APRUS Lua-断网续传 配置说明

概述

断网续传是 Aprus 适配器支持的功能之一；用于在网络状态不佳的情况下先保存采集到的点位数据，并在网络状态良好的情况下上传这些数据到云端，以此保证数据的连续性。

断网续传功能依赖于 httpsqs 数据库，aprus 适配器通过 Lua 接口和 httpsqs 数据库进行数据交互。若 mqtt 服务器断开链接时，向 httpsqs 发起请求保存数据；若 mqtt 服务器链接成功则向 httpsqs 发起请求获取数据，并上报给 mqtt 服务器。由于 httpsqs 数据库的存在，在 aprus 适配器断电的情况下，数据也能够保存下来，并在网络状态良好的情况下上传数据到云端。

1.1 httpsqs API

1.1.1 httpsqs .new

功能	创建 httpsqs 数据库实例对象
接口描述	httpsqsObj = httpsqs .new()
httpsqsObj	httpsqs .new()返回的实例对象

1.1.2 httpsqs .run

功能	运行 httpsqs 实例对象
接口描述	httpsqs.run(httpsqsObj)
httpsqsObj	httpsqs .new()返回的实例对象

1.1.3 httpsqs .maxnum

功能	设置 httpsqs 数据库最大队列数量; 需要在 httpsqs.run 之后运行
接口描述	httpsqs.maxnum(httpsqsObj, "qnname ", 1000)
qnname	<字符串>数据库队列名称
1000	要设置的数据库的队列数量

1.1.4 httpsqs .get

功能	获取数据库队列的数据

接口描述	httpsqs_data=httpsqs.get(httpsqsObj,"qnname ")
httpsqs_data	从数据库队列中获取到的数据
httpsqsObj	httpsqs .new()返回的实例对象
"qnname "	<字符串>数据库队列名称

1.1.4 httpsqs .put

功能	向数据库队列中存储数据
接口描述	httpsqs.put(httpsqsObj, cjson.encode(config.Httpsqs1))
httpsqsObj	httpsqs .new()返回的实例对象
cjson.encode(config.Httpsqs1)	将 config.Httpsqs1 打包为 json 格式； config.Httpsqs1 为 lua 的表，具体的参数如下 Httpsqs1 = { qnname = "test_qnname_1", data = "hello" }, qnname 为数据库队列名称 data 为要存储的数据

1.1.5 httpsqs .debug

功能	设置数据库 LOG 打印级别
接口描述	httpsqs.debug(httpsqsObj, 0x01010101)
httpsqsObj	httpsqs .new()返回的实例对象
0x01010101	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。

1.1.5 httpsqs .DBdebug

功能	设置数据库内部 LOG 打印级别
接口描述	httpsqs.DBdebug(httpsqsObj, 0)
httpsqsObj	httpsqs .new()返回的实例对象

0	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。

2 APRUS.lua 配置说明

```
package.cpath="./?.so"
package.cpath = "./?.so"
package.path = "./?.lua"
cjson = require "cjson"
config = require "config"
require "modbus"
require "httpsqs"

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

mqtt_connect = 0 --0 断开；1 链接
function mqttsys_handle(m, code)
	if code == 0 then
		config.logInfo.payload = "==================Mqtt Disconnect=================="
		user.log(cjson.encode(config.logInfo))
		mqtt_connect = 0
	elseif code == 1 then
		mqttDB_name = "httpsqs_DB"
		while true do
			mqtt_conndata = httpsqs.get(httpsqsObj, config.Httpsqs1.qnname)
			if mqtt_conndata then
				mqtt.publish(mqtt3Obj, mqttDB_name, "r", mqtt_conndata)
			else
				config.logInfo.payload = "=====Mqtt Connected and httpsqs DB is Empty===="
				user.log(cjson.encode(config.logInfo))
				break;
			end
		end
		mqtt_connect = 1
	end
end

function modbusevent_handle(session, code, style_L, val_L, style_E, val_E, z)
```

```
	if code == 15 then
		local json = cjson.encode(mbbuf[session])
		if mqtt_connect == 0 then
			config.Httpsqs1.data = json
			httpsqs.put(httpsqs0bj, cjson.encode(config.Httpsqs1))
		elseif mqtt_connect == 1 then
			mqtt.publish(mqtt30bj, mbstr[session], "r", json)
			mbbuf[session] = {}
		end
	elseif code > 10 then
		mbbuf[session][style_L] = val_L
		mbbuf[session][style_E] = val_E
	else
		local json = cjson.encode({ [style_L] = val_L, [style_E] = val_E, ["Z"] = z })
		if mqtt_connect == 0 then
			config.Httpsqs1.data = json
			httpsqs.put(httpsqs0bj, cjson.encode(config.Httpsqs1))
		elseif mqtt_connect == 1 then
			mqtt.publish(mqtt30bj, mbstr[session], "r", json)
			mbbuf[session] = {}
		end
	end
end

function modbus_load_collectnodes(session, nodes)
	for k, v in pairs(nodes)
	do
		modbus.add_collectnode(session, v.ID, v.reg, v.addr, v.cnt)
	end
end

function modbus_load_varnodes(session, nodes)
	for k, v in pairs(nodes)
	do
		modbus.add_varnode(session, v.ID, v.reg, v.addr, v.dtype, v.dBit, v.format, v.len, v.pMode, v.dStyle,
v.dOffset,
			v.dExt)
	end
end

function init()
	mqtt30bj = mqtt.new()
	mqtt.config(mqtt30bj, "", "gards.mixyun.com", "31883", "aprusdev@aprus", "mixlinker123") --V8
	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()
    init()
    ----------ModbusTcp Config----------
    MBCfg = config.Modbus

    --Init Object
    modbusObj = nil
    mbbuf = { modbusObj }
    mbstr = { modbusObj }

    --Config Object
    modbusObj = modbus.new(MBCfg.device.type)
    mbbuf[modbusObj] = {}
    mbstr[modbusObj] = "modbusObj"
    modbus.config(modbusObj, MBCfg.device.ip, MBCfg.device.port, MBCfg.device.ifname)
    --modbus.delay(modbusObj, 50)     --设置采集间隔。默认值为 20mS --必须在添加采集节点前调用
    modbus_load_collectnodes(modbusObj, MBCfg.node.collect)
  modbus_load_varnodes(modbusObj, MBCfg.node.variable)
    modbus.debug(modbus3Obj, 0x01010101)
    ----------------------------------
    mqtt.run(mqtt3Obj)

    modbus.run(modbusObj)


    print("<--------------------httpsqs func------------------------>")
    httpsqsObj = httpsqs.new()
    -- httpsqs.debug(httpsqsObj, 0x01010101)
    httpsqs.DBdebug(httpsqsObj, 0)
    -- httpsqs.config(httpsqsObj, mqtt3Obj, config.Httpsqs1.qnname)
    httpsqs.run(httpsqsObj)

    httpsqs.maxnum(httpsqsObj, config.Httpsqs1.qnname, 1000)
    print("<--------------------httpsqs run------------------------>")



    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 == "modbus" then
```

```
            modbusevent_handle(msg.session, msg.code, msg.style_L, msg.val_L, msg.style_E, msg.val_E, msg.z)
        end
    end
end

start()
```

2.1 此处仅解释一下断网续传相关的函数和操作：

1、mqtt 链接状态函数
mqtt_connect = 0 --0 断开；1 链接

```
function mqttsys_handle(m, code)
    if code == 0 then
        config.logInfo.payload = "====================Mqtt Disconnect===================="
        user.log(cjson.encode(config.logInfo))
        mqtt_connect = 0
    elseif code == 1 then
        mqttDB_name = "httpsqs_DB"
        while true do
            mqtt_conndata = httpsqs.get(httpsqsObj, config.Httpsqs1.qnname)
            if mqtt_conndata then
                mqtt.publish(mqtt3Obj, mqttDB_name, "r", mqtt_conndata)
            else
                config.logInfo.payload = "=====Mqtt Connected and httpsqs DB is Empty===="
                user.log(cjson.encode(config.logInfo))
                break;
            end
        end
        mqtt_connect = 1
    end
end
```

mqttsys_handle 函数为 mqtt 断开链接时会执行，code==0 为 mqtt 断开链接，将标志位置 0；code==1 为 mqtt 链接成功，将标志位置 1，并调用 httpsqs.get 函数从数据库拿数据，拿到数据后调用 mqqt.publish 发送数据；

2.modbus 协议产生数据函数（以 modbus 为例，其他协议同理）

```
function modbusevent_handle(session, code, style_L, val_L, style_E, val_E, z)
```

```
    if code == 15 then
        local json = cjson.encode(mbbuf[session])
        if mqtt_connect == 0 then
            config.Httpsqs1.data = json
            httpsqs.put(httpsqsObj, cjson.encode(config.Httpsqs1))
        elseif mqtt_connect == 1 then
            mqtt.publish(mqtt3Obj, mbstr[session], "r", json)
            mbbuf[session] = {}
        end
    elseif code > 10 then
        mbbuf[session][style_L] = val_L
        mbbuf[session][style_E] = val_E
    else
        local json = cjson.encode({ [style_L] = val_L, [style_E] = val_E, ["Z"] = z })
        if mqtt_connect == 0 then
            config.Httpsqs1.data = json
            httpsqs.put(httpsqsObj, cjson.encode(config.Httpsqs1))
        elseif mqtt_connect == 1 then
            mqtt.publish(mqtt3Obj, mbstr[session], "r", json)
            mbbuf[session] = {}
        end
    end
End
```

在 modbus 采集到数据即将上报时会调用该函数，在原有 modbus 协议基础上，增加判断 mqtt 链接状态（状态位由 mqttsys_handle 函数置 0 或 1），若 mqtt 链接成功则之间调用 mqtt.publish 发送数据；否则调用 httpsqs.put 函数将数据存起来，在 mqtt 链接成功时调用 httpsqs.get 函数从数据库拿数据，拿到数据后调用 mqqt.publish 发送数据。

3 config.lua 配置说明

```
return
{
	AprusX = {
		ipmode = "manual", --auto/manual/none
		inet_addr = "192.168.1.234",
		netmask = "255.255.255.0",
		luaver = "V00.R",
		devinfo = "ModbusTcpDev",
	},

	Httpsqs1 = {
		qnname = "test_qnname_1",
		data = "hello"
	},


	logInfo = {
		payload = "hello"
	},
	Modbus = {
		device = {
			type = "tcp",
			ip = "192.168.123.66",
			port = "1024",
			ifname = "eth1",
		},
		node = {
			collect = {
				{ ID = 1, reg = "1", addr = 0, cnt = 50 },
				-- { ID = 1, reg = "2", addr = 0, cnt = 50 },
				-- { ID = 1, reg = "3", addr = 0, cnt = 50 },
				-- { ID = 1, reg = "4", addr = 0, cnt = 50 },
			},
			variable = {
				{ ID = 1, reg = "1", addr = 0,  dtype = "bit", dBit = 0, pMode = { 1, 5 }, dStyle = { "L1_1_0" } },
				{ ID = 1, reg = "1", addr = 1,  dtype = "bit", dBit = 0, pMode = { 1, 5 }, dStyle = { "L1_1_1" } },
				{ ID = 1, reg = "1", addr = 2,  dtype = "bit", dBit = 0, pMode = { 1, 5 }, dStyle = { "L1_1_2" } },
				{ ID = 1, reg = "1", addr = 3,  dtype = "bit", dBit = 0, pMode = { 1, 5 }, dStyle = { "L1_1_3" } },
					},

		},
	},
}
```