# APRUS Lua-ModbusServer 配置说明

# 第一章 概述

## 1.1 文档说明

APRUS 适配器可配置为 Modbus 服务器，支持与其他支持 Modbus 协议的客户端设备进行通信。APRUS Lua-ModbusServer 主要包括两个文件：aprus.lua 和 config.lua。配置为 Modbus 服务器后，用户可将设备运行的过程数据和结果放置到服务器的数据中，供其他设备读取和写入。本文将介绍如何将 APRUS 适配器可配置为 Modbus 服务器，并解释各参数的含义。

## 1.2 参考资料

[1].《APRUS Lua-数据中心 配置说明》
[2].《APRUS Lua-AXIO 配置说明》

# 第二章 配置说明

## 2.1 apurs.lua 配置

说明：function start() 为主流程入口函数

```
package.cpath="./?.so"
package.cpath="./?.so"
package.path="./?.lua"
cjson = require "cjson"
config = require "config"
require "AXio"
require "ModbusServer"

function ModbusServer_init()
    mbSvrObj = ModbusServer.new("ModbusServer")
    local MBCfg = cjson.encode(config.ModbusServer.ServerConfig)
    local ret = ModbusServer.config(mbSvrObj, MBCfg)
    --ModbusServer.debug(mbSvrObj, 0x01010101)   --ModbusServer 显示详尽的调试信息
end

function MBserver_load_station(svr, cfg)
    for k,v in pairs(cfg)
    do
        print("load Station" .. cjson.encode(v))
```

```
            ModbusServer.setStation(svr, cjson.encode(v))
      end
end

function updateU16(num16)
            --小于 65535 的数，只需要保存在一个数据格子内就可以了。
            DataCenter.DBsetRecord(dbAXio_AI, 17 , num16)
end

function updateU32(num32)
            --大于 65535 的数，需要分段保存在两个或更多个数据格子内。
            --num32 = 0x12345678
            tl = math.floor(num32 % 0x10000)        --取余
            th,f = math.modf(num32 / 0x10000)       --取整
            DataCenter.DBsetRecord(dbAXio_AI, 16 , th)
            DataCenter.DBsetRecord(dbAXio_AI, 17 , tl)
end

function act_control(m, json)
      for k,v in pairs(json) do
            if k ~= "Act" then
            AXio.write(axioObj, k, v)
            end
      end
end

function mqttdata_handle(m, topic, data)
      local json = cjson.decode(data)
      if json.Act == "Control" then
            act_control(m, json)
      end
end

function mqttsys_handle(m, code)
      if code == 0 then
--          AXio.stop(axioObj)
      elseif code == 1 then
            --AXio.run(axioObj)
      end
end

function ioevent_handle(session, code, style_L, val_L, style_E, val_E, z)
      if code == 15 then
            local json = cjson.encode(iobuf[session])
            mqtt.publish(mqtt30bj, iostr[session], "r", json)
            iobuf[session] = {}
```

```
	elseif code > 10 then
		iobuf[session][style_L] = val_L
		iobuf[session][style_E] = val_E
	else
		local json = cjson.encode({[style_L] = val_L, [style_E] = val_E, ["Z"] = z})
		mqtt.publish(mqtt3Obj, iostr[session], "r", json)
	end
end

function io_load_nodes(session, nodes)
	for k,v in pairs(nodes)
	do
		AXio.add_node(session, v.m, v.index, v.interval, v.dtype, v.cntmode, v.pMode, v.dStyle, v.dOffset, v.dExt)
		--AXio.add_node(session, v.m, v.index, v.direction, v.interval, v.dtype, v.cntmode, v.pMode, v.dStyle,
v.dOffset, v.dExt)
	end
end

function loadDatabase(dbs)
	for k,v in pairs(dbs)
	do
		DataCenter.addDB(cjson.encode(v))
	end
end

function init()

	mqtt3Obj = mqtt.new()
	user.setluaver(config.AprusX.luaver)
	user.setdevinfo(config.AprusX.devinfo)
	user.ipconfig(config.AprusX.ipmode, config.AprusX.inet_addr, config.AprusX.netmask)
end

function start()
	print("<--------------------user lua start------------------------>")
	init()
	print("<--------------------DataCenter_load_database()------------------------>")
	loadDatabase(config.DataCenter)

	IOCfg = config.AXIO
	axioObj = nil
	iobuf = {axioObj}
	iostr = {axioObj}

	print("<--------------------AXio.new()------------------------>")
	axioObj = AXio.new()
```

```
    dbAXio_AI = DataCenter.getDB("AXio_AI")
    DataCenter.DBresize(dbAXio_AI,20)
    iobuf[axioObj] = {}
    iostr[axioObj] = "io"
    print("<--------------------io_load_nodes()------------------------>")
    io_load_nodes(axioObj, IOCfg.node)

    print("<--------------------ModbusServer_init()------------------------>")
    ModbusServer_init()
    print("<--------------------ModbusStation_init()------------------------>")
    MBserver_load_station(mbSvrObj, config.ModbusServer.Stations)
    print("<--------------------ModbusServer.run------------------------>")
    ModbusServer.run(mbSvrObj)
    AXio.run(axioObj)
    mqtt.run(mqtt3Obj)

    counter = 0
    while true do
        local msg = user.waitmsg()
        if msg.from == "mqtt-sys" then
            mqttsys_handle(msg.session, msg.code)
        elseif msg.from == "mqtt-msg" then
            mqttdata_handle(msg.session, msg.topic, msg.payload)
        elseif msg.from == "io" then
            ioevent_handle(msg.session, msg.code, msg.style_L, msg.val_L, msg.style_E, msg.val_E, msg.z)

            counter = counter+1
            DataCenter.DBsetRecord(dbAXio_AI, 18 , counter)
            --updateU32（counter）
            --print("update counter" .. counter)
        end
    end
end
start()
```

## 2.2 此处仅解释一下 ModbusServer 相关的函数和操作：

### 2.2.1 ModbusServer.new

功能：创建 ModbusServer 实例

示例：mbSvrObj = ModbusServer.new("ModbusServer")

| 参数 | 值 | 说明 |
|---|---|---|
| "ModbusServer" | | 模块运行实例的名称 |

### 2.2.2 ModbusServer.config

功能：配置 ModbusServer 实例

示例：ModbusServer.config(mbSvrObj, MBCfg)

| 参数 | 值 | 说明 |
|---|---|---|
| mbSvrObj | | ModbusServer. new 实例的返回值 |
| MBCfg | cjson.encode(config.ModbusServer.ServerConfig) | ModbusServer 配置参数（详见 config.lua） |

### 2.2.3 MBserver.setStation

功能：配置 Modbus 站点信息

示例：ModbusServer.setStation(mbSvrObj, cjson.encode(stationConfig))

| 参数 | 值 | 说明 |
|---|---|---|
| mbSvrObj | | ModbusServer. new 实例的返回值 |
| stationConfig | | ModbusServer 的站点配置（详见 config.lua） |

### 2.2.4 ModbusServer.run

功能：创建 ModbusServer 启动实例

示例：ModbusServer.run(mbSvrObj)

| 参数 | 值 | 说明 |
|---|---|---|
| mbSvrObj | | ModbusServer. new 实例的返回值 |

### 2.2.5、 MBserver_load_station(mbSvrObj, config.ModbusServer.Stations)

使 ModbusServer 加载 config.lua 中 ModbusServer.Stations 段配置的 ModbusServer 服务器。

函数原型如下（使用循环方式添加 config.lua 文件中 Stations 段的每一行）：

```
function MBserver_load_station(svr, cfg)
    for k,v in pairs(cfg)
    do
        print("load Station" .. cjson.encode(v))
        ModbusServer.setStation(svr, cjson.encode(v))
    end
end
```

### 2.2.6、ModbusServer.debug(mbSvrObj, 0x01010101)

启用详细的 Modbus 服务器调试信息。第一个参数为 ModbusServer. new 实例的返回值，第二个参数的值为 32 位十六进制数字，这个数值分成 4 个字节，单独每个字节为 1 时开启对应功能，为 0 时关闭对应功能。从低位到高 4 个字节的功能分别是：错误信息、连接信息和一般信息、接收内容详情、发送内容详情。此设置可以帮助您了解设备的运行情况，排查错误，优化性能。请根据需要选择要开启的调试信息级别。

例：ModbusServer.debug(mbSvrObj, 0x01000101)

LOG 中打印错误信息、一般信息和发送内容详情。

# 3 config.lua 配置说明

对于 ModbusServer 的 config.Lua 一般需要配置两大部分：**ModbusServer 参数（ServerConfig 段）、站点配置参数（Stations 段）**

config.lua 示例：

```
ModbusServer=
    {
            ServerConfig=
            {
                ifname="eth0",
                ip="192.168.123.88",
                port = 502,
            },
            Stations=
            {
               {station=1,db1="DB_1_100",db2="DB_2_100",db3="DB_3_100",
  db4="DB_4_100"},
                {station=2, db1="AXio_DO", db2="AXio_DI", db4="AXio_AI"},
                {station=8, db1="DB_1_100", db3="DB_3_100"},
            },
    },
```

3.1 ModbusServer 参数（ServerConfig 段）

ServerConfigr 段参数的功能如下表所示：

| 参数 | 值 | 说明 |
|---|---|---|
| ifname | " eth0" | 绑定物理网口到 xxx（非必要）。可配置有 usb0, eth0, eth0:n, eth1, wlan0。配置成非 eth0:n 时，请不要同时配置 IP 地址 C_ip="xxx"。如果不配置，使用任意可用网络连接 |
| ip | " 192.168.123.88" | 服务器 IP 地址（如果相应的物理网口已有地址，此处可不填写） |
| port | 502 | Modbus 服务端口 |

3.2 站点配置参数（Stations 段）

| 参数 | 值 | 说明 |
|---|---|---|
| station | 1 | station=站号（必要），对应 ModbusServer 的 ID 号 |
| db1 | " DB_1_100 " | 可读写开关量，对应功能码 01（非必要） |
| db2 | " DB_2_100" | 只读开关量，对应功能码 02（非必要） |
| Db3 | " DB_3_100 " | 可读写 16 位数据，对应功能码 03（非必要） |
| Db4 | "DB_4_100 " | 只读 16 位数据，对应功能码 04（非必要） |

config.lua 中每一行 station xxxxxx 表示新建一个站点，db1、db2、db3、db4 对应 modbus 的 01、02、03、04 四种功能码。如果不想提供 01 功能码，则 db1=xxx 这段不填就可以。相反，如果想实现全部 4 种功能码则需全部填写。db1=xxx 这个值来自于数据中心内的数据库的名称。服务器启动

时会到数据中心里查找对应名称的数据库。如果相应的数据库没有定义，则无法实现对应的功能码。另外，Modbus 服务器的站点对数据库的单元长度是有要求的。db1 和 db2 要求其数据库的单元长度为 1 字节，db3 和 db4 要求其数据库的单元长度为 2 字节。配置错误会产生无法预料的错误。

数据中心相关知识，请参考《APRUS 编程手册_数据中心.pdf》。