APRUS Lua-ppiServer 配置说明

第一章 概述

1.1 文档说明

APRUS 适配器可配置为 PPI 服务器，支持与其他支持 PPI 协议的客户端设备进行通信。APRUS Lua-PPIServer 主要包括两个文件：aprus.lua 和 config.lua。配置为 ppi 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。需要注意的是 ppiServer 共享的时候，通过 PPI 采集的数据是什么类型就以什么类型数据共享。本文将介绍如何将 APRUS 适配器可配置为 ppi 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》

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.path = "./?.lua"
cjson = require "cjson"
config = require "config"
require "ppi"
require "ppiServer"

function ppiServer_load_station(svr, cfg)
	for k, v in pairs(cfg)
	do
		print("ppiServer load Station" .. cjson.encode(v))
		ppiServer.setStation(svr, cjson.encode(v))
	end
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
            ppi.stop(ppiSvrObj)
        elseif code == 1 then
            --ppi.run(ppiSvrObj)
        end
end

function ppi_handle(obj, name, code, data)
        mqtt.publish(mqtt3Obj, 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()
        mqtt3Obj = mqtt.new()
        mqtt.config(mqtt3Obj, "", "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()
	print("<--------------------user lua start------------------------>")
	init()
	print("<--------------------ppi_init()------------------------>")
	ppiSvrObj = ppi.new("ppi")
	ppi.config(ppiSvrObj, cjson.encode(config.PPI.Device))
	ppi_load_collectnodes(ppiSvrObj, config.PPI.ColNode)
	ppi_load_varnodes(ppiSvrObj, config.PPI.VarNode)
	--ppi.debug(ppiSvrObj, 0x01010101)

	print("<--------------------ppiServer_init()------------------------>")
	ppiSvrObj = ppiServer.new("ppiSvrObj")
	ppiServer.config(ppiSvrObj, cjson.encode(config.PPI.Device))
	ppiServer_load_station(ppiSvrObj, config.PPI.ColNode)
	--ppiServer.debug(ppiSvrObj, 0x01010101)
	print("<--------------------ppi.run------------------------>")
	ppi.run(ppiSvrObj)
	print("<--------------------ppiServer.run------------------------>")
	ppiServer.run(ppiSvrObj)
	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 == "ppi" then
			ppi_handle(msg.obj, msg.name, msg.code, msg.data)
		end
	end

	print("<--------------------user lua over------------------------>")
end

start()
```

2.2 此处仅解释一下 ppiServer 相关的函数和操作：

2.2.1 ppiServer.new

功能：创建 ppiServer 实例

示例：ppiSvrObj= ppiServer.new("ppiServer")

参数	值	说明
"ppiServer"		模块运行实例的名称

2.2.2 ppiServer.config

功能：配置 ppiServer 实例

示例：ppiServer.config(ppiSvrObj, cjson.encode(config.PPI.Device))

参数	值	说明
ppiSvrObj		ppiServer. new 实例的返回值
cjson.encode(config.PPI.Device)		ppiServer 配置参数（详见 config.lua）

2.2.3 ppiServer.setStation

功能：配置 Modbus 站点信息

示例：ppiServer.setStation(ppiSvrObj, cjson.encode(stationConfig))

参数	值	说明
ppiSvrObj		ppiServer. new 实例的返回值
stationConfig		ppiServer 的站点配置（详见 config.lua）

2.2.4 ppiServer.run

功能：创建 ppiServer 启动实例

示例：ppiServer.run(ppiSvrObj)

参数	值	说明
ppiSvrObj		ppiServer. new 实例的返回值

2.2.5、 ppiServer_load_station(ppiSvrObj, config.PPI.ColNode)

使 ppiServer 加载 config.lua 中 PPI.ColNode 段配置的 ppiServer 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 ppiServer_load_station(svr, cfg)
    for k, v in pairs(cfg)
    do
        print("ppiServer load Station" .. cjson.encode(v))
        ppiServer.setStation(svr, cjson.encode(v))
    end
end
```

2.2.6、 设置 ppi 实例的调试打印级别

语句	ppi.debug(ppiObj, debugNumber)
说明	启用详细的 zyzn 调试信息。第一个参数为 ppi.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：ppiServer.debug(ppiSvrObj, 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 ppiServer 的 config.Lua 一般只需要配置 PPI 部分，需要在采集结点中含有 dbName 即可

config.lua 示例：

```
PPI = {
        Device = {
            portindex = 1,
            rate = 9600,
            databit = 8,
            stopbit = 1,
            parity = "Even", -- None/Odd/Even
            queryInterval = 100
        },
        ColNode = {
            --bit
            { ID = 2, reg = "V",  addr = 100, cnt = 1, ctype = "bit", bitn = 0, dbName
= "PPI_2V100L0" },
            { ID = 2, reg = "V",  addr = 100, cnt = 1, ctype = "bit", bitn = 1, dbName
= "PPI_2V100L1" },
            { ID = 2, reg = "V",  addr = 100, cnt = 1, ctype = "bit", bitn = 2, dbName
= "PPI_2V100L2" },
            { ID = 2, reg = "V",  addr = 100, cnt = 1, ctype = "bit", bitn = 3, dbName
= "PPI_2V100L3" },
            { ID = 2, reg = "V",  addr = 100, cnt = 1, ctype = "bit", bitn = 4, dbName
= "PPI_2V100L4" },
            { ID = 2, reg = "V",  addr = 100, cnt = 1, ctype = "bit", bitn = 5, dbName
= "PPI_2V100L5" },
            { ID = 2, reg = "V",  addr = 100, cnt = 1, ctype = "bit", bitn = 6, dbName
= "PPI_2V100L6" },
            { ID = 2, reg = "V",  addr = 100, cnt = 1, ctype = "bit", bitn = 7, dbName
= "PPI_2V100L7" },

            { ID = 2, reg = "V",  addr = 101, cnt = 1, ctype = "bit", bitn = 0, dbName
= "PPI_2V101L0" },
            { ID = 2, reg = "V",  addr = 101, cnt = 1, ctype = "bit", bitn = 1, dbName
= "PPI_2V101L1" },
            { ID = 2, reg = "V",  addr = 101, cnt = 1, ctype = "bit", bitn = 2, dbName
= "PPI_2V101L2" },
            { ID = 2, reg = "V",  addr = 101, cnt = 1, ctype = "bit", bitn = 3, dbName
```

```
= "PPI_2V101L3" },
            { ID = 2, reg = "V",  addr = 101, cnt = 1, ctype = "bit", bitn = 4, dbName
= "PPI_2V101L4" },
            { ID = 2, reg = "V",  addr = 101, cnt = 1, ctype = "bit", bitn = 5, dbName
= "PPI_2V101L5" },
            { ID = 2, reg = "V",  addr = 101, cnt = 1, ctype = "bit", bitn = 6, dbName
= "PPI_2V101L6" },
            { ID = 2, reg = "V",  addr = 101, cnt = 1, ctype = "bit", bitn = 7, dbName
= "PPI_2V101L7" },


            { ID = 2, reg = "V",  addr = 506, cnt = 1, ctype = "bit", bitn = 0, dbName
= "PPI_2V506L0" },
            { ID = 2, reg = "V",  addr = 506, cnt = 1, ctype = "bit", bitn = 1, dbName
= "PPI_2V506L1" },
            { ID = 2, reg = "V",  addr = 506, cnt = 1, ctype = "bit", bitn = 2, dbName
= "PPI_2V506L2" },
            { ID = 2, reg = "V",  addr = 506, cnt = 1, ctype = "bit", bitn = 3, dbName
= "PPI_2V506L3" },
            { ID = 2, reg = "V",  addr = 506, cnt = 1, ctype = "bit", bitn = 4, dbName
= "PPI_2V506L4" },
            { ID = 2, reg = "V",  addr = 506, cnt = 1, ctype = "bit", bitn = 5, dbName
= "PPI_2V506L5" },
            { ID = 2, reg = "V",  addr = 506, cnt = 1, ctype = "bit", bitn = 6, dbName
= "PPI_2V506L6" },
            { ID = 2, reg = "V",  addr = 506, cnt = 1, ctype = "bit", bitn = 7, dbName
= "PPI_2V506L7" },
            --bit

            --byte
             { ID = 2, reg = "V",  addr = 100, cnt = 10, ctype = "byte", dbName =
"PPI_2V100L10"},
             { ID = 2, reg = "V",  addr = 506, cnt = 1,  ctype = "byte",  dbName =
"PPI_2V506L1"},
            --byte

            --otherReg
            --{ ID = 2, reg = "S",  addr = 0, cnt = 30,  ctype = "byte",  dbName =
"PPI_2S0L30"},
            --{ ID = 2, reg = "SM",  addr = 0, cnt = 50,  ctype = "byte",  dbName =
"PPI_2SM0L50"},
        },
    },
```

PPI 服务器的 config.lua 和 PPI 的一致，无需额外配置。

数据中心相关知识，请参考《APRUS 编程手册_数据中心.pdf》。