



SF4202 JsonDom

作者	罗梦溪	<div> 中科时代 基于PC技术的工智机新时代</div> <div>深圳市南山区粤海街道百度国际大厦西塔楼</div> <div>官网：www.sinsegye.com.cn</div> <div>邮箱：Sales@sinsegye.com.cn</div> <div>热线电话：400-013-2158</div>
日期	2025.08.20	
版本	V1.0	
Email	Luomengxi@sinsegye.com.cn	

更新说明：

本文记录了<产品名称>的变更情况。

<2025.08.20> / <V1.0>

功能	变更类型	说明	相关文档
初稿	/	/	/

目录

一、前言	1
1. 文件说明	1
2. 安全声明	1
二、概述	3
1. 使用场景	3
2. 整体架构	3
3. 主要功能及产品组件	3
三、安装卸载	4
1. 安装要求	4
2. 安装过程	4
3. 更新安装	16
4. 卸载过程	18
四、技术说明	20
1. 快速启动	20
2. 本例实验操作步骤	21
3. 实验注意点	36
五、功能介绍	40
1. 结构体转为 Json	40
4. Json 转为结构体	41
5. 生成 Json 文档	42
6. 清空 Json 文档	42
7. 获取 Json 文档长度	43
8. 复制 Json 文档	44
六、附录	44
1. Linux 指令安装/卸载指南	44
2. 支持与服务	46

一、前言

1. 文件说明

本说明专为熟悉相关国家标准且经过专业培训的控制与自动化技术专家而制定。

在安装与调试部件时，务必仔细审阅所有相关文件及以下说明。

合格人员应始终采用最新的有效文档进行操作。

责任人员必须确保所述产品的应用或使用完全符合所有安全要求，涵盖所有相关法律法规、指导原则及标准。

1.1. 免责声明

本文件经过精心编制，但鉴于所描述产品处于持续的开发与升级过程中，中科时代（深圳）计算机系统有限公司保留随时对文件进行修改和更新的权利，且无需事先通知。请注意，禁止依据数据图及本文件描述对已交付的产品进行任何改动。

对于因使用或信赖本手册所载明或未明示的信息而造成的任何损失或损害，中科时代计算机系统有限公司不承担任何责任。

1.2. 版权所有

本手册的所有权归中科时代计算机系统有限公司所有。未经书面许可，任何人不得以任何形式复制、分发、翻译或以其他方式使用本手册的全部或部分内容。

本手册受版权法保护。任何对本手册内容的复制、分发、翻译、展示、表演、演绎或使用，无论出于何种目的，均需得到中科时代计算机系统有限公司的明确许可。未经许可，任何行为均视为侵犯中科时代计算机系统有限公司的版权。

2. 安全声明

2.1. 安全规程

为了您的安全，请阅读以下说明。始终遵守产品特定的安全说明，您可以在本文档的适当位置找到这些说明。

2.2. 责任免除

所有组件都提供了硬件和软件配置。不允许对文件中所述以外的硬件或软件配置进行修改，中科时代不对文件所述外的硬件或软件负责。

2.3. 人员资格

本说明仅适用于熟悉适用国家标准的经过培训的控制、自动化和驱动技术专家。

2.4. 信号词

文档中使用的信号词分类如下。为了防止人员和财产受到伤害和损害，请阅读并遵守安全和警告通知。

2.5. 个人伤害警示

	<div><div>警告</div><div>危险的类型 说明不避开危险的后果 说明如何避免危险的发生</div></div>	警告表示一种潜在的危險情况, 如果不加以避免, 可能会导致严重的伤害或死亡。
	<div><div>注意</div><div>危险的类型 说明不避开危险的后果 说明如何避免危险的发生</div></div>	注意表示潜在的危險情况, 如果不避免, 可能会导致轻度受伤或中度受伤, 或导致设备损坏。
<div><div>提醒</div><div>危险的类型 说明不避开危险的后果 说明如何避免危险的发生</div></div>	注意表示一种潜在的危險情况, 如果不加以避免, 可能只导致设备的损坏。	

2.6. 对财产或环境造成损坏的警告

<div><div>注意</div><div>危险的类型 说明不避开危险的后果 说明如何避免危险的发生</div></div>	环境、设备或数据可能会被损坏。
---	-----------------

2.7. 产品处理信息

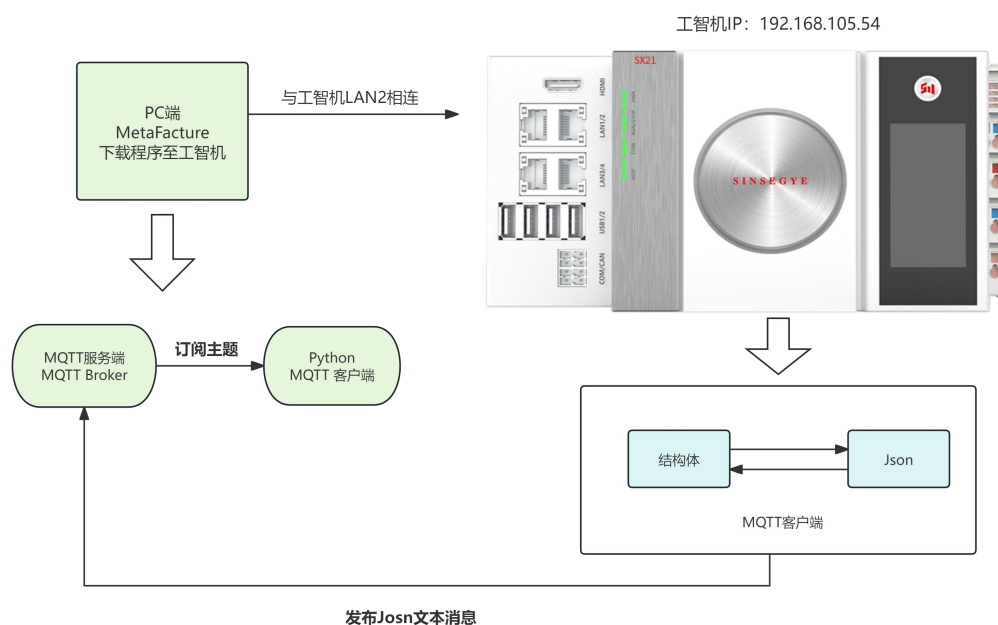
例如，这些信息包括：行动建议、援助或有关产品的进一步信息。

二、概述

1. 使用场景

- Json 格式数据转换为结构体数据
- 结构体数据转换为 Json 格式数据
- 使用 MQTT 通信传输 Json 格式数据

2. 整体架构



3. 主要功能及产品组件

注：本手册中用到的中科时代的软件包，均可以从官网的子页面获取。官网提供的版本可能比本手册中提到的版本更高，一般情况下这不会影响您按照本手册的例子执行相应的操作

- JsonDom 产品包括两部分组件

产品组件	描述说明
------	------

SF4202_JsonDom_1.0.6_adm64.deb	JsonDom RTE 组件
SF4202_JsonDom_1.0.1.1.library	上位机程序使用的库文件

- MQTT 产品包括两部分组件

产品组件	描述说明
SF4202_iotmqtt_1.0.8_amd64.deb	MQTT RTE 组件
SF4202_IoTMqtt_1.0.6.0.library	上位机程序使用的库文件

三、安装卸载

1. 安装要求

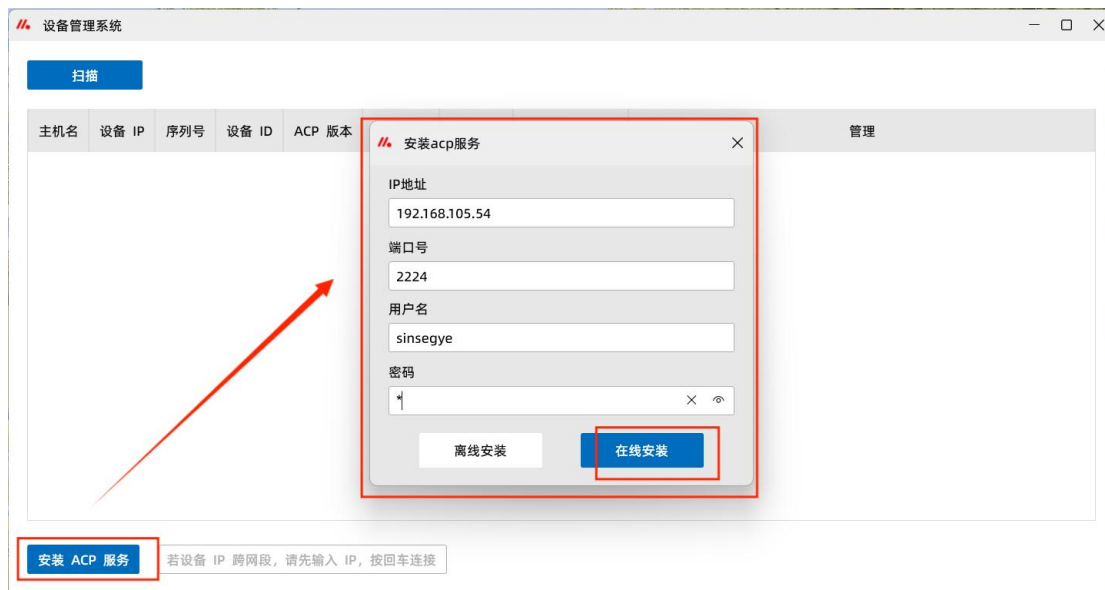
- 中科时代出厂的工智机；
- 安装 Device Manager 软件

2. 安装过程

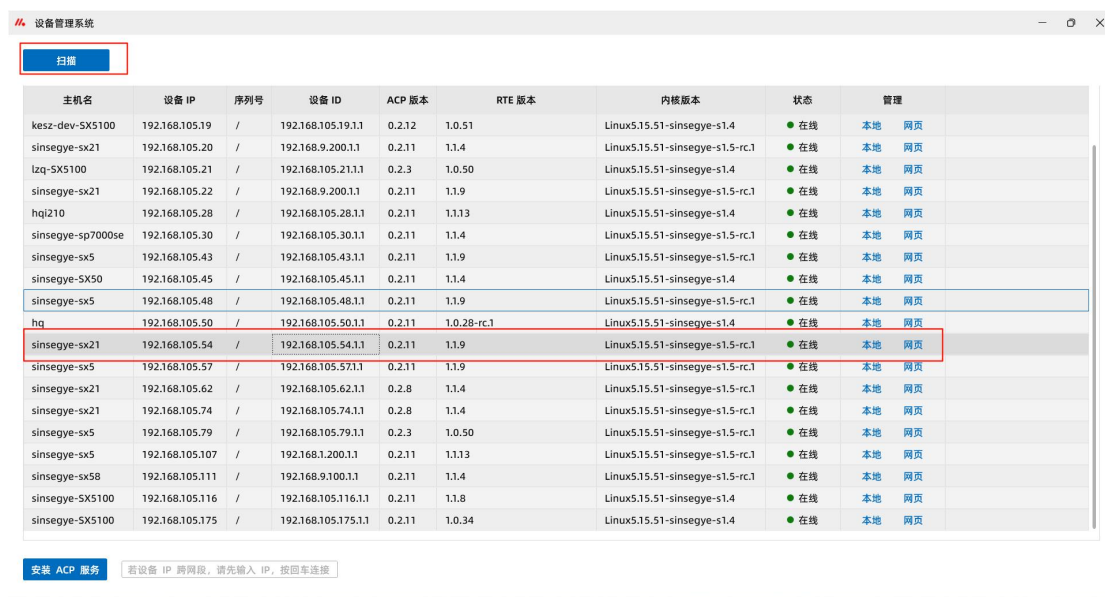
- JsonDom 产品有两个组件，所以要使用 JsonDom 需要从安装这两个组件开始，下面将详细介绍：

2.1. 工智机端安装 JsonDom RTE 组件

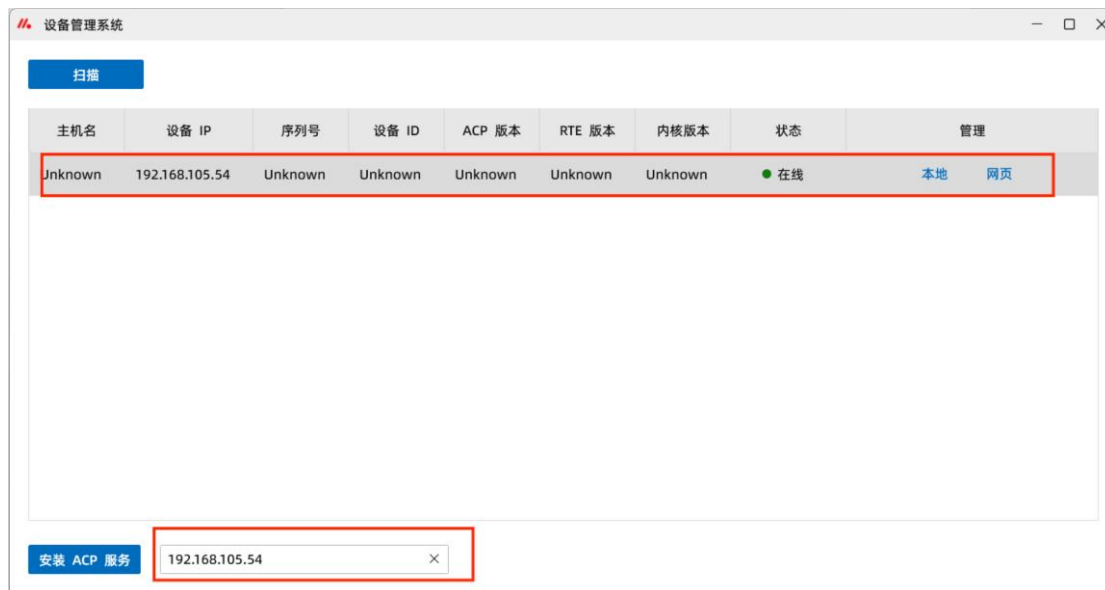
- 打开 Device Manager 软件；
- 安装 ACP 服务（以工智机（192.168.105.54）为例），选择“在线安装”；



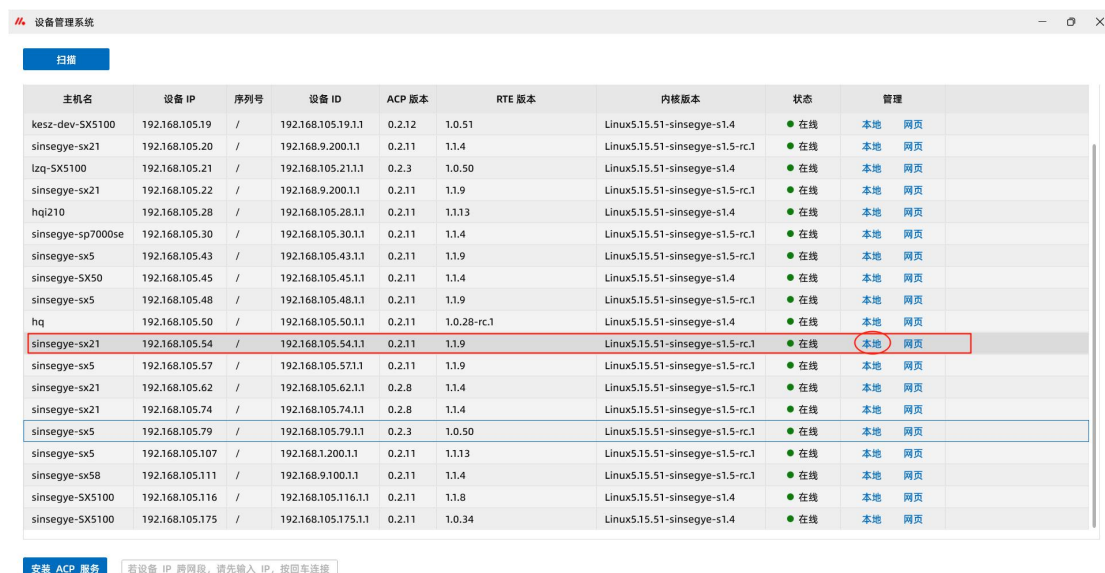
- 点击“扫描”，扫描到需要使用的工智机；



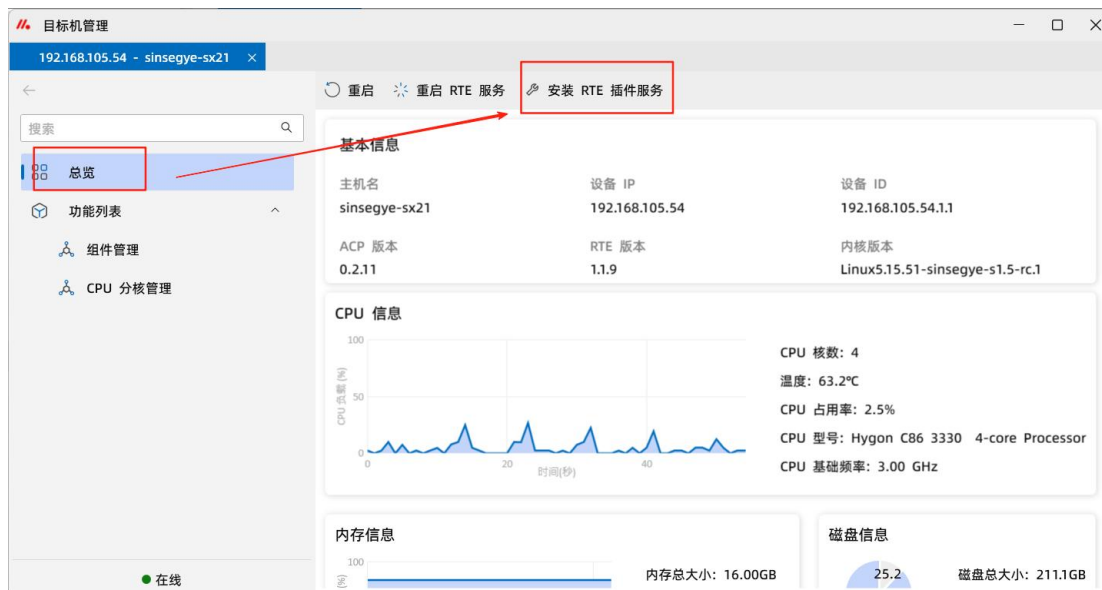
- 如果扫描不到工智机设备，可以输入工智机的 IP 地址手动添加；



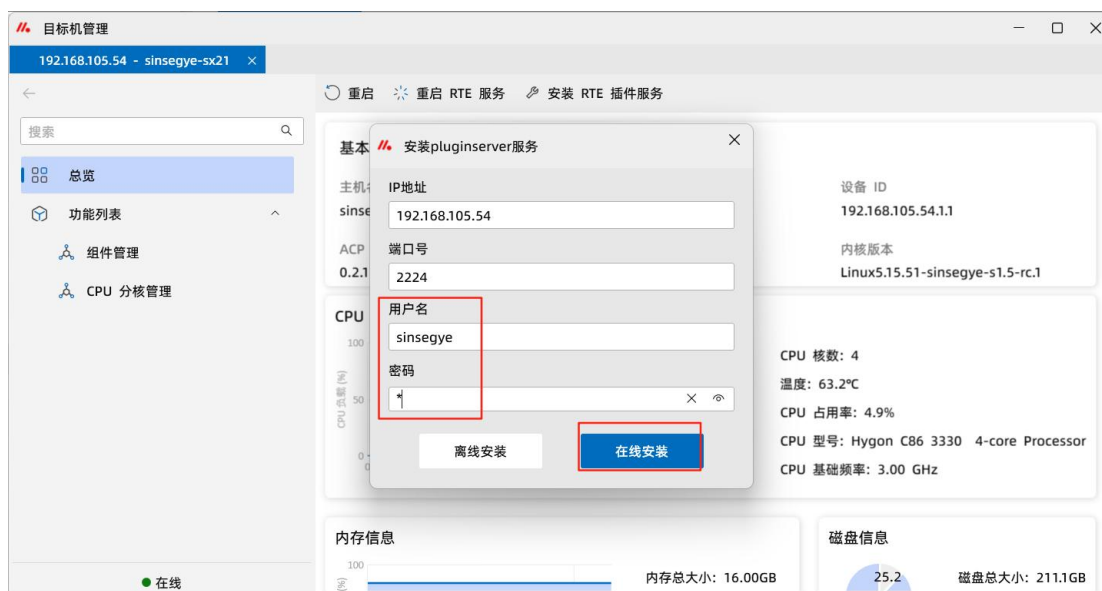
- 选择工智机（192.168.105.54），点击“本地”，进入工智机组件管理页面；



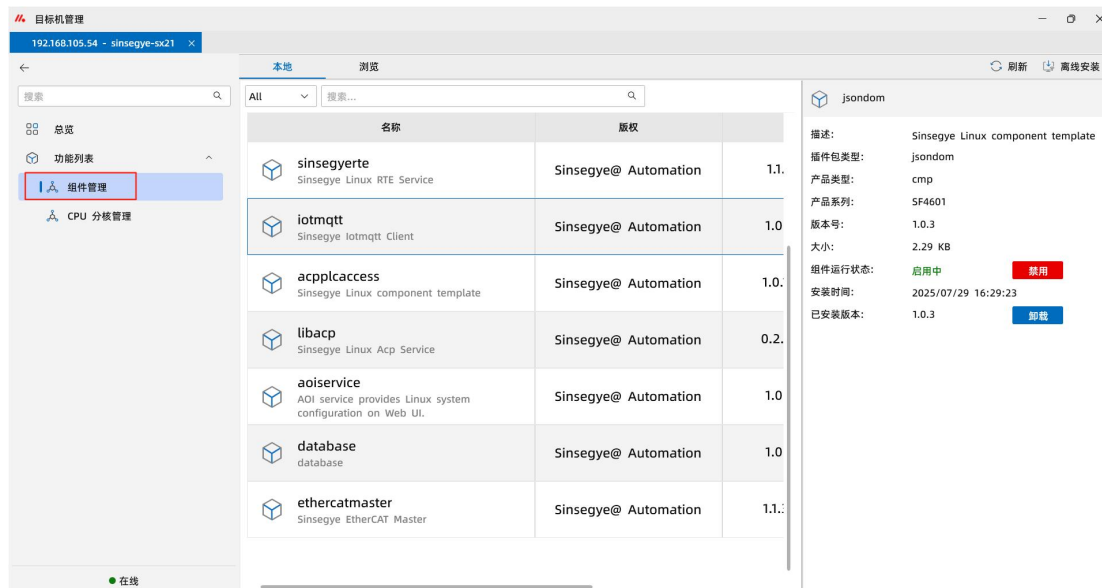
- 进入工智机设备管理器页面，点击“安装 RTE 插件服务”



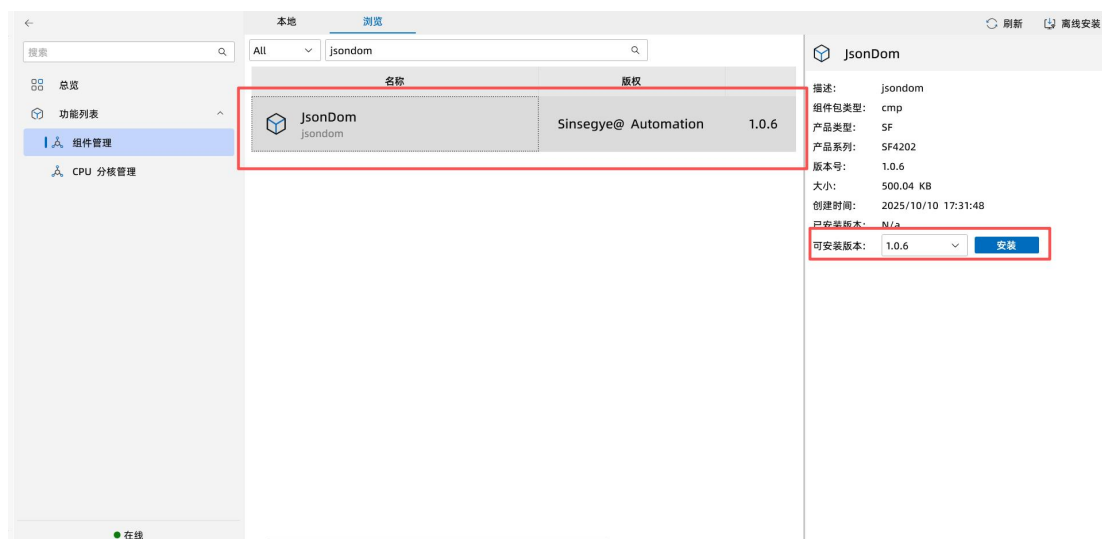
- 输入用户名和密码，点击“在线安装”



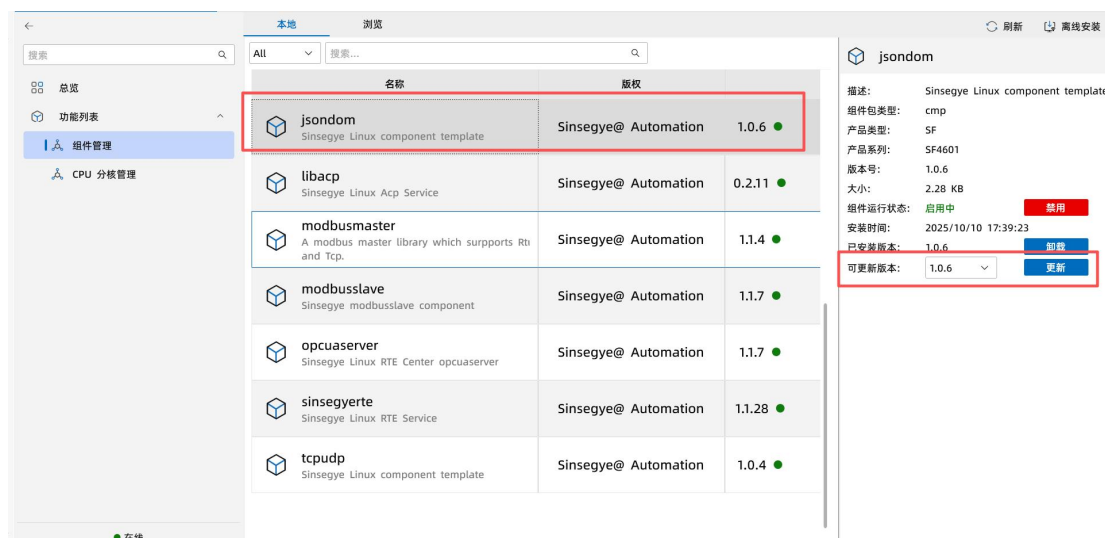
- 进入工智机设备管理器页面，点击“软件”下拉选择“组件管理”，进入如下页面；



- 点击“浏览”，选择“jsondom”查看版本进行安装；

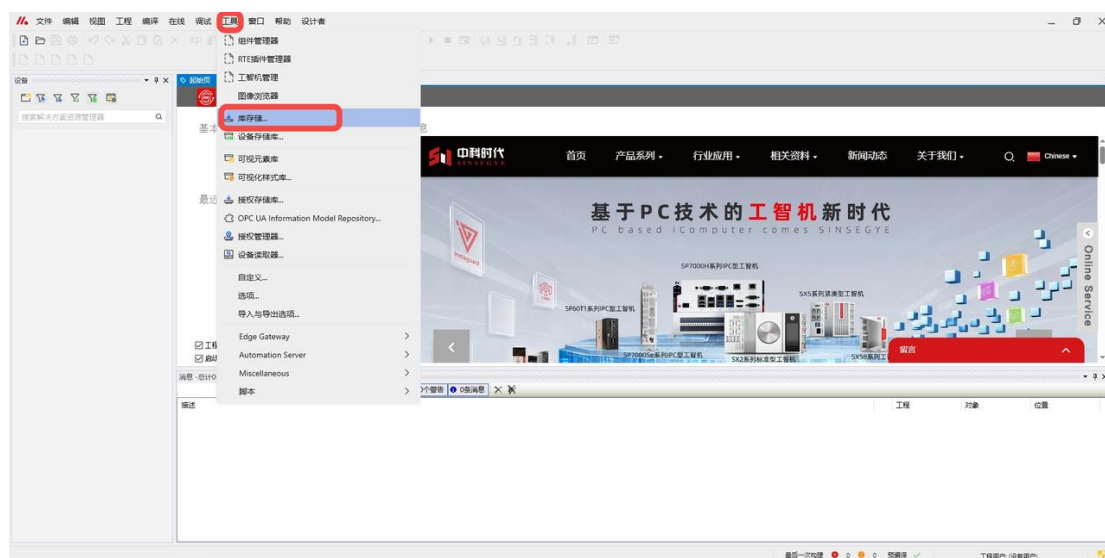


- 传输完成后，弹窗选择确定安装；安装完成后，选择确认重启组件；
- 安装完成后，本地新增 1.0.6 版本的“jsondom”；

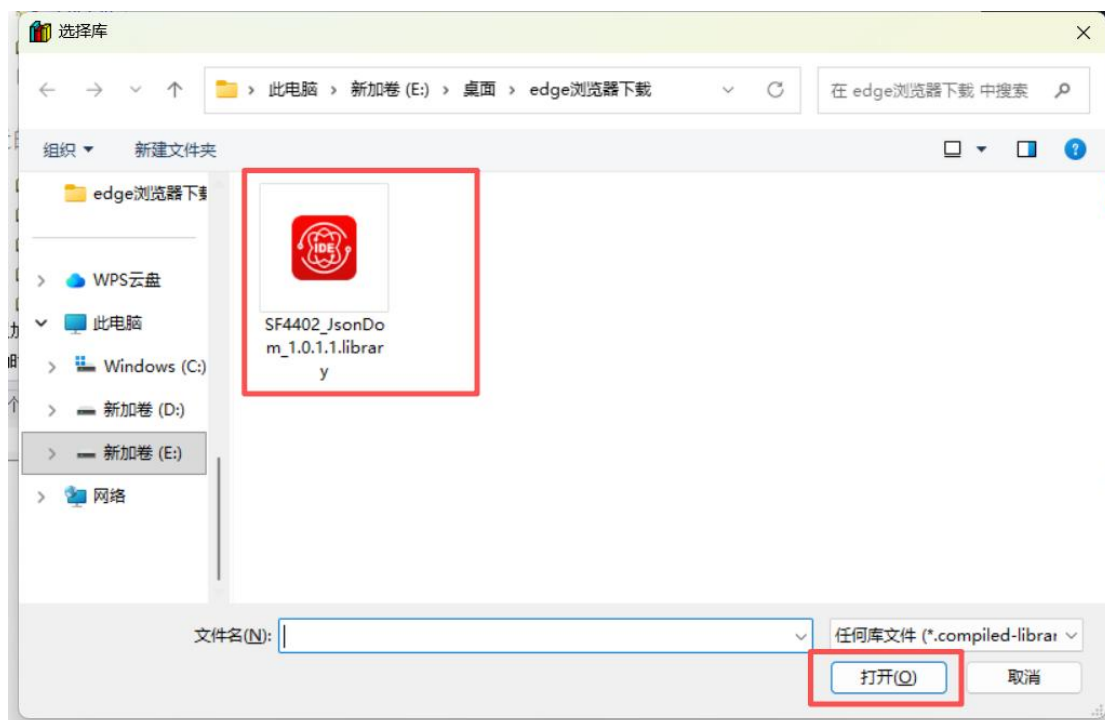
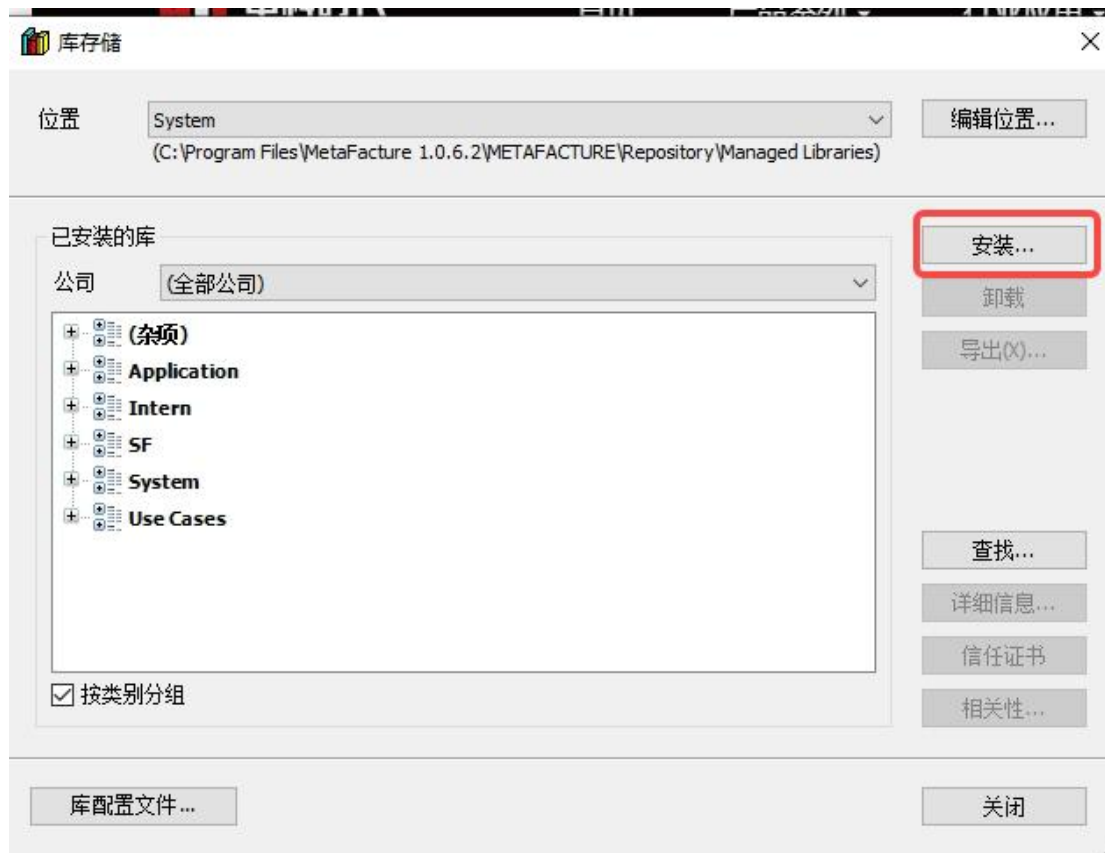


2.2. IDE 侧部署 JsonDom 的 library

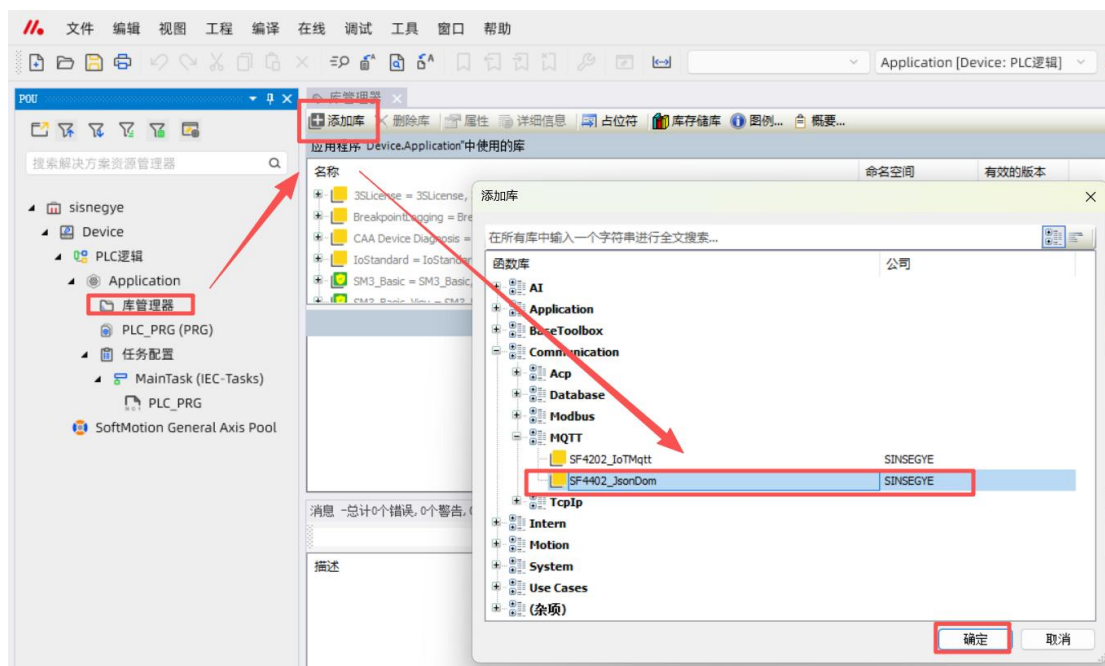
- MetaFactory 中点击最上面的菜单栏 “工具” -- “库存储” ；



- 弹出的对话框中点击“安装” -- 选中 SF4402_JsonDom_1.0.1.1.library -- 点击“打开”;

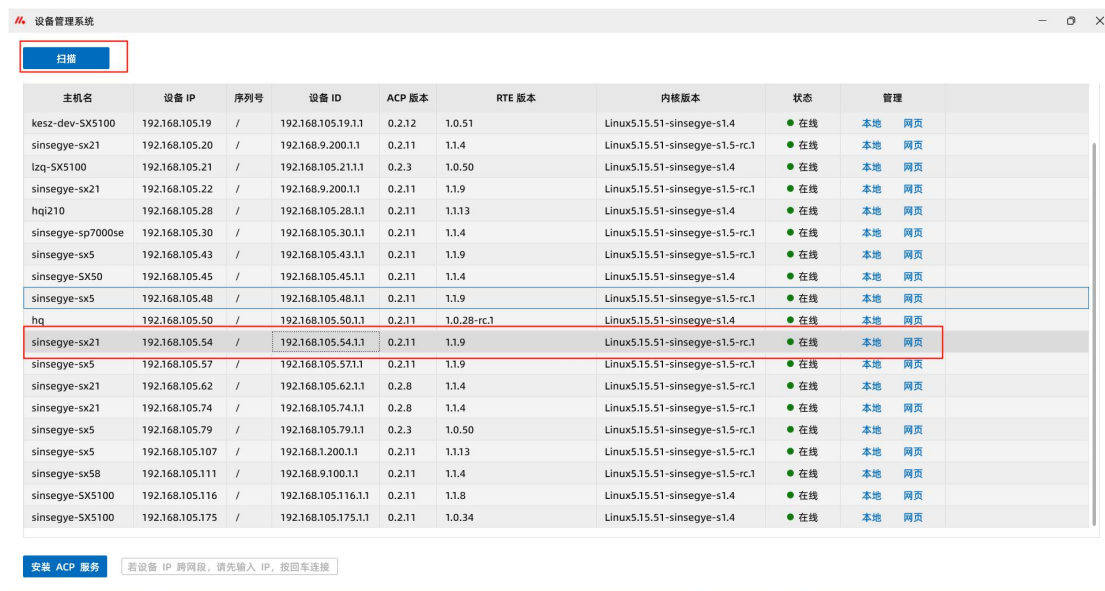


- 工程中双击“库管理器” -- “添加库” -- 双击“SF4402_JsonDom”，加载库完成；

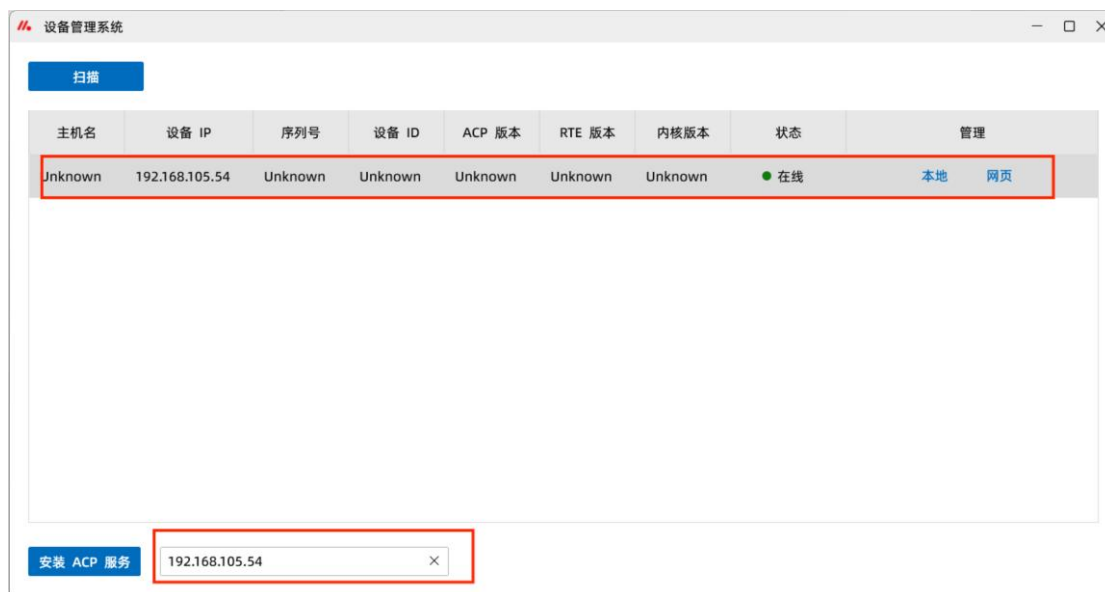


2.3. 工控机端安装 MQTT RTE 组件

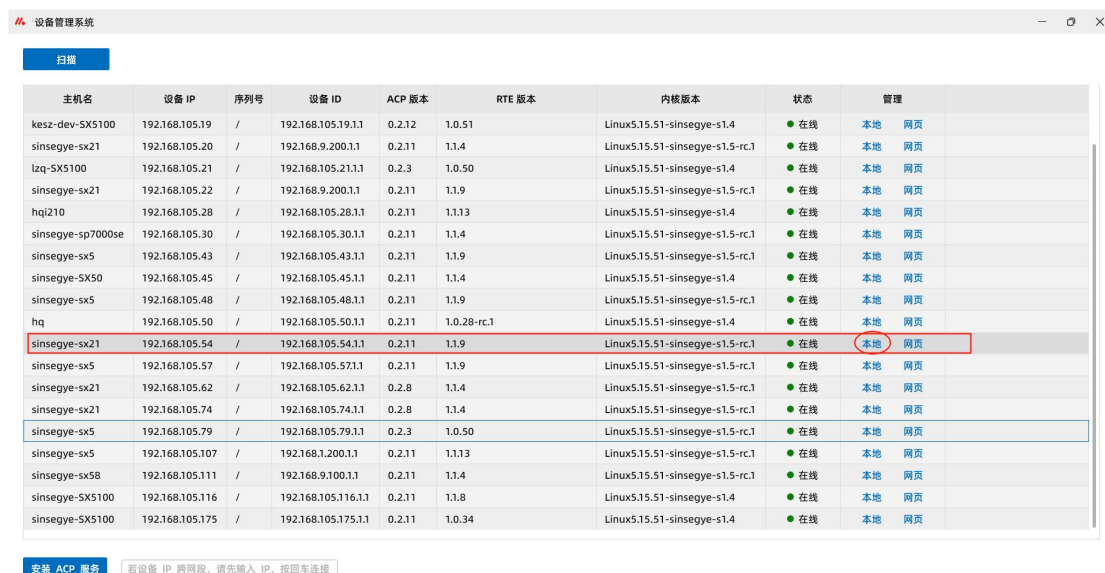
- 本例实验中会用到 MQTT 通信，所以需要安装 MQTT RTE 组件和库文件
- 打开 Device Manager 软件；
- 点击“扫描”，扫描到需要使用的工控机；



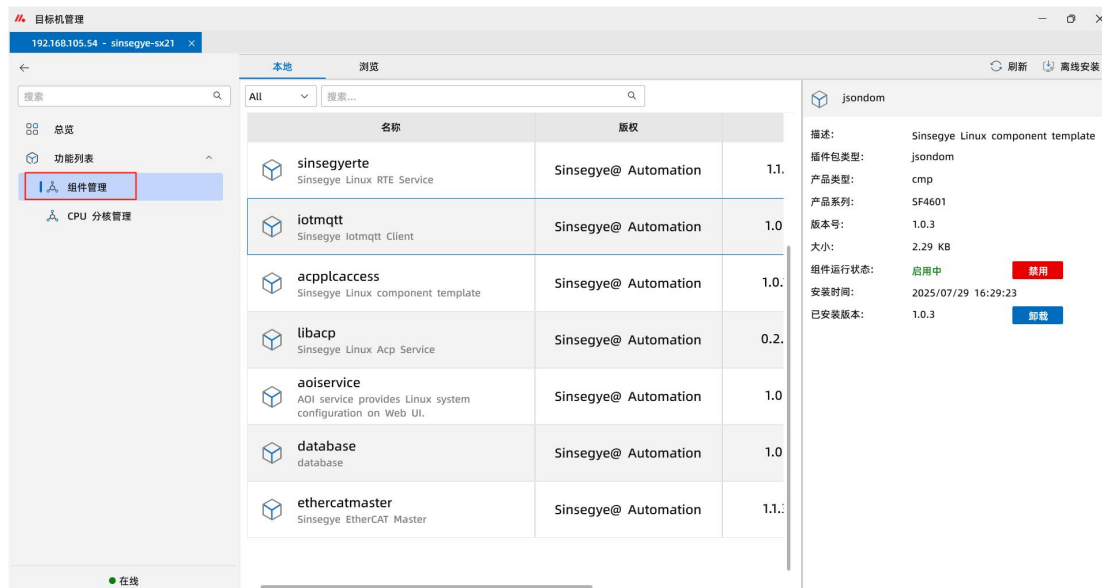
- 如果扫描不到工控机设备，可以输入工控机的 IP 地址手动添加；



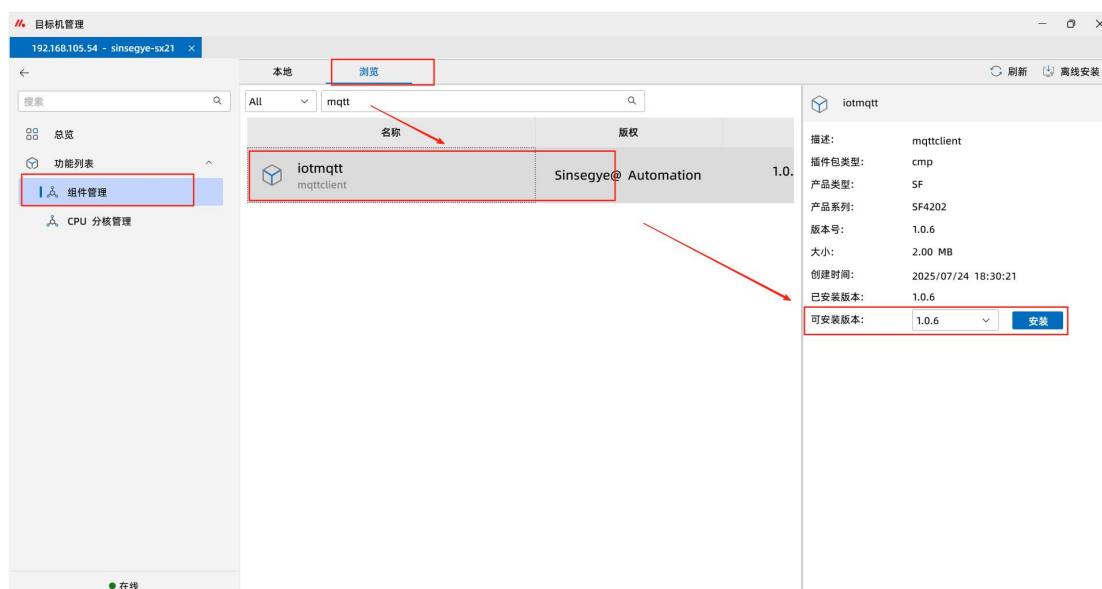
- 选择工智机（192.168.105.54），点击“本地”，进入工智机组件管理页面；



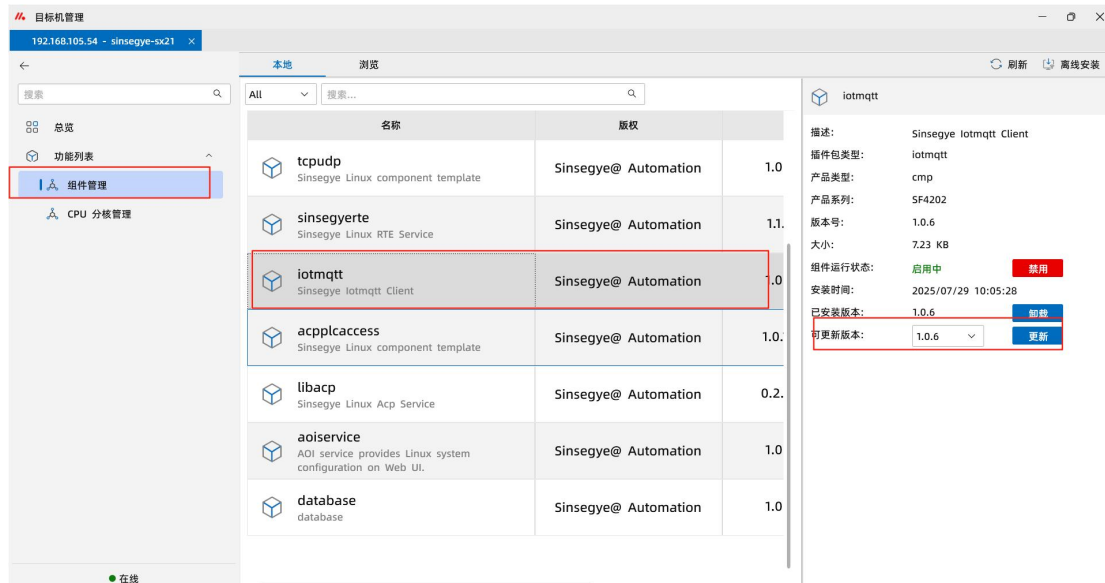
- 进入工智机设备管理器页面，点击“软件”下拉选择“组件管理”，进入如下页面；



- 点击“浏览”，选择“mqtt”查看版本进行安装；

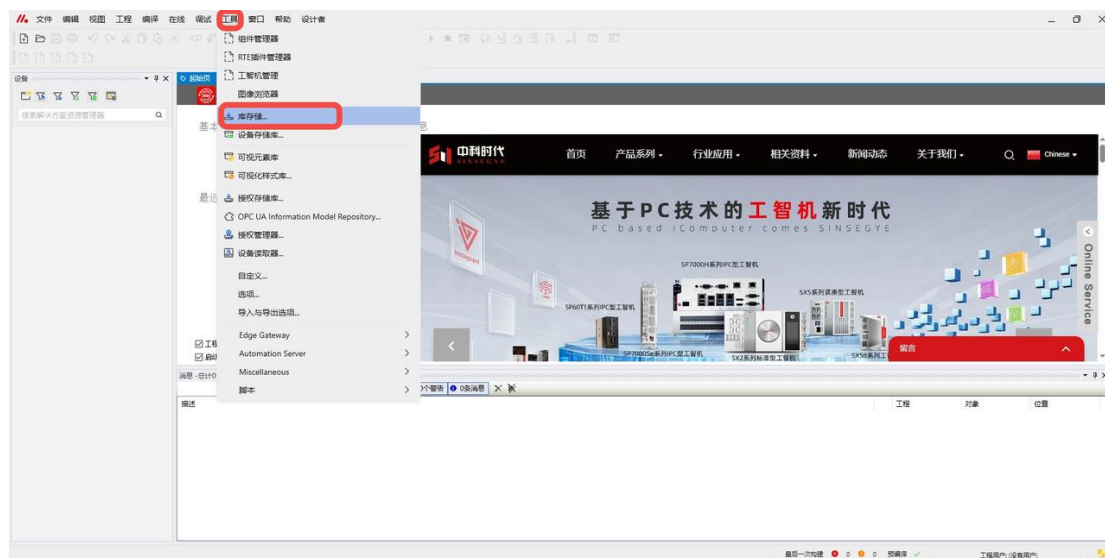


- 传输完成后，弹窗选择确定安装；安装完成后，选择确认重启组件；
- 安装完成后，本地新增 1.0.6 版本的“iotmqtt”；

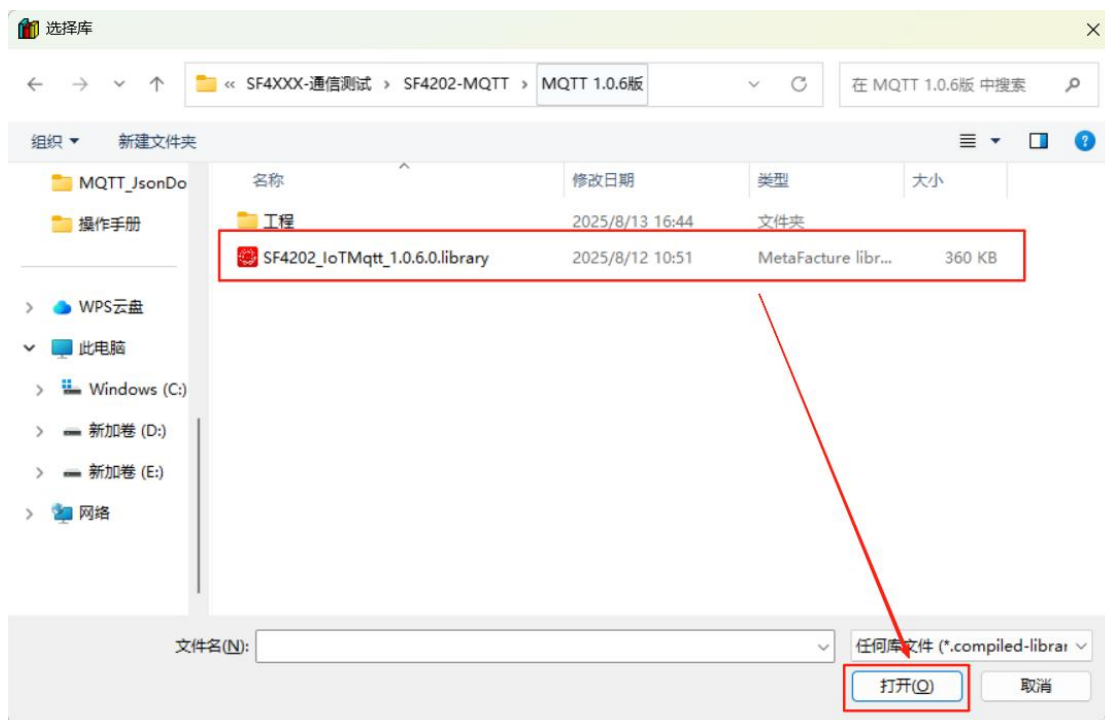
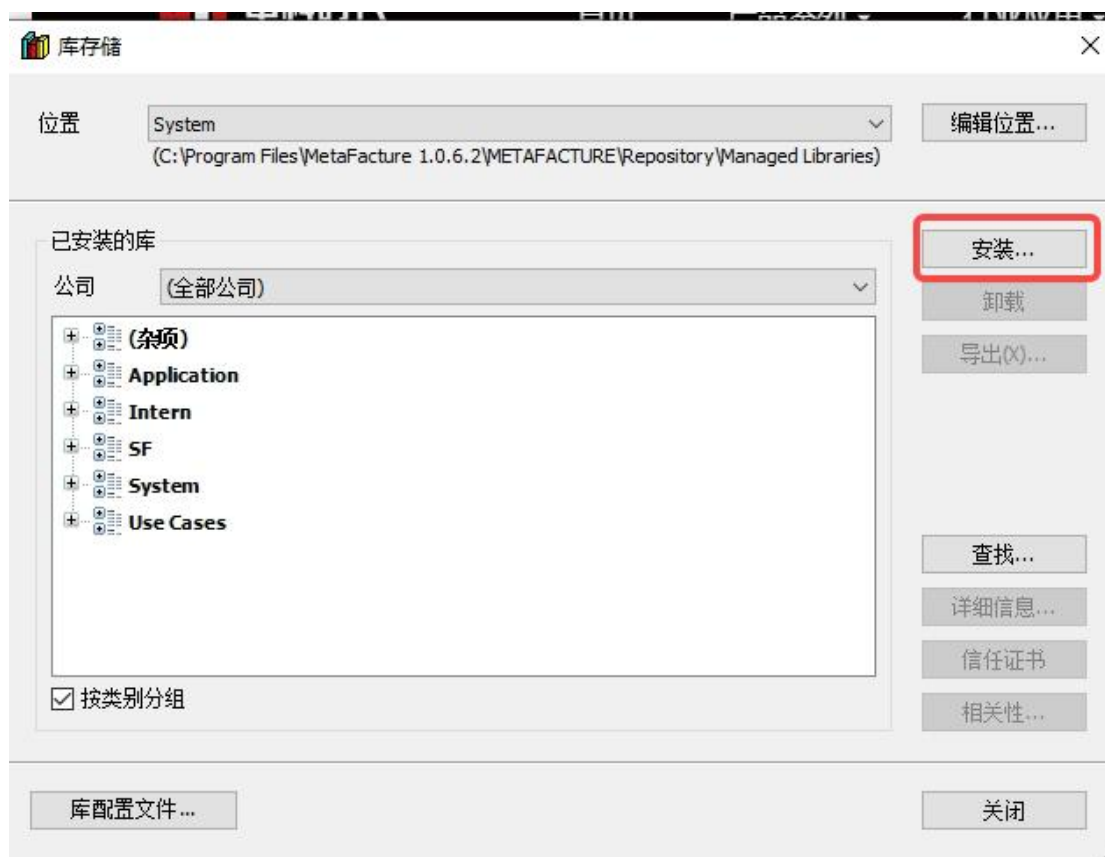


2.4. IDE 侧部署 MQTT 的 library

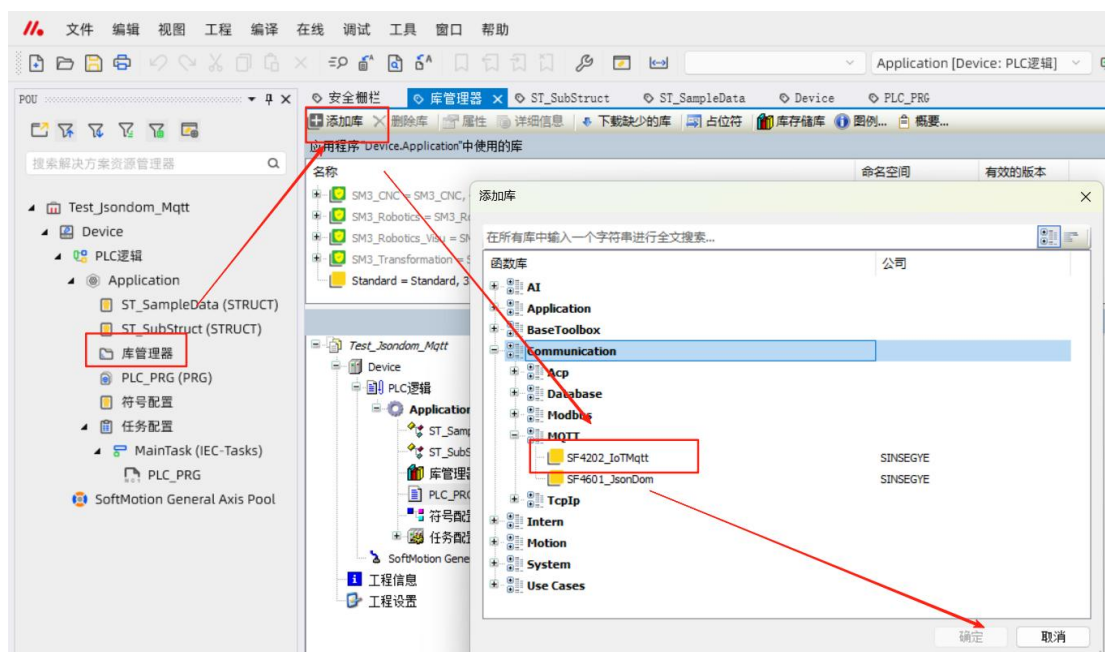
- MetaFactory 中点击最上面的菜单栏 “工具” -- “库存储” ；



- 弹出的对话框中点击“安装” -- 选中 SF4202_lotMqtt_1.0.6.0.library -- 点击“打开”;



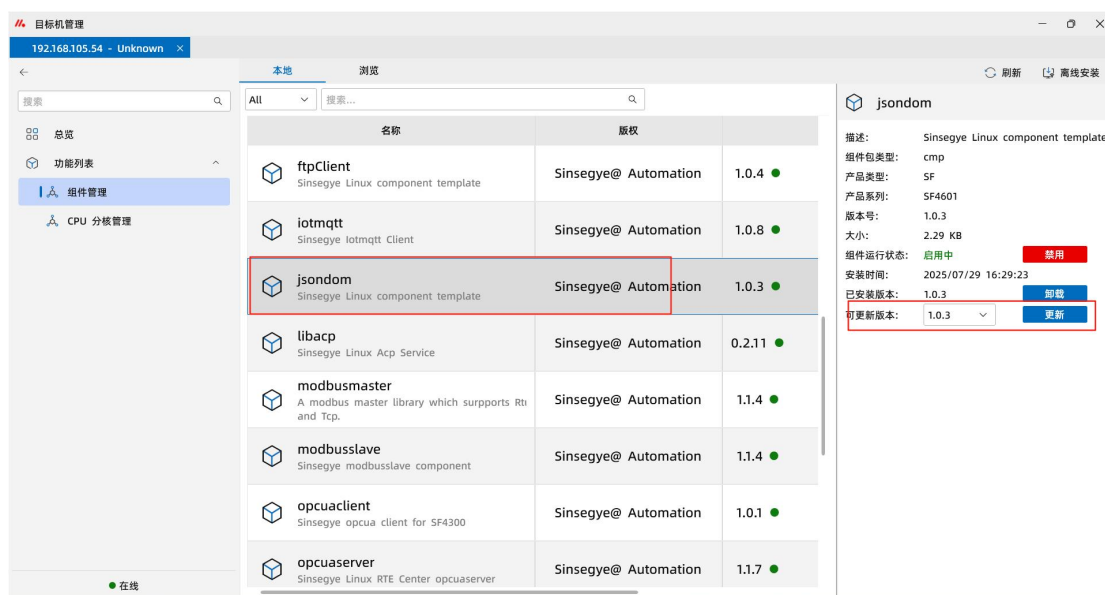
- 工程中双击“库管理器” -- “添加库” -- 双击“SF4601_IotMqtt”，加载库完成；



3. 更新安装

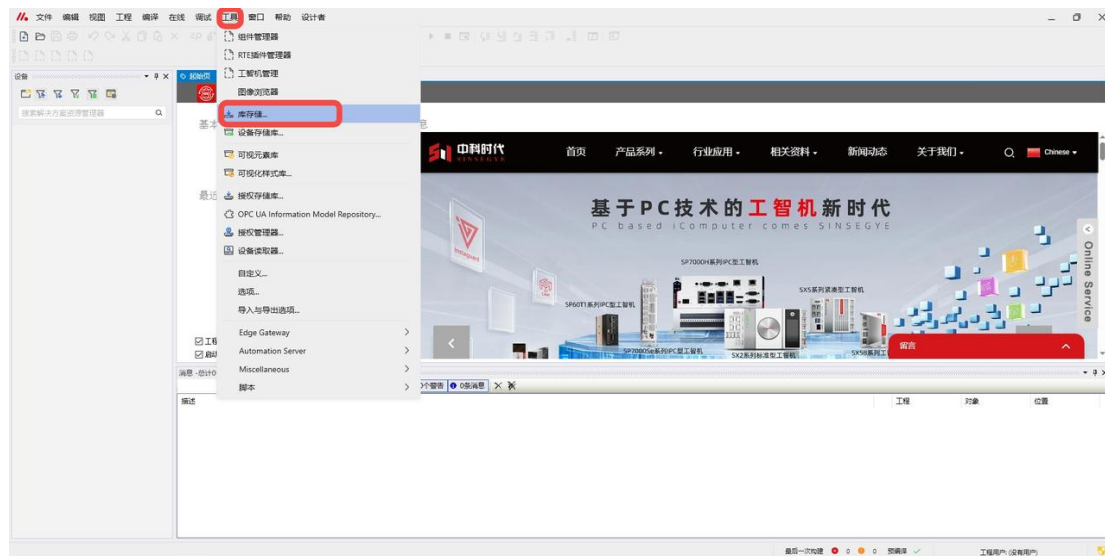
3.1. 升级工智机 JsonDom RTE 组件

- 打开 Device Manager，选择工智机（192.168.105.54），进入组件管理页面；
- 点击“jsondom”组件，选择需要更新的版本，点击更新；

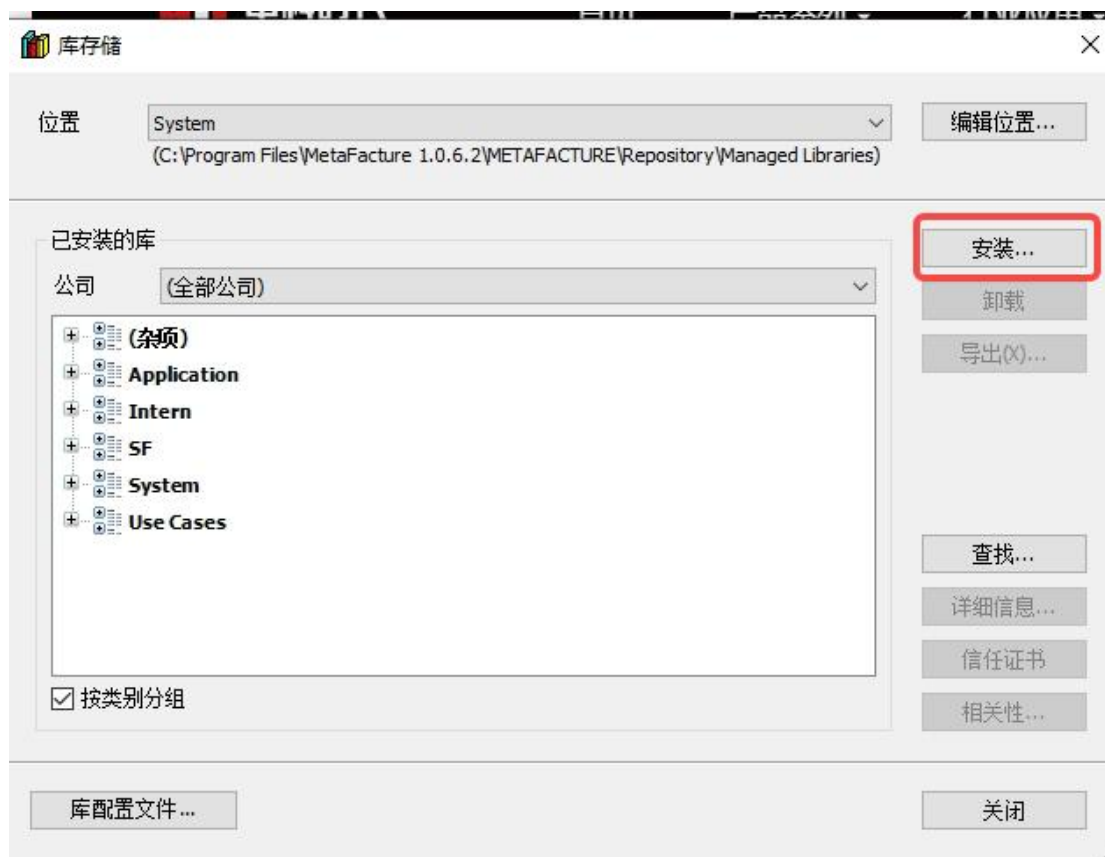


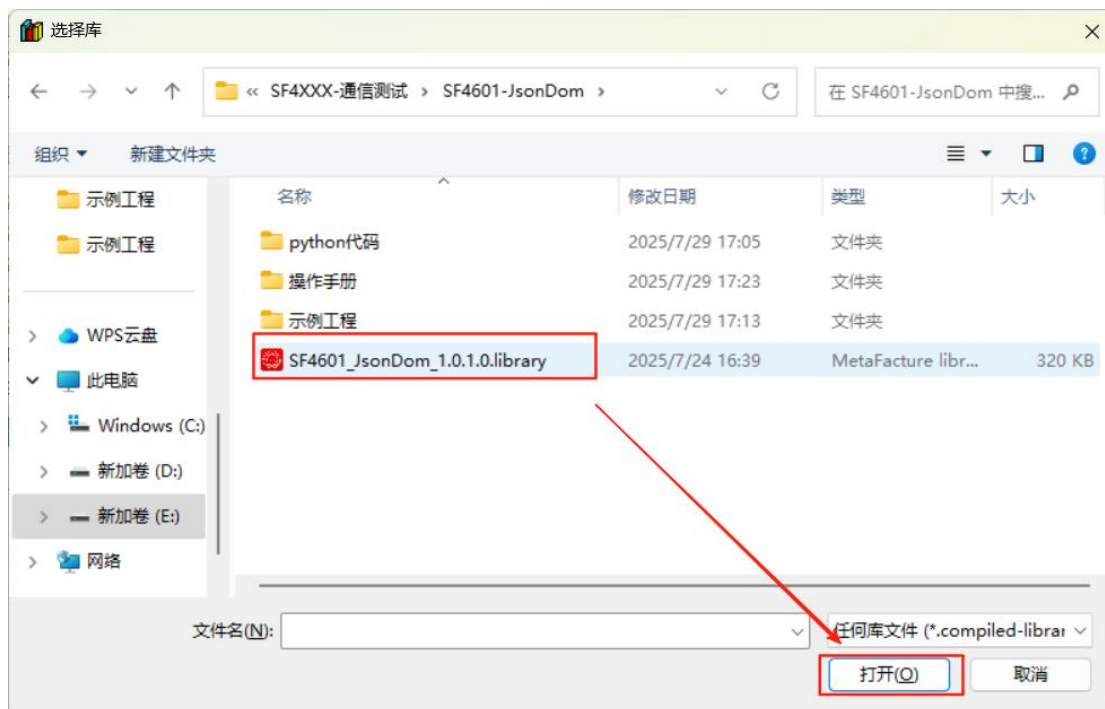
3.2. 升级 IDE 侧 JsonDom 的 library

- MetaFacture 中点击最上面的菜单栏 “工具” -- “库存储” ；

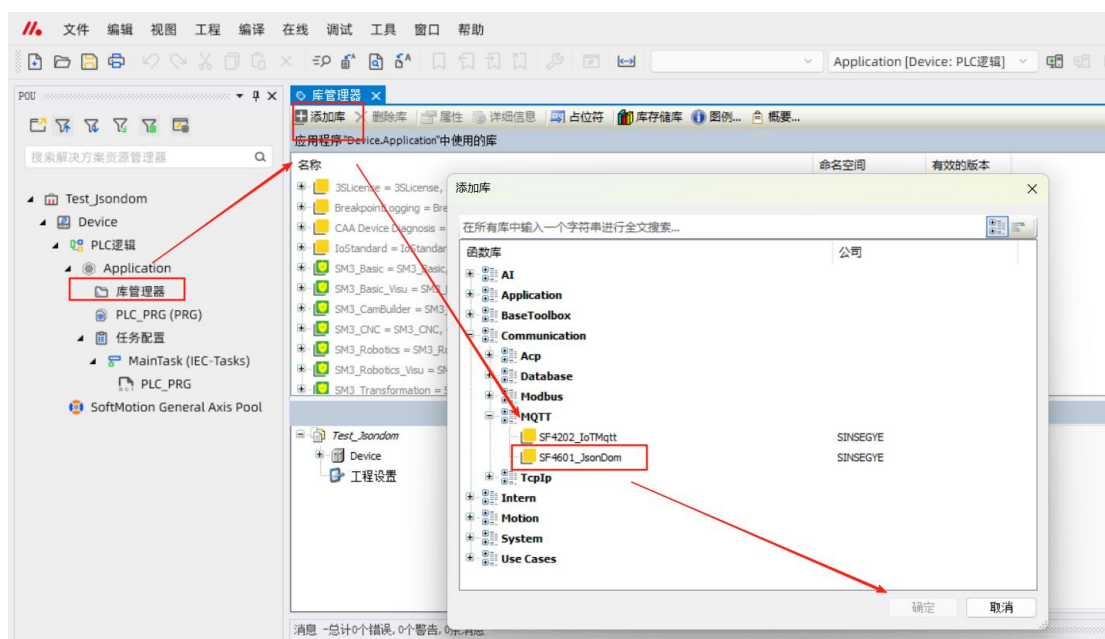


- 弹出的对话框中点击“安装” -- 选中 SF4601_JsonDom_1.0.1.0.library -- 点击“打开”;





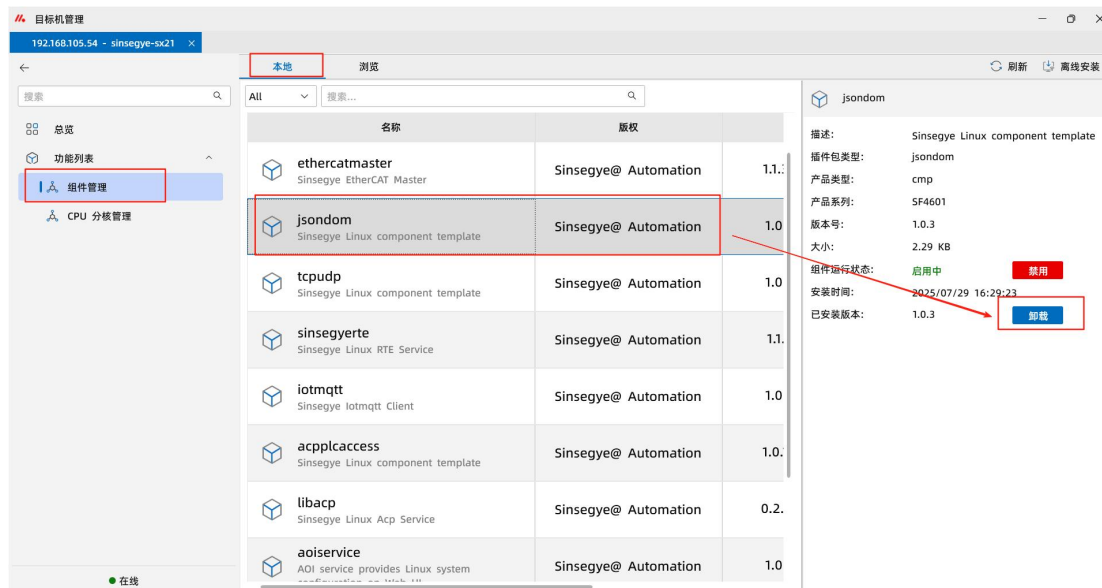
- 工程中双击“库管理器” -- “添加库” -- 双击“SF4601_JsonDom”，加载库完成；



4. 卸载过程

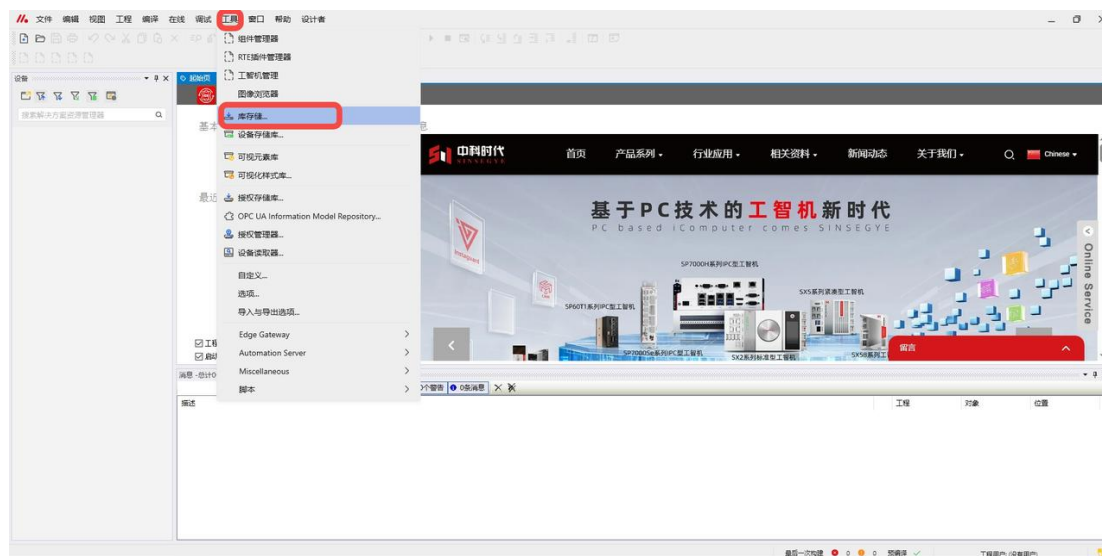
4.1. 卸载工智机 JsonDom RTE 组件

- 打开 Device Manager，进入工智机的组件管理页面，选择需要卸载的组件，点击卸载；

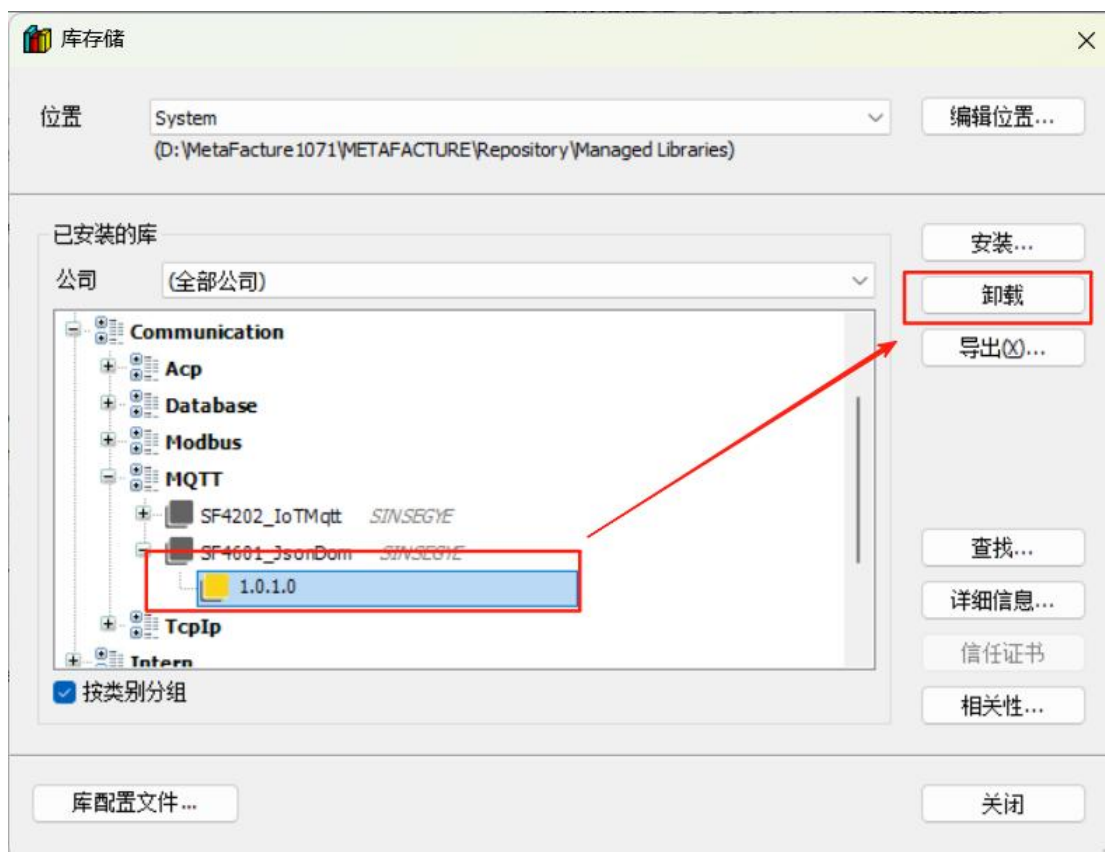


4.2. 卸载 IDE 侧的 JsonDom library

- MetaFacture 界面点击“工具”--“库存储”



- 对话框中选中安装的 SF4601_JsonDom 库，点击“卸载”



四、技术说明

1. 快速启动

1.1. 本例软、硬件配置

硬件：

1. SX21 工控机
2. Win11 PC

软件：

1. MetaFacutre V1.0.7.1
2. Spyder (Pyhton 编译软件)
3. EMQX Broker

1.2. 本例实验说明

- 本例实验测试了 JsonDom 库的 FB_JsonSaxWriter 和 FB_JsonReadWriterDataType 两个功能块，具体测试范围如下表所示：

库文件	功能块	功能块方法
SF4601_JsonDom	FB_JsonSaxWriter	ResetDocument()
		GetDocumentLength()
		CopyDocument()
	FB_JsonReadWriterDataType	SetSymbolFromJson()
		AddJsonKeyValueFromSymbol()

2. 本例实验操作步骤

2.1. 结构体数据转为 Json 格式数据的实验步骤如下：

- 建立结构体数据（涵盖所有基本数据类型：BOOL、INT、REAL、STRING、一维数组、二维数组、时间类型、嵌套结构体、结构体数组）

```

Shell
TYPE ST_SampleData :
STRUCT
    //基本数据类型
    bEnabled : BOOL;
    nCounter : INT;
    fTemperature : REAL;
    sName : STRING(20);

    //一维数组
    arrValues : ARRAY[0..2] OF DINT;

    //二维数组
    arrMatrix : ARRAY[0..1, 0..2] OF REAL;

```



```
//时间类型

dtTimestamp : DATE_AND_TIME;

tDuration : TIME;


// 嵌套结构体

stSubItem : ST_SubStruct;


// 结构体数组

arrStructs : ARRAY[0..1] OF ST_SubStruct;

END_STRUCT

END_TYPE
```

```
Shell

TYPE ST_SubStruct :

STRUCT

    bStatus : BOOL;

    nValue : INT;

    sDescription : STRING(15);

END_STRUCT

END_TYPE
```

- PLC_PRG 中声明区调用结构体

```
Shell

stData : ST_SampleData; //存放测试数据
```

- PLC_PRG 中程序区域为结构体数据赋值

```
Shell

// 初始化结构体数据
```

```
stData.bEnabled := TRUE;

stData.nCounter := 42;

stData.fTemperature := 23.5;

stData.sName := 'SampleDevice';


// 一维数组

stData.arrValues[0] := 10;

stData.arrValues[1] := 20;

stData.arrValues[2] := 30;


// 二维数组

stData.arrMatrix[0,0] := 1.1;

stData.arrMatrix[0,1] := 1.2;

stData.arrMatrix[0,2] := 1.3;

stData.arrMatrix[1,0] := 2.1;

stData.arrMatrix[1,1] := 2.2;

stData.arrMatrix[1,2] := 2.3;


// 时间类型

stData.dtTimestamp := DT#2023-05-15-14:30:00;

stData.tDuration := T#1H2M3S;


// 嵌套结构体

stData.stSubItem.bStatus := FALSE;

stData.stSubItem.nValue := 99;

stData.stSubItem.sDescription := 'Nested item';


// 结构体数组
```

```
stData.arrStructs[0].bStatus := TRUE;
stData.arrStructs[0].nValue := 100;
stData.arrStructs[0].sDescription := 'Array item 1';

stData.arrStructs[1].bStatus := FALSE;
stData.arrStructs[1].nValue := 200;
stData.arrStructs[1].sDescription := 'Array item 2';
```

- PLC_PRG 中声明区域调用 JsonDom 库的功能块 FB_JsonSaxWriter 和 FB_JsonReadWriteDataType

Shell

```
sJsonBuffer : STRING(10000); // JSON 缓冲区
fbJsonDom : MetaCore_JsonDom.FB_JsonSaxWriter;
fbJsonDataType : MetaCore_JsonDom.FB_JsonReadWriteDataType;
```

- PLC_PRG 中程序区域调用 fbJsonDom 和 fbJsonDataType，将结构体数据转为 Json 格式并存入 sJsonBuffer 中

Shell

```
fbJsonDom.ResetDocument();

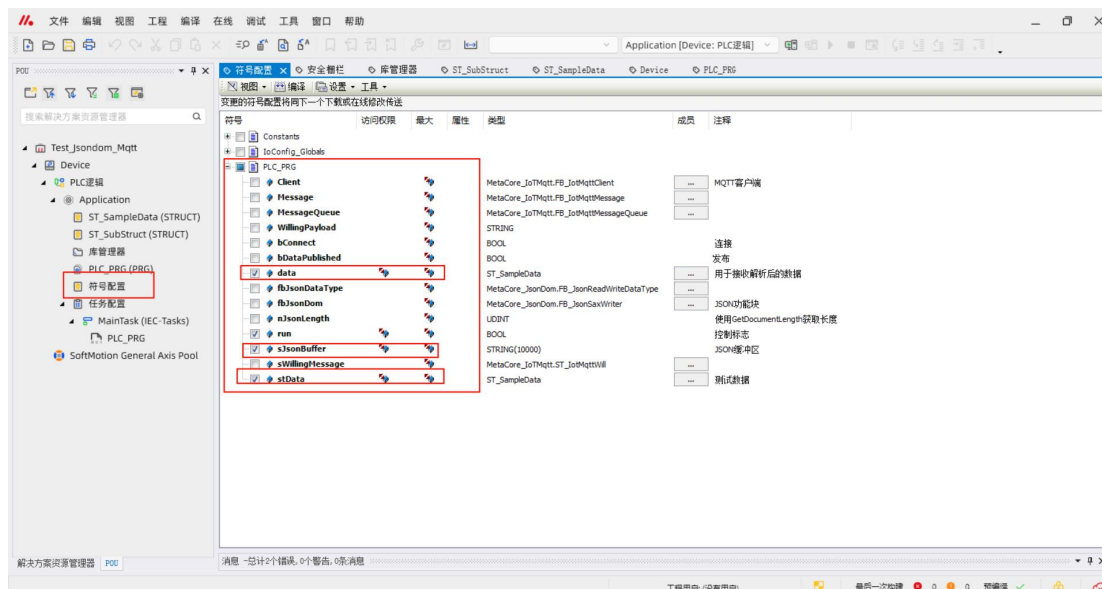
fbJsonDataType.AddJsonKeyValueFromSymbol(
    fbJsonDom,
    'data',
    'ST_SampleData',
    SIZEOF(stData),
    ADR(stData)
);

nJsonLength := fbJsonDom.GetDocumentLength(); // 获取 JSON 文档长度
fbJsonDom.CopyDocument(sJsonBuffer, nJsonLength);
```

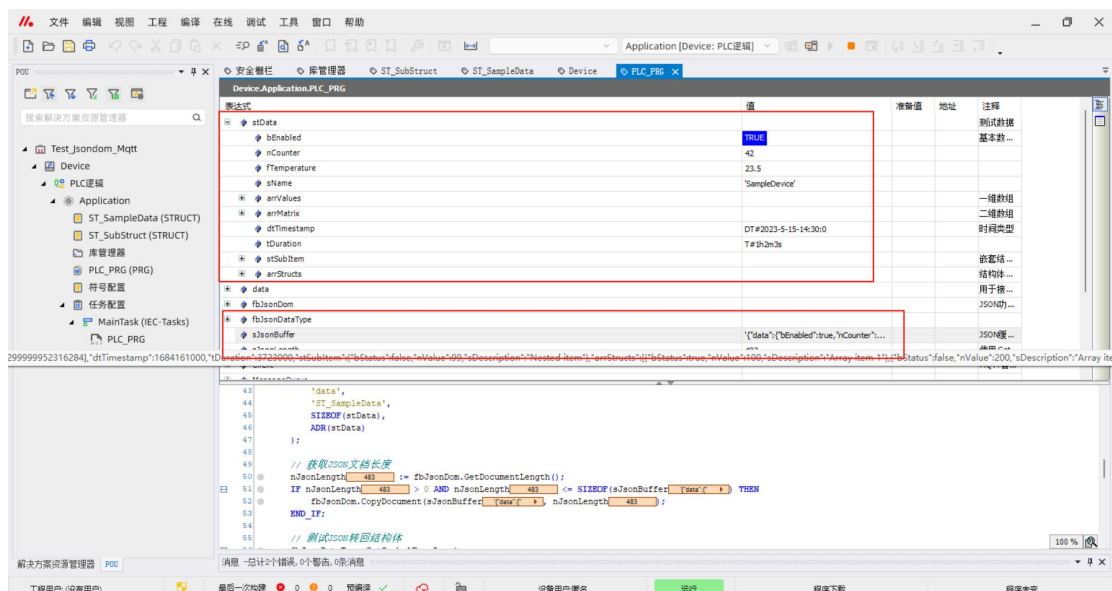
- 注意!必须添加符号配置才能成功转换

右键设备树的“Application”——>添加对象——>符号配置

勾选存放结构体数据和 Json 文本数据的变量



- 登录设备，下载程序，程序运行后可以看到 stData 是赋值后的结构体数据，而 sJsonBuffer 中为转换后的 Json 文本，nJsonLength 为 Json 文本的长度



2.2. Json 格式数据转换为结构体数据的实验步骤如下：

- POU 中声明区域创建存放转换后的结构体数据

Shell

```
data : ST_SampleData; // 用于接收解析后的数据
```

- PLC_PRG 中声明区域调用 JsonDom 库的功能块 FB_JsonReadWriteDataType

```
Shell
```

```
fbJsonDataType : MetaCore_JsonDom.FB_JsonReadWriteDataType;
```

- PLC_PRG 中程序区域调用 fbJsonDataType, 将 sJsonBuffer 转为结构体存入 data

```
Shell
```

```
// 测试 JSON 转回结构体
```

```
fbJsonDataType.SetSymbolFromJson(
    sJsonBuffer,
    SIZEOF(sJsonBuffer),
    ADR(data),
    SIZEOF(data)
);
```

- 登录设备，下载程序，程序运行后可以看到 sJsonBuffer 中为转换后的 Json 文本，而 data 中为 sJsonBuffer 转换为结构体的数据。

表达式	值	准备值	地址	注释
* stData				测试数据
data				用于接...
bEnabled	TRUE			基本数...
nCounter	42			
fTemperature	23.5			
sName	'SampleDevice'			
arrValues				一维数组
arrValues[0]	10			
arrValues[1]	20			
arrValues[2]	30			
arrMatrix				二维数组
arrMatrix[0, 0]	1.1			
arrMatrix[0, 1]	1.2			
arrMatrix[0, 2]	1.3			
arrMatrix[1, 0]	2.1			
arrMatrix[1, 1]	2.2			
arrMatrix[1, 2]	2.3			
dtTimestamp	DT#2023-5-15-14:30:0			时间类型
tDuration	T#1h2m3s			
stSubItem				嵌套结...
bStatus	FALSE			
nValue	99			
sDescription	'Nested item'			
arrStructs				结构体...
arrStructs[0]				
arrStructs[1]				
fbJsonDom				JSON功...

2.3. 将得到的 Json 数据通过 MQTT 通信传输并使用 python 获取数据的实验步骤如下：

- POU 中声明区域调用 MQTT 库的功能块

```
Shell
```

```
// MQTT 客户端

Client : MetaCore_IoTMqtt.FB_IotMqttClient;

MessageQueue : MetaCore_IoTMqtt.FB_IotMqttMessageQueue;

Message : MetaCore_IoTMqtt.FB_IotMqttMessage;

sWillingMessage : MetaCore_IoTMqtt.ST_IotMqttWill;

WillingPayload : STRING := 'PLC offline';


// 控制标志

run : BOOL := TRUE;

bPublish : BOOL := FALSE; // 手动发布触发标志

bDataPublished : BOOL := FALSE; //是否发布标志

bConnect : BOOL := FALSE; // 手动连接触发标志
```

- POU 中程序区域调用发布和连接功能块

```
// MQTT 连接管理

Client(

    sClientId := 'PLC_Client',

    sHostName := '192.168.105.16',

    nHostPort := 1883,

    nKeepAlive := 5,

    stWill := sWillingMessage,

    ipMessageQueue := ADR(MessageQueue),

    bError => ,

    hrErrorCode => ,

    eConnectionState => ,

    bConnected =>

);

Client.Execute(bCo
```

```
// 发布消息

IF Client.bConnected AND bPublish THEN

    Client.Publish(

        'plc/data',

        ADR(sJsonBuffer),

        nJsonLength, // 使用实际 JSON 长度

        2,           // QoS=2 确保可靠传输

        FALSE,

        FALSE

    );

    // 发布后重置触发标志

    bPublish := FALSE;

    bDataPublished := TRUE;

END_IF
```

- Python 代码部分，实现 MQTT 通信以及 Json 文本的输出

```
import paho.mqtt.client as mqtt

import json

from datetime import datetime

def on_connect(client, userdata, flags, rc):

    print(f" Connected to MQTT broker with code: {rc}")

    client.subscribe("plc/data")

def parse_duration(duration_str):
```

```
"""解析 CODESYS 的 TIME 格式（如 T1H2M3S）"""

if not isinstance(duration_str, str) or not
duration_str.startswith('T'):

    return None

hours = minutes = seconds = 0

time_str = duration_str[1:] # 去掉'T'

if 'H' in time_str:

    hours = int(time_str.split('H')[0])

    time_str = time_str.split('H')[1]

if 'M' in time_str:

    minutes = int(time_str.split('M')[0])

    time_str = time_str.split('M')[1]

if 'S' in time_str:

    seconds = int(time_str.split('S')[0])

return {

    'hours': hours,

    'minutes': minutes,

    'seconds': seconds,

    'total_seconds': hours*3600 + minutes*60 + seconds

}

def on_message(client, userdata, msg):

    try:

        # 原始数据检查

        raw_data = msg.payload.decode('utf-8')
```



```
print(f"\n Raw message length: {len(raw_data)} bytes")

# 调试: 打印前 200 个字符 (避免日志过长)

print(f" Sample data (first 200
chars):\n{raw_data[:200]}{'...' if len(raw_data)>200 else ''}")

try:

    data = json.loads(raw_data)

except json.JSONDecodeError as e:

    print(f" JSON 解析失败! 错误位置: {e.pos}, 可能原因: 数
据截断或格式错误")

    print(f" 错误附近的上下文: \n{raw_data[max(0,
e.pos-50):e.pos+50]}")

    return

# 美化打印完整 JSON

print("\n 完整 JSON 数据结构: ")

print(json.dumps(data, indent=2, ensure_ascii=False))

# 提取数据字段

plc_data = data.get('data', {})

# 基本字段

print("\n 基本数据: ")

print(f"• 设备名称: {plc_data.get('sName', 'N/A')}")

print(f"• 使能状态: {'ON' if plc_data.get('bEnabled') else
'OFF'}")

print(f"• 计数器值: {plc_data.get('nCounter', 0)}")

print(f"• 温度: {plc_data.get('fTemperature',
```

```
0.0):.2f}°C")

# 数组数据
print("\n 数组数据: ")
print(f"• 一维数组: {plc_data.get('arrValues', [])}")
print(f"• 二维数组: {plc_data.get('arrMatrix', [])}")

# 智能时间处理
print("\n 时间数据: ")
timestamp = plc_data.get('dtTimestamp')
if timestamp:
    if isinstance(timestamp, (int, float)): # Unix 时间戳
        dt = datetime.fromtimestamp(timestamp)
        print(f"• 时间戳 (Unix): {timestamp} → {dt.isoformat()}")
    elif isinstance(timestamp, str): # ISO 格式
        try:
            dt = datetime.strptime(timestamp,
"%Y-%m-%dT%H:%M:%S")
            print(f"• 时间戳 (ISO): {dt.isoformat()}")
        except ValueError:
            print(f"• 无法解析的时间格式: {timestamp}")

# 持续时间处理
duration = plc_data.get('tDuration')
if duration:
    if isinstance(duration, str):
        parsed = parse_duration(duration)
```

```
        if parsed:

            print(f"• 持续时间: {duration}")

            print(f" → 合计: {parsed['total_seconds']}秒
(H:M:S =
{parsed['hours']}:{parsed['minutes']}:{parsed['seconds']})")

        else:

            print(f"• 非标准持续时间格式: {duration}")

        elif isinstance(duration, (int, float)):

            print(f"• 持续时间 (秒): {duration}")

# 结构体数据
print("\n 结构体数据: ")

if 'stSubItem' in plc_data:

    sub = plc_data['stSubItem']

    print(f"• 子结构体: Status={sub.get('bStatus')},
Value={sub.get('nValue')},
Desc='{sub.get('sDescription','')}'")

if 'arrStructs' in plc_data:

    print("• 结构体数组:")

    for i, item in enumerate(plc_data['arrStructs']):

        print(f" [{i}] Status={item.get('bStatus')},
Value={item.get('nValue')},
Desc='{item.get('sDescription','')}'")

except Exception as e:

    print(f"\n!! 处理消息时发生严重错误: {str(e)}")

import traceback

traceback.print_exc()
```

```
# MQTT 客户端配置

client = mqtt.Client(
    client_id="PythonSubscriber",
    protocol=mqtt.MQTTv311,
    transport="tcp"
)

# 回调绑定

client.on_connect = on_connect
client.on_message = on_message

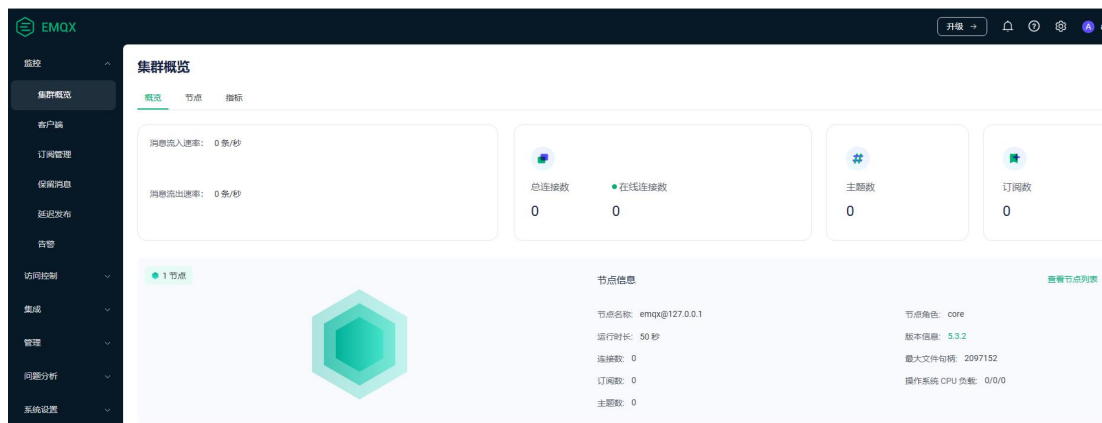
# 连接设置

broker_ip = "192.168.105.16"
client.connect(broker_ip, 1883, 60)

print(f" 开始监听 MQTT 主题 plc/data, 连接服务器: {broker_ip}...")

try:
    client.loop_forever()
except KeyboardInterrupt:
    print("\n 手动停止订阅")
    client.disconnect()
```

- 建立 MQTT 服务端 MQTT Broker 并登录（详细操作步骤见 3.实验注意点）



- 运行 Python 代码
- 登录工智机设备，下载程序，程序运行后置 bConnect 为 True，成功连接后将发布消息

表达式	值	准备值	地址	注释
sJsonBuffer	{'data': {'bEnabled': true, 'nCounter': ...			JSON 编...
nJsonLength	483			使用 Get...
Client				MQTT 客户端
sClientId	'PLC_Client'			default i...
sHostName	'192.168.105.16'			default i...
nHostPort	1883			default i...
sTopicPrefix	*			topic pr...
nKeepAlive	5			in seconds
sUserName	*			optional ...
sUserPassword	*			optional ...
stWill				optional ...
stTLS				optional ...
ipMessageQueue	16#00007F9C5F70B148			
bError	FALSE			
hrErrorCode	0			
eConnectionState	MQTT_ERR_SUCCESS			
bConnected	TRUE			
MessageQueue				
Message				
sWillingMessage				
WillingPayload	'PLC offline'			
run	FALSE			控制标志
bDataPublished	TRUE			发布
bConnect	TRUE			连接

- Python 端将收到主题为 plc/data 的消息，即转换后的 Json 文本并输出
- 这是 Python 端收到的前 200 个字符

```

Console 1/A X
client = mqtt.Client(
开始监听MQTT主题 plc/data, 连接服务器: 192.168.105.16...
Connected to MQTT broker with code: 0

Raw message length: 483 bytes
Sample data (first 200 chars):
{"data":{"bEnabled":true,"nCounter":42,"fTemperature":23.5,"sName":"SampleDevice","arrValues":[10,20,30],"arrMatrix":
[1.100000023841858,1.2000000476837158,1.2999999523162842,2.0999999046325684,2.200000...

```

- 这是 Python 端收到的完整的 Json 数据

```

完整JSON数据结构:
{
  "data": {
    "bEnabled": true,
    "nCounter": 42,
    "fTemperature": 23.5,
    "sName": "SampleDevice",
    "arrValues": [
      10,
      20,
      30
    ],
    "arrMatrix": [
      1.100000023841858,
      1.2000000476837158,
      1.2999999523162842,
      2.0999999046325684,
      2.200000047683716,
      2.299999952316284
    ],
    "dtTimestamp": 1684161000,
    "tDuration": 3723000,
    "stSubItem": {
      "bStatus": false,
      "nValue": 99,
      "sDescription": "Nested item"
    },
    "arrStructs": [
      {
        "bStatus": true,
        "nValue": 100,
        "sDescription": "Array item 1"
      },
      {
        "bStatus": false,
        "nValue": 200,
        "sDescription": "Array item 2"
      }
    ]
  }
}

```

- 这是 Python 端处理后的数据

```

基本数据:
• 设备名称: SampleDevice
• 使能状态: ON
• 计数器值: 42
• 温度: 23.50°C

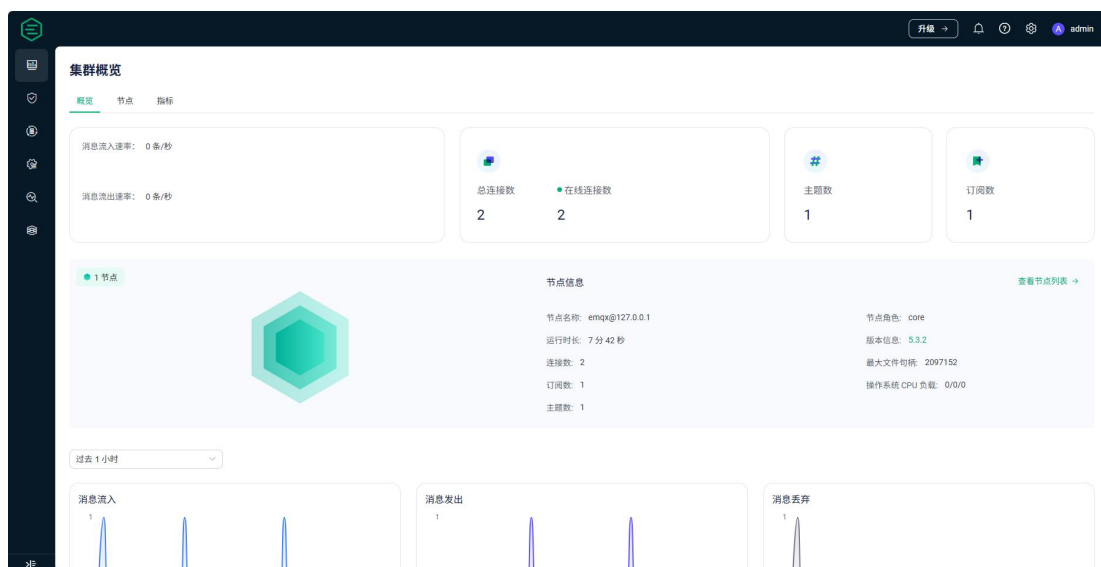
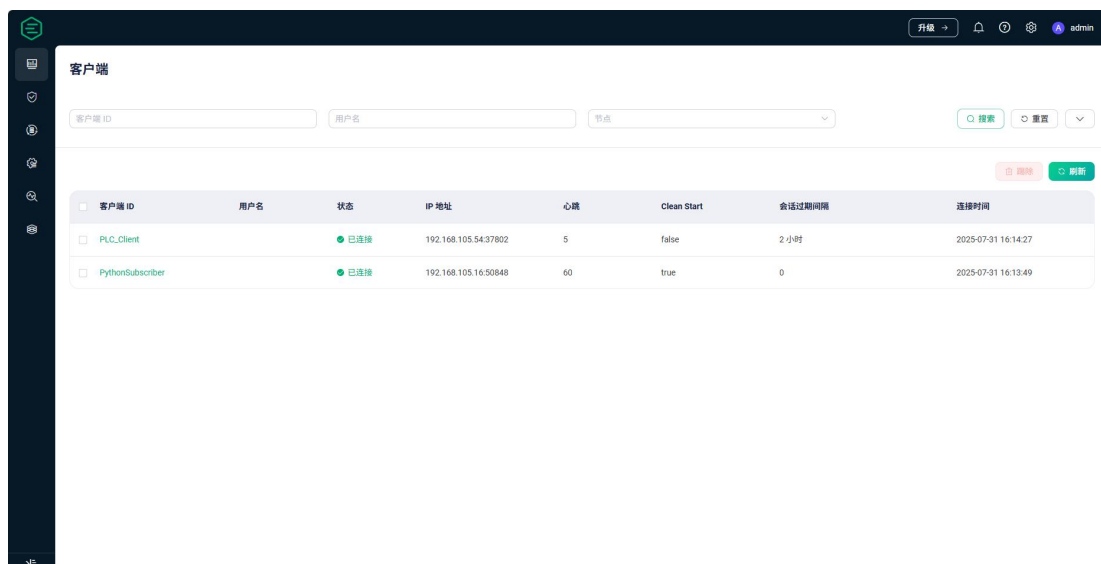
数组数据:
• 一维数组: [10, 20, 30]
• 二维数组: [1.100000023841858, 1.2000000476837158, 1.2999999523162842, 2.0999999046325684, 2.200000047683716, 2.299999952316284]

时间数据:
• 时间戳 (Unix): 1684161000 → 2023-05-15T22:30:00
• 持续时间 (秒): 3723000

结构体数据:
• 子结构体: Status=False, Value=99, Desc='Nested item'
• 结构体数组:
  [0] Status=True, Value=100, Desc='Array item 1'
  [1] Status=False, Value=200, Desc='Array item 2'

```

- 可以看到 MQTT 服务端的连接情况，此时有两个客户端连接，分别是工智机端和 Python 端
- 主题数为 1，即工程中发布消息的主题 plc/data
订阅数为 1.即 Python 端订阅了 plc/data 这个主题

客户端

客户端 ID: 用户名: 节点:

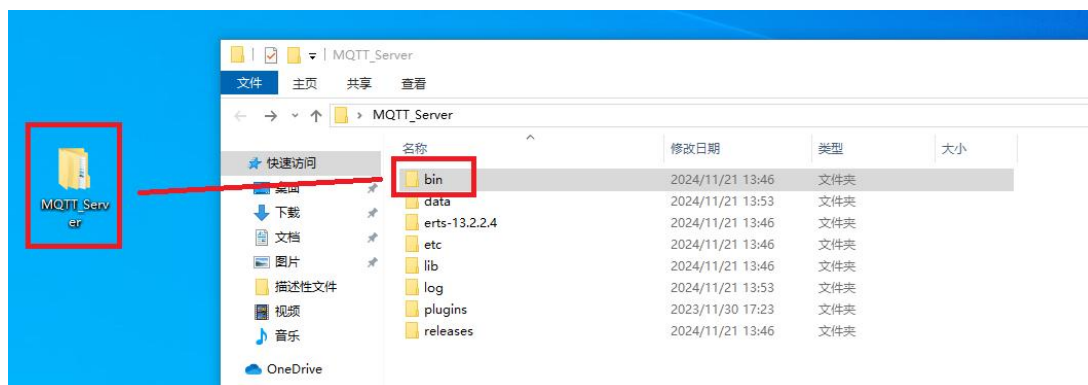
客户端 ID	用户名	状态	IP 地址	心跳	Clean Start	会话过期间隔	连接时间
<input type="checkbox"/> PLQ_Client		已连接	192.168.105.54:37802	5	false	2 小时	2025-07-31 16:14:27
<input type="checkbox"/> PythonSubscriber		已连接	192.168.105.16:50848	60	true	0	2025-07-31 16:13:49

3. 实验注意点

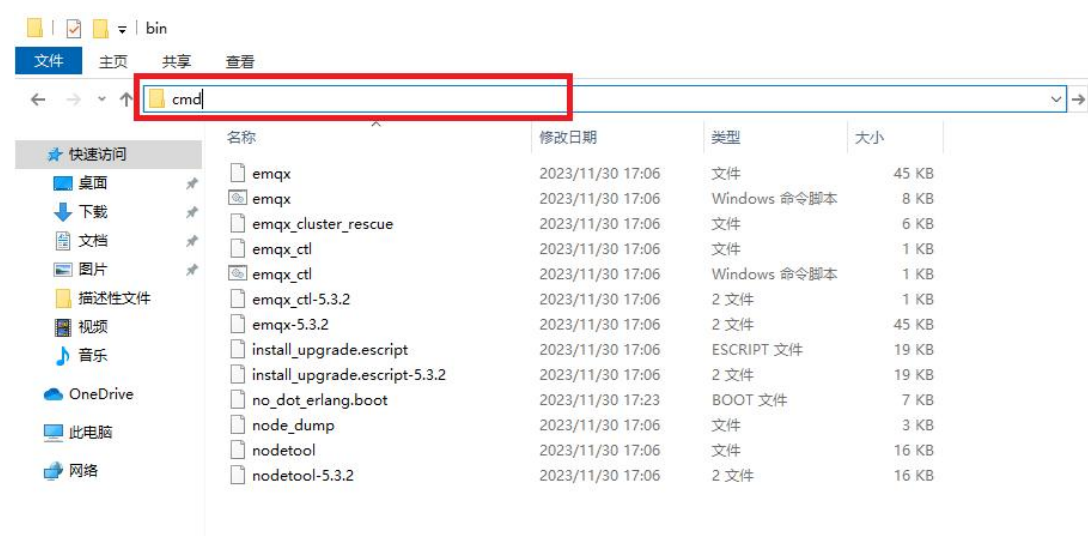
- 部分实验有先后顺序要求，建议按照文档中的顺序依次实验

3.1. 安装 EMQX(MQTT Broker)

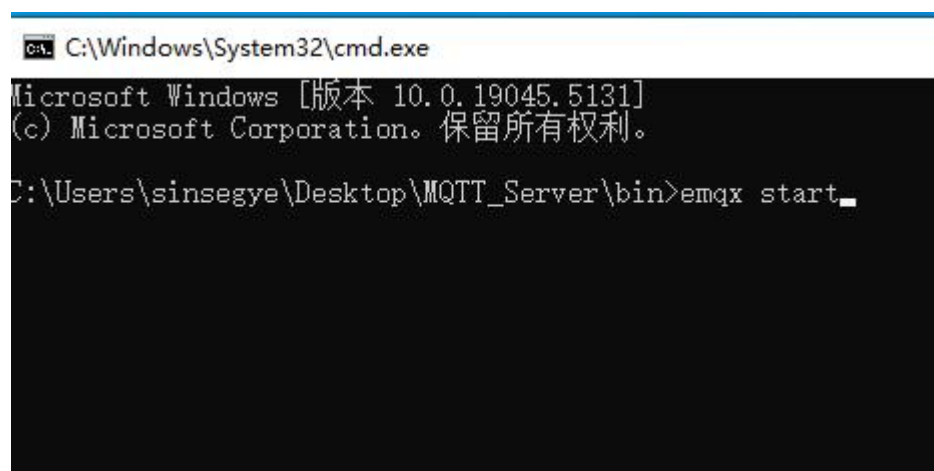
- 将官网下载的 emqx-5.3.2-windows-amd64.zip 进行解压缩，自行解压到相应的文件夹中。



- 打开文件夹目录 bin，在文件夹位置处输入“cmd”，回车。



- 在命令行中输入——>emqx start——>回车



- 启动完成


```

C:\Windows\System32\cmd.exe

Microsoft Windows [版本 10.0.19045.5131]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\sinsegye\Desktop\MQTT_Server\bin>emqx start
EMQX_NODE__DB_ROLE [node.role]: core
EMQX_NODE__DB_BACKEND [node.db_backend]: mnesia
C:\Users\sinsegye\Desktop\MQTT_Server>_

```

- EMQX 端口号:

端口号	说明
1833	TCP 端口
8883	WebSocket 端口
8884	WebSocket Secure 端口
8883	SSL/TLS 端口
18083	Broker 的 Dashboard 访问端口

- 可以通过 18083 端口访问 MQTT Broker 可视化界面

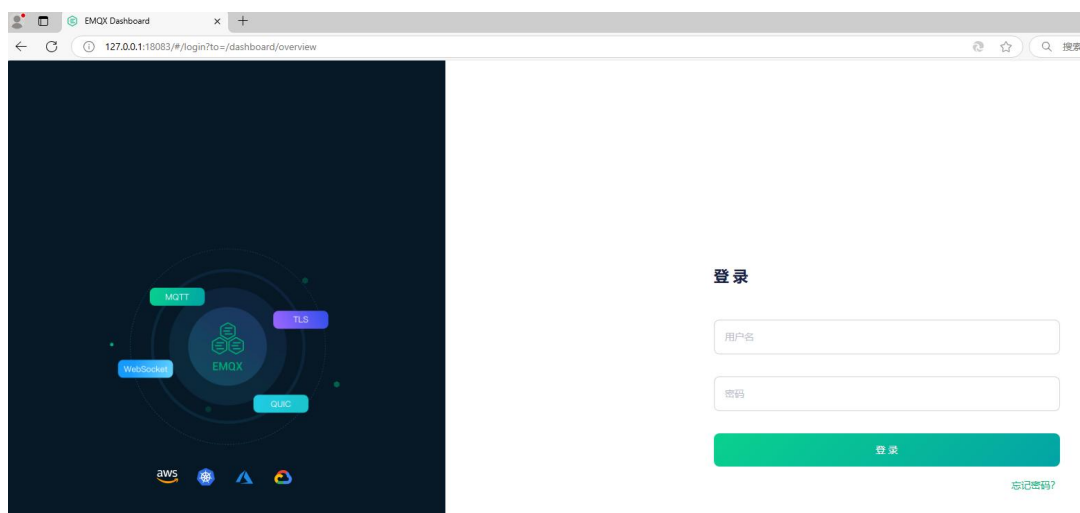
本机访问: 127.0.0.1:18083/

外部访问: 192.168.110.147:18083/

(本例 MQTT Broker 使用的是 127.0.0.1:18083/)

用户名: admin

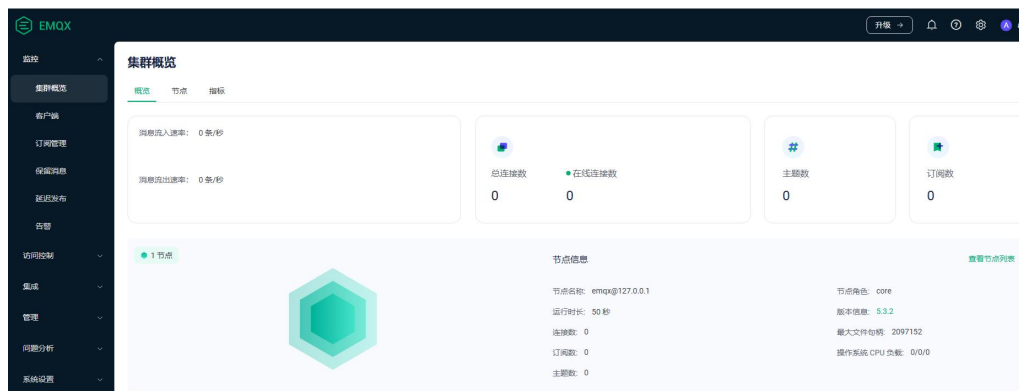
密码: public



- 首次登录会提示修改密码，也可以选择跳过。



- Dashboard 界面，可以进行 Broker 的设置以及查看客户端连接情况、消息订阅等等。



五、功能介绍

1. 结构体转为 Json

1.1. 功能块方法 FB_JsonReadWriterDataType.AddJsonKeyValueFromSymbol 介绍



1.2. 参数介绍

- 输入参数

参数名称	参数类型	描述
fbWriter	FB_JsonSaxWriter	JSON 文档写入器实例
sKey	STRING	要添加的 JSON 字段名称（键）
sDatatype	STRING	PLC 变量的数据类型名称（如"ST_SampleData"）
nData	UINT	目标变量的内存大小（字节数）
pData	POINTER TO UDINT	指向源变量的内存地址

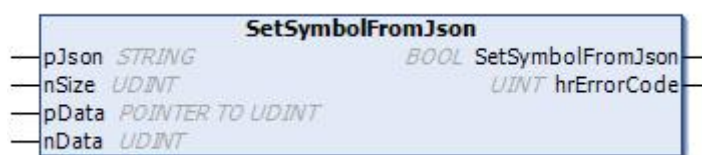
- 输出参数

参数名称	参数类型	描述
AddJsonKeyValueFromSym	BOOL	当功能块被激活时，被置为 True，一直保持到收到确认信号

bol		
hrErrorCo de	UINT	返回操作结果状态码

2. Json 转为结构体

2.1. 功能块方法 FB_JsonReadWriterDataType.SetSymbolFromJson 介绍



2.2. 参数介绍

- 输入参数

参数名称	参数类型	描述
pJson	STRING	包含要解析的 JSON 文本
nSize	UDINT	指定 pJson 字符串的实际长度（字节数）
pData	POINTER TO UDINT	指向目标变量的内存地址
nData	UINT	指定目标变量的内存大小（字节数）

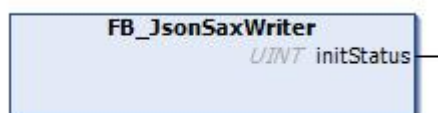
- 输出参数

参数名称	参数类型	描述
SetSy mbolFr omJso n	BOOL	

hrError Code	UINT	返回操作结果状态码
-----------------	------	-----------

3. 生成 Json 文档

3.1. 功能块 FB_JsonSaxWriter 介绍



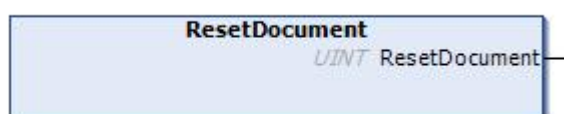
3.2. 功能介绍

FB_JsonSaxWriter 是用于流式生成 JSON 文档的功能块，采用 SAX (Simple API for XML/JSON) 模型，特别适合高效处理大尺寸数据或动态构建复杂 JSON 结构。其核心特性包括：

- 低内存消耗：通过逐步生成 JSON 内容，避免一次性加载全部数据到内存；
- 全面数据类型支持：可处理嵌套对象/数组及 PLC 特殊类型（如 TIME、DATE_AND_TIME）；
- 精准状态控制：通过 initStatus 状态码实时监控初始化与执行状态；
- 灵活文档构建：提供 AddString 等方法，支持结构化生成 JSON。

4. 清空 Json 文档

4.1. 功能块方法 FB_JsonSaxWriter.ResetDocument 介绍



4.2. 功能介绍

ResetDocument 是 FB_JsonSaxWriter 功能块的核心方法，用于清空当前 JSON 文档并重置生成器状态，为构建新 JSON 文档做准备。

特性	说明
初始化作用	清除内存中的 JSON 数据，将生成器恢复到初始状态
必须首调用	在开始构建新 JSON 文档前必须调用
不影响配置	保留之前设置的参数（如浮点数精度）
无参数设计	直接调用，无需输入/输出参数

5. 获取 Json 文档长度

5.1. 功能块 FB_JsonSaxWriter.GetDocumentLength 介绍



5.2. 功能介绍

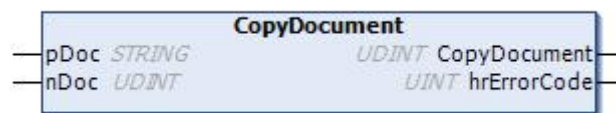
GetDocumentLength 用于 获取当前生成的 JSON 文档的精确字节长度（不含终止符\0），是内存管理和安全传输的关键方法。

- 输出参数

参数名称	参数类型	描述
GetDocumentLength	UINT	返回当前 JSON 文档的长度（字节数）
hrErrorCode	UINT	返回操作结果状态码

6. 复制 Json 文档

6.1. 功能块 FB_JsonSaxWriter.CopyDocument 介绍



6.2. 功能介绍

CopyDocument 用于 将生成的 JSON 文档复制到指定的字符串缓冲区，是获取最终 JSON 数据的核心方法。

- 输入参数

参数名称	参数类型	描述
pDoc	STRING	目标缓冲区（接收 JSON 文本）
nDoc	UDINT	目标缓冲区的最大容量（需用 SIZEOF(buffer)获取）

- 输出参数

参数名称	参数类型	描述
CopyDocument	UDINT	返回值
hrErrorCode	UINT	返回操作结果状态码

六、附录

1. Linux 指令安装/卸载指南

1.1. 安装要求

- 中科时代出厂的工控机;
- 熟悉基础的 Linux 操作命令; 、

开始安装前, 请熟读 [linux 基础操作](#) 中的操作示例

1.2. 安装过程

工控机端安装 SF4601_JsonDom RTE 组件

- 上传 deb 包到工控机 Linux 环境的/home/sinsegye 目录下
- 上传完成后在工控机上执行命令安装 (参考下方截图, 如果模块文件名发生变化则命令行中的文件名做相应更改)

```
Shell
cd $HOME
sudo dpkg -i SF4601_JsonDom_1.0.3_amd64.deb
```

```
sinsegye@sinsegye-sx21:~$ cd $HOME
sinsegye@sinsegye-sx21:~$ sudo dpkg -i SF4601_JsonDom_1.0.3_adm64.deb
```

- 修改 RTE 的配置文件, ComponentManger 模块下加入 SF4200

```
Shell
sudo nano /usr/local/etc/SinsegyeRTE/SinsegyeRTE.cfg
```

```
Shell
[ComponentManager]
Component.0=retainDeamon
Component.1=CmpCanBusUtils
Component.2=CmpSinsegyeLibs
Component.3=SinsegyeCmp
Component.4=jsondom
```

- 重启 RTE 服务, 使新加入的 JsonDom 被调用

```
Shell
sudo systemctl restart sinsegyerte.service
```


1.3. 更新安装

工智机端升级 JsonDom RTE 组件

- 上传升级版 deb 包到工智机 Linux 环境的/home/sinsegye 目录下, 上传方法参考附录;
- 上传完成后在工智机上执行命令安装 (参考下方截图, 如果模块文件名发生变化则命令行中的文件名做相应更改)

```
Shell
cd $HOME
sudo dpkg -i SF4601_JsonDom_1.0.3_amd64.deb
```

- 重启 RTE 服务, 使新升级的的 JsonDom 被调用

```
Shell
sudo systemctl restart sinsegyerte.service
```

1.4. 卸载过程

卸载工智机 JsonDom RTE 组件

- 工智机上执行命令卸载

```
Shell
sudo dpkg -r jsondom
```

- 修改 RTE 的配置文件, ComponentManger 模块下去掉 jsondom

```
Shell
sudo nano /usr/local/etc/SinsegyeRTE/SinsegyeRTE.cfg
```

- 重启 RTE 服务

```
Shell
sudo systemctl restart sinsegyerte.service
```

2. 支持与服务

中科时代为公司产品及解决方案提供全方位支持与服务, 确保针对相关问题给予快速且专业的响应。

资料下载

我们的资料下载专区涵盖了丰富的文件资源，包括应用案例、技术文档、产品介绍等，满足您的多样化需求。

资料下载地址：<https://help.sinsegye.com.cn/>

获取支持

如需中科时代产品的本地支持与服务，请随时联系我们。您可以通过访问我们的官方网站 www.sinsegye.com.cn，查找中科时代的分公司地址，并获取更多关于中科时代的信息。

此外，您还可以通过以下方式联系我们：

- 热线电话：400-013-2158
- 邮箱地址：support@sinsegye.com.cn