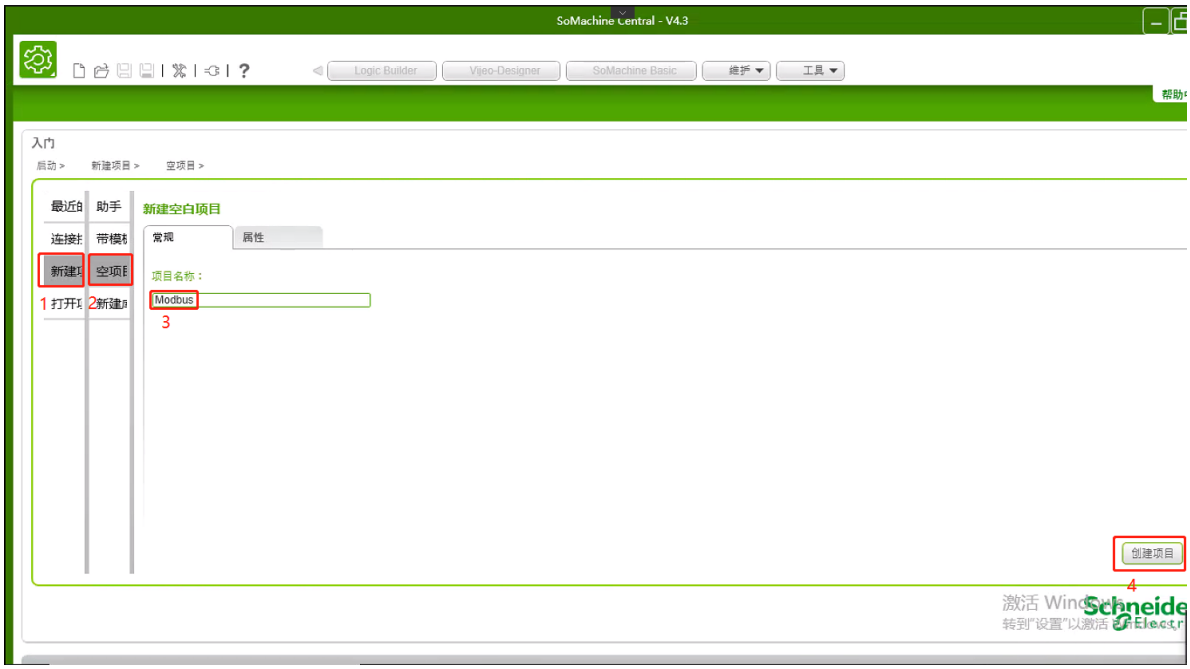
IG501 串口端子接线说明如下图：



TM241 串口端子接线说明请通过 PLC 说明文档获取。

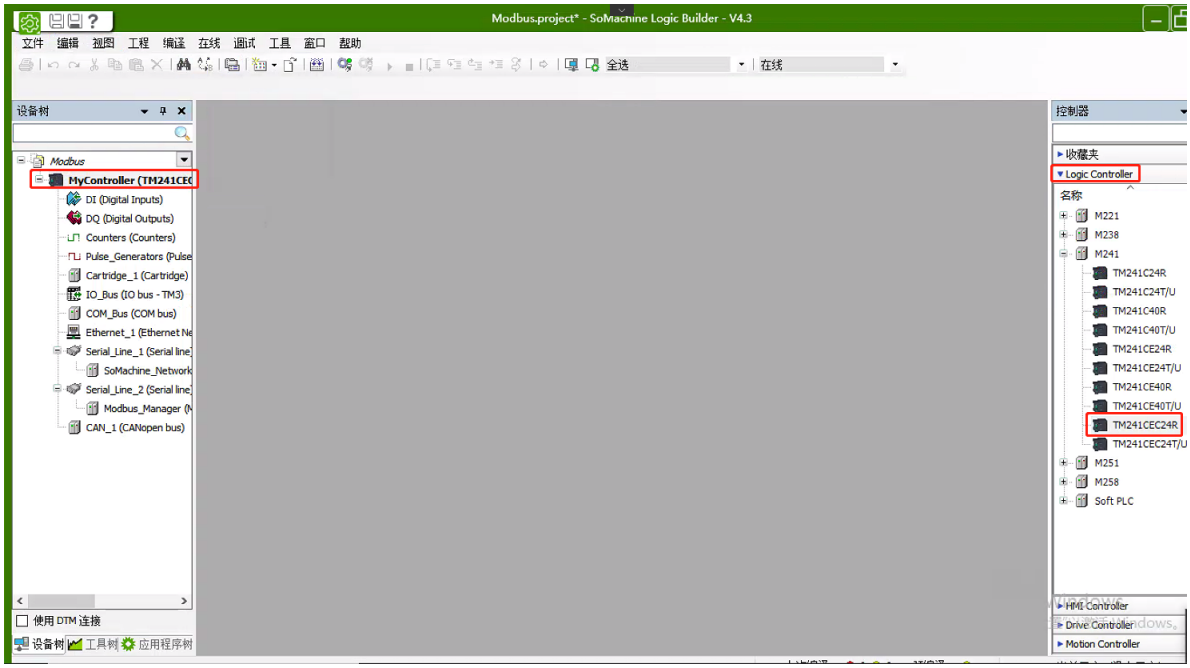
- 步骤 2：配置 TM241

  打开 SoMachine 软件，并点击“新建项目 > 空项目”以创建一个新的项目。

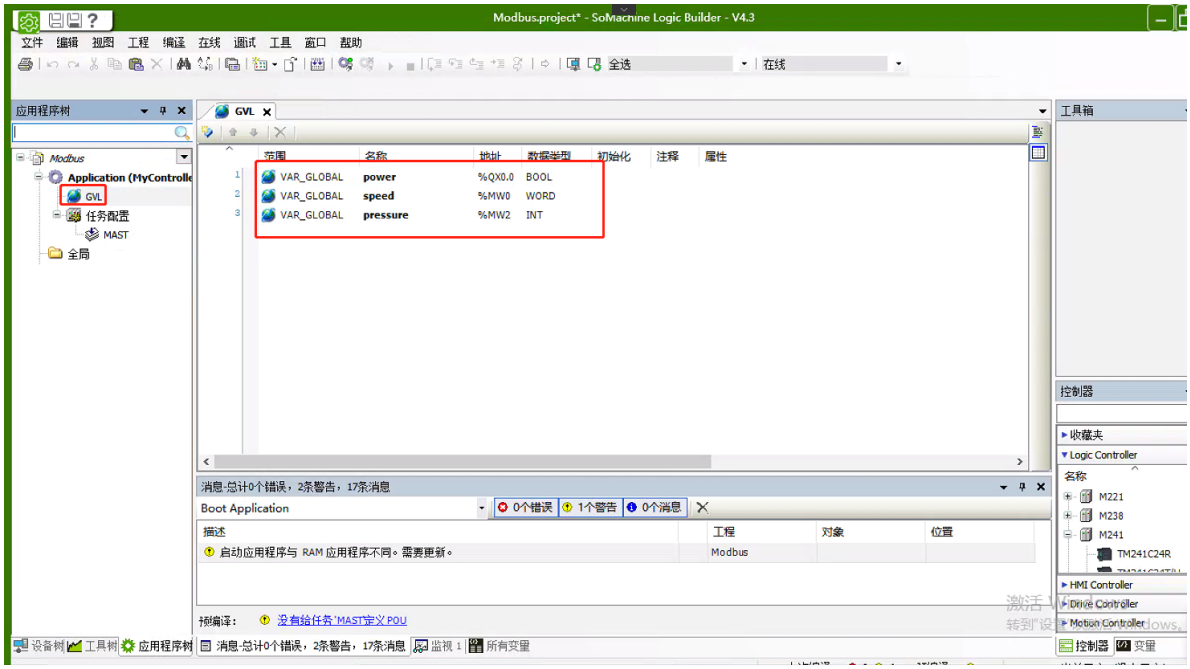


随后双击“编程一个或多个控制器”，进入项目视图。

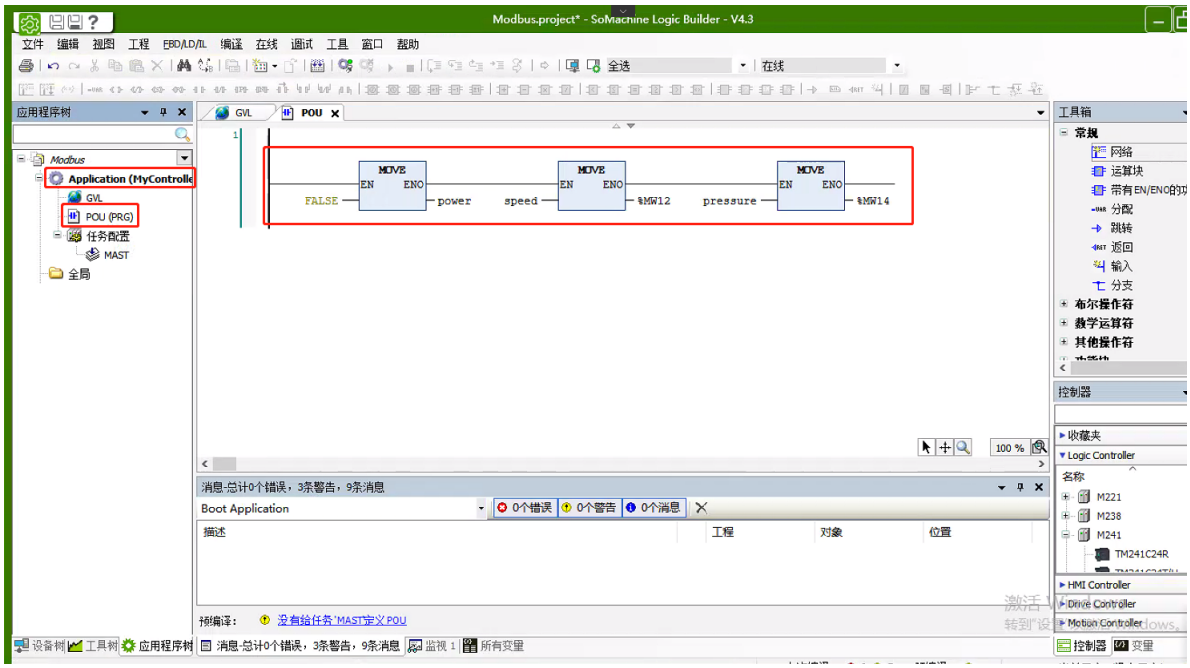
选择相应的 PLC 并添加至“设备树”中。如下图所示：



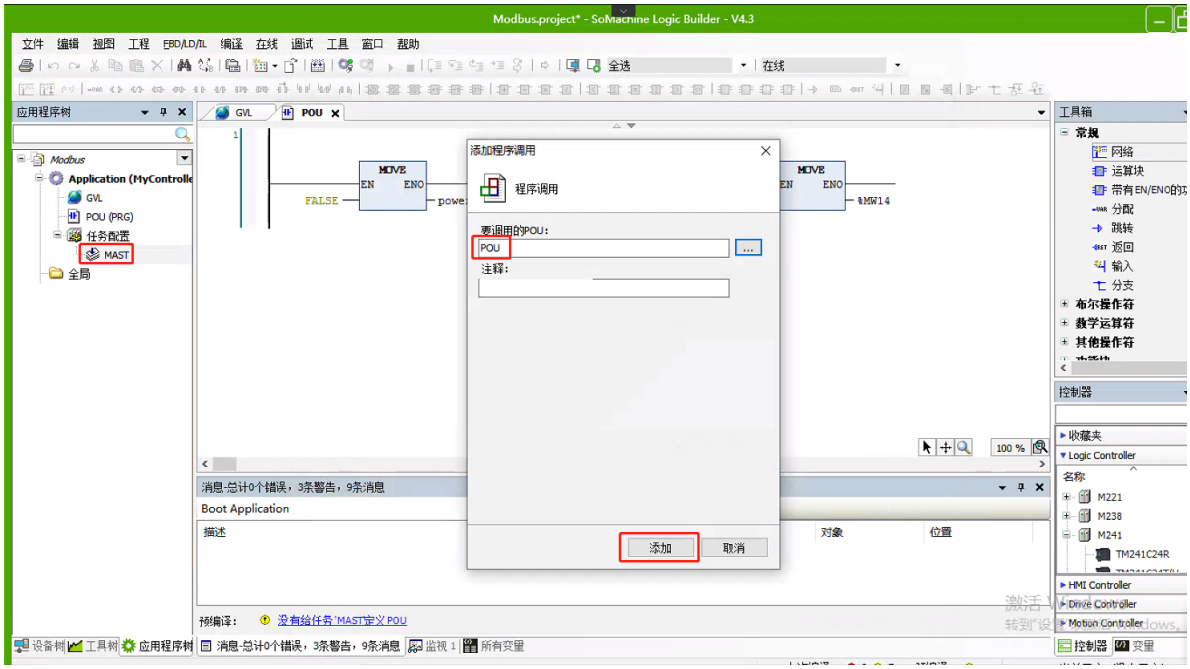
在“应用程序树”的“GVL”中添加如下变量：（`%QX0.0` 对应 modbus 地址为 `1`；`%MW0` 对应 modbus 地址为 `40001`；`%MW2` 对应 modbus 地址为 `40003`）



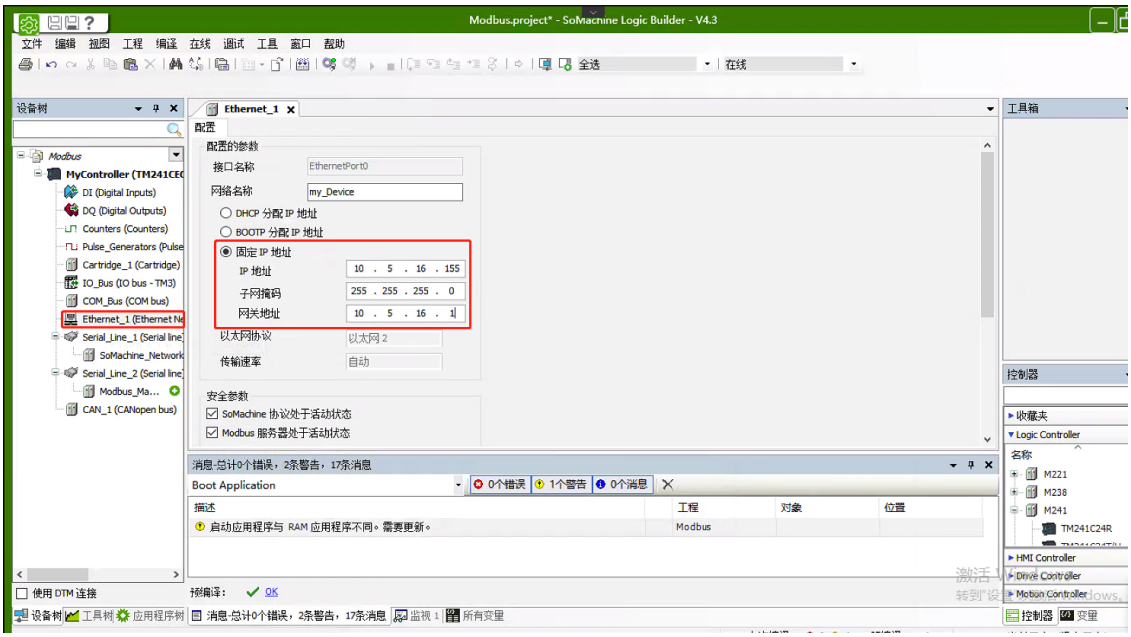
右击“Application”，选择“添加对象 > POU”。添加一个“POU”对象，并在“POU”中配置简单的赋值规则。



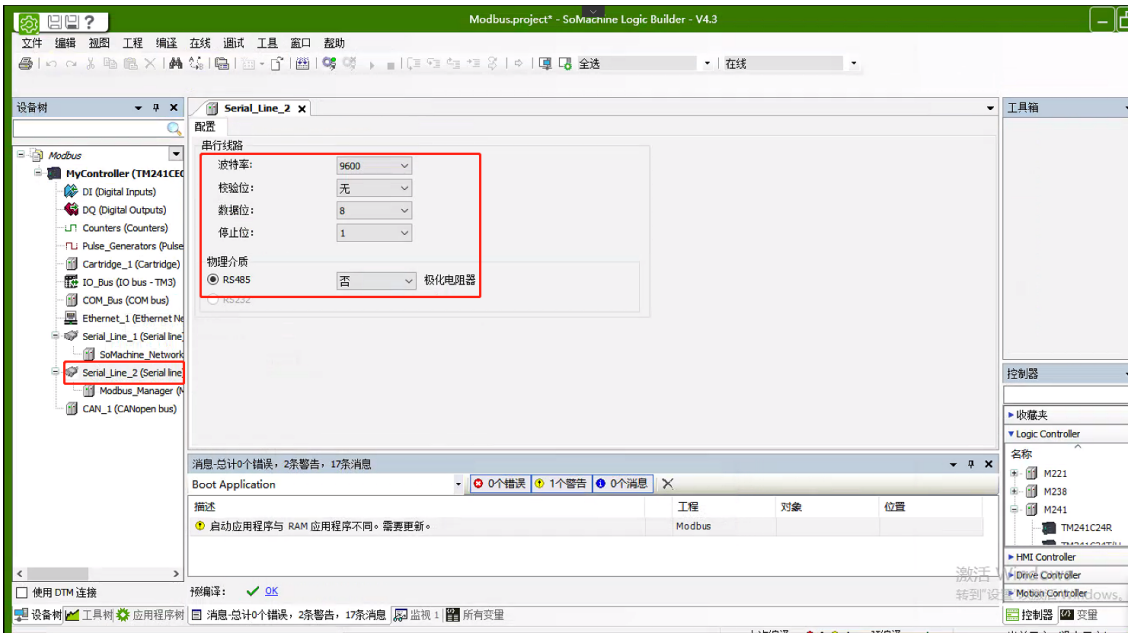
右击“MAST”，选择“添加对象 > 程序调用”。添加上一步中创建的“POU”对象并点击“添加”。



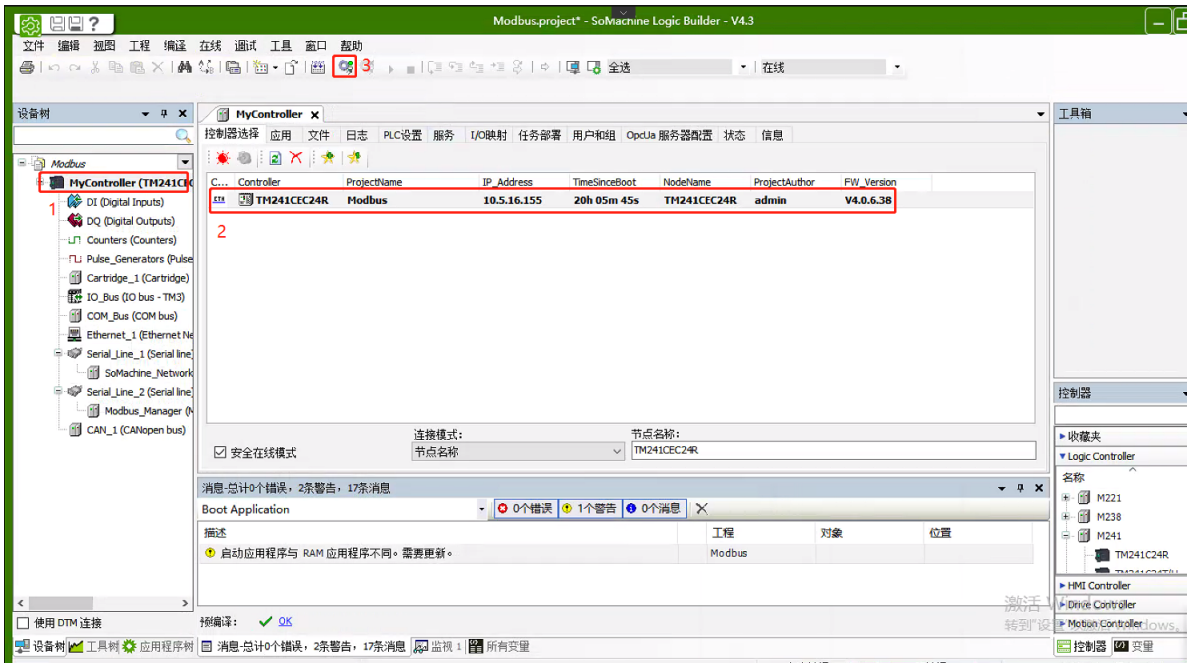
– Modbus TCP 使用 PLC 进行 Modbus TCP 通讯时，在“设备树”的” Ethernet_1”中设置 PLC 的 IP 地址（本 demo 中为 `10.5.16.155`）。



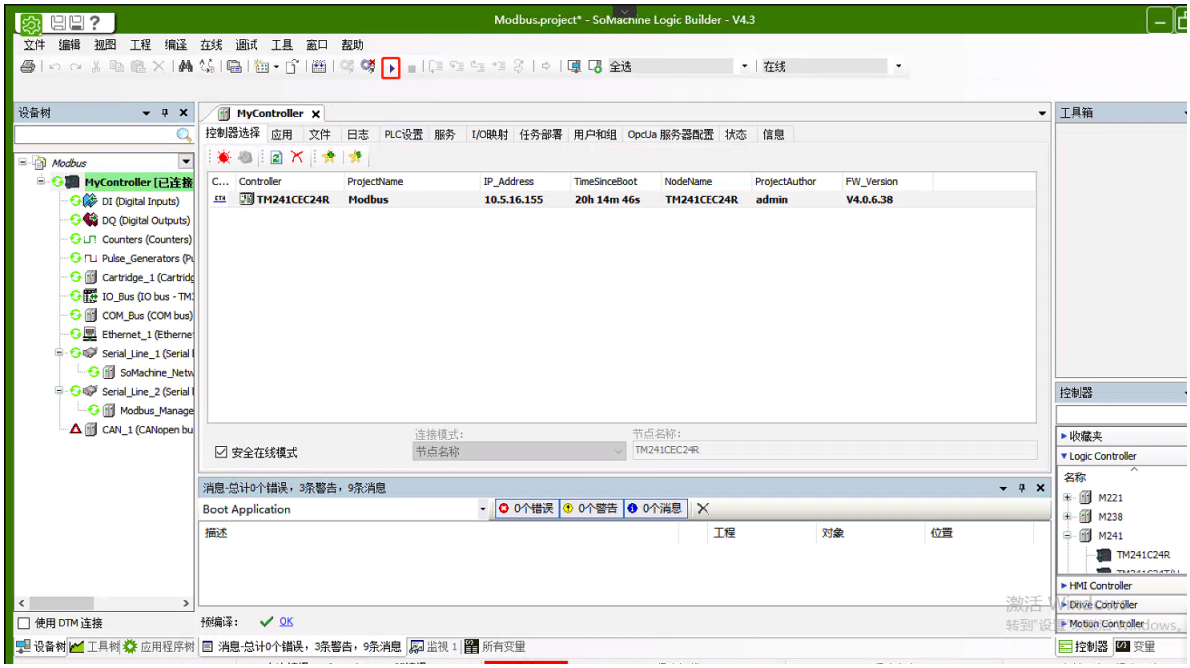
– Modbus RTU 使用 PLC 进行 Modbus RTU 通讯时，选择“设备树”中相应的串口并设置串口通讯参数（本 demo 为 `Serial_Line_2`，使用 RS485 串口通讯）。串口通讯参数需与 `modbus_example.py` 中的 modbus 参数一致，见*修改 Modbus slave 参数*。



配置完毕后双击 PLC，在控制器页面中选中需要更新程序的 PLC 并点击“登录”以下载程序。



下载成功后点击“开始”使 PLC 开始运行。



配置开发环境

- *InGateway* 配置
- 建立项目文件夹
- InGateway 配置设备联网、软件更新、IDE 软件获取等基础的配置操作请查看MobiusPi Python Development Quick Start。以下操作我们将假设你已经完成了 InGateway 的软件更新、设备联网、开启调

试模式等配置。

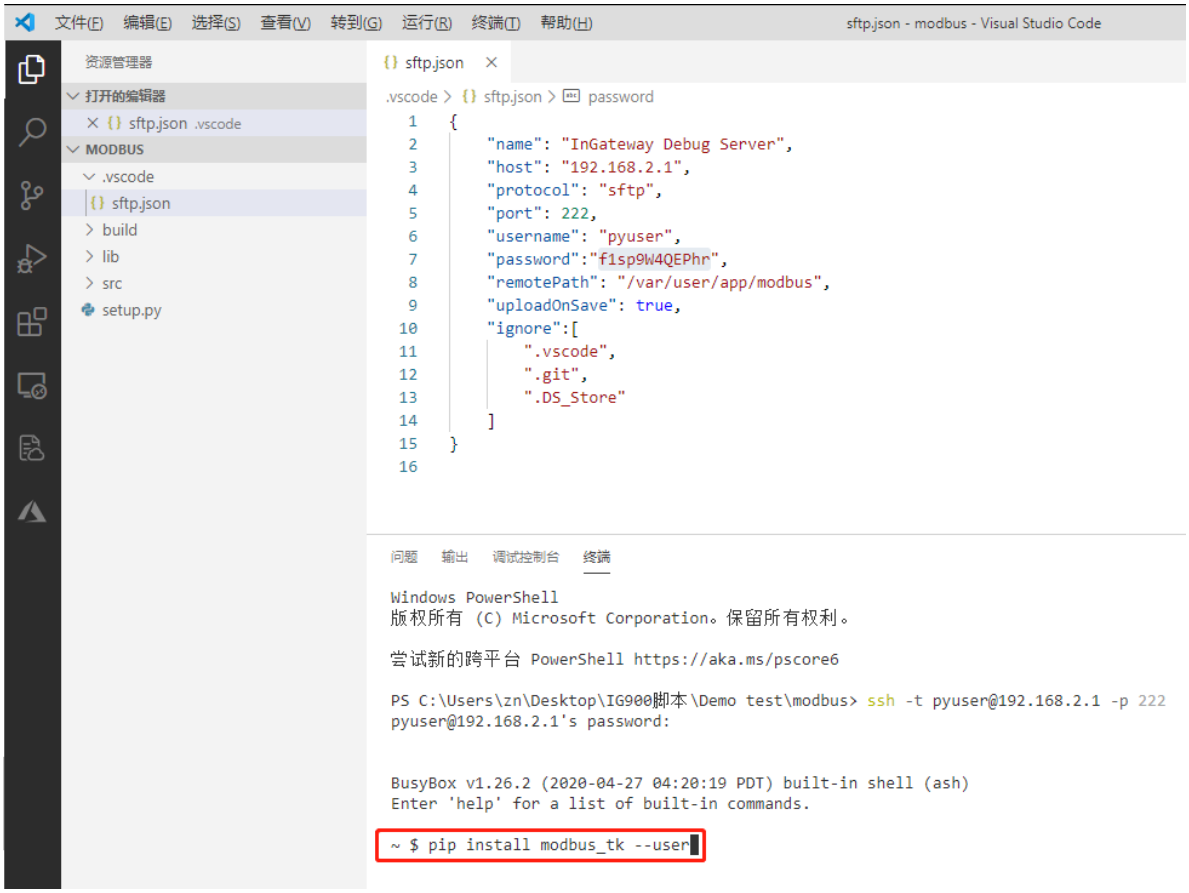
- 建立项目文件夹建立一个名为“modbus”文件夹作为项目文件夹，最终项目文件夹的结构如下：

```
.vscode
  sftp.json
build
lib
src
  main.py
  modbus_example.py
setup.py
```

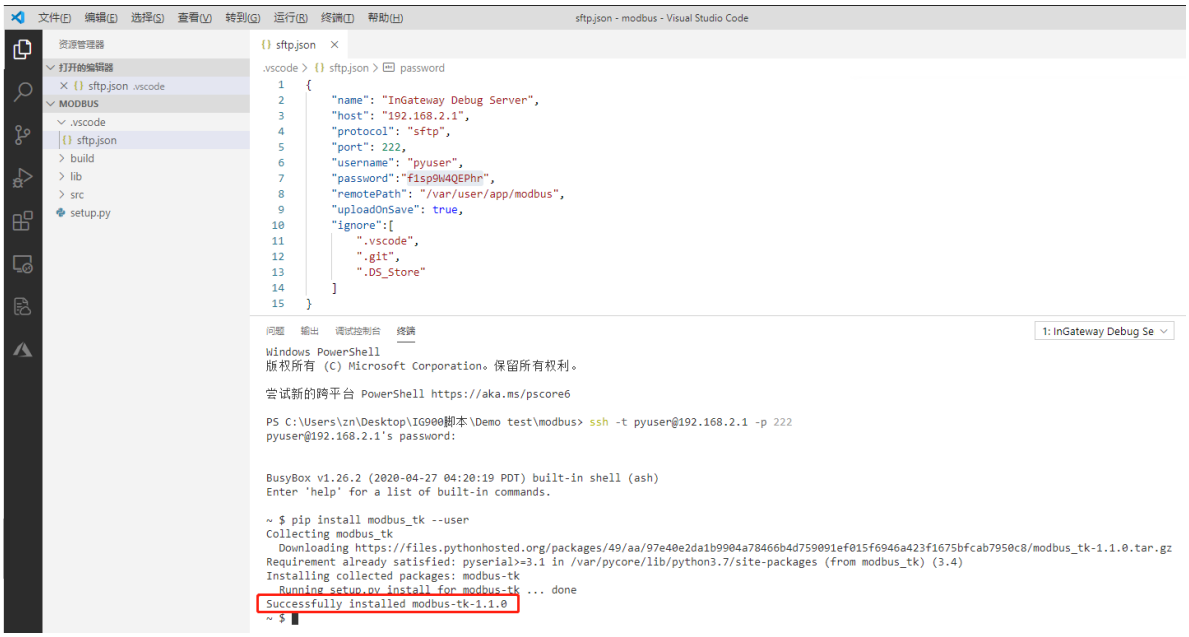
  - `.vscode`：VS Code 配置文件夹
    - `sftp.json`：与 InGateway 建立 SFTP 连接所需的 SFTP 配置文件。
  - `build`：App 发布包文件夹。
  - `lib`：App 第三方依赖库文件夹。
  - `src`：App 源码文件夹
    - `main.py`：App 入口。
    - `modbus_example.py`：主要基于 `modbus_tk` 实现了 InGateway 作为 Modbus master 通过 Modbus TCP 和 Modbus RTU 协议读写 Modbus slave 的 01、02、03 和 04 功能码的 Modbus 变量。
  - `setup.py`：App 说明文件。

1.5.4 开始测试

- *安装 modbus_tk*
- *修改 Modbus slave 参数*
- *调试代码*
- *核对数据读写*
- 步骤 1：安装 `modbus_tk` 建立与 InGateway 的 SFTP 连接，操作步骤见建立 SFTP 连接。SFTP 连接建立成功后在终端执行 `pip install modbus_tk --user` 安装 `modbus_tk` 依赖库。



安装成功后如下图所示：

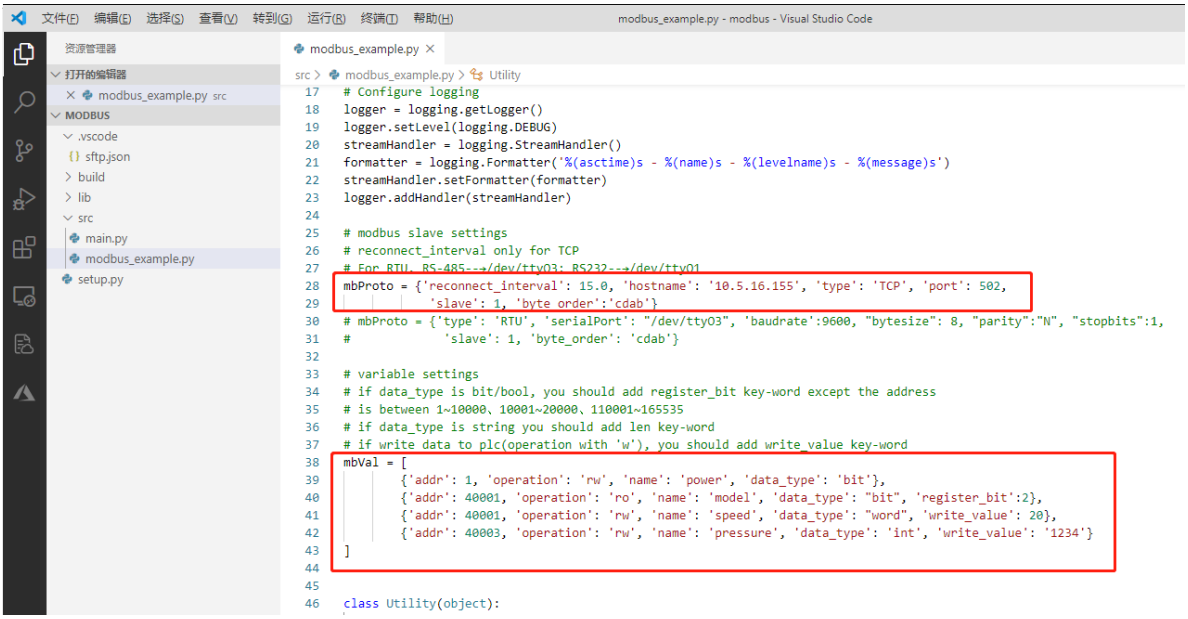


- 步骤 2：修改 Modbus slave 参数本 demo 的默认配置为使用 `Modbus TCP` 协议采集 IP 地址为 `10.5.16.155`，端口号为 `502`，从站地址为 `1`，字节序为 `cdab`，超时时间 `15` 秒的 Modbus slave 中的以下

变量数据：

1. 读取寄存器地址为 1 的 bit 类型的 modbus 数据，数据名称为 power
2. 读取寄存器地址为 40001 的 bit 类型的 modbus 数据的第三位，数据名称为 model
3. 读取寄存器地址为 40001 的 word 类型的 modbus 数据并写入数值 20，数据名称为 speed
4. 读取寄存器地址为 40003 的 int 类型的 modbus 数据并写入数值 1234，数据名称为 pressure

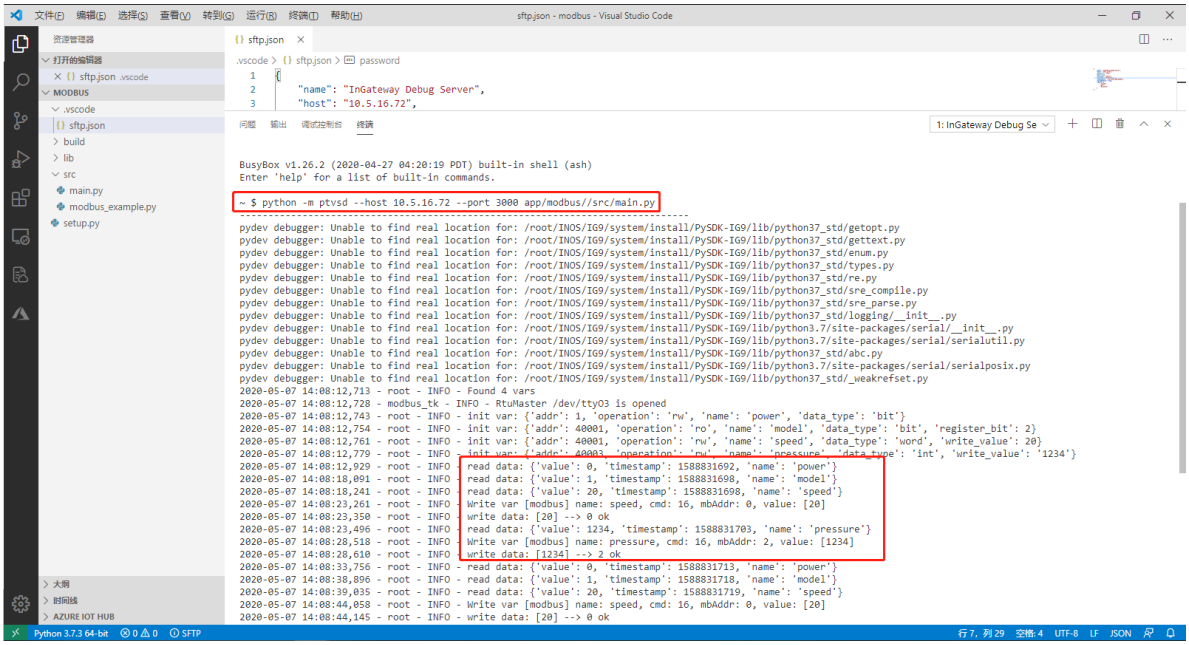
如果你的 PLC 上中配置了以上 modbus 地址变量，则可以直接使用 demo 代码。否则请在 modbus_example.py 中根据你的实际情况调整以下配置：



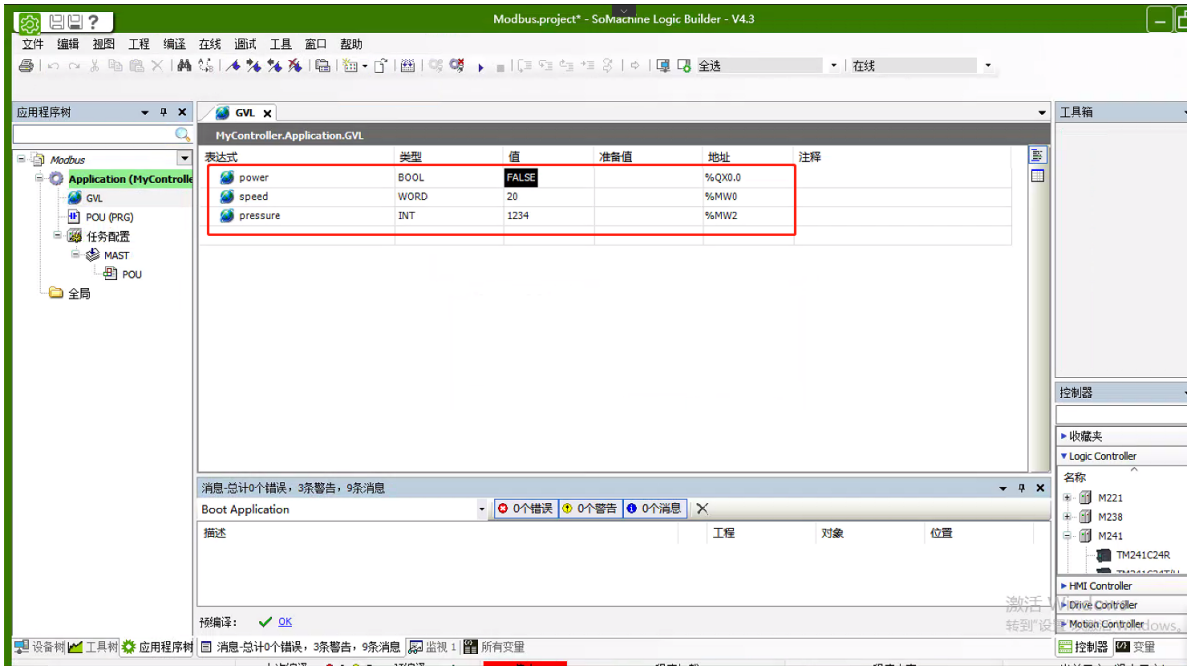
- mbProto（Modbus TCP）
  - reconnect_interval：超时时间
  - hostname：Modbus 模拟器或 PLC 的 ip 地址
  - type：通讯类型，以太网通讯时为 TCP
  - port：通讯端口号
  - slave：站地址
  - byte_order：字节序，包括 abcd、badc、cdab、dcba 四种
- mbProto（Modbus RTU）
  - type：通讯类型，串口通讯时为 RTU
  - serialPort：串口号，使用 RS485 串口时为/dev/tty03；使用 RS232 串口时为/dev/ttyO1
  - baudrate：波特率
  - bytesize：数据位

    * parity：校验位，无校验为 N 、偶校验为 E、奇校验为 O
    * stopbits：停止位
    * slave：站地址
    * byte_order：字节序，包括 abcd、badc、cdab、dcba 四种
  - mbVal
    * addr：modbus 寄存器地址
    * operation：可读可写 rw、只读 ro、只写 wo
    * len：读写的数据长度，仅对 string 数据类型有效
    * name：数据名称
    * data_type：数据类型
    * register_bit：03 和 04 功能码数据的数据类型为 bit 或 bool 时，通过此项参数设置读取寄存器地址的哪一位。可配置 0-15 中的任意一位
    * write_value：当数据可写时，写入该 modbus 寄存器地址的数值

- 步骤 3：调试代码如何使用 VS Code 调试代码请参考调试代码。main.py 运行结果如下图所示：



- 步骤 4：核对数据读写在 SoMachine 软件中登录相应 PLC 后，进入“应用程序树”的“GVL”页面，查看各变量数值，与代码读取到的数据一致。



至此，完成了在 InGateway 上实现 modbus 数据采集。

Volume resources Files or directories on the root file system (except under `/sys`, `/dev`, or `/var`). These include:

- Folders or files used to read or write information across Greengrass Lambda functions (for example, `/usr/lib/python2.x/site-packages/local`).
- Folders or files under the host' s /proc file system (for example, `/proc/net` or `/proc/stat`). Supported in v1.6 or later. For additional requirements, see *Volume resources under the /proc directory*. To configure the `/var`, `/var/run`, and `/var/lib` directories as volume resources, first mount the directory in a different folder and then configure the folder as a volume resource. When you configure volume resources, you specify a *source* path and a *destination* path. The source path is the absolute path of the resource on the host. The destination path is the absolute path of the resource inside the Lambda namespace environment. This is the container that a Greengrass Lambda function or connector runs in. Any changes to the destination path are reflected in the source path on the host file system.Files in the destination path are visible in the Lambda namespace only. You can' t see them in a regular Linux namespace.

Device resources Files under `/dev`. Only character devices or block devices under `/dev` are allowed for device resources. These include:

- Serial ports used to communicate with devices connected through serial ports (for example, `/dev/ttyS0`, `/dev/ttyS1`).
- USB used to connect USB peripherals (for example, `/dev/ttyUSB0` or `/dev/bus/usb`).
- GPIOs used for sensors and actuators through GPIO (for example, `/dev/gpiomem`).
- GPUs used to accelerate machine learning using on-board GPUs (for example, `/dev/nvidia0`).

- Cameras used to capture images and videos (for example, `/dev/video0`). `/dev/shm` is an exception. It can be configured as a volume resource only. Resources under `/dev/shm` must be granted `rw` permission.

AWS IoT Greengrass also supports resource types that are used to perform machine learning inference. For more information, see Perform machine learning inference.

1.6 Greengrass Discovery RESTful API

All devices that communicate with an AWS IoT Greengrass core must be a member of a Greengrass group. Each group must have a Greengrass core. The Discovery API enables devices to retrieve information required to connect to a Greengrass core that is in the same Greengrass group as the device. When a device first comes online, it can connect to the AWS IoT Greengrass service and use the Discovery API to find:

- The group to which it belongs. A device can be a member of up to 10 groups.
- The IP address and port for the Greengrass core in the group.
- The group CA certificate, which can be used to authenticate the Greengrass core device.

NoteDevices can also use the AWS IoT Device SDKs to discover connectivity information for a Greengrass core. For more information, see AWS IoT Device SDK.

To use this API, send HTTP requests to the Discovery API endpoint. For example:

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```