APRUS Lua-PPI 配置说明

概述

本章主要为 APRUS Lua-PPI 协议的相关配置说明，该协议主要针对支持(西门子)PPI 协议的设备。APRUS 适配器可通过 PPI 协议与对接设备进行通信。而其中 APRUS 的 Lua 包含 apru.lua 和 config.lua 两个文件，客户只需配置 config.lua 就可以对支持 PPI 协议的设备进行数据采集。如需修改 APRUS.lua 文件的内容时，请咨询相关的技术人员，随意修改会导致适配器不能正常工作，所以此文档主要介绍 config.lua 内容。

1 APRUS.lua 配置说明

demo 示例

```
package.cpath="/home/zdw/test/lib/x86/output/lua-cjson/lib/?.so;./?.so"
package.path="./?.lua"
cjson = require "cjson"
config = require "config"
require "ppi"

function act_control(m, json)
    for k,v in pairs(json) do
        if k == "Act" then
            ppi.write(ppiobj, cjson.encode({["name"] = k, ["val"] = 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
        ppi.stop(ppiobj)
    elseif code == 1 then
        ppi.run(ppiobj)
    end
end

function ppi_handle(obj, name, code, data)
    mqtt.publish(m1, name, "r", data)
end

function ppi_load_collectnodes(obj, nodes)
    for k,v in pairs(nodes)
    do
        ppi.addcnode(obj,cjson.encode(v))
    end
end

function ppi_load_varnodes(obj, nodes)
    for k,v in pairs(nodes)
    do
        ppi.addvnode(obj,cjson.encode(v))
    end
end
```

```
function init()
    m1 = 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()
    init()
    ppiobj = ppi.new("ppi_1")
    ppi.config(ppiobj, cjson.encode(config.PPI.Device))
    ppi_load_collectnodes(ppiobj,config.PPI.ColNode)
    ppi_load_varnodes(ppiobj,config.PPI.VarNode)
    mqtt.run(m1)
    while true do
        local r = user.waitmsg()
        if r.from == "mqtt-sys" then
            mqttsys_handle(r.session, r.code)
        elseif r.from == "mqtt-msg" then
            mqttdata_handle(r.session, r.topic, r.payload)
        elseif r.from == "ppi" then
            ppi_handle(r.obj, r.name, r.code, r.data)
        end
    end
end
start()
```

内容较多，不逐行描述，仅对关键部分(已标红)进行解释

1.1 导入 ppi 协议支持库

语句	require "ppi"
说明	导入 ppi 协议支持库

1.2 加载 config.lua 配置文件

语句	config = require "config"
说明	导入 config.lua 中的配置信息(下一节介绍 config.lua)

1.3 创建 ppi 对象

语句	ppiobj = ppi.new("ppi_1")
说明	创建 ppi 对象，返回 ppiobj 供全局使用

1.4 配置 ppi 对象接口参数

语句	ppi.config(ppiobj, cjson.encode(config.PPI.Device))
说明	配置 ppi 通信接口 RS485 参数

1.5 添加 ppi 采集节点

语句	ppi_load_collectnodes(ppiobj,config.PPI.ColNode)
说明	添加 config.lua 中所配置的采集节点

1.6 添加 ppi 上报变量节点

语句	ppi_load_varnodes(ppiobj,config.PPI.VarNode)
说明	添加 config.lua 中所配置的变量节点

1.7 运行 ppi 实例

语句	ppi.run(ppiobj)
说明	在 mqtt 建立连接后 运行 ppi 实例 即开始采集+运算

1.8 暂停 ppi 实例

语句	ppi.stop(ppiobj)
说明	在 mqtt 连接断开时 暂停 ppi 实例

1.9 等待 ppi 事件

语句	local r = user.waitmsg()
说明	启动事件等待，该接口会返回全局各种事件，包括 ppi 对象事件

1.10 ppi 事件处理

语句	elseif r.from == "ppi" then ppi_handle(r.obj, r.name, r.code, r.data)
说明	当接收到 ppi 对象事件时，调用处理函数 ppi_handle

	r.obj 即 new 时所返回的对象 r.name 即 new 时所配置的对象名称 r.code 事件码 区分改变上报事件 周期上报事件等等 r.data 事件数据

1.11 mqtt 数据上报

语句	function ppi_handle(obj, name, code, data) mqtt.publish(m1, name, "r", data) end
说明	若无需特殊处理，则直接将事件数据通过 mqtt 发送 如需要二次处理可将 data 展开做分析

1.12 反向控制

语句	ppi.write(ppiobj, cjson.encode({["name"] = k, ["val"] = v}))
说明	反向控制时调用的 ppi 写入接口

2 config.lua 配置说明

Demo 示例：

```
APRUS={
        ipmode="manual",                        --auto/manual/none
        inet_addr="192.168.1.198",
        netmask="255.255.255.0",
        luaver="V00.R",
        devinfo="ppiDev",
},

PPI={
    Device={
        portindex=1,
        rate=9600,
        databit=8,
        stopbit=1,
        parity="Even",        -- None/Odd/Even
    },

    ColNode={
        {ID=2,reg="V",addr=100,ctype="bit",bitn=0,cnt=1},
        {ID=2,reg="V",addr=100,ctype="bit",bitn=1,cnt=1},
        {ID=2,reg="V",addr=100,cnt=50},
        {ID=2,reg="S",addr=0,cnt=30},
    },

    VarNode={
        {ID=2,reg="V",addr=100,ctype="bit",dtype="bit",bitn=0,cycle=0,name="L2_V_100_
    0"},
        {ID=2,reg="V",addr=100,ctype="bit",dtype="bit",bitn=1,cycle=5,name="L2_V_100_
```

```
    1"},
        {ID=2,reg="V",addr=101,dtype="bit",bitn=1, cycle=0.5,name="L2_V_101_1"},
        {ID=2,reg="V",addr=101,dtype="bit",bitn=2, cycle=0,name="L2_V_101_2"},
        {ID=2,reg="V",addr=101,dtype="bit",bitn=3, cycle=5,name="L2_V_101_3"},
        {ID=2,reg="V",addr=102,dtype="short",cycle=0.5,offset="*10;+5.123;~1",name="L
    2_V_102"},
        {ID=2,reg="S",addr=0,dtype="bytes",len=10, cycle=7,name="L2_S_0_bytes"}
    },
}
```

2.1 APRUS：接口属性

参数	值	说明
ipmode	"auto"/"manual"/"none"	ip 获取方式
inet_addr	"192.168.1.234"	APRUS 的 IP 地址
netmask	"255.255.255.0"	子网掩码
luaver	"MAX.LUA.V032700.R"	APRUS 的 Lua 版本信息，根据实际脚本自行填写
devinfo	"ppiDev"	与 APRUS 对接设备，可根据需求自行填写

2.2 PPI-Device：接口属性

参数	值	说明
portindex	整型：1/2	RS485-1/2 接口
rate	整型：2400/4800/.../15200	波特率
databit	整型：5/6/7/8	数据位
stopbit	整型：1/2	停止位
parity	字符串：Even/Odd/None	校验

2.3 PPI-ColNode ：采集节点属性

参数	值	说明
ID	整型：1~n	设备 ID
reg	字符串： S/SM/AI/AQ/C/T/I/Q/M/V	寄存器类型
addr	整型：0~n	采集起始地址
ctype(选填)	字符串：byte(默认)/bit	bit:以位为单位采集 Byte:以字节为单位采集

bitn	整型：0~7	位偏移量，在 addr 的基础上做偏移 即采集起始地址=addr*8+bitn
cnt	整型：1~n	采集单位数

2.4 PPI-VarNode:上报节点属性

参数	值	说明
ID	整型：1~n	设备 ID
reg	字符串： S/SM/AI/AQ/C/T/I/Q/M/V	寄存器类型
addr	整型：0~n	数据起始地址
ctype(选填)	字符串：byte(默认)/bit	用于关连对应的采集节点
dtype	字符串： bit/byte/ubyte/short/ushort/int/uint/long/ulong/float/double/bytes	变量类型
len(选填)	整型：1~n	数据长度，仅当数据类型位 bytes 时有效，用于指定字符串长度
bitn(选填)	整型：0~n	位偏移量，当数据类型为 bit 时有效
cycle	整型：0~n	单位：秒　精度：0.1 等于 0：即指定上报类型为改变上报 大于 0：即指定上报周期
name	字符串：上报名称	
offset	+ - * / ~ 数字	例如：offset="*10; +5.123; ~1" 表示乘以 10 再加 5.123 结果保留 1 位小数